The VLDB Journal (2019) 28:351–375
https://doi.org/10.1007/s00778-019-00540-5

REGULAR PAPER

Incremental maintenance of maximal cliques in a dynamic graph

Apurba Das[1] · **Michael Svendsen**[1] · **Srikanta Tirthapura**[1]

Received: 5 March 2018 / Revised: 21 January 2019 / Accepted: 2 April 2019 / Published online: 12 April 2019

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 when edges are added to the graph, as well as the first change-sensitive algorithm for incremental clique maintenance under edge additions, whose runtime is proportional to the magnitude of the change in the set of maximal cliques, when the number of edges added is small. Our algorithm can also be applied to the decremental case, when edges are deleted from the graph. We present experimental results showing these algorithms are efficient in practice and are faster than prior work by two to three orders of magnitude.

Keywords Graph mining · Maximal clique · Incremental algorithm · Dynamic graph

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 [6] through mining maximal cliques in an appropriately 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 [23], identification of groups of closely linked people in a social network [19,28,31], identification of web communities [18,27,38], and even in the construction of the phylogenetic tree of life [11,39,52].

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 long-standing 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. [6] on maintaining common intervals

✉ Apurba Das
adas@iastate.edu

Michael Svendsen
michael.sven5@gmail.com

Srikanta Tirthapura
snt@iastate.edu

[1] Iowa State University, Ames, USA

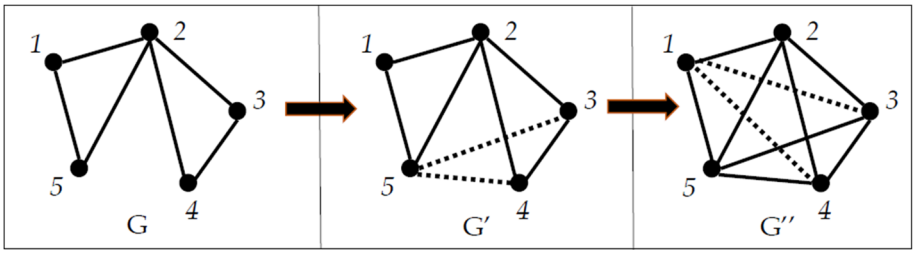


Fig. 1 Change in maximal cliques due to addition of edges. On the left is the initial graph G with maximal cliques $\{1, 2, 5\}$ and $\{2, 3, 4\}$; on the middle is the graph G' after adding edges $(3, 5)$ and $(4, 5)$ to G resulting in new maximal clique $\{2, 3, 4, 5\}$ and only subsumed maximal clique $\{2, 3, 4\}$; 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 $\{2, 3, 4, 5\}$

among genomes, Duan et al. [12] on incremental k-clique clustering, Hussain et al. [22] 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 Fig. 1 for an example. Let $\Lambda^{new}(G, G') = \mathcal{C}(G')\backslash\mathcal{C}(G)$ denote the set of maximal cliques that were newly formed when going from G to G', and $\Lambda^{del}(G, G') = \mathcal{C}(G)\backslash\mathcal{C}(G')$ 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 $\Lambda(G, G')$? Can we 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 [34] 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 [34] 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 our results on incremental algorithms. Results for decremental algorithms are similar. The key algorithmic contributions in this work are as follows:

(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 for enumerating the change (in the set of maximal cliques) is based on an exploration of a carefully chosen 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 to enumerate the change, either for enumerating new maximal cliques, or for subsumed maximal cliques. The key aspect of the algorithm is that through exploring this subgraph, it is able to directly "zero in" on maximal cliques that have changed (either added or subsumed). This approach reduces wasteful effort in enumeration, when compared to an approach that iterates through the set of existing maximal cliques. Based on our 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 Sect. 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 the order of a hundred thousand vertices and millions of edges within a few seconds. Our comparison with prior and recent works shows that `IMCE` significantly outperforms prior solutions, including the ones due to Stix [42], Ottosen and Vomlel [37], and Sun et al. [43]. For example, on the `flickr-growth` graph, our algorithms are faster than those in [37,42,43] 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 min to 2 h. Further details are in Sect. 6.

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. [48], leading to worst-case optimal time complexity $O(3^{n/3})$ for an n-vertex graph [34]. Another work on refinements of [4,48] includes [25], who presents several strategies for pivot selection to enhance the algorithm in [4], and a fixed parameter tractable algorithm parameterized by the graph degeneracy [13,14].

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 [8,32,50], 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 [32] 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 $n \times n$ matrices, which is $O(n^{2.376})$. Further work in this direction includes [26] and [24], which consider the enumeration of maximal independent sets in lexicographic order, [7], which considers the external memory model, and [36], which considers uncertain graphs. Extensions to parallel frameworks such as MapReduce, MPI, or shared memory are presented in [9,35,44]. Note that there is a long line of prior work on finding the maximum clique in a graph, e.g., [46,47,49] on finding a maximum clique in a graph. However, these algorithms are not directly useful to our work on enumerating maximal cliques, since the maximum clique is a related, but different concept. While every maximum clique is a maximal clique, there may be maximal cliques that are not maximum.

Maximal Clique Enumeration in a Dynamic Graph. In [42], 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 [42] 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 [37] 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 to our work, there are no provable performance bounds for this algorithm. Another difference is that the algorithm of [37] 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. [43] 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 max-

imal 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 [30,40], k-truss communities [20], densest subgraph [3,33], and maximal bicliques in a bipartite graph [10]. The main differences between this work and [10] are that the latter focuses on bicliques, which are complete structures in a bipartite graph, while our work focuses on maximal cliques. The theory of the size of change in the number of substructures is substantially different between cliques and bicliques. Further, the manner in which the change in the set of substructures is derived is different in the two cases. For instance, when new edges are added, the change in the set of maximal cliques only involves vertices that are in the 1-neighborhood of the newly added edges, but this is not necessarily the case with bicliques. Other structures such as 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. In [1], the definition of a dense structure is based on the weights of the edges, and dense subgraphs are defined as those whose total weight exceeds a certain threshold. Clearly, maximal cliques are different than such structures, due to their requirement of complete connectivity among vertices.

Roadmap: We present preliminaries in Sect. 2, followed by bounds on magnitude of change in Sect. 3, algorithms for enumerating the change in Sect. 4, discussions in Sect. 5, and experimental results in Sect. 6.

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.

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 due to a change in the set of edges of the graph is linear in the magnitude of change (in P), and polynomial in the size of the input graph, and the number of edges added to or deleted from the graph.

By the phrase "magnitude of change," we mean the number of structures that have changed with respect to the property P. Note that the notion of a change-sensitive algorithm for a dynamic graph is similar to the notion of an "output-sensitive" algorithm for a static graph, whose time complexity depends on the size of the output and the size of the graph.

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. [41] is an incremental algorithm for graph connectivity, an algorithm due to Thorup [45] is a decremental algorithm, and one due to Wulff-Nilsen [51] is a fully dynamic algorithm. We present change-sensitive incremental and decremental algorithms for maximal clique maintenance. Our fully dynamic algorithm for maximal clique maintenance is, however, not change-sensitive.

Results for Static Graphs: We present some known results about maximal cliques on static graphs. Nearly 50 years ago, Moon and Moser [34] 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 [34])

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

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. [50], and due to Makino and Uno [32]. We use the following result (Theorem 2) 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 [32], and our algorithm can use other methods as a subroutine also. Let the arboricity of a graph be defined as the minimum number of forests into which the edges of the graph can be partitioned. The arboricity of a graph is no more than the maximum vertex degree, but could be significantly smaller [8].

Theorem 2 (Chiba and Nishizeki [8]) *There is an algorithm* MCE(G) *that enumerates all maximal cliques in graph G in time $O(\alpha m \mu)$ where μ is the number of maximal cliques in G, and α and m are, respectively, the arboricity of G and the number of edges in G. The space complexity of the algorithm is $O(n+m)$, where n is the number of vertices in G.*

Note that the space complexity excludes the size of the output, which may be much larger.

3 Magnitude of change

From prior work [34], the maximum number of maximal cliques in an n-vertex graph, denoted by $f(n)$, is known (see Theorem 1). The result of [34] 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 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

$$\begin{aligned} \frac{16}{9} f(n) \le \lambda(n) < 2f(n) \quad & \text{if} (n \mod 3) = 0 \\ \lambda(n) = 2f(n) \quad & \text{if} (n \mod 3) = 1 \\ \frac{11}{6} f(n) \le \lambda(n) < 2f(n) \quad & \text{if} (n \mod 3) = 2 \end{aligned}$$

Proof We first note that $\lambda(n) \le 2f(n)$ 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 $|\mathcal{C}(G)| \le f(n)$ and $|\mathcal{C}(G+H)| \le f(n)$. Since $|\Lambda^{new}(G, G+H)| \le |\mathcal{C}(G+H)|$ and $|\Lambda^{del}(G, G+H)| \le |\mathcal{C}(G)|$, we have $|\Lambda(G, G+H)| = |\Lambda^{new}(G, G+H)| + |\Lambda^{del}(G, G+H)| \le |\mathcal{C}(G)| + |\mathcal{C}(G+H)| \le 2f(n)$.

The result of Moon and Moser [34] states that for $n \ge 2$, 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 in which $(n \mod 3)$ 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).

Next, we 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 Fig. 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

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

We add edge set H to the graph 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

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

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

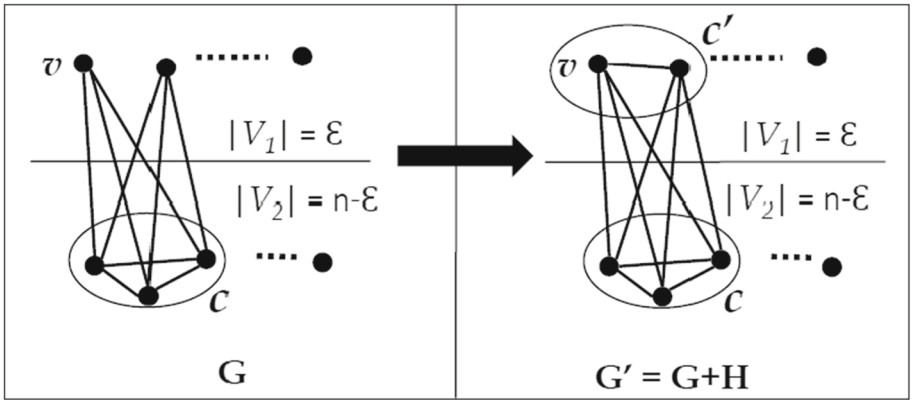


Fig. 2 A large change in set of maximal cliques when a few edges are added. The vertex set is partitioned into V_1 and V_2. On the left is G, the original graph on n vertices where each vertex in V_1 is connected to each vertex in V_2, and V_1 is an independent set. In G, the induced subgraph G_2 on 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 $c \cup c'$

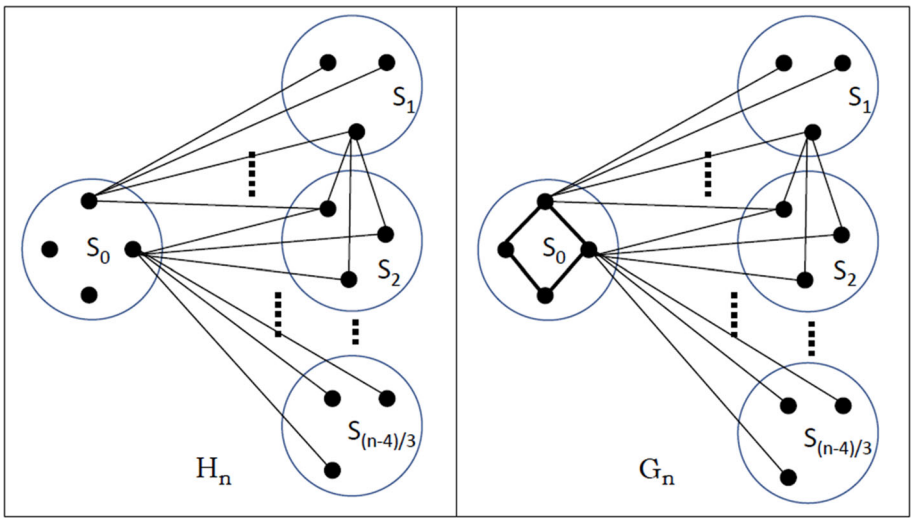


Fig. 3 On the left is H_n where each vertex v in S_i is connected to each vertex u in $S_j, i \neq j$. On the right is G_n which is formed from H_n by adding four edges to S_0. For the case $(n \mod 3) = 1$, H_n and G_n are non-isomorphic graphs on n vertices, with $f(n)$ maximal cliques each, showing a counterexample to Theorem 2 of Moon and Moser [34]

$G'(V_1)$ 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

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

Putting together Eqs. 1, 2, and 3 we get

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

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. □

3.2 An error in a result of Moon and Moser (1965)

Moon and Moser [34], in Theorem 2 of 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 Fig. 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 [34], 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, 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* [34]*, 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 Sect. 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 Sect. 4.2. In Sect. 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 Sect. 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 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 [8], suppress cliques that were also present in G, and output the rest. The above approach is not change-sensitive. To see why, consider a case when the initial graph G is the union of a Moon–Moser graph on $(n-3)$ vertices, along with three isolated vertices a, b, and c. Suppose three edges are added in H so that vertices $\{a, b, c\}$ form a triangle. In going from graph G to $G+H$, the set of maximal cliques has changed as follows: A new clique $\{a, b, c\}$ has been formed and three existing (trivial) cliques $\{a\}$, $\{b\}$, and $\{c\}$ have been subsumed. Following the above approach, all cliques in G and in $G+H$ are enumerated, which has a cost of $\Omega(3^{\frac{n}{3}})$, since there are $\Theta(3^{\frac{n}{3}})$ cliques in G and in $G+H$. The size of change is small (a constant), while the cost of enumeration is very large (exponential in the number of vertices). Approaches that involve enumerating maximal cliques in a certain graph, followed by suppressing cliques that do not belong to Λ^{new}, run the 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 that leads to a change-sensitive algorithm for new cliques. At its core, our algorithm constructs a set of subgraphs of $G+H$ such that each maximal clique in any of these subgraphs is a new maximal clique, i.e., belongs to $\Lambda^{new}(G, G')$. Further, each element of $\Lambda^{new}(G, G')$ is a maximal clique in one of these subgraphs. This construction allows the algorithm to directly output cliques from $\Lambda^{new}(G, G')$, without enumerating cliques that do not belong to $\Lambda^{new}(G, G')$. This can form the basis of a change-sensitive algorithm. There is an additional duplicate elimination step in our algorithm, whose goal is to only to suppress enumerating the same clique multiple times.

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 $\cup_{e \in H} C'(e) \subseteq \Lambda^{new}$. □

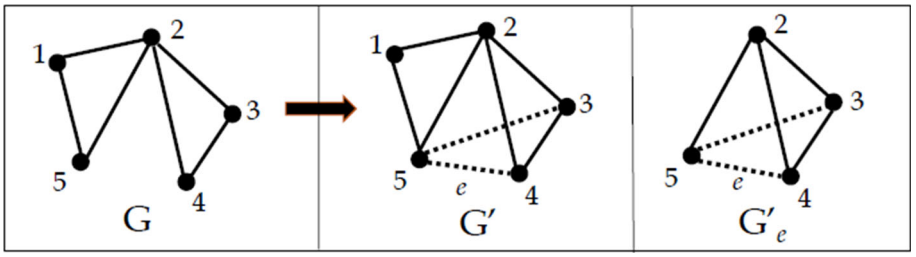


Fig. 4 Illustration of Lemma 2 that, the set of new maximal cliques in G' containing $e = (4, 5)$, i.e., the single clique $\{2, 3, 4, 5\}$, is exactly the set of all maximal cliques in G'_e

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 Fig. 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 both to 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 Fig. 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 [8], 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

```
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)
    V_e ← {u, v} ∪ {Γ_G'(u) ∩ Γ_G'(v)}
    G'_e ← graph induced by V_e on G'
    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
```

We now present the time–space complexity analysis of `IMCENewClq`. Our analysis shows that `IMCENewClq` is a change-sensitive algorithm for enumerating new maximal cliques, since its time complexity is polynomial in the maximum degree Δ and linear in the number of new maximal cliques $|\Lambda^{new}|$ and in the number of new edges ρ.

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 may be 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 predetermined ordering of edges in H.

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 `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 $O(\Delta^3 |\Lambda^{new}|)$.

Within each iteration, the time taken to generate the subgraph $G'(e)$ is $O(\Delta^2)$, which is dominated by the term $O(\Delta^3|\Lambda^{new}|)$. For each new edge added, there must be a new maximal clique that contains this edge. Hence, as long as $\rho > 0$, i.e., at least one new edge is added, Λ^{new} is a non-empty set. The overall runtime of each iteration is bounded by $O(\Delta^3|\Lambda^{new}|)$. 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 MCE(G'_e), from Theorem 2. Hence, the total space requirement is linear in the number of edges in $G + H$. □

4.2 Practical algorithm for enumerating new maximal cliques

Now we present an efficient algorithm FastIMCENewClq for enumerating new maximal cliques when new edges are added. This is based on the theoretically efficient algorithm IMCENewClq and incorporates improvements that we discuss next.

The algorithm IMCENewClq uses as a subroutine Algorithm MCE (Chiba and Nishizeki [8]) to enumerate maximal cliques within a subgraph of G. While MCE is theoretically output-sensitive, in practice, it is not the most efficient algorithm for maximal clique enumeration. The most efficient algorithms for maximal clique enumeration in a static graph are typically based on depth-first search using a technique called "pivoting," such as the algorithm due to Tomita et al. [48]. While the runtime of these algorithms are not provably output-sensitive, they are faster in practice than those algorithms that are provably output-sensitive.

In particular, the algorithm due to Tomita et al. [48], which we call TTT, is a recursive algorithm based on backtracking that takes as input a graph G and enumerates $\mathcal{C}(G)$. In its recursive procedure, TTT maintains a currently found clique c, not necessarily maximal. It tries to extend c to get a larger clique, declaring c to be maximal when no further vertices can be added. Vertices are considered for addition to c in a carefully chosen order, and cliques that were enumerated in prior recursive calls are not explored again through the maintenance of a set of vertices whose exploration is complete. The recursive search procedure backtracks upon reaching a maximal clique and continues until all maximal cliques are enumerated. TTT is shown to be worst-case optimal with a runtime of $O(3^{n/3})$ for an n vertex graph [48]. It is possible to directly improve the performance of the IMCENewClq algorithm by using TTT in place of MCE. In the following, we show how to do even better.

Reducing Redundant Clique Computation: Note that IMCENewClq (Algorithm 1) may compute the same clique c multiple times. For example, if $c \in C'(e_1)$ and $c \in C'(e_2)$ for two distinct edges e_1 and e_2, c will be enumerated (at least) twice, once when considering e_1 in the for loop, and once while considering edge e_2. In line 7, duplicates are suppressed prior to emitting the cliques, by outputting c only for one of the edges among $\{e_1, e_2\}$. However, the algorithm still pays the computational cost of computing a clique such as c multiple times.

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

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

```
1  if (cand = ∅) & (fini = ∅) then
2      Output K and return
3  pivot ← (u ∈ cand ∪ fini) such that u maximizes the size of cand ∩ Γ_G(u)
4  ext ← cand − Γ_G(pivot)
5  for q ∈ ext do
6      K_q ← K ∪ {q}
7      if K_q ∩ E ≠ ∅ then
8          cand ← cand − {q} ; fini ← fini ∪ {q}
9          continue
10     cand_q ← cand ∩ Γ_G(q) ; fini_q ← fini ∩ Γ_G(q)
11     TTTExcludeEdges(G, K_q, cand_q, fini_q, E)
12     cand ← cand − {q} ; fini ← fini ∪ {q}
```

We now present a method, Algorithm FastIMCENewClq, to avoid such redundant clique computation. The idea is to consider the edges in H in a specific order $e_1, e_2, \ldots$. When enumerating all cliques in $C(e_i)$, the algorithm prunes out search paths that lead to cliques containing edge e_j, $j < i$. This way, each new clique is enumerated exactly once.

For this purpose, Algorithm FastIMCENewClq uses as a subroutine Algorithm TTTExcludeEdges (Algorithm 2), an extension of the TTT algorithm, which enumerates all maximal cliques of an input graph that avoid a given set of edges. While TTT simply takes a graph as input and enumerates all maximal cliques within the graph, 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 a recursive version of TTTExcludeEdges, which takes as input five parameters—an input graph G, three sets of vertices K, cand, and fini, and a set of edges $\mathcal{E}$. The algorithm outputs every maximal clique in G that contains (a) all vertices in K, (b) zero or more vertices in cand, (c) none of the vertices in fini, and (d) none of the edges in $\mathcal{E}$.

Algorithm 3: `FastIMCENewClq`(G, H)

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

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 $\quad e \leftarrow e_i = (u, v)$
5 $\quad V_e \leftarrow \{u, v\} \cup \{\Gamma_{G'}(u) \cap \Gamma_{G'}(v)\}$
6 $\quad \mathcal{G} \leftarrow$ Graph induced by V_e on G'
7 $\quad K \leftarrow \{u, v\}$
8 $\quad$ `cand` $\leftarrow V_e \backslash \{u, v\}$; `fini` $\leftarrow \emptyset$
9 $\quad S \leftarrow$ `TTTExcludeEdges`$(\mathcal{G}, K,$ `cand`, `fini`, $\mathcal{E})$
10 $\quad \Lambda^{new} \leftarrow \Lambda^{new} \cup S$
11 $\quad \mathcal{E} \leftarrow \mathcal{E} \cup e_i$

A description of `TTTExcludeEdges` is presented in Algorithm 2, and an example of its output is presented in Fig. 5. 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 `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.

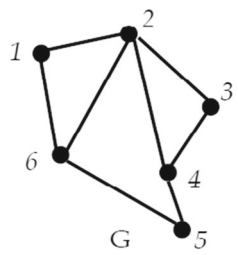
Our algorithm for enumerating new maximal cliques `FastIMCENewClq` (Algorithm 3) is an adaptation of `IMCENewClq` (Algorithm 1) where we use `TTTExcludeEdges` instead of the output-sensitive `MCE`. In particular, while enumerating all new cliques containing edge e_i, `FastIMCENewClq` enumerates only those cliques that exclude edges $\{e_1, e_2 \ldots, e_{i-1}\}$. Note that in `FastIMCENewClq`, there is no further duplicate suppression required, since the call to `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 `FastIMCENewClq` more efficient in practice than `IMCENewClq`.

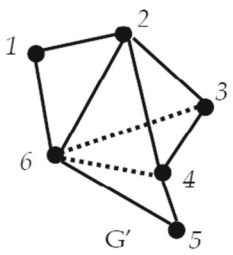
The correctness of `FastIMCENewClq` follows in a similar fashion to that of Algorithm `IMCENewClq` proved in Theorem 4, except that we also need a proof of the guarantee provided by Algorithm `TTTExcludeEdges`, which we establish in the following lemma.

Lemma 3 `TTTExcludeEdges`$(\mathcal{G}, K,$ `cand`, `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* `cand`*, (3) c contains no vertex from* `fini`*, and (4) c does not contain any edges in $\mathcal{E}$.*

Proof We note that `TTTExcludeEdges` matches the original `TTT` algorithm, except for lines 7 to 9. Hence, if we do not consider lines 7 to 9 in `TTTExcludeEdges`, the algorithm becomes `TTT`, and by the correctness of `TTT` ([48, Theorem 1]), all maximal cliques c in $\mathcal{G}$ are returned. Now consider lines 7 to 9 in `TTTExcludeEdges`. Clearly, (1), (2), (3) are preserved for each maximal clique c generated by `TTTExcludeEdges`. Now to complete the correctness proof of `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 `TTTExcludeEdges`. Assume there exists a maximal clique c in $\mathcal{G}$, which contains an edge in $\mathcal{E}$, which is output by the `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.

Next, we show 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 `TTTExcludeEdges`. The only reason for c not being generated is the inclusion of lines 7 to 9 (of `TTTExcludeEdges`) to `TTT` resulting in `TTTExcludeEdges`, because, otherwise, c would be generated due to correctness of `TTT`. So in





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 from G to G' due to addition of new edges (3, 6) and (4, 6). Order the new edges as (3, 6) followed by (4, 6). There are two new maximal cliques containing edge (4, 6), {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)

TTTExcludeEdges, during the expansion of K toward c, there exists a vertex $q \in c$ such that line 7 in TTTExcludeEdges is satisfied and c never gets a chance to be generated as q is excluded from cand and included in 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 and completes the proof. □

4.3 Enumeration of subsumed maximal cliques

We now consider the enumeration of subsumed cliques, i.e., the set $\mathcal{C}(G)\backslash\mathcal{C}(G+H)$. 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.

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 to storing the clique itself. The procedure is as 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 that the signatures of two different cliques C_1 and C_2 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.[1] 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 describe 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 $c - H$, 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 using an induction on the number of edges in H those are present in c:

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

Proof Note that we only consider the set of all edges $H_1 \subseteq H$ which are present in c (line 4). It is clear that computing maximal cliques in $c-H$ is equivalent to computing maximal cliques in $c - H_1$.

We prove the lemma using induction on k, the number of edges in H_1. Suppose $k = 1$ so that H_1 is a single edge, say $e_1 = \{u, v\}$. Note that $c-H_1$ has two maximal cliques, $c\backslash\{u\}$ and $c\backslash\{v\}$. It can be verified that in Algorithm 4 cliques $c\backslash\{u\}$ and $c\backslash\{v\}$ are inserted into S, thus 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 generate 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. In the following lemma (Lemma 5), we present the time and space complexity of IMCESubClq where we use an induction on ρ for proving the time complexity. Note that the time complexity is change-sensitive when ρ is a constant because the time complexity is linear on the size of Λ^{new}.

Lemma 5 *Algorithm* IMCESubClq *(Algorithm 4) enumerates all cliques in $\Lambda^{del} = \mathcal{C}(G)\backslash\mathcal{C}(G')$ 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'

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

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

```
Input: G - Input Graph
       H - Edge set being added to G
       D - Set of maximal cliques in G
       Λ^new - set of new maximal cliques in G + H
Output: All cliques in Λ^del = C(G)\C(G + H)
Λ^del ← ∅
for c ∈ Λ^new do
    S ← {c}
    for e = (u, v) ∈ E(c) ∩ H do
        S' ← φ
        for c' ∈ S do
            if e ∈ E(c') then
                c1 = c'\{u} ; c2 = c'\{v}
                S' ← S' ∪ c1 ; S' ← S' ∪ c2
            else
                S' ← S' ∪ c'
        S ← S'
    for c' ∈ S do
        if c' ∈ D then
            Λ^del ← Λ^del ∪ c'
            D ← D\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 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, $c\backslash\{u\}$ and $c\backslash\{v\}$, 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 clique $c \in \Lambda^{new}$, 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.

Algorithm 5: IMCED(G, H)

```
Input: G - Input Graph, H - Set of ρ edges being deleted
Output: All cliques in Λ^new(G, G − H) ∪ Λ^del(G, G − H)
Λ^new ← ∅, Λ^del ← ∅, G'' ← G − H
Λ^del ← IMCENewClq(G'', H)
Λ^new ← IMCESubClq(G'', H, C(G''), Λ^del)
```

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

Next, we consider the case when a set of edges H is deleted from G. A set of edges H is deleted from graph G, 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 algorithm for maximal cliques is outlined in Algorithm 5 (IMCED).

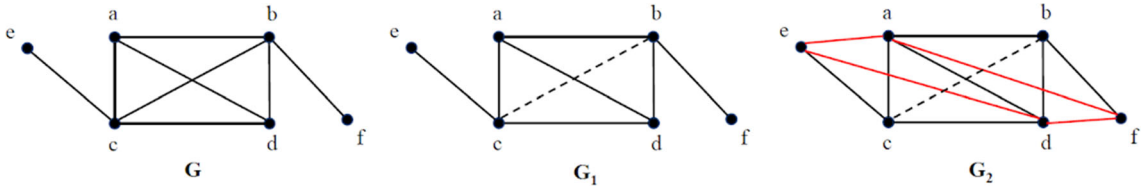


Fig. 6 Change in the maximal cliques due to both addition and deletion of edges. The initial graph G, graph G_1 after deleting edge (b, c) from G, resulting in new maximal cliques $\{a, c, d\}$ and $\{a, b, d\}$ and one deleted maximal clique $\{a, b, c, d\}$, graph G_2 after adding edges (a, e), (e, d), (a, f), and (d, f) from G_1 resulting in new maximal cliques $\{a, e, d, c\}$ and $\{a, b, d, f\}$ and subsumed cliques $\{a, c, d\}$, $\{a, b, d\}$, $\{c, e\}$, and $\{b, f\}$. Note that the intermediate new cliques (at state G_1) $\{a, c, d\}$ and $\{a, b, d\}$ are only "transient" maximal cliques and are not in the final graph G_2

4.5 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. See Fig. 6 for an example.

5 Discussion

Our incremental algorithm `IMCE` can be adapted to related problems such as maintaining top-k maximal cliques. Also, the techniques developed in the work can be used for the maintenance of maximal cliques with additional search context such as graph with labels at nodes or vertices.

Maintenance of top-k maximal cliques: Observe that the vertices that correspond to the top-k maximal cliques are of a high degree. More precisely, if the smallest size of the clique among the current top-k maximal cliques is s, then we only need to consider the vertices of the original graph whose degree is at least $(s - 1)$. Thus, given top-k maximal cliques of G, we can update the top-k maximal cliques of the graph $G' = G + H$ using our incremental algorithm as follows: (1) For computing new maximal cliques, we only enumerate those with size at least s. We can do this by recursively deleting vertices of degree smaller than $s - 1$ from the subgraph used ($\mathcal{G}$ at line 6 of Algorithm 3), adding these vertices to the `fini` set (line 8 of Algorithm 3), and then enumerating maximal cliques of the updated graph containing the rest of the vertices (by adding them to the `cand` set at line 8 of Algorithm 3). This ensures that each maximal clique such enumerated is of size at least s. (2) It is possible that some of the maximal cliques in top-k may be subsumed by larger new maximal cliques when new edges are added. The algorithm for subsumed cliques can deal with this situation in the following manner: Instead of checking for the containment in the set of maximal cliques of the original graph, check for the containment in the set of top-k maximal cliques (line 14 of Algorithm 4 where the set C contains only top-k maximal cliques of G instead of all maximal cliques of G). Eventually, as a result of subsumption, it might happen that the number of maximal cliques is less than k in the final set. For handling this situation, it is required to generate the set of all new maximal cliques and sort them in decreasing order of size.

Maintenance of maximal cliques with search context: Search contexts are relevant in "keyword"-based or "topic"-based searches, or in the combinations of the two, such as finding communities with people interested in a specific topic [17,21]. The network in this context contains labels attached to nodes/edges. We can use `IMCE` for maintaining maximal cliques such that each of the nodes in the maximal cliques contains one or a group of specified keywords/labels. There might be two cases.

First, consider the case when vertices contain labels. Here, we should consider only those vertices to add to `cand` set (line 8 of Algorithm 3) from the graph G'_e that contains the specified labels and put rest of the vertices in `fini` set. This way, we can use `IMCE` for maintaining only those maximal cliques that contain specified labels to the vertices. Next, we consider the case when edges contain labels. For generating maximal cliques with each of the edges containing specified labels, we consider Algorithm 2 and add an additional check for constraints on edge labels whenever we add a vertex q to K (in line 6 of the algorithm).

Boundedness of incremental computation: In the context of a recent theoretical framework for incremental graph algorithms [16], the time complexity of `IMCE` is *bounded* when the size of the batch ρ is fixed. Our analysis shows that when the original graph is large and the changes in the graph are

small, the time complexity of computing the change is proportional to the size of the change in the set of maximal cliques, which follows the definition of bounded computation as defined in [16]. Also, IMCE admits *localizable computations* [16] as we focus on the subgraphs local to the changes in the graph structure for enumerating the changes.

6 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 computation time and memory usage of our algorithms? (2) How does the computation time compare with the magnitude of the change when new edges are added (incremental algorithm), when existing edges are deleted (decremental algorithm), and in the fully dynamic case, when edges are both added and deleted? (3) What is the impact on the computation time of our incremental algorithm when the stream of new edges are located around high-/low-degree vertices of the original graph? (5) In the incremental case, can we achieve a space–time trade-off in subsumed clique computation, depending on whether or not we store the (signatures) of the set of maximal cliques? (6) How do our algorithms compare with prior works?

6.1 Datasets

We consider graphs from the Stanford large graph database [29], KONECT—The Koblenz Network Collection,[2] and Network Repository[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. ca-cit-HepTh is a citation network in high-energy physics theory in a period from January 1993 to April 2003 where each vertex represents a paper and there is an edge from "a" to "b" if paper "a" cited paper "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 $wiki_1$. facebook-friendship is a friendship network where vertex represents user 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. soc-livejournal is a social network of LiveJournal where vertex represents user and there is an edge between two users when they are friends. As the original graph does not contain time stamps at its edges, we synthetically generate time stamps of edges by assigning an integer uniformly chosen at random between 0 and the number of edges in the network to each edge. A summary of the graphs used in this experiment is given in Table 1. In our experiments, for incremental computation we start with the empty graph and at each iteration, we add a batch of new edges (in the increasing order of time stamps) and enumerate the change in maximal cliques after the addition. For decremental computation, we start with the original graph and in each iteration, delete a batch of existing edges (in the decreasing order of time stamps), and enumerate the change in maximal cliques after the deletion.

Next, we generate three synthetic RMAT [5] graphs. An RMAT-n-m graph has n vertices and m edges. We use RMAT-50K-5M and RMAT-100K-10M for addressing graphs with specific edge stream patterns (edge stream around high-/low-degree vertices) and a high-density graph RMAT-100-4000 for addressing the behavior of the maintenance algorithms with the change in the density of the graph.

For generating the edge stream for the fully dynamic case, we used the first two RMAT graphs. We first randomly assign a label of either 0 (for edge deletion) or 1 (for edge addition) to each edge. We then assign a randomly chosen time stamp to each edge of the graph. For creating the initial graphs RMAT-50K-5M-INIT and RMAT-100K-10M-INIT, we remove all the edges from the original graph those are marked 1. We then arrange all edges in increasing order of time stamps for creating the edge stream for RMAT-50K-5M-INIT. For RMAT-100K-10M-INIT, we order all the edges by grouping them based on the source vertex. Note that a batch of edges contains a mix of new edges to add and existing edges to delete. For experimenting with the stream of edges around high-degree vertices, we choose the 1000 highest degree vertices of the initial graph and consider all the edges with a label of 1 that are attached to at least one high-degree vertex. Similarly for experimenting with the stream of edges around low-degree vertices, we choose the 10,000 lowest degree vertices of the initial graph and consider all the edges with label 1 that are attached to at least a low-degree vertex. For creating the edge stream of RMAT-100-4000, we follow an approach similar to soc-livejournal.

We also consider 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 [15], and we "plant" cliques of a certain size. We call these graphs ER-1M-20M with 1M

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

[3] http://networkrepository.com/.

Table 1 Input graphs and their aggregate statistics

Dataset	Nodes	Edges	Density	# Maximal cliques	Maximum degree	Degeneracy
dblp-coauthor	1,282,468	5,179,996	6.3×10^{-6}	1,219,320	1, 522	118
flickr-growth	2,302,925	22,838,276	8.6×10^{-6}	> 400B	27, 937	600
wikipedia-growth	1,870,709	36,532,531	2.08×10^{-5}	131,652,971	226, 073	206
soc-livejournal	4,033,137	27,933,062	3.4×10^{-6}	38,413,665	2, 651	213
ca-cit-HepTh	22,908	2,444,798	0.0093	> 400B	8, 718	561
facebook-friendship	63,731	817,035	4×10^{-4}	1,539,038	1, 098	52
RMAT-100-4000	100	4,000	0.8	10,180	99	65
RMAT-50K-5M	50K	5M	0.004	232,400,002,455	10, 496	328
RMAT-100K-10M	100K	10M	0.002	144,600,002,154	15, 408	371
ER-1M-20M	1M	20M	4×10^{-5}	19,978,809	81	29
ER-2M-15M	2M	15M	7.5×10^{-6}	14,998,954	59	29

vertices and 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.

6.2 Experimental setup and implementation details

We implemented all the algorithms in Java on a 64-bit Intel(R) Xeon(R) CPU with 16G DDR3 RAM with 13G JVM heap memory.

Algorithm Implementations: We first evaluate our incremental algorithm IMCE for maintenance of maximal cliques when new edges are added. IMCE consists of FastIMCENewClq for enumerating new maximal cliques and IMCESubClq for enumerating subsumed maximal cliques. We also implemented the theoretically efficient algorithm IMCENewClq for enumerating new maximal cliques. Since FastIMCENewClq performed better in all cases, we present results for FastIMCENewClq. We also implemented a variant of IMCENewClq by replacing MCE with TTT and name this variant as IMCENewClqTTT.

We evaluate a variant of IMCE where we use a different strategy for computing subsumed cliques in IMCESubClq. Note that deciding subsumed cliques by checking if it is in the set of all maximal cliques of the graph before update requires us to store the set of maximal cliques (or their signatures) of the original graph as in IMCESubClq. In this variant, we modify line 14 of IMCESubClq where instead of checking for containment, we directly check for maximality of each c' (line 13 of algorithm IMCESubClq). We name this variant of IMCE as IMCE − NoCliqueStore. IMCE − NoCliqueStore uses less memory than IMCE, but has to pay an additional overhead to check for maximality for each candidate subsumed clique.

Next, we evaluate Algorithm 5 (IMCED) which handles the decremental case. When the graph changes to $G - H$ starting from G due to the deletion of a batch H, we compute the new maximal cliques and delete maximal cliques following Observation 2. We also experimentally evaluate the fully dynamic case with a mixture of addition and deletion of edges. For dealing with the fully dynamic case, we first remove all edges that are both added and deleted (as these edges do not contribute to the change in the set of maximal cliques). We then run IMCED for computing the changes due to the deletion of edges, followed by IMCE − NoCliqueStore for computing the changes due to the addition of edges. Finally, we generate the overall changes in the set of maximal cliques due to the deletion and addition of edges.

We consider the following prior algorithms for comparison with IMCE: (1) STIX (Stix [42]) computes on a dynamic graph by incrementally adding one edge at a time; (2) OV (Ottosen and Vomlel [37]) computes on a dynamic graph by incrementally adding a set of edges; and (3) MCMEI (Sun et al. [43]) computes on a dynamic graph by incrementally adding one edge at a time. We also consider the following prior algorithms for comparison with our decremental algorithm IMCED: (1) STIXD (Stix [42]) computes on a dynamic graph by deleting one edge at a time and (2) MCMED (Sun et al. [43]) computes on a dynamic graph by deleting one edge at a time. For the algorithms (STIX, MCMEI, STIXD, MCMED) that support only single edge addition/deletion, we simulate the addition (deletion) of a batch of edges by inserting (deleting) the edges one at a time. We also compare IMCE and IMCED with baseline algorithms Naive and NaiveD, respectively, where Naive handles the incremental case by running a static algorithm TTT each time a set of new edges is added to the graph and explicitly computing the symmetric difference; NaiveD similarly handles the decremental case

by running TTT each time a set of existing edges is deleted from the graph.

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. For defining the size of change in the theoretical analysis (Sect. 4), we used the total number of cliques that were added and deleted. We call this metric "change-in-number." Note that there are other natural ways to quantify the size of change. If one were to actually enumerate the change, each clique that is added (or deleted) could be written as a set of its constituent vertices. Hence, it is natural to consider another metric for the size of change, equal to the sum of the sizes of all cliques that are a part of the change. We call this metric "change-in-nodes." We further consider another metric "change-in-edges," defined as the sum of the numbers of edges in all the cliques that are a part of the change. For example, suppose there are two new maximal cliques of sizes 3 and 4, and one subsumed clique of size 2. The change-in-number is 3, since there are a total of three cliques to enumerate. The change-in-nodes is $3+4+2=9$. The change-in-edges is $\binom{3}{2}+\binom{4}{2}+\binom{2}{2}=10$, since a clique on k vertices has $\binom{k}{2}$ edges. We consider all three metrics, change-in-number, change-in-nodes, and change-in-edges, to measure the size of change. (3) Memory 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.

6.3 Discussion of experimental results

Incremental computation time: Figure 7 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 $\rho = 1000$. The size of the change is shown on the left y-axis, and the time for computing the change is shown on the right y-axis. We see that the time for computing the change in the set of maximal cliques becomes greater as iterations progress for graphs flickr-growth, soc-livejournal, and facebook-friendship and remains roughly the same for other graphs. Figure 8 shows the breakdown of computation time of IMCE into computation time for new maximal cliques (FastIMCENewClq) and computation time for subsumed maximal cliques (IMCESubClq).

Strategies for subsumed cliques: Next, we compare the computation time of IMCE with IMCE − NoCliqueStore as shown in Fig. 9. Clearly, IMCE is faster than IMCE − NoCliqueStore, because cost of checking for maximality in computing subsumed cliques as in IMCE − NoCliqueStore is higher than that of check-

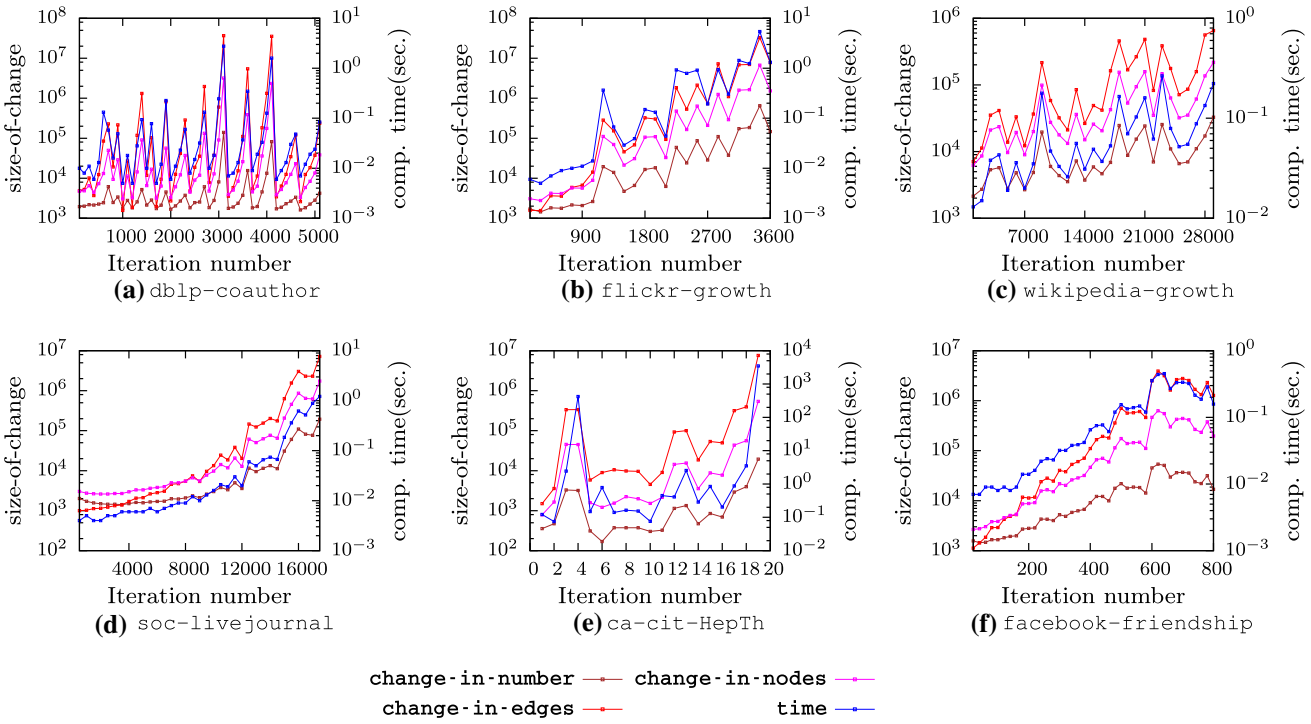


Fig. 7 Computation time for enumerating the change in set of maximal cliques for IMCE, and size-of-change per batch (batch size $\rho = 1000$). The left y-axis shows the size of change, and the right y-axis shows the computation time in seconds

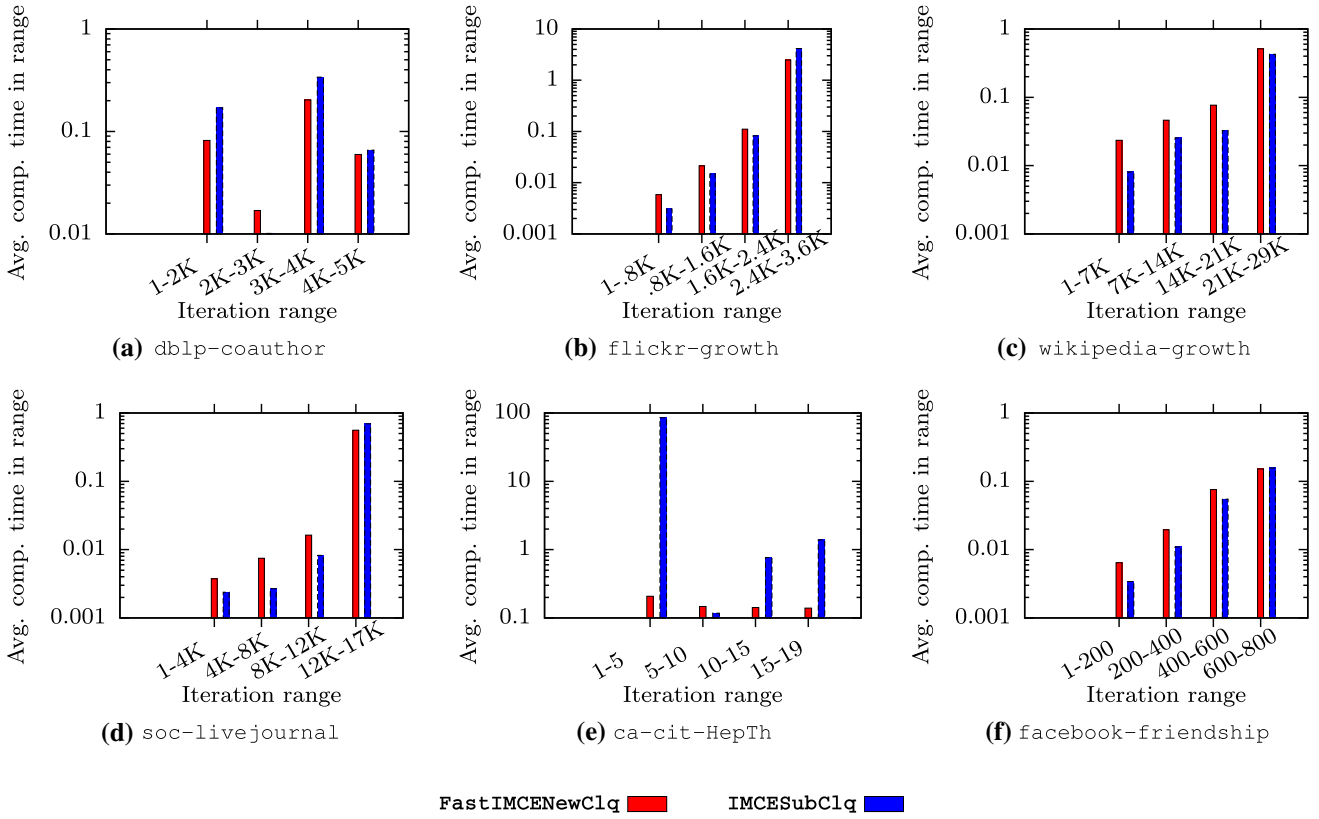


Fig. 8 Computation time (in s) broken down into time for new and subsumed cliques with batch size $\rho = 1000$. Average time in the y-axis is the average taken over the total computation time (new + subsumed) of the iterations in each of the ranges on the x-axis

ing for containment in the set of maximal cliques in computing subsumed cliques as in IMCE. We do not observe much difference in computation time for the graphs dblp-coauthor and ca-cit-HepTh because dblp-coauthor is small and sparse; ca-cit-HepTh is small and sparse at the initial states of the computation compared to the other graphs. Therefore, the sizes of neighborhood of the vertices are small that makes the maximality checking easier.

Impact of edge insertion pattern on computation time: We study the computation time of IMCE when the new edges center around high-degree and low-degree vertices. We have used synthetic graphs RMAT-50K-5M-INIT and RMAT-100K-10M for this evaluation. We consider new edges around 1K highest degree nodes for creating the edge stream around high-degree nodes and consider new edges around 10K least degree nodes for creating the edge stream around low-degree nodes. We observe that the changes in the set of maximal cliques are large (Fig. 10) when the new edges center around high-degree vertices of the initial graph. On the other hand, the changes in the set of maximal cliques are small when the new edges center around low-degree vertices of the initial graph and the computation time is small. For instance, addition of 655 batches (with batch size 100) of edges around low-degree vertices, starting with RMAT-50K-5M-INIT, takes around 0.6 s with cumulative size of change (in the number of maximal cliques) 93K, whereas addition of 655 batches of new edges (with the same batch size) centering around high-degree vertices takes around 137 s with cumulative size of change (in the number of maximal cliques) 1.8×10^7 starting with the same initial graph.

Benefits of using TTTExcludeEdges ***for new maximal cliques:*** We compare the computation times of IMCENewClq (which uses MCE to enumerate cliques), IMCENewClqTTT (which uses TTT), and FastIMCENewClq (which uses TTTExcludeEdges), and the results are shown in Table 2. We observe that FastIMCENewClq is significantly faster than IMCENewClqTTT—the difference can be attributed to the additional pruning in TTTExcludeEdges when compared to TTT. Further, IMCENewClqTTT is much faster than IMCENewClq—the difference can be attributed to the use of TTT which is faster than MCE.

On finding planted cliques in synthetic graphs: We observe that IMCE can find all "planted" cliques in the synthetic $G(n, N)$ graphs in approximately 20 min, whereas the other algorithms (STIX, OV, MCMEI) could not find a single planted clique in an hour. Results are shown in Table 3.

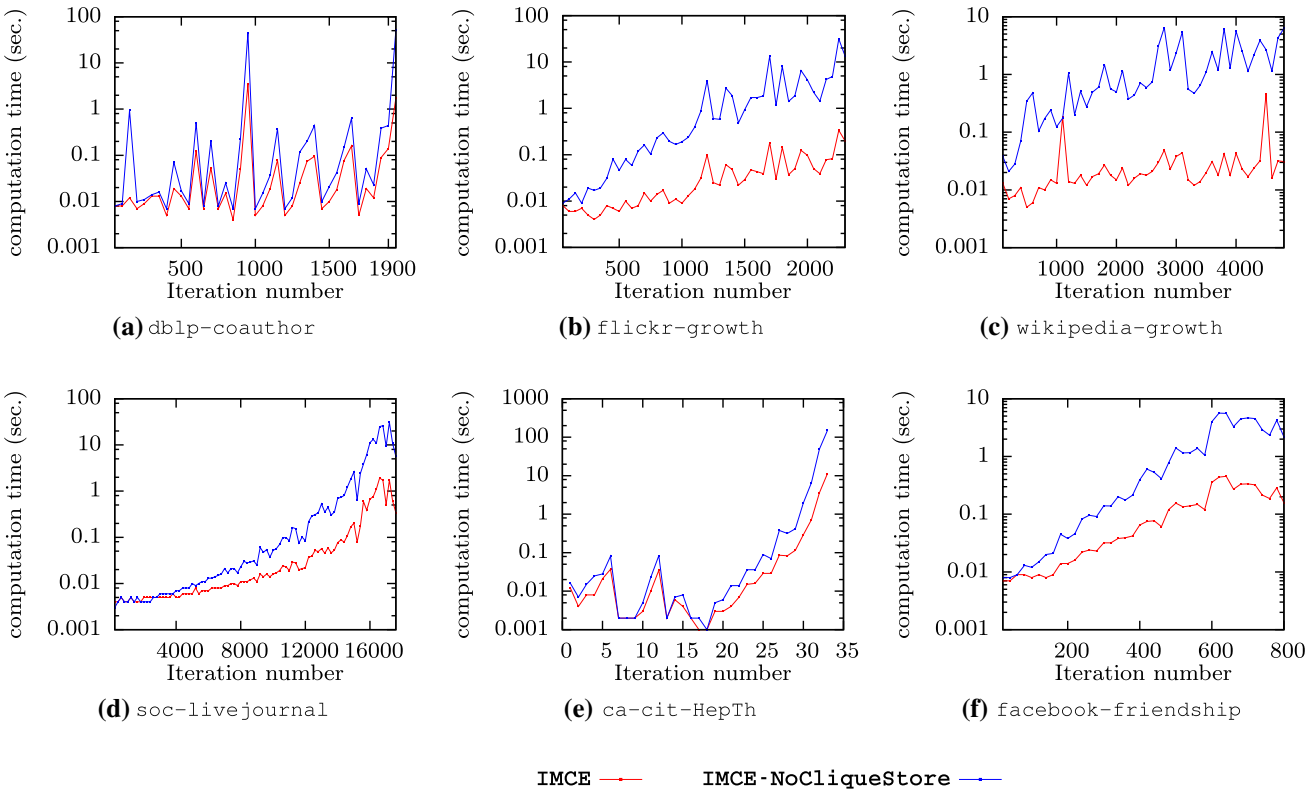


Fig. 9 Difference in computation time due to different strategies for subsumed cliques computation: once by storing the maximal cliques and another by directly checking for maximality (without storing the maximal cliques). We use batch size 1000 for all graphs except for `ca-cit-HepTh` where we use batch size of 100

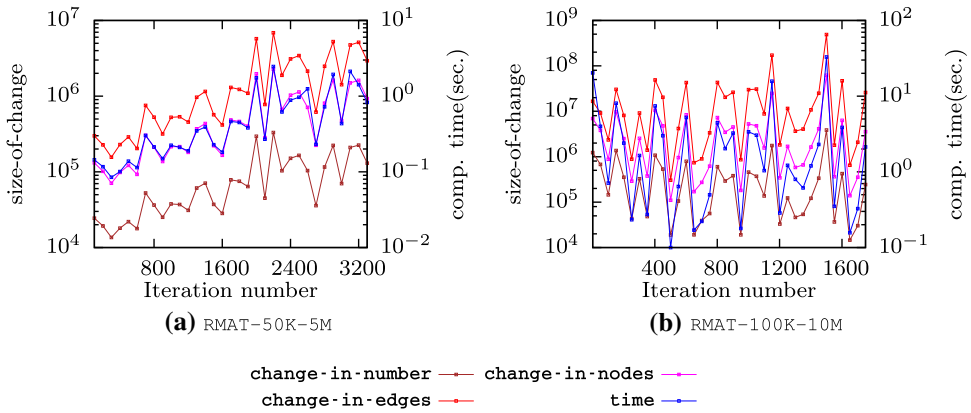


Fig. 10 Performance of `IMCE` with edge stream centering around 1K highest degree vertices considering batch size 100

Table 2 Cumulative computation time (in s) for new maximal cliques with batch size $\rho = 100$

Dataset	IMCENewClq	IMCENewClqTTT	FastIMCENewClq
dblp-coauthor (9603)	7774	62	24
flickr-growth (25,000)	7161	343	125
wikipedia-growth (26,795)	446	310	30
soc-livejournal (151,997)	7200	474	302
ca-cit-HepTh (233)	7464	71	15
facebook-friendship (8171)	2100	89	55

The number of batches for which the cumulative time is computed is in parentheses

Table 3 Total time taken to find all the planted cliques incrementally ($\rho = 100$)

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

Other algorithms (STIX, OV, MCMEI) cannot find a single planted clique within an hour

Decremental computation time: Figure 11 shows the computation time of IMCED for computing the changes in the set of maximal cliques when batches of edges are deleted. For this experiment, we choose a batch size of 100 edges. For the graph flickr-growth, we had to prune the graph down by 15 million edges, to get a reasonable turnaround time for the computation. We used a batch size of 10 for this graph. Similarly, in the case of ca-cit-HepTh, we had to prune the graph down by 1.9 million edges and used the rest of the graph as the initial graph and used the batch size of 100 for performing the decremental computation starting from that point.

Fully dynamic computation time: Figure 12 shows the behavior of the algorithm in a fully dynamic setting, where a batch contains both the edges for addition and the edges for deletion. For this experiment, we considered synthetic RMAT graphs RMAT-50K-5M and RMAT-100K-10M.

Impact of graph density on incremental case: Figure 13 shows the computation time of IMCE − NoCliqueStore upon adding a batch of 100 edges of the initial graphs with different densities. For generating the initial graphs, at every iteration i, we add $100,000 \times i$ edges in stream to the empty graph and we insert the batch of next 100 edges to evaluate the performance of IMCE − NoCliqueStore. For this experiment, we do not use IMCE because for most of the graphs (flickr-growth, wikipedia-growth, soc-livejournal, ca-cit-HepTh) the number of maximal cliques at different densities is so large that those cannot fit in the main memory and IMCE requires the set of maximal cliques of the initial graph for computing the subsumed cliques. We observe that the cost of computing the changes increases as the graph becomes denser. The changes are especially noticeable for graphs flickr-growth, soc-livejournal, ca-cit-HepTh. Sometimes, the computation time is lower in the denser state of the graph. This is because (1) the changes in the set of maximal cliques due to the insertion of a batch are smaller than the others and (2) the computation time is dominated by the size of the changes in the set of maximal cliques.

Impact of graph density on decremental case: Tables 4 and 5 show the computation time of the algorithm IMCED when the density of the initial graph changes. For doing

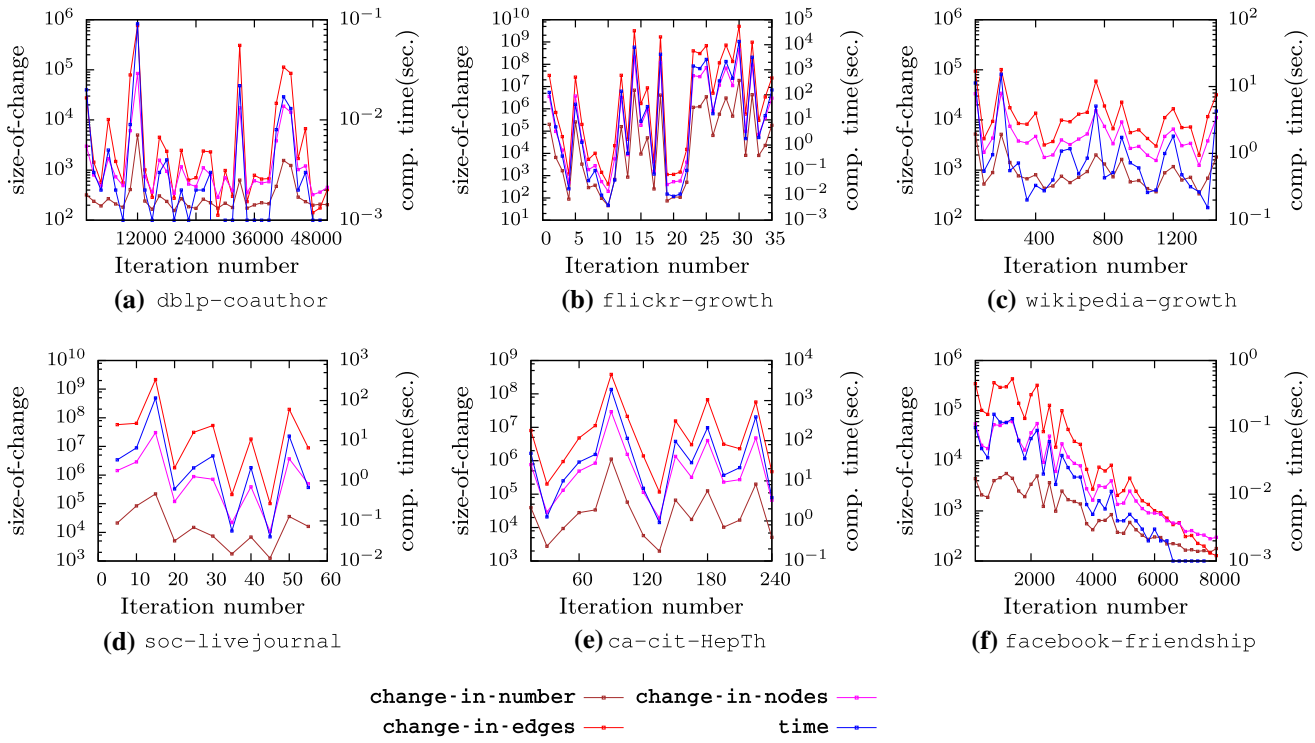
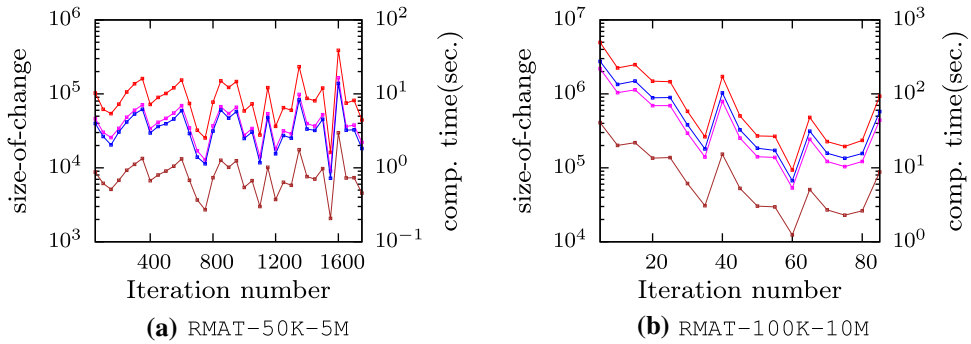


Fig. 11 Computation time for enumerating the change in set of maximal cliques for decremental case when the edges are deleted from the graph instead of insertion, and size-of-change per batch (batch size $\rho = 100$ except for flickr-growth where the batch size is 10). The left y-axis shows the size of change, and the right y-axis shows the computation time in seconds



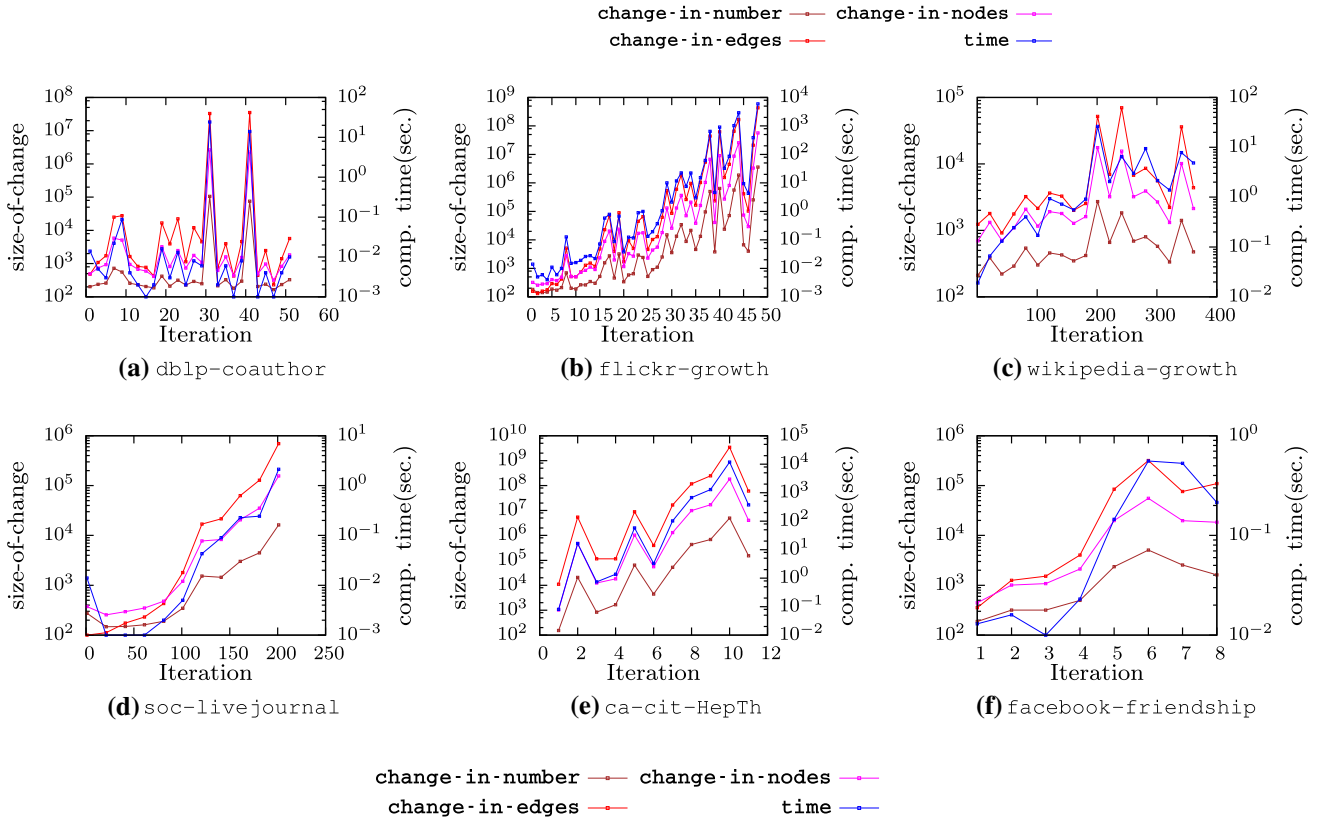


Fig. 12 Fully dynamic case where both addition and deletion of edges are performed in a streaming manner. Each batch (of size 100) in the stream consists of mixed edges

Fig. 13 Performance of IMCE when the density of the graph changes over time

Table 4 Decremental computation time (in s) of different algorithms upon changing density of RMAT-100-4000 by deleting edges in reverse order (of the stream for incremental computation) starting from the original graph

Initial edges	Initial density	IMCED	STIXD	MCMED
3.5K	0.7	243.8	1,496	> hour
3K	0.6	4.2	15	94
2.5K	0.5	0.3	1.2	5
2K	0.4	0.06	0.6	0.6
1K	0.2	0.003	0.5	0.05

The reported computation time is for deleting a batch of next 100 edges from initial graph (at different densities)

Table 5 Decremental computation time (in s) of different algorithms upon changing density of dblp-coauthor by deleting edges in reverse order (of the stream for incremental computation) starting from the original graph

Initial edges	Initial density	IMCED	STIXD	MCMED
4,179,996	5×10^{-6}	0.001	7.8	26.5
3,179,996	3.9×10^{-6}	0.001	6.7	21.2
2,179,996	2.7×10^{-6}	0.002	5.7	20.4
1,179,996	1.4×10^{-6}	0.005	4.9	15.8
179,996	2.2×10^{-7}	0.002	4.1	11.4

The reported computation time is for deleting a batch of next 100 edges from initial graph (at different densities)

Table 6 Cumulative computation time for adding the same set of edges once in incremental computation and then in decremental computation

Dataset	IMCE	IMCED
dblp-coauthor	1113	7219
wikipedia-growth	224	7627
facebook-friendship	101	342
RMAT-100-4000	564	4274

The initial state of each graph for the incremental computation is the final state for the same graph in the decremental computation and vice versa. Batch size is 1000 for all graphs except RMAT, where batch size is 100

this, at each iteration, we remove the edges from the original graph for decreasing the density of the graph and then perform the decremental computation on deleting a batch of next 100 edges. For example, for dblp-coauthor graph, at first iteration we remove 1 million initial edges and on the rest of the graph we perform the decremental computation, and in second iteration, we remove 2 million initial edges and then perform the decremental computation. Similarly, we remove multiple of 500 edges from the original graph RMAT-100-4000 in each iteration. We observe that for RMAT-100-4000 graph, the decrease in computation time is noticeable when the density of the initial graph changes, whereas for dblp-coauthor graph, no such trend is observable. This is because RMAT-100-4000 graph is very dense (with density more than 0.8), whereas dblp-coauthor graph is very sparse (with density 6.3×10^{-6}). Therefore, the change in the density due to the deletion of edges is not significant in dblp-coauthor as in RMAT-100-4000.

Incremental versus decremental computation: Table 6 shows the comparison of IMCE and IMCED on dblp-coauthor, facebook-friendship, wikipedia-growth, and RMAT-100-4000. In this study, we started the incremental computation from the empty graph, and then starting from the point where we stopped the incremental computation, we started decremental computation with reverse order of the edges of the incremental stream. We observe that the overall computation time for decremental computation is higher than the incremental computation on the same set of edges. This is because the computation cost for generating subsumed cliques is lower in the incremental computation than the computation cost of generating new cliques (that are subsumed in the incremental computation) in the decremental computation as in the decremental computation, we directly check for maximality instead of containment check by presenting the set of maximal cliques of the graph before update as in the incremental computation.

Change-sensitiveness: The change in the computation time as a function of the size of change is shown in Figs. 7, 13, and 10 for incremental computation; in Fig. 11 for decremental computation; and in Fig. 12 for fully dynamic computation.

We observe that the computation time is almost proportional to the size of change. Note that the metrics change-in-nodes and change-in-edges better capture the notion of change-sensitiveness than the metric change-in-number because the actual cost of computation depends on the neighborhood structure of the vertices.

As we have discussed in Sect. 4.4 that the fully dynamic case might not become change-sensitive because there might be many intermediate cliques computed that are not in the final output, we tried to reproduce this case on RMAT-100K-10M graph by creating edge stream grouped by the source vertex. Indeed, we observed that there are many intermediate cliques generated as a result of the change in the graph that are not in the final output, but this size (of intermediate cliques that are not in the final output) is much smaller than the actual changes that are in the final output. Therefore, this wasteful computation time is dominated by the time for computing the actual change and thus we see the change-sensitive behavior in this case as well.

Memory consumption: Figure 14 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 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, wikipedia-growth, soc-livejournal and facebook-friendship, 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. We do not show the memory consumption for ca-cit-HepTh because the number of maximal cliques is small compared to the other graphs till the state we executed the incremental computation for this graph, and therefore, the difference in memory consumption with storing the maximal cliques and with storing the hashes of the maximal cliques is not noticeable (less than 1 MB).

Cumulative computation time versus 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

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

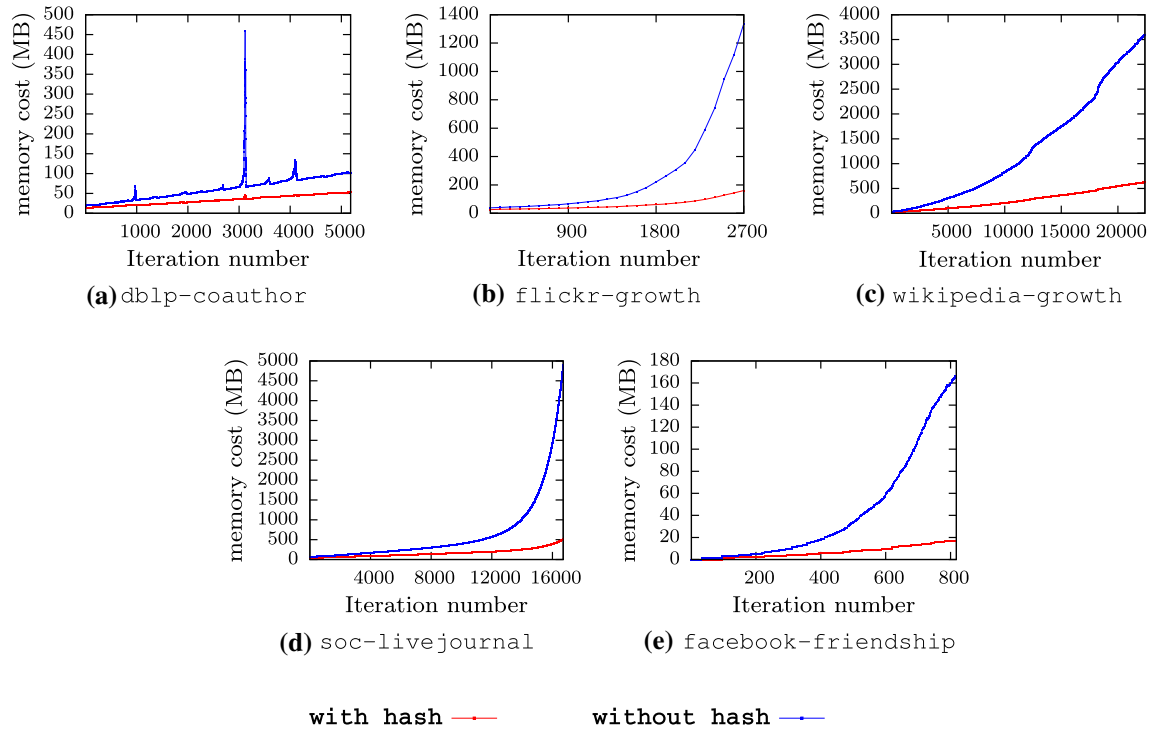


Fig. 14 Memory cost of IMCE with and without using hash function ($\rho = 1000$)

Table 7 Cumulative computation time (in s) of IMCE with different batch sizes

Dataset	$\rho = 1$	$\rho = 10$	$\rho = 100$	$\rho = 1000$	$\rho = 3 \log_2 \Delta$
dblp-coauthor (5,179,996)	1659	1335	1198	1289	1252
flickr-growth (3649×10^3)	7028	6784	6465	7159	6062
wikipedia-growth ($28{,}798 \times 10^3$)	6973	6567	7160	6995	6513
soc-livejournal ($17{,}633 \times 10^3$)	6892	6876	7176	7095	6871
ca-cit-HepTh (19,000)	29	24	1076	3728	22
facebook-friendship (817,035)	100	97	97	94	96

Note that Δ is the maximum degree of the graph before update. Numbers in parentheses indicate the total number of edges inserted incrementally

Table 8 Comparison of incremental computation time(s) of IMCE and Naive for adding a single batch with different batch sizes starting from a graph with 1 million initial edges

Dataset	$\rho = 100$		$\rho = 1000$		$\rho = 10000$		$\rho = 100,000$		$\rho = 1,000,000$	
	Naive	IMCE	Naive	IMCE	Naive	IMCE	Naive	IMCE	Naive	IMCE
dblp-coauthor	4.5	0.001	4.6	0.009	4.8	0.07	5	1.5	9.5	67.3
flickr-growth	10	0.002	10	0.03	9.6	0.4	11	3.4	50.9	127
wikipedia-growth	8.7	0.003	8.5	0.02	9.4	0.2	9.7	2.4	21.9	25.9

ρ indicates the batch size

Table 9 Comparison of decremental computation time(s) of IMCED and NaiveD for deleting a single batch with different batch sizes starting from the original graph

Dataset	$\rho = 100$		$\rho = 1000$		$\rho = 10000$		$\rho = 100,000$	
	NaiveD	IMCED	NaiveD	IMCED	NaiveD	IMCED	NaiveD	IMCED
dblp-coauthor	22.6	0.02	22.3	0.2	22.6	6.4	25.3	>720
facebook-friendship	19.4	0.3	19.5	0.6	19.2	4.8	14.9	192

ρ indicates the batch size

Table 10 Cumulative time (in s) for enumerating new and subsumed cliques

Dataset	STIX	OV	MCMEI	IMCE
dblp-coauthor (499)	3870	295	7212	**0.6**
flickr-growth (288)	3911	306	7214	**0.2**
wikipedia-growth (305)	4258	309	7216	**0.3**
soc-livejournal (156)	4077	299	7234	0.1
ca-cit-HepTh (310)	7291	17	11	1
facebook-friendship (1895)	7776	193	6853	1

Bold values highlight the improved runtime over prior work
The number of batches is shown in parentheses. Batch size $\rho = 100$ except ca-cit-HepTh, where $\rho = 10$ edges

100 batches if we used a batch size of 100. Table 7 shows the results for different batch sizes. There is no observable trend found by varying the batch size for all the input graphs except for ca-cit-HepTh. For this graph, we found that with the increase in batch size from 10 to 100, the number of subsumed clique candidates that are not actually subsumed increases quite a lot. This affects the overall computation time. The possible reason is that the density of other graphs is smaller, the sizes of new cliques for those graphs are smaller, and the search for subsumed cliques is short, even for large batch size. From this observation, it seems, for dense graph, it is good to use small batch size to reduce the redundant computation in the subsumed clique computation such as in the case of ca-cit-HepTh graph.

Comparison with Baseline: We compare IMCE (IMCED) with algorithm Naive (NaiveD), which recomputes the set of maximal cliques each time there is an addition (deletion) of edges, and show the results in Table 8 (Table 9). As expected, both IMCE and IMCED significantly outperform Naive and NaiveD when the number of edges inserted/deleted in a batch is small (less than 1 million for Naive and less than 10^5 for NaiveD, in our experiments). When the number of edges is larger than 1 million, Naive outperforms IMCE. This is not surprising, since as the number of edges in a batch increases, the size of the change also increases, and there is lesser benefit in using an incremental algorithm.

However, Naive and NaiveD have an additional problem related to memory consumption, since they have to store the set of maximal cliques in order to compute the symmetric difference. For instance, NaiveD cannot execute even for a single edge deletion for wikipedia-growth, due to insufficient memory. On flickr-growth and ca-cit-HepTh, NaiveD also comes to a near standstill since it tries to store the set of maximal cliques in memory. Since IMCE and IMCED do not store the set of all maximal cliques, they do not run into similar issues of memory consumption.

Comparison with prior works: We also compare the computation time of IMCE with prior works as shown in Table 10. Clearly, IMCE is many orders of magnitude (more than 1000) faster than prior algorithms for most of the input graphs except for ca-cit-HepTh and facebook-friendship because (1) these graphs are of small sizes compared to the other graphs and (2) the number of maximal cliques at the initial states of these graphs is small compared to the other graphs. 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 IMCE with STIX, *ov*, and MCMEI upon changing (increasing) the density of the input graphs and we present the results in Tables 11 and 12, and we compare IMCED with STIXD and MCMED upon changing (decreasing) the density of the input graph and we present the results in Tables 4 and 5. Note that we cannot compare other larger graphs because all of the prior works require the set of maximal cliques of the initial graph to start the computation and the number of maximal cliques at different states of graph is so large that they cannot fit in the main memory. We observe that both IMCE and IMCED are magnitude of order faster than the prior works which is as expected except for RMAT-100-4000 where the performance of OV is similar to that of IMCE. This is because the graph RMAT-100-4000 is small and the number of maximal cliques of this graph is also small compared to the other graphs.

Summary of results: To summarize the results of our experiments, we note the following: (1) IMCE and IMCED are change-sensitive: The computation time 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 and IMCED are two to three orders of magnitude faster than prior algorithms. (3) The computation time

Table 11 Incremental computation time (in s) of different algorithms upon changing the density of `dblp-coauthor` at each computation with batch size 100

Initial edges	Initial density	IMCE	STIX	OV	MCMEI
1M	1.2×10^{-6}	< 1 ms	16	1.6	16.2
2M	2.4×10^{-6}	< 1 ms	20.8	0.9	21.3
3M	3.6×10^{-6}	0.02	376.2	4.8	28.8
4M	4.9×10^{-6}	0.002	26.2	1.6	33
5M	6×10^{-6}	0.003	42.4	2.4	33.8

Table 12 Incremental computation time (in s) as a function of the density of `RMAT-100-4000` using batch size 100

Initial edges	Initial density	IMCE	STIX	OV	MCMEI
1K	0.2	0.005	1.4	0.01	0.05
2K	0.4	0.04	544	0.07	1
2.5K	0.5	0.2	> 1 h	0.3	10
3K	0.6	2.6	> 1 h	3	220.8
3.5K	0.7	146	> 1 h	129	> 1 h

for the maintenance increases when the density of the graph increases. (4) The use of hash signatures for storing maximal cliques greatly reduces the memory consumption.

7 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 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.

Acknowledgements AD and ST were supported in part by NSF Grants 1527541, 1725702, and 1632116.

References

1. Angel, A., Koudas, N., Sarkas, N., Srivastava, D., Svendsen, M., Tirthapura, S.: Dense subgraph maintenance under streaming edge weight updates for real-time story identification. VLDB J. **23**, 1–25 (2013)
2. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Appl. Math. **65**, 21–46 (1993)
3. Bahmani, B., Kumar, R., Vassilvitskii, S.: Densest subgraph in streaming and mapreduce. VLDB **5**(5), 454–465 (2012)
4. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM **16**(9), 575–577 (1973)
5. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-mat: a recursive model for graph mining. In: Proceedings of the 2004 SIAM International Conference on Data Mining, pp. 442–446. SIAM (2004)
6. Chateau, A., Riou, P., Rivals, E.: Approximate common intervals in multiple genome comparison. In: 2011 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 131–134. IEEE (2011)
7. Cheng, J., Ke, Y., Fu, A.W.-C., Yu, J.X., Zhu, L.: Finding maximal cliques in massive networks. TODS **36**(4), 21 (2011)
8. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM J. Comput. **14**, 210–223 (1985)
9. Das, A., Sanei-Mehri, S.-V., Tirthapura, S.: Shared-memory parallel maximal clique enumeration. ArXiv preprint arXiv:1807.09417 (2018)
10. Das, A., Tirthapura, S.: A change-sensitive algorithm for maintaining maximal bicliques in a dynamic bipartite graph. CoRR. arXiv:abs/1707.08272 (2017)
11. Driskell, A.C., Ané, C., Burleigh, J.G., McMahon, M.M., O'Meara, B.C., Sanderson, M.J.: Prospects for building the tree of life from large sequence databases. Science **306**(5699), 1172–1174 (2004)
12. Duan, D., Li, Y., Li, R., Lu, Z.: Incremental k-clique clustering in dynamic social networks. Artif. Intell. Rev. **38**, 1–19 (2012)
13. Eppstein, D., Löffler, M., Strash, D.: Listing all maximal cliques in sparse graphs in near-optimal time. In: ISAAC, pp. 403–414 (2010)
14. Eppstein, D., Strash, D.: Listing all maximal cliques in large sparse real-world graphs. In: Pardalos, P., Rebennack, S. (eds.) Experimental Algorithms, LNCS, vol. 6630, pp. 364–375. Springer, Berlin (2011)
15. Erds, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci **5**, 17–61 (1960)
16. Fan, W., Hu, C., Tian, C.: Incremental graph computations: doable and undoable. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 155–169. ACM (2017)
17. Galbrun, E., Gionis, A., Tatti, N.: Overlapping community detection in labeled graphs. Data Min. Knowl. Discov. **28**(5–6), 1586–1610 (2014)
18. Gibson, D., Kumar, R., Tomkins, A.: Discovering large dense subgraphs in massive graphs. In: VLDB, pp. 721–732 (2005)
19. Hanneman, R.A., Riddle, M.: Introduction to Social Network Methods. http://faculty.ucr.edu/~hanneman/nettext/. Textbook on the web
20. Huang, X., Cheng, H., Qin, L., Tian, W., Yu. J.X.: Querying k-truss community in large and dynamic graphs. In: SIGMOD, pp. 1311–1322 (2014)
21. Hung, S.-C., Araujo, M., Faloutsos, C.: Distributed community detection on edge-labeled graphs using spark. In: 12th International Workshop on Mining and Learning with Graphs (MLG), vol. 113 (2016)
22. Hussain, M.M.-U., Wang, A., Trajcevski, G.: Co-maxrs: continuous maximizing range-sum query. Sciences **305**, 110–129 (2015)
23. Java, A., Song, X., Finin, T., Tseng, B.L.: Why we twitter: an analysis of a microblogging community. In: WebKDD/SNA-KDD, pp. 118–138 (2007)

24. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Inf. Process. Lett. **27**(3), 119–123 (1988)
25. Koch, I.: Enumerating all connected maximal common subgraphs in two graphs. Theor. Comput. Sci. **250**(1), 1–30 (2001)
26. Kose, F., Weckwerth, W., Linke, T., Fiehn, O.: Visualizing plant metabolomic correlation networks using clique-metabolite matrices. Bioinformatics **17**(12), 1198–1208 (2001)
27. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Trawling the web for emerging cyber-communities. Comput. Netw. **31**(11), 1481–1493 (1999)
28. Lehmann, S., Schwartz, M., Hansen, L.K.: Biclique communities. Phys. Rev. E **78**, 016108 (2008)
29. Leskovec, J., Krevl, A.: SNAP datasets: stanford large network dataset collection. http://snap.stanford.edu/data (2014)
30. Li, R., Yu, J.X., Mao, R.: Efficient core maintenance in large dynamic graphs. TKDE **26**(10), 2453–2465 (2014)
31. Lo, D., Surian, D., Zhang, K., Lim, E.-P.: Mining direct antagonistic communities in explicit trust networks. In: CIKM, pp. 1013–1018 (2011)
32. Makino, K., Uno, T.: New algorithms for enumerating all maximal cliques. In: SWAT, pp. 260–272 (2004)
33. McGregor, A., Tench, D., Vorotnikova, S., Vu, H.T.: Densest subgraph in dynamic graph streams. In: MFCS, pp. 472–482 (2015)
34. Moon, J.W., Moser, L.: On cliques in graphs. Isr. J. Math. **3**(1), 23–28 (1965)
35. Mukherjee, A.P., Tirthapura, S.: Enumerating maximal bicliques from a large graph using mapreduce. IEEE Trans. Serv. Comput. **10**(5), 771–784 (2017)
36. Mukherjee, A.P., Xu, P., Tirthapura, S.: Enumeration of maximal cliques from an uncertain graph. IEEE Trans. Knowl. Data Eng. **29**(3), 543–555 (2017)
37. Ottosen, T.J., Vomlel, J.: Honour thy neighbour: clique maintenance in dynamic graphs. In: PGM, pp. 201–208 (2010)
38. Rome, J.E., Haralick, R.M.: Towards a formal concept analysis approach to exploring communities on the world wide web. In: Ganter, B., Wille, R. (eds.) Formal Concept Analysis, LNCS, vol. 3403, pp. 33–48. Springer, Berlin (2005)
39. Sanderson, M.J., Driskell, A.C., Ree, R.H., Eulenstein, O., Langley, S.: Obtaining maximal concatenated phylogenetic data sets from large sequence databases. Mol. Biol. Evol. **20**(7), 1036–1042 (2003)
40. Sariyüce, A.E., Gedik, B., Jacques-Silva, G., Wu, K., Çatalyürek, Ü.V.: Streaming algorithms for k-core decomposition. PVLDB **6**(6), 433–444 (2013)
41. Simsiri, N., Tangwongsan, K., Tirthapura, S., Wu, K.-L.: Work-efficient parallel union-find with applications to incremental graph connectivity. In: European Conference on Parallel Processing, pp. 561–573. Springer (2016)
42. Stix, V.: Finding all maximal cliques in dynamic graphs. Comput. Optim. Appl. **27**(2), 173–186 (2004)
43. Sun, S., Wang, Y., Liao, W., Wang, W.: Mining maximal cliques on dynamic graphs efficiently by local strategies. In: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 115–118. IEEE (2017)
44. Svendsen, M., Mukherjee, A.P., Tirthapura, S.: Mining maximal cliques from a large graph using mapreduce: tackling highly uneven subproblem sizes. J. Parallel Distrib. Comput. **79–80**, 104–114 (2015)
45. Thorup, M.: Decremental dynamic connectivity. J. Algorithms **33**(2), 229–243 (1999)
46. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Glob. Optim. **44**(2), 311 (2009)
47. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique. In: Proceedings of the WALCOM: Algorithms and Computation, 4th International Workshop, WALCOM 2010, Dhaka, Bangladesh, February 10–12, 2010, pp. 191–203 (2010)
48. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theor. Comput. Sci. **363**(1), 28–42 (2006)
49. Tomita, E., Yoshida, K., Hatta, T., Nagao, A., Ito, H., Wakatsuki, M.: A much faster branch-and-bound algorithm for finding a maximum clique. In: Proceedings of the Frontiers in Algorithmics, 10th International Workshop, FAW 2016, Qingdao, China, June 30–July 2, 2016, pp. 215–226 (2016)
50. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A new algorithm for generating all the maximal independent sets. SIAM J. Comput. **6**(3), 505–517 (1977)
51. Wulff-Nilsen, C.: Faster deterministic fully-dynamic graph connectivity. In: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1757–1769. SIAM (2013)
52. Yan, C., Burleigh, J.G., Eulenstein, O.: Identifying optimal incomplete phylogenetic data sets from sequence databases. Mol. phylogenetics Evol. **35**(3), 528–535 (2005)