Finding All Maximal Connected s-Cliques in Social Networks

Rachel Behar
The Rachel and Selim Benin School of Computer Science and Engineering, Hebrew University of Jerusalem, Israel
rachel.beharharr@mail.huji.ac.il

Sara Cohen
The Rachel and Selim Benin School of Computer Science and Engineering, Hebrew University of Jerusalem, Israel
sara@cs.huji.ac.il

ABSTRACT

Cliques are commonly used for social network analysis tasks, as they are a good representation of close-knit groups of people. For this reason (as well as for others), the problem of enumerating, i.e., finding, all maximal cliques in a graph has received extensive treatment. However, considering only complete subgraphs is too restrictive in many real-life scenarios where "almost cliques" may be even more useful. Hence, the notion of an *s-clique*, a clique relaxation that allows every node to be at distance at most s from every other node, has been introduced. *Connected s-cliques* add the natural requirement of connectivity to the notion of an s-clique.

This paper presents efficient algorithms for finding all maximal connected s-cliques in a graph. We present a provably efficient algorithm, which runs in polynomial delay. In addition, we present several variants of the well-known Bron-Kerbosch algorithm for maximal clique generation. Extensive experimentation over both real and synthetic datasets shows the efficiency of our algorithms, and their scalability with respect to graph size, density, and choice of s.

1 INTRODUCTION

Maximal cliques have long been considered a key component in the analysis of social networks [34]. Cliques are indeed highly cohesive sets of nodes, and as such are used to detect close-knit overlapping communities [13, 29, 36]. For this reason (among others), there has been extensive work on algorithms for finding all maximal cliques in a given graph, e.g., [1, 6, 8, 11, 17].

While the notion of a clique captures a completely cohesive group of nodes within a graph, this definition is often overly restrictive. In practice, it is obvious that sets of nodes can represent cohesive groups even if several links are missing; for example, within a community not all pairs of people will be friends. In addition, as networks are often built from observation of empirical data, there may be real-life links that are missing within the data captured. Searching for groups of nodes that are cliques will miss highly related groups of nodes for which links have been omitted from the dataset. To overcome these limitations, relaxations to the notion of a clique have been studied [30].

One useful relaxation to the notion of a clique, called an *s-clique*, was introduced over 65 years ago [24] to describe and measure connectivity in social groups. Given a graph G, we say that a set of nodes U is an s-clique, where s is a (typically small) natural number, if every pair of nodes $u, v \in U$ is at distance at most s one from another in G. In particular, when $s = 1$, the notions of a clique and an s-clique coincide. An s-clique is *maximal* if it cannot be extended with additional nodes, while retaining the required distances property. Unlike cliques, s-cliques may be unconnected. Connected s-cliques add the natural requirement of connectivity.

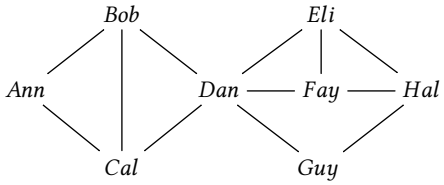


Figure 1: Example of a small social network G.

Example 1.1. Consider the example of a small social network G in Figure 1. Graph G contains six maximal cliques, namely $\{a, b, c\}$, $\{b, c, d\}$, $\{d, e, f\}$, $\{e, f, h\}$, $\{d, g\}$, $\{g, h\}$, where a is a shorthand for Ann, b is a shorthand for Bob, and so on. This graph contains three maximal 2-cliques $\{a, b, c, d\}$, $\{b, c, d, e, f, g\}$ and $\{d, e, f, g, h\}$. Intuitively, the 2-cliques seem to better capture the graph communities, as they are a bit coarser. They also highlight the fact that d is a bridge between the communities.

Graph G contains two maximal 3-cliques $\{a, b, c, d, e, f, g\}$ and $\{b, c, d, e, f, g, h\}$, which (by their symmetric difference) indicate the people who, if linked, could help merge the communities. Thus, such a link might be suggested to Ann and Hal. Finally, we note that there is a single maximal 4-clique in G, as the diameter of G is four.

This paper studies the problem of enumerating (i.e., finding) all maximal connected s-cliques in a graph. Maximal clique enumeration has been shown to be useful in other areas (beyond social network analysis), e.g., finding subgraphs common to a set of input graphs [19], genome mapping and protein clustering in bioinformatics [14, 25], clustering for wireless sensornetworks [4], and statistical analysis of financial networks [5]. As s-cliques are relaxations of cliques, they allow significantly greater flexibility, and may be useful for the above applications, e.g., to find subgraphs that are "almost common" to input graphs (i.e., that appear in slight variations in the various input graphs), to integrate genome mappings based on very similar subportions or to cluster protein sequences while allowing more flexibly for missing information.

An algorithm for enumerating maximal connected s-cliques can be used for new and interesting applications, such as link prediction in social networks [22], since missing direct links in large s-cliques are prime candidates for link suggestion. Note that large cliques could not be used for this purpose, as they are missing no links at all, by definition. Similarly, another possible application would be to help identify hidden connections in a social network, by finding maximal s-cliques that may form unidentified communities. We leave the development of such applications to future work, and focus in this paper on algorithms for efficiently enumerating all maximal s-cliques from a given graph.

The main contributions of this paper are three new algorithms for enumerating all maximal connected s-cliques from a given graph G. While it may seem from Example 1.1 that graphs have a small (polynomial) number of maximal s-cliques, this is in fact not always the case. As we demonstrate later, a graph may have exponentially many maximal s-cliques. (This was already well known for $s = 1$, and is true for larger values of s as well.) Hence, we cannot hope to derive a polynomial time algorithm for the problem at hand, as it may take exponential time to simply print the output. Instead, our first algorithm guarantees polynomial delay between results, i.e., the time to produce the first result, between every pair of subsequent results, and from the final result until completion, is polynomial.

The other two algorithms we present are adaptations of the well-known Bron-Kerbosch method, originally developed for finding all maximal cliques, to the problem at hand. While Bron-Kerbosch clique enumeration does not run in polynomial delay, it is known to be the fastest method, in practice, for maximal clique enumeration (when used with some specific optimizations). Hence, adapting this method to s-cliques is of interest. Optimizations for our adaptations, including pivoting and checking for feasibility, are studied. Extensive experimentation, over both real and synthetic datasets, proves the efficiency of our techniques, as well as their suitability for use over social network data.

2 RELATED WORK

Due to their numerous uses, the problem of finding all maximal cliques of a graph has received extensive attention. In the worst case, there can be exponentially many maximal cliques in a graph. In fact, [26] shows that the maximal number of cliques in graph with n nodes is $O(3^{n/3})$. Thus, the focus is on finding maximal cliques in time that is efficient with respect to the input and output. One well-known algorithm is that of Bron-Kerbosch [6], which, with the *pivoting* improvement of [32], guarantees a worst-time complexity of $O(3^{n/3})$ for graphs of size n. Hence the total time spent is no worse than required to return all maximal cliques on the graph with the most possible cliques.

Additional work on maximal clique enumeration has focused on output-efficient algorithms, i.e., algorithms whose runtime is a function of the number of maximal cliques in the given input graph [1, 17]. Several works have studied enumeration over sparse graphs [7, 12], as such graphs tend to be common in practice. Recent work has also focused on maximal clique enumeration over uncertain graphs [38], and over massive networks [9, 11].

Enumeration of maximal graphs for several relaxations of the notion of a clique has also been studied. The problem of mining all maximal k-plexes was studied in [3, 35], and enumeration of maximal c-isolated cliques was studied in [15]. There has also been work on mining quasi-cliques (i.e., densest subgraphs) in a single graph [23, 33, 37], and over a set of graphs [16], as well as mining locally dense subgraphs [31].

Among clique relaxations, both quasi-cliques and s-clubs appear to be most related to s-cliques. Formally, *quasi-cliques* are parameterized by a value γ, i.e., a subset S of nodes in a graph G is a γ-quasi-clique if every node in S is connected to at least $\gamma(|S|-1)$ nodes in S. It has been shown [16] that there is a strong relationship between the parameter γ, and the diameter of the induced subgraph of G on S. For example, if $\frac{1}{2} \leq \gamma \leq \frac{|S|-2}{|S|-1}$, then the induced subgraph on S will have diameter at most 2. At first glance, it would seem then that this property can be utilized to enumerate (connected) s-cliques, e.g., by enumerating γ-quasi-cliques with an appropriate (s-dependent) choice of γ. In fact, this is not the case, and previous algorithms for γ-quasi-cliques do not enumerate s-cliques. The difference is subtle, as every pair of nodes in an s-clique S is of distance at most s in G, but may be of larger distance in the graph induced by S.

A subset U of nodes in a graph G is an *s-club*, if the diameter of U is at most s, i.e., if there is a path between every two nodes in U that only traverses nodes in U, that is of length at most s. This definition is different from that of s-cliques, where the distance between nodes is determined by the shortest path in the entire graph G. Similar to the maximum clique problem, the problem of finding an s-club of maximum size is also NP-complete [2]. However, unlike cliques and s-cliques, s-clubs are not hereditary (i.e., a subgraph of an s-club is not necessarily an s-club), and indeed s-club maximality testing is NP-complete [28]. Since maximal s-clubs cannot be efficiently recognized, enumerating maximal s-clubs cannot be achieved in polynomial delay (unlike enumerating maximal connected s-cliques, as we show in this paper).

The enumeration problem for connected s-cliques has not yet been studied. However, the optimization problem for s-cliques, i.e., the NP-complete problem of finding an s-clique of maximal size, was studied in [2]. Enumerating s-cliques with given labels, over a labeled graph, is also NP-complete problem, and is studied in [18]. Finally, [28] has shown that for graphs with some special properties, the notions of connected s-cliques and s-clubs coincide.

3 FORMAL FRAMEWORK

Graphs and Induced Subgraphs. We use G, H (possibly with subscripts or superscripts) to denote simple undirected graphs. We use $V(G)$ to denote the nodes of G and $E(G)$ to denote the edges of G. Note that an edge is a pair $\{v, u\}$ where v and u are two different nodes in $V(G)$.

We will often be interested in *induced subgraphs* of a given graph G. Formally, a subset of nodes $U \subseteq V(G)$ defines the *induced subgraph* $G[U]$ of G consisting of precisely the set of nodes U, and the edges in $E(G)$ that are incident only on nodes in U. In notation, we have $V(G[U]) = U$ and $E(G[U]) = E(G) \cap U^2$. We say that H is an induced subgraph of G if $H = G[U]$, for some U, and denote this fact by $H \sqsubseteq G$.

We use $dist_G(u, v)$ to denote the number of edges on the shortest path between u and v in G and $N^i_G(v)$ to denote the set

$$N^i_G(v) := \{u \mid dist_G(v, u) \leq i \text{ and } u \neq v\}$$

of the nodes at distance at most i from v in G. We use $N_G(v)$ for the special case where $i = 1$, i.e., $N_G(v)$ contains all direct neighbors of v. Extending this notation, we use $N^{\forall,i}_G(V)$ and $N^{\exists,i}_G(V)$ to denote the set of nodes at distance at most i from all and at least one, respectively, v in V, i.e.,

$$N^{\forall,i}_G(V) := \{u \mid \forall v \in V, dist_G(v, u) \leq i \text{ and } u \notin V\},$$
$$N^{\exists,i}_G(V) := \{u \mid \exists v \in V, dist_G(v, u) \leq i \text{ and } u \notin V\}.$$

If G is clear from the context, we will omit the subscript.

Example 3.1. Consider graph G from Figure 1. Let $V = \{e, h\}$. We have:

$$N^{\exists,1}(V) = \{d, f, g\} \qquad N^{\forall,1}(V) = \{f\}$$
$$N^{\exists,2}(V) = N^{\exists,1}(V) \cup \{b, c\} \qquad N^{\forall,2}(V) = N^{\exists,1}(V).$$

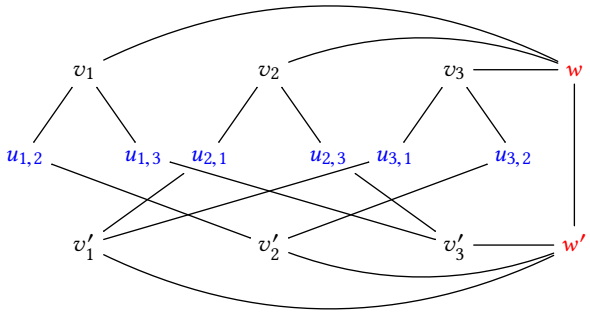


Figure 2: Graph G'.

Cliques and s-Cliques. A set of nodes U in a graph G is a *clique* if all pairs $u, v \in U$ are adjacent one to another in G. Cliques are useful in many problem areas, as they represent fully cohesive portions of G. As discussed earlier, in many scenarios, the requirement that a set of nodes form a clique may be overly restrictive.

Previous work [30] has studied various relaxations of the notion of a clique. In the following, let s be an integer and U be a set of nodes in a graph G. We say that

- U is an *s-clique* if, for all $u, v \in U$, it holds that $dist_G(u, v) \leq s$.
- U is a *connected s-clique* if U is an s-clique and the induced graph $G[U]$ is connected.

When $s = 1$, cliques coincide with both s-cliques and connected s-cliques.

Example 3.2. Consider graph G from Figure 1. The set of nodes $\{a, b, c, d, e, f, g\}$ is a (connected) 3-clique, but is not a 2-clique, e.g., $dist_G(a, f) = 3 > 2$. The set of nodes $\{a, b, c, d\}$ is a connected 2-clique, and the set $\{a, d\}$ is a 2-clique, but is not a connected 2-clique.

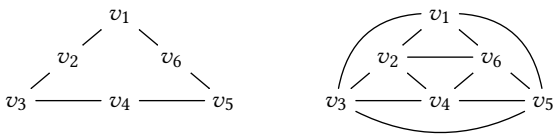
We say that U is a *maximal (connected) s-clique* in G, if U is a (connected) s-clique, and for all U' such that $U \subsetneq U'$, it holds that U' is not a (connected) s-clique. It is natural to focus on *maximal* (connected) s-cliques. Indeed, every (connected) s-clique is contained in some maximal (connected) s-clique. Hence, maximal (connected) s-cliques can be viewed as a succinct representation for all (connected) s-cliques.

Example 3.3. Consider graph G', from Figure 2. Let

$$V = \{v_1, v_2, v_3\} \qquad U = \{u_{1,2}, u_{1,3}, u_{2,1}, u_{2,3}, u_{3,1}, u_{3,2}\}$$
$$V' = \{v'_1, v'_2, v'_3\} \qquad W = \{w, w'\}$$

Now, it is easy to observe that every subset C of $V \cup V'$ that does not contain both v_i and v'_i for some $i \leq 3$ is a 2-clique. Such 2-cliques are not maximal, however. A subset $C \subseteq V \cup V' \cup W$ will be a maximal connected 2-clique if (1) C contains precisely one among v_i, v'_i for each $i \leq 3$, and (2) C contains w, w'. Thus, for example, $\{v_1, v_2, v'_3\}$ is a 2-clique (but not maximal nor connected), and $\{v_1, v_2, v'_3, w, w'\}$ is a maximal connected 2-clique in G. Note that there are additional ways to form maximal connected 2-cliques, when taking nodes from U. For example, $\{v_1, v'_2, w, w', u_{1,2}\}$ is also a maximal connected 2-clique.

We can now formally state our problem of interest: *Given a graph G and an integer s, enumerate (i.e., find, one after another) all maximal connected s-cliques in G.* As larger s-cliques can, naturally, be more interesting, we will also briefly consider a related problem, i.e., that of finding maximal s-cliques of size at least k, for some given number k.



(a) H **(b)** H^2

Figure 3: Graph H and corresponding graph H^2.

Remark 1. *We do not consider enumeration of maximal s-cliques that are not necessarily connected. This is because enumeration of maximal s-cliques over a graph G can be reduced to maximal clique enumeration: Define G^s as the graph containing an edge between nodes u, v if they are of distance at most s in G. Then, the maximal cliques in G^s are precisely the maximal s-cliques in G. However, this reduction is not applicable for* connected *s-cliques, as cliques in G^s can correspond to unconnected sets in G. Hence, enumeration of maximal* connected *s-cliques is more difficult, as the following example demonstrates.*

Consider the graphs H, H^2 in Figures 3 (a) and (b). The graph H^2 contains an edge between every pair of nodes in H that are of distance at most 2 one from another. Every 2-clique in H is a clique (in the standard sense) in H^2. Observe, for example, that the sets $C_1 = \{v_1, v_2, v_6\}$ and $C_2 = \{v_1, v_3, v_5\}$ are 2-cliques in H and cliques in H^2. Unlike the set C_1, the set C_2 is not *a connected 2-clique. Indeed no two nodes in C_2 are connected in H, and thus, C_2 forms an unconnected subgraph of H. This cannot be seen when looking at H^2 alone; the information about connectedness is lost in the given graph transformation.*

We note that due to the fact that the number of sets in the result can be exponential in the size of G, the problem of enumerating all maximal connected s-cliques cannot be solved in polynomial time. Hence, exponential time may be needed just to print the output. Therefore, we focus on finding algorithms whose runtime is either provably efficient with respect to the output size (e.g., polynomial delay) or of high efficiency in practice.

Example 3.4. We demonstrate a graph with exponentially many maximal connected s-cliques for $s = 2$. It is easy to extend this idea to derive a graph with exponentially many maximal connected s-cliques for other values of s. Let n be an integer. Let

$$V = \{v_i \mid i \leq n\} \qquad U = \{u_{i,j} \mid i \neq j \leq n\}$$
$$V' = \{v'_i \mid i \leq n\} \qquad W = \{w, w'\}$$

be sets of nodes. We add edges $\{v_i, u_{i,j}\}$, $\{u_{i,j}, v'_j\}$ for all $i \neq j \leq n$, as well as edges $\{v_i, w\}$, $\{v'_i, w'\}$ for all $i \leq n$. Finally, we add the edge $\{w, w'\}$. Graph G' from Figure 2 has precisely this structure, for $n = 3$.

Every pair of nodes v_i, v'_j where $i \neq j$ have distance 2 one from another, while v_i, v'_i have distance 3. Nodes w, w' are at distance at most 2 from every node in the graph. Thus, it is easy to see that every choice of nodes including precisely one among v_i, v'_i, for all $i \leq n$, as well as nodes w, w', yields a maximal connected 2-clique. (Note that nodes $u_{i,j}$ cannot be added to such sets.) Thus, the graph derived has at least 2^n maximal connected 2-cliques, while it has only $2n + n(n-1) + 2$ nodes, i.e., the number of maximal connected 2-cliques is exponential in the size of the graph.

Algorithm PolyDelayEnum(G, s)
1. $\mathcal{Q} \leftarrow$ EmptyQueue()
2. $\mathcal{I} \leftarrow$ EmptyIndex()
3. $C \leftarrow$ ExtendMax($\emptyset, G, s$)
4. Enqueue($\mathcal{Q}, C$)
5. Insert($\mathcal{I}, C$)
6. **while** NotEmpty($\mathcal{Q}$)
7. **do** $C \leftarrow$ Dequeue($\mathcal{Q}$)
8. Print(C)
9. **for** $v \in N_G^{\exists,1}(C)$
10. $C' \leftarrow$ ExtendMax($\{v\}, G[C \cup \{v\}], s$)
11. $C'' \leftarrow$ ExtendMax(C', G, s)
12. **if** $C'' \notin \mathcal{I}$
13. **then** Enqueue($\mathcal{Q}, C''$)
14. Insert($\mathcal{I}, C''$)

Algorithm ExtendMax(C, G, s)
1. **if** $C = \emptyset$
2. **then** add an arbitrary node to C
3. **while** $\exists v \in N_G^{\forall,s}(C) \cap N_G^{\exists,1}(C)$
4. **do** $C \leftarrow C \cup \{v\}$
5. **return** C

Figure 4: An polynomial delay algorithm for enumerating all maximal connected s-cliques.

4 A POLYNOMIAL DELAY ALGORITHM

We present a provably efficient algorithm for enumerating all maximal connected s-cliques in a given graph G. This algorithm is inspired by the general purpose algorithm for enumerating maximal subgraphs satisfying some connected-hereditary property, appearing in [10]. Our algorithm, called PolyDelayEnum, appears in Figure 4.

The algorithm PolyDelayEnum uses two data structures:

- $\mathcal{Q}$, a queue, containing maximal connected s-cliques that must still be processed. Later, in Section 6, we will also consider using a priority queue for $\mathcal{Q}$.
- $\mathcal{I}$, an index containing maximal connected s-cliques that have already been generated. In order to achieve the required runtime, access to $\mathcal{I}$ (both insertions and membership checks) must be in time that is at most logarithmic in the size of $\mathcal{I}$. Thus, for example, $\mathcal{I}$ can be implemented as a BTree.

PolyDelayEnum uses a sub-procedure, called ExtendMax, which is given, as input, a set C, a graph G and the integer s. We note that the set C provided as input is always a connected s-clique. ExtendMax returns a set C' such that

- $C \subseteq C'$ (this is ensured as C' is created by adding nodes to C);
- C' is a connected s-clique (as we only add a node that is connected to and of distance at most s from the nodes presently in C);
- C' is maximal in G, with respect to the above two properties (as we continue to add nodes as long as possible).

For example, when calling ExtendMax with the empty set (Line 3 of PolyDelayEnum), a single maximal connected s-clique is returned. In general, the output of ExtendMax may differ, depending on the order in which we iterate over the nodes of G. In the special case that G is almost a connected s-clique, i.e., contains a single node that contradicts this property (as occurs in the invocation of ExtendMax in Line 10 of PolyDelayEnum), there is only one possible output for ExtendMax.

Now, PolyDelayEnum begins by finding a single maximal connected s-clique, using ExtendMax, and adding this set to both $\mathcal{Q}$ and $\mathcal{I}$ (Lines 1–5). While $\mathcal{Q}$ is not empty, we remove a maximal connected s-clique C from $\mathcal{Q}$, and print C (Lines 6–8). Now, for each node v that is a neighbor of some node in C, we proceed as follows. First (Line 10) we find the s-clique C' containing v that is maximal with respect to $G[C \cup \{v\}]$. Next (Line 11), we extend this set so as to derive an s-clique C'' that is maximal with respect to G. If we have not created C'' yet (i.e., $C'' \notin \mathcal{I}$), we add it to $\mathcal{Q}$ and $\mathcal{I}$ (Lines 12–14). We note that a key aspect of the algorithm is the fact that ExtendMax is called twice, consecutively (Lines 10 and 11) with different graphs as input. The first invocation guides the creation of a new connected s-clique C' to contain portions from C. The second invocation ensures maximality with respect to the input graph G.

Example 4.1. Consider calling PolyDelayEnum with the graph G from Figure 1 and $s = 2$. At first, an arbitrary 2-clique will be created, such as $C = \{a, b, c, d\}$. This set C will be added into the queue $\mathcal{Q}$. When removed from the queue (Line 7), we will print C and then iterate over the set $N_G^{\exists,1}(C) = \{e, f, g\}$. Consider the case where we choose $v = e$ (in Line 9). We will call ExtendMax($\{e\}, G[C \cup \{e\}], 2$), deriving the set $C' = \{b, c, d, e\}$. There is only one way to extend C' in order to derive a maximal connected 2-clique. This extension will be returned from the next call to ExtendMax in Line 11, $C'' = \{b, c, d, e, f, g\}$, and inserted into the queue. Execution will proceed similarly, first for all other neighbors of C, which will return the same result C'', that will not be enqueued again. Then, the next graph dequeued will be C'', that will form the last s-clique $\{d, e, f, g, h\}$ when h is added.

The following result holds. (In practice, the delay between answers is typically lower than the theoretical result.)

Theorem 4.2. *Given a connected graph G and an integer s, the algorithm* PolyDelayEnum *prints every maximal connected s-clique in G, precisely once. In addition, the delay before printing the first answer, and between every two consecutive answers, and from the time the last answer is printed until termination, is $O(|V(G)|^3)$.*

Proof. We start by showing correctness of the algorithm. First, it is immediate from the structure of ExtendMax that every set printed must be a maximal connected s-clique. Second, observe that every set in printed at most once, as we only insert C into $\mathcal{Q}$ if it was not already generated in the past (i.e., does not appear in $\mathcal{I}$). It remains to show that every maximal connected s-clique is indeed printed.

Let v be an arbitrary node in G. Then, there is some set C, printed by PolyDelayEnum, that contains v. (This can be shown by induction on the distance of v from the closest node in the first set generated by PolyDelayEnum.)

We can now prove that every maximal connected s-clique is printed. Let C_* be some maximal connected s-clique. Let C_m be the set printed by PolyDelayEnum for which the largest connected component in $C_* \cap C_m$ is maximal. (If there are ties, choose C_m arbitrarily among all such sets.) If $C_m = C_*$, we have indeed printed C_*. Suppose otherwise. Observe first that $C_m \cap C_*$ cannot be empty, as PolyDelayEnum must print some set containing each node in C_*. Now, since $C_m \neq C_*$, there is some node $v \in C_* - C_m$, such that v is connected to the largest connected component in $C_m \cap C_*$. After dequeuing C_m from $\mathcal{Q}$ in Line 7, we will iterate over the node v in Line 9. Then, Line 10 will return a

Algorithm CLIQUES(R, P, X)
1. **if** $P = \emptyset$ and $X = \emptyset$
2. **then** PRINT(R)
3. **for** $v \in P$
4. **do** CLIQUES($R \cup \{v\}, P \cap N(v), X \cap N(v)$)
5. $P \leftarrow P - \{v\}$
6. $X \leftarrow X \cup \{v\}$

Figure 5: Bron-Kerbosch algorithm for enumerating maximal cliques.

set C' containing v, as well as the largest connected component in $C_m \cap C_*$ (since these together form a connected s-clique). This set will be enlarged to C'' in Line 11. Now, C'' is either inserted into Q (and eventually printed) or was already inserted in Q (and eventually printed). Observe that the largest connected component in $C'' \cap C_*$ must be larger than that in $C_m \cap C_*$, which contradicts the maximality of choice of C_m. Hence, it must be $C_m = C_*$, and C_* is indeed printed.

We will now show that the delay is $O(|V(G)|^3)$. First, note that by a preprocessing step we can compute $N_G^s(v)$ for every node $v \in V(G)$ in time $O(|V(G)|^3)$. In addition, EXTENDMAX runs in $O(|V(G)|^2)$, since it traverses each edge at most once, in increasing distance from the set C. Now, the delay before printing the first answer is determined by *(1)* the preprocessing step which finds the distances between nodes and *(2)* running EXTENDMAX in Line 3. Therefore the delay before printing the first answer is $O(|V(G)|^3)$.

The delay between every two consecutive answers, and from the time the last answer is printed until termination, is determined by the runtime of a single iteration of the while loop in Line 6 in POLYDELAYENUM, because in each iteration a single answer is printed and the last answer is printed in the last iteration. In each iteration, the for loop in Line 9 goes over the set $N_G^{\exists,1}(C)$, that is of size $O(|V(G)|)$, and for each node v in the set calls EXTENDMAX, which runs in $O(|V(G)|^2)$. In total we derive that each iteration runs in $O(|V(G)|^3)$. □

5 ADAPTATION OF BRON-KERBOSCH ALGORITHM

We start by reviewing the Bron-Kerbosch algorithm for enumerating maximal cliques. We then present strategies and optimization techniques to adapt this algorithm for enumerating maximal connected s-cliques.

5.1 Maximal Cliques

The Bron-Kerbosch algorithm for enumerating maximal cliques appears in Figure 5. This algorithm, called CLIQUES, recursively searches for all maximal cliques. When called with sets R, X and P, it searches for all maximal cliques containing all nodes in R, possibly some nodes in P, and no nodes in X. In particular, in the first invocation, we send the empty set for R and X, and the set $V(G)$ for P.

Throughout the execution, R is always a clique. The sets P and X are disjoint, and always satisfy that $P \cup X$ contains precisely every node that is connected to all nodes in R (i.e., those can potentially be used to extend R). Therefore, when P and X are both empty, R is a maximal clique. Otherwise, if P is not empty, the algorithm attempts to add each node $v \in P$ in turn to R, updating P and X to contain only nodes that are connected to v (in addition to being connected to the rest of R). After the recursive call that includes v, the node v is added to X, so as to create additional unseen cliques—those that exclude v. (Cliques containing v will be produced in the recursive call.)

The algorithm CLIQUES, as presented in Figure 5, runs quite poorly in practice. To improve the runtime, [32] presented a technique called *pivoting* to reduce the branching factor of the recursion, by iterating over only a subset of P. In particular, instead of iterating over P, their algorithm:

- first, chooses a *pivot* node $u \in P \cup X$.
- then, iterates only over the nodes v in $P - N(u)$.

The intuition behind this improvement is that, for any node u, every maximal clique must either contain u, or contain some node that is not a neighbor of u. (Otherwise, if the maximal clique contains only neighbors of u, it must also contain u.) Hence, it is sufficient to iterate over the nodes in P, other than the neighbors of u. The pivot node is chosen so as to minimize the set $P - N(u)$. This optimization ensures a worst case runtime of $O(3^{n/3})$, which is order of the largest number of cliques possible in a graph of n nodes. Previous work has shown that this improvement causes the algorithm to run very well in practice.[1]

5.2 Maximal Connected s-Cliques

We adapt the Bron-Kerbosch algorithm to return all maximal connected s-cliques, instead of all maximal cliques. There are, perhaps, two different natural approaches to adapt the algorithm CLIQUE to this new setting:

(1) R **is always a connected s-clique:** In the CLIQUE algorithm, we have seen that R is always a clique. It is natural to adapt this algorithm so as to preserve the invariant that R is always a connected s-clique. This approach is taken in CSCLIQUES1 in Figure 6. Thus, when choosing a node v with which to extend R (Line 3), we will only choose nodes from P that are adjacent to some node in R. After choosing v, we perform a recursive call, with v added to R and we intersect sets P and X with the nodes of distance at most s from v. Thus, throughout the algorithm, P and X always contain precisely the nodes at distance at most s from every node in R. Only such nodes may possibly be used to extend R in the future.
Note the final change in the algorithm, in the condition for printing R (Line 1). We print R only if it is maximal, i.e., there are no nodes at distance at most s from R (i.e., nodes in P or X) that are adjacent to R.

(2) R **is always an s-clique, but may be unconnected:** In the second adaptation, we do not require R to be connected. In the algorithm CSCLIQUES2 in Figure 7, the set R may be unconnected. To observe this, see that in Line 3 we can add any node v from P to R, even if v is not adjacent to R. Thus, R will be an s-clique throughout the execution, but may be unconnected. This requires an additional change to the condition for printing R in Line 1—we print R only if it is connected.

Example 5.1. Consider using CSCLIQUES1 to find all maximal connected 2-cliques for graph H in Figure 3. When first called, $R = X = \emptyset$, $P = \{v_1, \ldots, v_6\}$ and $s = 2$. Since, R is empty, we iterate over all nodes in P. Suppose the first node chosen is v_1.

[1] Additional optimizations to the algorithm have been considered in the past, such as iterating over v in the outermost recursion according to a degeneracy ordering of G [12] and early recognition of special branching cases [27]. Such optimizations can also be included in our algorithm.

Algorithm CsCliques1(R, P, X, s)
1. **if** $P \cap N^{\exists,1}(R) = \emptyset$ and $X \cap N^{\exists,1}(R) = \emptyset$
2. **then** print(R)
3. **for** $v \in P \cap N^{\exists,1}(R)$
   ▷ If $R = \emptyset$, then take $N^{\exists,1}(R) = G(V)$
4. **do** CsCliques1$(R \cup \{v\}, P \cap N^s(v), X \cap N^s(v), s)$
5. $P \leftarrow P - \{v\}$
6. $X \leftarrow X \cup \{v\}$

Figure 6: Adaptation of Bron-Kerbosch algorithm for enumerating all maximal connected s-cliques. Throughout the execution, R is always a connected s-clique.

Algorithm CsCliques2(R, P, X, s)
1. **if** $P \cap N^{\exists,1}(R) = \emptyset$ and $X \cap N^{\exists,1}(R) = \emptyset$ and R is connected
2. **then** print(R)
3. **for** $v \in P$
4. **do** CsCliques2$(R \cup \{v\}, P \cap N^s(v), X \cap N^s(v), s)$
5. $P \leftarrow P - \{v\}$
6. $X \leftarrow X \cup \{v\}$

Figure 7: Adaptation of Bron-Kerbosch algorithm for enumerating all maximal connected s-cliques. Throughout the execution, R is always an s-clique, but may be unconnected.

In the recursive call to CsCliques1, we will have $R = \{v_1\}$, $P = \{v_2, v_3, v_5, v_6\}$ and $X = \emptyset$. When a recursive call is made for the second node v_2, we will have $R = \{v_2\}$, $P = \{v_3, v_4, v_6\}$ and $X = \{v_1\}$. (Note that v_1 was removed from P in Line 5 and added to X in Line 6, in the previous iteration.) Now, consider the execution of CsCliques1$(\{v_2\}, \{v_3, v_4, v_6\}, \{v_1\}, 2)$. We will iterate over nodes that are neighbors of $\{v_2\}$ in $\{v_3, v_4, v_6\}$, i.e., only over v_3. This will cause a single recursive call to CsCliques1$(\{v_2, v_3\}, \{v_4\}, \{v_1\}, 2)$, which will eventually produce the maximal connected 2-clique $\{v_2, v_3, v_4\}$.

We contrast this execution with the execution of algorithm CsCliques2. At first, the algorithms will proceed in the same fashion, as $R = \emptyset$ and both algorithms will iterate over all nodes in the graph. However, consider the execution of the recursive call to CsCliques2$(\{v_2\}, \{v_3, v_4, v_6\}, \{v_1\}, 2)$. Instead of iterating only over neighbors of v_2 in $\{v_3, v_4, v_6\}$, we will iterate over all nodes in this set. Assuming the order of iteration is $v_3 < v_4 < v_6$, this will cause two additional recursive calls: CsCliques2$(\{v_2, v_4\}, \emptyset, \{v_3\}, 2)$ and CsCliques2$(\{v_2, v_6\}, \emptyset, \{v_1, v_4\}, 2)$. Neither of these calls will produce maximal connected s-cliques.

When comparing these two approaches, it is immediately obvious that CsCliques2 does extra work that is avoided by CsCliques1, as CsCliques2 can create many sets that will never be printed, as they are unconnected. The algorithm CsCliques1 completely avoids this by ensuring that R is connected throughout its execution. Notwithstanding the fact that CsCliques1 would seem to be much superior to CsCliques2, in fact, we will see that the latter is much more amenable to optimizations (including the pivoting technique). This will be discussed further in Section 5.3. However, before discussing this further, we prove correctness of both algorithms.

In order to prove correctness, let $\prec$ be an arbitrary ordering of the nodes in G. We assume that the iteration over (the subset of) P in Line 3 of both cscliquesi and CsCliques2 follows this ordering, i.e., if $v, v' \in P$ (resp. $v, v' \in P \cap N^{\exists,1}(R)$) and $v \prec v'$, then we will choose to iterate over v before iterating over v'. The ordering $\prec$ implies two types of total orderings over nodes in a connected s-clique C. Let $\omega_2(C) = v_1, \ldots, v_k$ be the total ordering over C, defined by $\prec$. Let $\omega_1(C) = u_1, \ldots, u_k$ be the total ordering over C, defined as follows:

- $u_1 = v_1$;
- for all $i > 1$, it holds that $u_i = v_j$ where v_j in the first node according to $\omega_2(C)$ that is not already among $u_1, \ldots, u_{i-1}$, for which $G[\{u_1, \ldots, u_i\}]$ is connected.

Example 5.2. Consider graph G' from Figure 2. Let $\prec$ be the total ordering that orders the nodes as seen in the graph from left to right, top to bottom, i.e.,

$$v_1 \prec \cdots \prec w \prec u_{1,2} \prec \cdots \prec u_{3,2} \prec v_1' \prec \cdots \prec w' .$$

Consider the set $C = \{v_1, v_2', w, w', u_{1,2}\}$. Then, we have $\omega_1(C) = v_1, w, u_{1,2}, v_2', w'$.

We consider the *execution tree* $\mathcal{T}_i$ formed by (recursive) calls to CsCliques$i$, for $i = 1, 2$. To be precise, this tree has nodes of the form (R, P, X) where R, P, X are subsets of $V(G)$. An edge from a node (R, P, X) to a node (R', P', X') is labeled by a node v. This tree is defined recursively as follows: The root of $\mathcal{T}_i$ is $(\emptyset, V(G), \emptyset)$. A node (R, P, X) has a child (R', P', X') with a connecting edge labeled v if the call to CsCliques$i$ with parameters (R, P, X) results in a recursive call with parameters (R', P', X') when the node v is chosen in Line 3.

Given a series of nodes $\bar{u} = u_1, \ldots, u_n$, we say that $\bar{u}$ is a path in $\mathcal{T}_i$ if there is a path of edges starting from the root of $\mathcal{T}_i$ with labels $u_1, \ldots, u_n$ (precisely in that order). A node (R, P, X) in $\mathcal{T}_i$ is an *output node* if R is printed by CsCliques$i$, during the call with parameters (R, P, X).

Several important properties of $\mathcal{T}_i$ hold.

Lemma 5.3. *Let G be a graph and let $\mathcal{T}_i$ be the execution tree derived by calling* CsCliques$i(\emptyset, V(G), \emptyset, s)$. *Let (R, P, X) be a node in $\mathcal{T}_i$.*

(1) *The set R is an s-clique;*
(2) *If $i = 1$, the set R is connected;*
(3) *$P \cup X = N^{\forall,s}(R)$;*
(4) *Let $u_1, \ldots, u_k$ be the path to (R, P, X) in execution tree $\mathcal{T}_i$. If $u_1, \ldots, u_k$ is a prefix of $\omega_i(C)$ for some maximal connected s-clique C, then $C - R \subseteq P$;*
(5) *For every other node (R', P', X') in $\mathcal{T}_i$ it holds that $R \neq R'$;*
(6) *If C is a maximal connected s-clique, then $\omega_i(C)$ is a path in $\mathcal{T}_i$.*

Proof. We show Properties 1–4 by induction on the depth (i.e., distance from the root) of the node (R, P, X). For base case of distance 0, i.e., the root node, all four properties are immediate. (For Property 3, note that $R = \emptyset$, and thus, every node in G is of distance at most s from all nodes in R. Indeed $P = V(G)$).

Now, assume that Properties 1–4 hold for nodes of distance at most k from the root. We prove the required for nodes at distance $k + 1$. Let (R, P, X) be a node at distance $k + 1$ from the root, and let (R', P', X') be its parent. Let v be the label of the edge from (R', P', X') to (R, P, X). By the induction hypothesis, R' is an s-clique, if $i = 1$, R' is connected, and every node in P' is of distance at most s from all nodes in R'. Since v is chosen from

P', and is chosen so as to be connected to R', if $i = 1$, it follows immediately that R satisfies Properties 1 and 2.

Let $S' = N^{\forall,s}(R')$. By the induction hypothesis, $S' = P' \cup X'$. During the loop of Line 3, before the recursive call CsCliques$i(R, P, X, s)$, we remove nodes from P' and add them to X'. Thus, when node v is chosen in Line 3, it still holds that $S' = P' \cup X'$. Let S be the set of nodes that are of distance at most s from all nodes in R'. Clearly, $S = S' \cap N^s(v)$. Now, since $P = P' \cap N^s(v)$ and $X = X' \cap N^s(v)$, it follows that $S = P \cup X$, as required in Property 3.

Finally, suppose that $u_1, \ldots, u_{k+1}$ is a prefix of $\omega_i(C)$ for some maximal connected s-clique C. This implies that also $u_1, \ldots, u_k$ is also a prefix of $\omega_i(C)$. Hence, $C - R' \subseteq P'$. Since we only choose, in Line 3, a node that is in P' (and hence, by Property 3 is of distance at most s from every node in R') and if $i = 1$, we only continue with nodes that are connected to R, it follows that no node in C is removed from P in Line 5. Otherwise, this would contradict the minimality of choice of u_{k+1} in the definition of $\omega_i(C)$. Hence, it follows that $C - R \subseteq P$, as required in Property 4.

Now, we show Property 5. Let (R'', P'', X'') be the lowest common ancestor of (R, P, X) and (R', P', X'). We consider two cases:

- Case 1: (R'', P'', X'') is the node (R, P, X). (The case is which (R'', P'', X'') is the node (R', P', X') is identical.) In this case, Let v be the node on the outgoing edge of (R, P, X) on the path leading to (R', P', X'). Clearly, $v \notin R$ and $v \in R'$, and therefore $R \neq R'$.
- Case 2: Neither node among (R, P, X) and (R', P', X') is an ancestor of the other. Let v and v' be the nodes on the outgoing edges of (R'', P'', X'') on the paths leading to (R, P, X) and (R', P', X'), respectively. Suppose, without loss of generality, that the node v was chosen before v' in the loop of Line 3. Clearly, R contains v. However, since v is removed from the set P'' before the recursive call with v', it follows that v is not in R'. Hence, $R \neq R'$.

Finally, we show Property 6. Let $\omega_i(C) = u_1, \ldots, u_n$. We show, by induction, that for every prefix $u_1, \ldots, u_k$ of $\omega_i(C)$, it holds that $u_1, \ldots, u_k$ is a path in $\mathcal{T}_i$. Obviously, this holds for the empty prefix. Assume the required for a prefix of length k, and we show that the claim holds for a prefix of length $k + 1$. By the induction hypothesis, $u_1, \ldots, u_k$ is a path in $\mathcal{T}_i$ leading to a node (R, P, X). Observe that $R = \{u_1, \ldots, u_k\}$. By Property 4, it holds that $C - R \subseteq P$, i.e., $u_{k+1} \in P$. For $i = 1$, by the definition of $\omega_i(C)$, it holds that u_{k+1} is connected to R. Hence, at some point, the node u_{k+1} will be chosen in Line 3. Therefore, there will be a recursive call made where u_{k+1} is added, i.e., $u_1, \ldots, u_{k+1}$ is a path in $\mathcal{T}_i$. □

We can now show correctness of the algorithm.

Theorem 5.4. *Let G be a graph. Then, calling the procedure* CsCliques$i(\emptyset, V(G), \emptyset, s)$ *will result in the printing of every maximal connected s-clique in G precisely once, and no other sets of nodes will be printed.*

Proof. First observe that no set will be printed more than once. This follows immediately from the Property 5 of Lemma 5.3, since we print the set R, and there are no two nodes in the execution tree that contain the same set R.

Next, we show that every set that is printed is a maximal s-clique. Assume that R is printed. Let (R, P, X) be the node of $\mathcal{T}_i$ in which R is printed. By Property 1 of Lemma 5.3, it follows that R is an s-clique. If $i = 1$, R is also connected (Property 2 of Lemma 5.3). If $i = 2$, R must be connected if it is printed, as this is part of the requirement before printing (Line 1). Hence, R is a connected s-clique. Suppose, by way of contradiction, that R is not maximal. Then there is a node v that is connected to R and is of distance at most s from every node in R. By Property 3 of Lemma 5.3, it holds that $v \in P \cup X$. Hence, either $P \cap N(R) \neq \emptyset$ or $X \cap N(R) \neq \emptyset$, and thus, R will not be printed, in contradiction to the assumption.

Finally, we show that every maximal connected s-clique will be printed. Let C be a maximal connected s-clique. By Property 6 of Lemma 5.3, it holds that $\omega_i(C)$ is a path in $\mathcal{T}_i$ leading to a node (R, P, X). Assume, by way of contradiction, that this node is not an output node. It then follows that there is a node $v \in (P \cap N(R)) \cup (X \cap N(R))$, i.e., v is connected to R and is in P or X. By Property 3 of Lemma 5.3, it follows that v is of distance at most s from every node in R. Hence, $R \cup \{v\}$ is a connected s-clique. However, $R = C$, and thus, this is a contradiction to the assumption that C is maximal. Hence, it follows that if C is a maximal connected s-clique, C will be printed during execution. □

5.3 Optimizing the Algorithm

We consider two different strategies to optimize the algorithm CsCliques2. Unfortunately, neither of these strategies are applicable to CsCliques1, as will be made clear.

Pivoting. As discussed earlier, a critical improvement to Cliques, which renders the algorithm efficient in practice, is that of pivoting [32]. This strategy reduces the branching factor in the execution tree by avoiding iteration over the entire set P, and iterating over $P - N(u)$, for some $u \in P \cup X$, instead. Node u is called the *pivot*. This technique can be applied when generating maximal connected s-cliques, due to the following proposition.

Proposition 5.5. *Let C be a maximal connected s-clique and let $u \in V(G)$ be a node. Then, one of the following conditions must hold: (1) $u \in C$, (2) $C \nsubseteq N^s(u)$ or (3) $C \cap N(u) = \emptyset$.*

Proof. Suppose, by way of contradiction, that none of the above conditions hold. Then,

(1) $u \notin C$;
(2) $C \subseteq N^s(u)$;
(3) C contains a node in $N(u)$.

It immediately follows that C is not maximal, in contradiction to the assumption, as u can be added to C (since it is close enough to all nodes in C, and has a neighbor in C). □

It may not be immediately obvious how to utilize this property, to improve the runtime, as it states that every clique must either contain u, or contain some node that is not at distance s from u, or cannot contain any neighbor of u. Thus, if instead of iterating over P, we iterate over $P - N^s(u)$ (in a similar fashion to the pivoting of Clique), we can miss maximal connected s-cliques that do not contain any node from $P - N^s(u)$ (and also contain no neighbor of u).

To overcome this problem, we will always choose the pivot as a neighbor of some node in R. To be precise, we choose a pivot node $u \in (P \cup X) \cap N^{\exists,1}(R)$ that minimizes $P - N^s(u)$. Then, in Line 3 of CsCliques2, instead of iterating over P, we iterate over $P - N^s(u)$. Now, every maximal connected s-clique containing R must either contain u, or must contain some node that is not within $N^s(u)$. (Otherwise, if it does not contain any node in $N^s(u)$, we could always add u, since u is connected to R and of distance at most s from every node in R.)

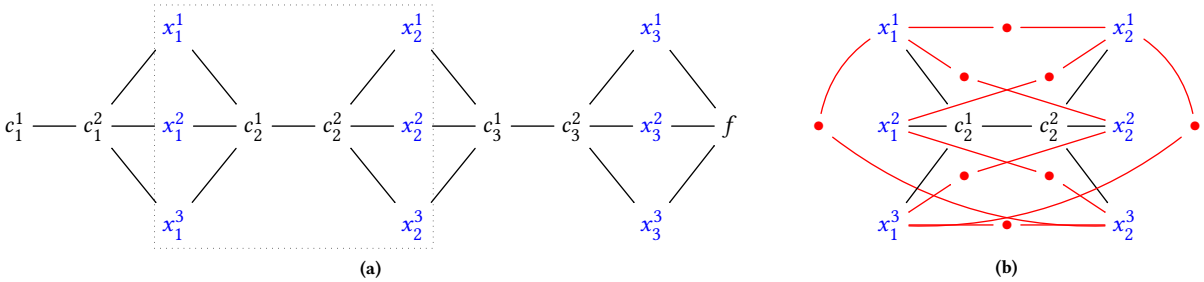


Figure 8: Graph used to demonstrate the reduction of Theorem 5.6

We note that this improvement cannot be integrated with CsCliques1, as it requires us to iterate over nodes that may not be neighbors of R. The algorithm CsCliques1 always preserves the invariant that R is connected, and iterating over such nodes would cause them to be added to R, thereby, losing the correctness of the invariant.

To summarize, integrating pivoting into CsCliques2 is performed by replacing Line 3 with the following two lines:

3.1 $u \leftarrow \arg\min_u \{|P - N^s(u)| \mid u \in (P \cup X) \cap N^{\exists,1}(R)\}$
3.2 **for** $v \in P - N^s(u)$

Checking for Feasibility. CsCliques1 never creates a set R that is unconnected. On the other hand, CsCliques2 may go deep into the recursion tree even if a set R is being considered, for which it is not possible to add nodes and derive a connected maximal s-clique. It would be preferable to prune branches for calls of the form (R, P, X, s) if there is no connected s-clique C such that $R \subseteq C$ and $C \subseteq R \cup P$. (Recall that we will only try to add nodes from P to R.) In such cases, the branches cannot lead to a solution. Unfortunately, this cannot be verified efficiently, in the general case, as shown stated by the following theorem.

Theorem 5.6. *Let G be a graph, $s > 1$ be an integer, and R be an s-clique. Determining whether there exists a connected s-clique C such that $R \subseteq C$ is NP-complete.*

Proof. Membership in NP is immediate, as we can guess a set C and check whether it satisfies all requirements. We show NP-hardness by a reduction from 3-SAT.

Let ψ be a 3-SAT formula with m clauses $C_1, \ldots, C_m$, over the variables $X_1, \ldots, X_k$. We assume, without loss of generality, that no clause contains both a variable and its negation.

Let V_0 be the set of nodes:

$$\{c_i^k \mid i \leq m, k \leq s\} \cup \{x_i^j \mid i \leq m, j \leq 3\} \cup \{f\}$$

Let G_0 be the graph derived by taking the nodes in V_0, and adding the following edges:

- $\{c_i^j, c_i^{j+1}\}$ for every $i \leq m$ and $j < s$;
- $\{c_i^s, x_i^j\}$ for every $i \leq m$ and $j \leq 3$;
- $\{x_i^j, c_{i+1}^1\}$ for every $i < m$ and $j \leq 3$;
- $\{x_m^j, f\}$ for every $j \leq 3$.

We say that a pair of nodes in V_0 is *conflicting* if they are of the form $x_i^j, x_{i'}^{j'}$ and the j-th literal in C_i is the negation of the j'-th literal in $C_{i'}$. Let G be the graph derived by taking the graph G_0, and adding a path of length s (using new nodes) between every pair of non-conflicting nodes in V_0 that are at distance greater than s in G_0.

Figure 8 (a) demonstrates graph G_0 for $s = 2$, $m = 3$ and the 3-SAT formula $\psi = (X_1 \vee \bar{X}_2 \vee X_3) \wedge (X_1 \vee X_2 \vee X_3) \wedge (\bar{X}_1 \vee \bar{X}_2 \vee X_3)$. Figure 8 (b) focuses on the framed subgraph in (a) and shows all the nodes and edges that will appear in G. Observe that there is no path of length $s = 2$ between x_1^2 and x_2^2 as they correspond to $\bar{X}_2$ and X_2.

Now, we make several observations about the graph G.

(1) Let U be any subset of V_0 that does not contain nodes corresponding to a literal and its negation. Then, U is an s-clique (although may be unconnected). This is immediate, since we have paths of length s between every two nodes in U.
(2) Let U be any subset of $V(G)$ that does contain nodes corresponding to a literal and its negation. Then, U is not an s-clique. This is also apparent, from a careful analysis of the graph. We did not include paths of length s between such nodes, in the beginning. Furthermore, when adding edges between nodes in V_0, we never create a path of length at most s between such nodes. Indeed, the use of s nodes $c_i^1, \ldots, c_i^s$ is precisely to avoid such cases.
(3) Let U be any subset of $V(G)$ containing the nodes $c_1^1, \ldots, c_m^1, f$. If U is an s-clique, then U must be a subset of V_0, i.e., cannot contain the additional new nodes. To see why this is so, observe that every node that is not among V_0 is on a path of length s between two nodes in V_0, and will be at distance greater than s from at least one node among $c_1^1, \ldots, c_m^1, f$.

Now, consider s-clique $R = \{c_1^1, \ldots, c_1^s, \ldots, c_m^1, \ldots, c_m^s, f\}$. We claim that there exists a connected s-clique C such that $R \subseteq C$ if and only if ψ is satisfiable. Suppose first that ψ is satisfiable, and let μ be a satisfying assignment for ψ. Let C be the set containing R, as well as all nodes corresponding to literals that are satisfied in μ. Clearly, C is an s-clique, by our first observation. In addition, since C must include at least one node x_i^j corresponding to a satisfied literal in each clause c_i, the set C is connected (as x_i^j will connect between c_i^s and c_{i+1}^1).

For the other direction, suppose there exists such a set C. By Observation 3, C can contain only nodes from V_0, and by Observation 2, C will not contain nodes corresponding to a literal and its negation. Therefore, C defines a truth assignment for ψ, in which true is assigned to literals corresponding to nodes in C. As before, the fact that C is connected implies that the truth assignment gives the value of true to at least one literal in each C_i, i.e., ψ is satisfiable. □

Theorem 5.6 implies that we cannot have an efficient algorithm that prunes useless branches whenever possible. Instead, we

apply a simple optimization, that is sufficient, but not complete, for pruning (i.e., branches are pruned only if they cannot lead to a result, but not all such branches will be pruned). In particular, we remove from P each node v for which the set of nodes $R \cup \{v\}$ is not completely contained in a single connected component of

$$G[R \cup \{v\} \cup (P \cap N^s(v))] .$$

Intuitively, v can be added to R, only if eventually, we may find nodes in $P \cap N^s(v)$ $(= N^{\forall,s}(R \cup \{v\}))$ that can fill in the gaps between the nodes in R to derive a single connected graph. Therefore, if we discover that this is not the case, we can remove v from P, as it can never be in a connected s-clique together with R.

Example 5.7. Consider running CsCliques2 algorithm with the above feasibility check, on graph H from Figure 3 (a). When first called, $R = X = \emptyset$, $P = \{v_1, \ldots, v_6\}$ and $s = 2$. Since, R is empty, we iterate over all nodes in P. Suppose the first node chosen is v_1. In the recursive call to CsCliques2, we will have $R = \{v_1\}$, $P = \{v_2, v_3, v_5, v_6\}$ and $X = \emptyset$. We will iterate over all nodes in P. In the second iteration of the loop of Lines 3-6, $v = v_3$ (v_2 was removed from P at the end of the first iteration) and $P \cap N^s(v) = v_5$. Now $R \cup \{v\} = \{v_1, v_3\}$ is not a connected component in $G[R \cup \{v\} \cup (P \cap N^s(v))] = \{v_1, v_3, v_5\}$ and v_3 will be removed from P without calling the recursion.

6 FINDING LARGE RESULTS

We now consider the problem of finding large maximal connected s-cliques. In particular, assume we are given an integer $k \geq 0$, and our goal is to find all maximal connected s-cliques C such that $|C| \geq k$. When $s = 1$, s-cliques are standard cliques. For this case, it is well known that determining whether there exists a clique of size k is NP-complete. Therefore, it is interesting to consider k as a parameter of the problem and determine whether a fixed parameter algorithm exists, i.e., whether there is an algorithm that runs in time $O(f(k) \cdot |G|^{O(1)})$, for an arbitrary function f. Since k is expected to be much smaller than $|G|$, such an algorithm would be useful. Unfortunately, determining existence of a clique is known to be $W[1]$-complete, i.e., we cannot expect to find an algorithm with time $O(f(k) \cdot n^{O(1)})$.

We now consider the case in which $s > 1$. Interestingly, determining whether there exists a s-clique of size k *is* fixed parameter tractable with respect to k, for both the case of connected s-cliques, and for arbitrary s-cliques. This gives hope that large maximal connected s-cliques can be enumerated with delay between answers that is exponential in k, but not in $|G|$. Unfortunately, for general s-cliques, this is not the case, as the following theorem states. (The proof has been omitted due to space limitations.) For connected s-cliques this problem is still open.

Theorem 6.1. *Let k and $s > 1$ be integers.*

(1) *The problem of determining whether a graph G contains a (connected) s-clique of size k is NP-complete, but is fixed parameter tractable with respect to k.*
(2) *It is not possible to enumerate all maximal s-cliques of size at least k in a graph G with fixed parameter delay with respect to k, unless $W[1] = W[0]$.*

Our algorithms can already mine all maximal connected s-cliques of size at least k, by simply finding all maximal connected s-cliques, and then filtering out all those that do not satisfy the size bound. This process may be highly inefficient, as many smaller s-cliques will be generated. We consider optimizations that can be made to our algorithms to speed up the process of finding all maximal connected s-cliques of size at least k. In PolyDelayEnum, we replace the queue with a priority queue that returns larger maximal connected s-cliques first. In the algorithms CsCliques1 and CsCliques2, we prune (i.e., do not make recursive calls) the cases in which $|R| + |P| < k$. Clearly, in such cases it is not possible to create maximal connected s-cliques with k nodes at least.

7 EXPERIMENTAL RESULTS

Our algorithms were implemented in 32bit C++, as an extension of the SNAP library [21]. All experimentation was run on a Win7 desktop with 16GB RAM and an Intel i5-4570 processor, with 2GB of memory allocated to the program.

We run our algorithms on synthetic Erdős-Réyni (ER) graphs and scale-free (SF) graphs (which simulate social networks) of varying sizes, generated by the SNAP library. All data points in our figures are derived by generating three random graphs of the same size, and taking the average runtime in seconds. We also use several real datasets, as follows, taken from [20]. (Some of these datasets are directed graphs, but for our experimentation, we ignore the direction of the edges.)

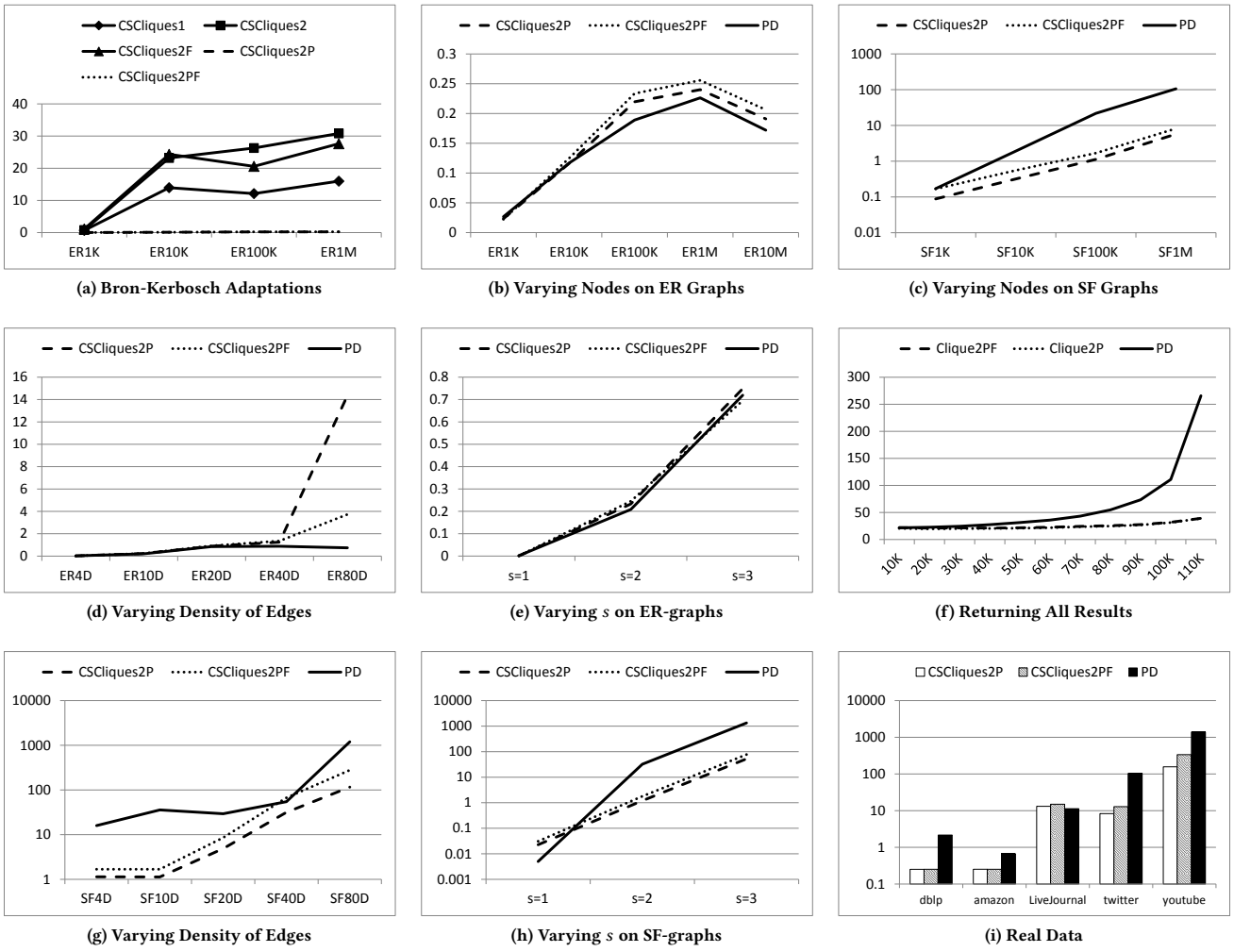
- DBLP, with 317,080 nodes and 1,049,866 edges
- Amazon, with 334,863 nodes and 925,872 edges
- LiveJournal, with 3,997,962 nodes and 34,681,189 edges
- Twitter, with 81,306 nodes and 1,768,149 edges
- YouTube, with 1,134,890 nodes and 2,987,624 edges

One of the most costly operations in all algorithms is computing the set $N^s(v)$ for various nodes v. To save time, whenever we compute this set, we store it in a hash table, to be reused again later on, if needed. For large data sets, there is insufficient memory to store all neighbors of distance s within the hash table. When memory begins to run low, we remove some entries from the hash table (using an LRI ordering) to make room for new neighbor results.

Comparing Bron-Kerbosch Adaptations. We start by comparing the baseline versions of our Bron-Kerbosch adaptations, i.e., CSClique1 and CSClique2, with the versions including the two optimizations considered. We append the letter "P" and "F" to CSClique2 to indicate that pivoting is used and that our feasibility check is performed.

Figure 9a shows the result of running our adaptations of the Bron-Kerbosch algorithm to find 100 connected 2-cliques, with and without various optimizations, for random ER graphs of varying numbers of nodes (between one thousand and one million), and average node degree 10. As was to be expected, CSClique1 is significantly superior to CSClique2. However, all three versions that do not use pivoting are much slower that the pivoting versions (the two overlapping lines at the bottom of the graph), running 30 to 60 times slower than the pivoting versions, for graphs of one million nodes. For SF graphs, the gap is even larger, with the non-pivoting versions running approximately 120 times slower on graphs with only one thousand nodes (and average node degree 10). For this reason, in the remainder of the experimentation, we only consider the algorithms CSClique2P and CSClique2PF, along with PolyDelayEnum (which will be denoted PD in our graphs).

Varying Node Size. We continue in our study of how the size of the graph (as determined by the number of its nodes) affects runtime. As before, we generated random graphs with between



(a) Bron-Kerbosch Adaptations

(b) Varying Nodes on ER Graphs

(c) Varying Nodes on SF Graphs

(d) Varying Density of Edges

(e) Varying s on ER-graphs

(f) Returning All Results

(g) Varying Density of Edges

(h) Varying s on SF-graphs

(i) Real Data

Figure 9: Execution times for varying parameters.

one thousand and one million nodes, and average node degree 10. We chose $s = 2$ and measured the time to return 100 connected s-cliques.

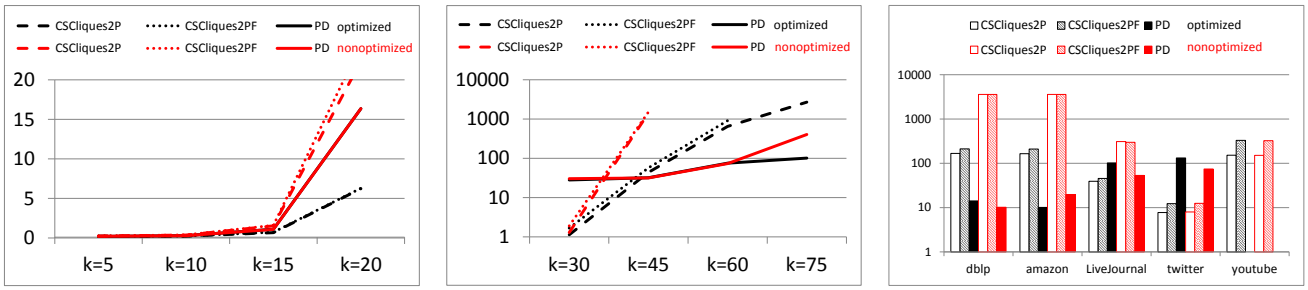
This experiment appears in Figures 9b and 9c. All times are in seconds, and that in Figure 9c the times are in log-scale. Algorithm PolyDelayEnum (PD in the graphs) slightly outperforms CsCliques2P and CsCliques2PF on the ER graphs, but is significantly worse on the SF graphs. In addition, all algorithms perform worse on the latter, than on the former, probably due to the fact that average size of connected 2-cliques generated is much larger in a scale-free graph. For example, PolyDelayEnum returned connected 2-cliques of average size 10.79 and 105.83, respectively, on the ER and SF graphs of size one million. The change in the trend of Figure 9b between 1M and 10M follows since the average size of connected 2-cliques is smaller over graphs with 10 million nodes, as the graph is sparser.

Varying Edge Density. We consider how the density of the edges affects the speed in which results can be returned. We generated random graphs with 100 thousand nodes, and an average degree of 4, 10, 20, 40 and 80. We chose $s = 2$, and measure the time to return 100 connected 2-cliques.

Figures 9d and 9g contain the result of this experiment for ER and SF graphs, respectively. As the density increases, algorithm PolyDelayEnum once again outperforms CsCliques2P and CsCliques2PF on the ER graphs. CsCliques2PF is superior to CsCliques2P for degree density 80, as it avoids many recursive calls. (This is in contrast to the many other cases in which we observe that its time is inferior, as the overhead for testing feasibility is large.) On the SF graphs, on the other hand, PolyDelayEnum is the slowest and CsCliques2PF performs slightly worse than CsCliques2P probably do to the high connectivity of scale-free graphs, which causes the feasibility check to always return true.

Varying s. We study how the choice of s affects the runtime. Once again, we generated a random graph with 100K nodes and average degree of 10. We measure the time to generate 100 connected s-cliques for values of s varying from 1 to 3. Recall that when $s = 1$, we actually return cliques.

The runtime appears in Figures 9e and 9h. Runtime increases as s increases, as it is increasingly more expensive to find neighbors of distance s. As before, runtime is significantly slower for SF graphs, as they have s-cliques that are much larger. The algorithm PolyDelayEnum is slower than the others, but returns s-cliques that are larger, on average. (For example, for $s = 3$, on SF graphs,



(a) ER Graphs (b) SF Graphs (c) Real Data

Figure 10: Execution time for returning large results.

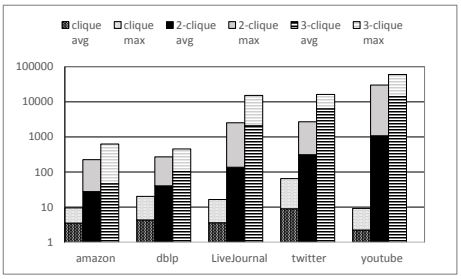


Figure 11: Average and max sizes.

PolyDelayEnum returns 3-cliques of average size 1084, while the other algorithms return 3-cliques with average size 668.) Even for $s = 3$, the runtime remains reasonable, and in practice, larger values of s are usually not of interest.

Returning all Results. We consider the runtime when returning *all* connected s-cliques in a graph. For this experiment, we generated an ER graph with 100 thousand nodes and average degree of 10. This graph has 112,134 connected 2-cliques. We measure time elapsed between generating every 10 thousand results, until all results are generated.

Figures 9f contains the result of this experiment. Recall that PolyDelayEnum runs in polynomial delay (Theorem 4.2), i.e., the delay between results is polynomial in $|V(G)|$. However, the CsCliques$i$ variations have no such guarantee. It is therefore somewhat surprising to see that, in practice, the delay between successive sets of 10 thousand results grows for PolyDelayEnum while for CsCliques2P and CsCliques2PF remaining almost steady. Perhaps this can be explained by the high memory requirements of PolyDelayEnum (e.g., as it must store all results created), which makes access to auxiliary data structures more expensive as more results are created.[2]

Non-synthetic Datasets. In Figure 9i we show the results of returning 100 connected 2-cliques over the real datasets presented in the beginning of this section. Once again, the algorithms CsCliques2P and CsCliques2PF outperform PolyDelayEnum almost consistently. PolyDelayEnum performs better only over the LiveJournal dataset, but the difference in runtime is small.

Returning Large Results. We consider the heuristics presented in Section 6 to find 100 connected 2-cliques that are larger than k, for some given size k. Once again, we generated random graphs with 100K nodes and average degree of 10 and used the real datasets as well. Our range of values for k for the random graphs was determined by the average size of answers returned by the algorithms when no size restriction was given. (With no size restriction, the average result size for the ER graph was approximately 10, and for the SF graph was between 35 and 60, depending on the algorithm used.) For the real graphs, we set $k = 50$.

In Figures 10a, 10b and 10c, the result of this experiment appears, comparing the optimized version of the algorithms (in black) against the regular algorithms (in red) until a hundred large results are found. Interestingly, the time for PolyDelayEnum on SF graphs is steady, as this algorithm naturally creates large results in this setting. The Bron-Kerbosch adaptations timed out on SF graphs with large k and PolyDelayEnum timed out on the youtube graph, running over an hour. Clearly, the optimizations for CsCliques2P and CsCliques2PF improve the runtime significantly in most cases, but the optimization for algorithm PolyDelayEnum is not consistently superior.

We note that the optimizations discussed in Section 6 were very important in achieving the runtimes shown. The speedup with respect to the regular (non-optimized) versions were significant in most cases.

Comparing cliques and s-cliques size. To further motivate enumerating s-cliques for $s > 1$, we compare the sizes of s-cliques for $s = 1, 2, 3$ over the real datasets. For this experiment we randomly sampled 100 s-cliques, for each of the datasets and each of the values of s. We then calculated both average and maximum sizes. The result of this experiment appears in Figure 11. The datasets are organized by increasing edge density, and for each graph, we plot the average and maximum size of cliques, 2-cliques and 3-cliques. Unsurprisingly, the size of s-cliques is larger (for all choices of s) when the graph is more dense. In addition, observe that as s grows larger, the average and maximum size of the s-cliques increases. Note that the sizes are in log scale, and hence, differ significantly.

Depending on the application, finding highly cohesive sets of larger (or smaller) sizes may be more useful. For example, in many settings, communities that are very small, or very large, may be less useful. Smaller communities may not well-represent the actual facts on the ground, while huge communities may contain people who are not sufficiently related. The ability to choose a value for s, and then enumerate (as our algorithms do) gives the user maximum flexibility.

[2] Also, the reader may note that while the delay given by Theorem 4.2 is polynomial in the input, it is still quite large.

Summary of Results. As is apparent from the results above, all algorithms PolyDelayEnum, CsCliques2P and CsCliques2PF have runtimes that are quite reasonable. Usually, the adaptations of Bron-Kerbosch have superior runtime to that of the algorithm PolyDelayEnum, but the latter may be preferable for returning larger s-cliques, particularly on sparse graphs. When comparing CsCliques2P and CsCliques2PF, one can observe that usually the overhead of checking for feasibility (done in the latter algorithm) is larger than the gains derived by avoiding unnecessary recursive calls. (The only exception was for highly dense ER-graphs, in which feasibility checking clearly pays off.)

Another important difference between PolyDelayEnum and the Bron-Kerbosch adaptations is in memory requirements. Running PolyDelayEnum requires additional data structures (e.g., the queue Q and index $\mathcal{I}$). Thus, its memory requirements are linear in the size of the output. This contrasts with the Bron-Kerbosch adaptations, which require memory that is linear in the input. In our implementation, we assumed that the structures for PolyDelayEnum fit in main memory, but for larger inputs (or when desiring to run the algorithm until all results have been found), it would be necessary to use external memory structures.

8 CONCLUSION

This paper studied the problem of finding maximal connected s-cliques in a graph—a problem of high interest, due to the usefulness of clique relaxations. We have presented the first algorithms for this problem, by taking two completely different approaches for solving this problem. The correctness of our algorithms is proven and experimentation shows the efficiency of our approaches.

As future work, we intend to study applications in which connected s-cliques can be useful, such as community detection and link prediction. Optimizations of the algorithms, for special types of graphs (e.g., sparse graphs or bipartite graphs) are also an interesting direction for future work. Another important direction is adapting the algorithms to a distributed environment, as returning all s-cliques for large graphs can become infeasible for a single machine.

ACKNOWLEDGMENTS

The authors were partially supported by the Israel Science Foundation (Grant 879/16).

REFERENCES

[1] E. A. Akkoyunlu. 1973. The Enumeration of Maximal Cliques of Large Graphs. *SIAM J. Comput.* 2, 1 (1973), 1–6. https://doi.org/10.1137/0202001
[2] Balabhaskar Balasundaram, Sergiy Butenko, and Svyatoslav Trukhanov. 2005. Novel Approaches for Analyzing Biological Networks. *J. Comb. Optim.* 10, 1 (2005), 23–39. https://doi.org/10.1007/s10878-005-1857-x
[3] Devora Berlowitz, Sara Cohen, and Benny Kimelfeld. 2015. Efficient Enumeration of Maximal k-Plexes. In *SIGMOD*.
[4] Kamanashis Biswas, Vallipuram Muthukkumarasamy, and Elankayer Sithirasenan. 2013. Maximal clique based clustering scheme for wireless sensor networks. In *ISSNIP*.
[5] Vladimir Boginski, Sergiy Butenko, and Panos M. Pardalos. 2005. Statistical analysis of financial networks. *Comput. Statistics and Data Analysis* 48, 2 (2005), 431–443. https://doi.org/10.1016/j.csda.2004.02.004
[6] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: Finding All Cliques of an Undirected Graph. *Commun. ACM* 16, 9 (Sept. 1973), 575–577. https://doi.org/10.1145/362342.362367
[7] Lijun Chang, Jeffrey Xu Yu, and Lu Qin. 2013. Fast Maximal Cliques Enumeration in Sparse Graphs. *Algorithmica* 66, 1 (2013), 173–186. https://doi.org/10.1007/s00453-012-9632-8
[8] James Cheng, Yiping Ke, Ada Wai-Chee Fu, Jeffrey Xu Yu, and Linhong Zhu. 2011. Finding maximal cliques in massive networks. *ACM Trans. Database Syst.* 36, 4 (2011), 21:1–21:34. https://doi.org/10.1145/2043652.2043654
[9] James Cheng, Yiping Ke, Ada Wai-Chee Fu, Jeffrey Xu Yu, and Linhong Zhu. 2011. Finding Maximal Cliques in Massive Networks. *ACM Trans. Database Syst.* 36, 4, Article 21 (Dec. 2011), 34 pages. https://doi.org/10.1145/2043652.2043654
[10] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. 2008. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *J. Comput. Syst. Sci.* 74, 7 (2008), 1147–1159.
[11] Alessio Conte, Roberto De Virgilio, Antonio Maccioni, Maurizio Patrignani, and Riccardo Torlone. 2016. Finding All Maximal Cliques in Very Large Social Networks. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016*. 173–184. https://doi.org/10.5441/002/edbt.2016.18
[12] David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing All Maximal Cliques in Large Sparse Real-World Graphs. *ACM Journal of Experimental Algorithmics* 18 (2013). https://doi.org/10.1145/2543629
[13] Y. Fried, D.A. Kessler, and N.M. Shnerb. 2016. Communities as Cliques. *Nature Scientific Reports* 6, 35648 (2016).
[14] E. Harley, A. Bonner, and N. Goodman. 2001. Uniform integration of genome mapping data using intersection graphs. *Bioinformatics* 17, 6 (2001), 487–494.
[15] Hiro Ito and Kazuo Iwama. 2009. Enumeration of Isolated Cliques and Pseudo-cliques. *ACM Trans. Algorithms* 5, 4, Article 40 (Nov. 2009), 21 pages. https://doi.org/10.1145/1597036.1597044
[16] Daxin Jiang and Jian Pei. 2009. Mining Frequent Cross-graph Quasi-cliques. *ACM Trans. Knowl. Discov. Data* 2, 4, Article 16 (Jan. 2009), 42 pages. https://doi.org/10.1145/1460797.1460799
[17] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. 1988. On Generating All Maximal Independent Sets. *Inf. Process. Lett.* 27, 3 (1988), 119–123.
[18] Mehdi Kargar and Aijun An. 2011. Keyword Search in Graphs: Finding R-cliques. *Proc. VLDB Endow.* 4, 10 (July 2011), 681–692. https://doi.org/10.14778/2021017.2021025
[19] Ina Koch. 2001. Enumerating All Connected Maximal Common Subgraphs in Two Graphs. *Theor. Comput. Sci.* 250, 1-2 (Jan. 2001), 1–30. https://doi.org/10.1016/S0304-3975(00)00286-3
[20] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data. (June 2014).
[21] Jure Leskovec and Rok Sosič. 2014. SNAP: A general purpose network analysis and graph mining library in C++. http://snap.stanford.edu/snap. (June 2014).
[22] David Liben-Nowell and Jon M. Kleinberg. 2007. The link-prediction problem for social networks. *JASIST* 58, 7 (2007), 1019–1031. https://doi.org/10.1002/asi.20591
[23] Guimei Liu and Limsoon Wong. 2008. Effective Pruning Techniques for Mining Quasi-Cliques. In *ECML PKDD*.
[24] R.Duncan Luce. 1950. Connectivity and generalized cliques in sociometric group structure. *Psychometrika* 15, 2 (1950), 169–190. https://doi.org/10.1007/BF02289199
[25] S. Mohseni-Zadeh, P. Brézellec, and J.-L. Risler. 2004. Cluster-C, an algorithm for the large-scale clustering of protein sequences based on the extraction of maximal cliques. *Comput. Biology and Chemistry* 28, 3 (2004), 211 – 218. https://doi.org/10.1016/j.compbiolchem.2004.03.002
[26] J.W. Moon and L. Moser. 1965. On cliques in graphs. *Israel Journal of Mathematics* 3, 1 (1965), 23–28. https://doi.org/10.1007/BF02760024
[27] Kevin A. Naudé. 2016. Refined pivot selection for maximal clique enumeration in graphs. *Theoretical Computer Science* 613 (2016), 28 – 37.
[28] F. Mahdavi Pajouh and B. Balasundaram. 2012. On inclusionwise maximal and maximum cardinality k-clubs in graphs. *Discrete Optimization* 9, 2 (2012), 84–97. https://doi.org/10.1016/j.disopt.2012.02.002
[29] G. Palla, I Derenyi, and T. Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (2005), 814–818.
[30] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. 2013. On clique relaxation models in network analysis. *European Journal of Operational Research* 226, 1 (2013), 9–18. https://doi.org/10.1016/j.ejor.2012.10.021
[31] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. 2015. Locally Densest Subgraph Discovery. In *KDD*.
[32] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The Worst-case Time Complexity for Generating All Maximal Cliques and Computational Experiments. *Theor. Comput. Sci.* 363, 1 (Oct. 2006), 28–42. https://doi.org/10.1016/j.tcs.2006.06.015
[33] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. 2013. Denser Than the Densest Subgraph: Extracting Optimal Quasi-cliques with Quality Guarantees. In *SIGKDD*.
[34] Stanley Wasserman and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications*. Cambridge University Press.
[35] Bin Wu and Xin Pei. 2007. A Parallel Algorithm for Enumerating All the Maximal k -Plexes. In *PAKDD*.
[36] B. Yan and S. Gregory. 2009. Detecting communities in networks by merging cliques. In *ICIS*.
[37] Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. 2007. Out-of-core Coherent Closed Quasi-clique Mining from Large Dense Graph Databases. *ACM Trans. Database Syst.* 32, 2 (June 2007).
[38] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. 2010. Finding top-k maximal cliques in an uncertain graph. In *ICDE*.