Theoretical Computer Science 899 (2022) 1–24

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the overall and delay complexity of the CLIQUES and Bron-Kerbosch algorithms

Alessio Conte [a,*], Etsuji Tomita [b]

[a] University of Pisa, Largo Bruno Pontecorvo 3, Pisa, 56127, Italy
[b] The Advanced Algorithms Research Laboratory, The University of Electro-Communications, Chofugaoka 1–5–1, Chofu, Tokyo, 182–8585, Japan

ARTICLE INFO

Article history:
Received 16 April 2021
Received in revised form 7 October 2021
Accepted 7 November 2021
Available online 12 November 2021

Keywords:
Maximal cliques
Graph enumeration
Output sensitive
Delay

ABSTRACT

We revisit the maximal clique enumeration algorithm CLIQUES by Tomita et al. that appeared in Theoretical Computer Science in 2006. It is known to work in $O(3^{n/3})$-time in the worst-case for an n-vertex graph. This is worst-case optimal with respect to the input size, but there is little knowledge about its performance with respect to the output. In this paper, we extend the time-complexity analysis with respect to the maximum size and the number of maximal cliques, and to its delay, solving issues that were left as open problems since the original paper. In particular, we prove that CLIQUES has $\Omega(3^{n/6})$ delay and that, even if we allow to change the pivoting strategy, a variant having polynomial delay cannot be designed unless $P = NP$. These same results apply to the related Bron-Kerbosch algorithm. On the positive side, we show that the complexity of CLIQUES and Bron-Kerbosch is amortized polynomial on graphs with logarithmic clique number. As these algorithms are widely used and regarded as fast "in practice", we are interested in observing their practical behavior: we run an evaluation of CLIQUES and three Bron-Kerbosch variants on over 130 real-world and synthetic graphs, observing how the clique number almost always satisfies our logarithmic constraint, and that their performance seems far from its theoretical worst-case behavior in terms of both total time and delay.[1]

1. Introduction

A *clique* is defined to be a subgraph in which all vertices are pairwise adjacent. In particular, it is *maximal* if it is not contained in a strictly larger clique. Given a graph, the enumeration of all its maximal cliques is a fundamental and important problem in graph theory [25] and has many practical applications in clustering, data mining, bioinformatics, social networks, and more, mostly related to community detection (see [13] for more details). An *independent set* of a graph G is a clique of the complement graph $\bar{G}$.

Tsukiyama et al. [32] gave the first algorithm MIS for enumerating maximal independent sets with a theoretical time-complexity analysis. For a graph G with n vertices and m edges, MIS enumerates all maximal independent sets in time $O(nm)$ per maximal independent set; this can be adapted to enumerate maximal cliques in the same complexity per solution [19].

* Corresponding author.
E-mail address: conte@di.unipi.it (A. Conte).

[1] Preliminary versions containing part of this work appeared in [10] and [27].

https://doi.org/10.1016/j.tcs.2021.11.005

Tomita et al. [28,29] and Bron and Kerbosch [3] independently presented different algorithms for the problem, although it was later understood that their pruning techniques were the same. These algorithms do not have output-sensitive guarantees but boast good practical performance.

Furthermore, while the complexity of Bron-Kerbosch is as of today still unknown, CLIQUES [28], based on *depth-first search* algorithms for finding a *maximum* clique [14,26], was the first maximal clique enumeration algorithm with proven worst-case optimal time (as a function of n): indeed its $O(3^{n/3})$-time worst-case time complexity matches the number of maximal cliques in Moon-Moser graphs [21]. The results in [28] were also reviewed in [23] and [2].

The algorithms modeled after Tsukiyama et al. typically follow the *reverse-search* framework [1] and enumerate the cliques in an *output-sensitive* fashion: if a problem has size n and α solutions, an algorithm is output-sensitive if its complexity is $O(poly(\alpha, n))$-time, for some polynomial function $poly(\cdot)$, and *amortized polynomial time* if it is just $O(\alpha\ poly(n))$.

In this line, steady improvements have been made by Chiba and Nishizeki [6], Johnson et al. [16], Makino and Uno [19], Chang et al. [5], Comin and Rizzi [7], and Conte et al. [9] and Manoussakis [20]. Most of these algorithms prove a stronger result than output-sensitivity, that is *polynomial delay*, *i.e.*, the time elapsed between two consecutive outputs of a solution is polynomial. Some rely on matrix multiplication, like the one by Comin and Rizzi [7], with $O(n^{2.094})$-time delay, while others on combinatorial techniques, such as the one by Conte et al. [9], with $O(qd\Delta)$-time delay, where q is the size of a maximum clique, d is the *degeneracy* of G (the smallest number such that every subgraph of G contains a vertex of degree at most d), and Δ the maximum degree. Based on CLIQUES, Eppstein et al. proposed an improved algorithm for sparse graphs that runs in $O(d(n-d)3^{d/3})$-time. In general, it is experimentally observed that algorithms based on CLIQUES and Bron-Kerbosch are fast in practice [31,11,8,24]. However, no theoretical time-complexity analysis with respect to the number of maximal cliques is made for CLIQUES in 2004 [30], 2006 [31], where the problem is noted in [30,31] as an important open problem. This is in contrast to the time-complexity analysis in reverse-search approach.

It is natural to ask whether CLIQUES is output sensitive, and whether it has polynomial delay. Furthermore, the question is motivated not just by theoretical interest in the algorithm, but by its use: while CLIQUES and related algorithms are widely used for their practical performance on real-world networks, there could be some "bad" graphs that require extensive time to be processed while having only a small number of maximal cliques. If this were the case, some applications may need to rely on guaranteed approaches for an overall more balanced performance, whereas if CLIQUES turns out to be output sensitive, then no such concern is necessary. For these reasons, this paper proposes a new complexity analysis of CLIQUES and related algorithms, that takes into account the number and the maximum size of maximal cliques.

2. Definitions and notation

We consider a simple undirected *graph* $G = (V(G), E(G))$, or simply (V, E) when G is clear from the context, with a finite set V of *vertices* and a finite set E of *unordered* pairs (v, w) of distinct vertices, called *edges*.[2] A pair of vertices v and w are *adjacent* if $(v, w) \in E$. For a vertex $v \in V$, let $\Gamma(v)$ be the set of all vertices that are adjacent to v in $G = (V, E)$, *i.e.*, $\Gamma(v) = \{w \in V \mid (v, w) \in E\}$ ($\not\ni v$).

For a subset $W \subseteq V$ of vertices, $G(W) = (W, E(W))$ with $E(W) = E \cap (W \times W)$ is called a *subgraph* of $G = (V, E)$ *induced* by W. For a set W of vertices, $|W|$ denotes the number of elements in W.

Given a subset $Q \subseteq V$ of vertices, if $(v, w) \in E$ for all $v, w \in Q$ with $v \neq w$ then the induced subgraph $G(Q)$ is called a *clique*. In this case, we may simply say that Q is a clique. If a clique is not a proper subgraph of another clique then it is called a *maximal* clique.

3. Maximal clique enumeration algorithm CLIQUES

We revisit a depth-first search algorithm, CLIQUES [28,31], which enumerates all maximal cliques of an undirected graph $G = (V, E)$, with $|V| = n$ vertices, giving the output in a tree-like form. The basic framework of CLIQUES is almost the same as that for finding a *maximum* clique, but *without the bounding condition* [14,26].

The algorithm (detailed in Algorithm 1) consists of a recursive call procedure based on two vertex sets *SUBG* and *CAND*. Initially, we set $SUBG \leftarrow V$ and $CAND \leftarrow V$, and the recursive task of CLIQUES(*SUBG*, *CAND*) is to enumerate all maximal cliques in $G(SUBG)$ which are fully contained in *CAND*.

We maintain a global variable $Q = \{p_1, p_2, ..., p_h\}$ that consists of the vertices of a current clique, and $SUBG = V \cap \Gamma(p_1) \cap \Gamma(p_2) \cap \cdots \cap \Gamma(p_h)$. The cliques found in a recursive subtree correspond to extensions of Q, which is initially empty.

The algorithm selects a certain vertex p from *SUBG* and adds p to Q. Then, we compute $SUBG_p = SUBG \cap \Gamma(p)$ as the new set of vertices in question, and generate a child recursive call. When this backtracks, we remove p from *CAND*, but not from *SUBG*, so that cliques contained in *CAND* that are maximal in $G(SUBG)$ correspond to cliques that we had not already found.

The algorithm employs two pruning methods to avoid unnecessary recursive calls, which happen to be the same as in the Bron-Kerbosch algorithms [3].

[2] For clarity, we will use the term *nodes* instead of *vertices* when talking about elements of the recursion tree of the algorithm.

Avoiding duplication: Let *FINI* (short for FINISHED) refer to $SUBG \setminus CAND$, *i.e.*, a subset of vertices of *SUBG* that were already processed by the algorithm, whereas *CAND* is the set of remaining CANDIDATE for expansion: $CAND = SUBG \setminus FINI$. Initially, we set $FINI \leftarrow \emptyset$, $CAND \leftarrow V$, $SUBG \leftarrow V$. In the subgraph $G(SUBG_p)$ with $SUBG_p = SUBG \cap \Gamma(p)$, compute

$$CAND_p = CAND \cap \Gamma(p), \quad FINI_p = FINI \cap \Gamma(p).$$

Then only the vertices in $CAND_p$ can be candidates for expanding the clique $Q \cup \{p\}$ to find *new* larger cliques.
Pivoting: Let u be the first node in *SUBG* selected in a recursive node, and call it the *pivot*. Any maximal clique Q' in $G(SUBG \cap \Gamma(u))$ is *not* maximal in $G(SUBG)$, since $Q' \cup \{u\}$ is a larger clique in $G(SUBG)$. Therefore, any maximal clique either contains u or at least a vertex in $SUBG \setminus \Gamma(u)$: this means we can skip the expansion of all vertices in $\Gamma(u)$, and only expand those in $SUBG \setminus \Gamma(u)$.

In order to minimize $|CAND \setminus \Gamma(u)|$, CLIQUES chooses the pivot $u \in SUBG$ that *maximizes* $|CAND \cap \Gamma(u)|$: this is *crucial* to the worst-case complexity of the algorithm.

And indeed, algorithm CLIQUES [28,31] (Algorithm 1) enumerates all maximal cliques in $O(3^{n/3})$-time, which is optimal in the worst-case, printing the output in a tree-like form (shown in Fig. 1).

Algorithm 1: Algorithm CLIQUES in [31].

```
   Input  : A graph G = (V, E).
   Output: All maximal cliques in G.
   /* Q ← ∅ is a global variable representing a clique */
1  CLIQUES (V, V)
2  Function CLIQUES(SUBG, CAND)
3      if SUBG = ∅ then
4          print ("clique,")   /* Q is a maximal clique */
5      else
6          u ← a vertex in SUBG maximizing |CAND ∩ Γ(u)|
           /* FINI ← ∅ */
7          while CAND \ Γ(u) ≠ ∅ do
8              p ← a vertex in CAND \ Γ(u)
9              print (p,",")   /* Q ← Q ∪ {p} */
10             SUBG_p ← SUBG ∩ Γ(p)
11             CAND_p ← CAND ∩ Γ(p)
12             CLIQUES(SUBG_p, CAND_p)
13             CAND ← CAND \ {p}
               /* FINI ← FINI ∪ {p} */
14             print ("back,")   /* Q ← Q \ {p} */
```

We can easily obtain a tree representation of all the maximal cliques from the output, where a dummy root is added to form a tree (Fig. 4 of [31]). The tree-like output format also has the practical advantage of producing a smaller output file. This is also important practically, if we want to store the result, since it saves space in the output file.

3.1. Bron-Kerbosch algorithms

For completeness, we recall the antecedent algorithm by Bron and Kerbosch for enumerating maximal cliques [3]. In the following, we will call BK the version of the Bron-Kerbosch algorithm *without* pivoting, and BKP a variant of the Bron-Kerbosch algorithm with pivoting (where no specific pivot choice is mandated).

Similarly to CLIQUES, the Bron-Kerbosch algorithm uses a recursive backtracking strategy that tentatively expands a clique in all possible ways and use pivoting to prune some recursive calls. However, we will highlight some key differences.

One difference between the algorithms is that CLIQUES outputs all maximal cliques in a *tree*-like format (Lines 4, 9, 14), to avoid the time for outputting a maximal clique every time it is found that is proportional to the size of a maximal clique found. Another is the choice of the pivot vertex u as the one in *SUBG* maximizing $|CAND \cap \Gamma(u)|$. The pivot selection of the maximum-degree and the tree-like outputting are crucial in CLIQUES so that it accomplishes the worst-case optimal $O(3^{n/3})$-time complexity.

Without these components, the running time of BKP is $\Omega(n3^{n/3})$, e.g., on a Moon-Moser graph [21] plus one edge; non-trivial upper bounds are not known for its complexity and finding one is an interesting open question. On the other hand, BK can be trivially observed to run in $\Omega(2^n)$ on a complete graph, as it essentially generates all subsets of every clique.

While BK and CLIQUES handle vertices differently (see the pseudo code in [3]), we observe that the recursion tree of BK and BKP can be simulated by modifying Algorithm 1: we can simulate BK by entirely removing Line 6, and replacing Line 7 with "**while** $CAND \neq \emptyset$ **do**". To simulate BKP, instead, we can replace Line 6 with some arbitrary choosing strategy. While this operation does not yield exactly the BK and BKP algorithms, it is sufficient to get an understanding of the algorithms and observe that the results we prove for the complexity of CLIQUES apply to those algorithms as well.

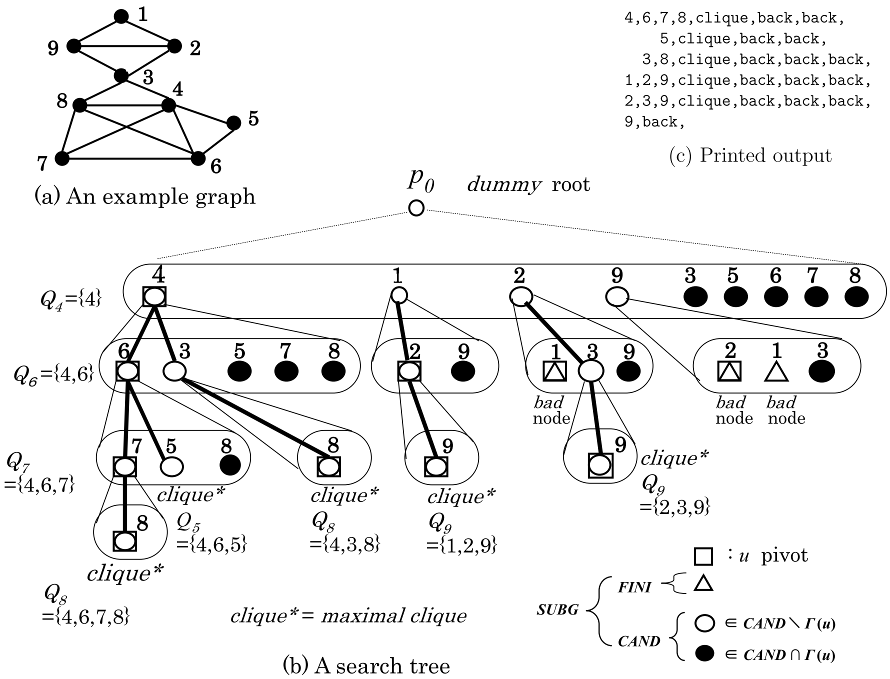

Fig. 1. An example run of CLIQUES from [31].

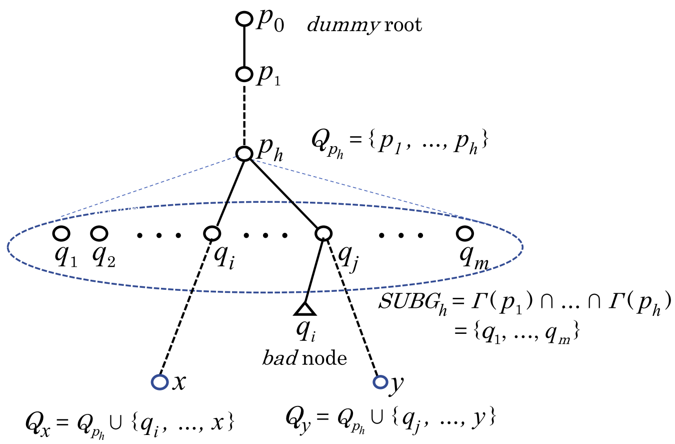

Fig. 2. Part of a search tree of CLIQUES.

3.2. Search tree

We here recall the *search tree* from [31], as a useful formalism to represent the enumeration process of CLIQUES. This will then be useful to analyze its complexity.

- The root of the search tree is a newly introduced *dummy* root p_0 ($\notin V$) to form a tree, corresponding to the start of the algorithm; every other node of the tree represents a nested recursive call.
- Every vertex in V is a child of the dummy root p_0, and every node of the search tree except the root corresponds to a vertex in V.
- Assume we have a path from the dummy root p_0 to a certain node p_h in the search tree as a sequence of nodes $p_0, p_1, p_2, \ldots, p_h$, corresponding to the vertices $v_{p_0}, v_{p_1}, v_{p_2}, \ldots, v_{p_h}$ of V: the node p_h corresponds to the recursive call having $Q = \{v_{p_0}, v_{p_1}, v_{p_2}, \ldots, v_{p_h}\}$.
- Taking the node p_h above, let $SUBG_h = V \cap \Gamma(p_1) \cap \Gamma(p_2) \cap \cdots \cap \Gamma(p_h)$. Then, every vertex in $SUBG_h$ is a child of p_h in the search tree.

Suppose $u \in SUBG_h$ maximizing $CAND \cap \Gamma(u)$ is chosen as a pivot in $SUBG_h$, in the recursive call corresponding to p_h: then every node in $\Gamma(u) \cap SUBG_h$ corresponds to a *leaf* of p_h since it should not be expanded by CLIQUES (in the corresponding recursive call) according to the second pruning method (pivoting). Such a leaf in $\Gamma(u)$ is called a *black node* or *black leaf*, and its associated recursive call is *not* performed by the algorithm.

Suppose $SUBG_h = FINI_h \cup CAND_h$, $q_i, q_j \in SUBG_h \setminus \Gamma(u)$, where $i < j$, q_i and q_j are adjacent ($(q_i, q_j) \in E(G)$), and $q_i \in FINI_h$. Then $q_j \in SUBG_h$ has a child q_i since q_j and q_i are adjacent in $SUBG_h$, but the child q_i should not be expanded by CLIQUES according to the first pruning method (avoiding duplication), and hence it is a leaf of the search tree. Such a leaf q_i is called a *bad node* or *bad leaf*. Note that if a bad node were expanded it could not lead to a *new* maximal clique.

When the above $SUBG_h$ is a singleton $\{q_1\}$ then the q_1 is a leaf in the search tree. The search tree consists of the nodes from $V \cup \{p_0\}$ and the parent-child relationship holds iff one of the above conditions holds.

Let q_i be a child of p_h, then the set $\{p_1, p_2, ..., p_h, q_i\}$ constitutes a *clique*, called an *accompanying clique* and is denoted by $Q_{p_1,...,q_i}$, or simply Q_{q_i}, or Q when it is clear.

Fig. 1 shows an example run of CLIQUES [31] (b) on an example graph (a), and the resulting printed output with appropriate indentations (c). Fig. 2 shows a part of a general search tree.

4. Overall complexity of CLIQUES

The time-complexity of CLIQUES directly depends on the size of the search tree. Suppose we have a path from the dummy root p_0 to a certain node p_h in the search tree as a sequence of nodes $p_0, p_1, p_2, ..., p_h$. Then the set $\{p_1, p_2, ..., p_h\}$ is an *accompanying clique*.

We are interested in the accompanying clique Q across different search tree nodes, and in particular, let us observe the following:

Lemma 4.1. *The accompanying clique Q is distinct in any internal (non-leaf) node of a search tree of* CLIQUES.

Proof. Let x and y be any pair of non-root internal distinct nodes in the search tree of CLIQUES, where x is generated before y in the search tree. Let the nearest common ancestor of x and y be p_h, and let the path from the dummy root p_0 to node p_h be $p_0, p_1, p_2, ..., p_h$ with the accompanying clique $Q_{p_h} = \{p_1, p_2, ..., p_h\}$. In addition, let the path from p_h to x be $p_h, q_i, ..., x$ and that from p_h to y be $p_h, q_j, ..., y$, respectively, and let $SUBG_h = \Gamma(p_1) \cap \Gamma(p_2) \cap ... \cap \Gamma(p_h) = \{q_1, q_2, ..., q_i, ..., q_j, ..., q_m\}$ ($i < j$). See Fig. 2. So, $Q_x = Q_{p_h} \cup \{q_i, ..., x\}$ and $Q_y = Q_{p_h} \cup \{q_j, ..., y\}$. When we visit node q_j we see that node q_i is in $FINI_h = SUBG_h \setminus CAND_h$ where $CAND_h = \{q_j, ..., q_m\}$ at that moment. If $q_i \notin \Gamma(q_j)$ then it clearly follows that $q_i \notin Q_y$. When $q_i \in \Gamma(q_j)$, q_i in the descendants of q_j is a *bad* node (leaf) since the previous node q_i is in $FINI_h$. Moreover, the *bad* node q_i is not in the path $q_j, ..., y$ since all nodes $q_j, ..., y$ are internal nodes by the assumption. Thus, $q_i \notin Q_y$. In any case, it follows that $q_i \in Q_x \setminus Q_y$ and the lemma is proved. □

Now let q be the size of a maximum clique and α the number of maximal cliques; we can prove the following on the complexity of CLIQUES:

Theorem 4.2. *The search tree of* CLIQUES *has at most $(1 + \Delta)\alpha 2^q$ nodes. Consequently, the running time of* CLIQUES is $O(\alpha 2^q n^2 \Delta)$.

Proof. Lemma 4.1 implies that the number of internal nodes is bounded by the number of possible accompanying cliques, *i.e.*, distinct non-maximal cliques of G. Each maximal clique has at most 2^q distinct subsets, so the internal nodes are at most $\alpha 2^q$, and the number of leaves of the search tree is at most $\Delta\alpha 2^q$. The number of nodes of a search tree is thus at most $(1 + \Delta)\alpha 2^q$. Since each node can be executed in $O(n^2)$-time [31], the statement follows. □

Theorem 4.2 has the following noteworthy consequence (the proof is straightforward, but reported in [27] for completeness):

Corollary 4.3. *The running time of* CLIQUES *on a graph G with n vertices and maximum clique size $q = O(\log n)$ is amortized polynomial.*

Proof. If $q = O(\log n)$ then $q \le c \log_2 n$ for some constant c. From Theorem 4.2 the running time of cliques is $O(\alpha 2^{c \log_2 n} n^2 \Delta)$. As $2^{c \log_2 n} = n^c$, $\Delta \le n - 1$, and 2^c is a constant, the running time is $O(\alpha n^{c+3})$, that is amortized polynomial. □

This condition immediately applies to many sparse graphs, where Δ is assumed to be small, and even in graphs with large Δ but small degeneracy d, as $q \le d + 1 \le \Delta + 1$. Corollary 4.3 however claims more: even dense graphs may satisfy this property. A simple example is the complete bipartite graph $K_{\frac{n}{2},\frac{n}{2}}$, which is by no means sparse as all vertices have degree $n/2$, but the size of a maximum clique is 2. It is often observed that the size q of a maximum clique is $O(\log n)$ in real-world graphs (with n vertices). To support this claim, and the scope of Corollary 4.3, we analyzed over a hundred real-world graphs, reporting our findings in Section 7. Finally, we recall that this corollary also holds for BKP.

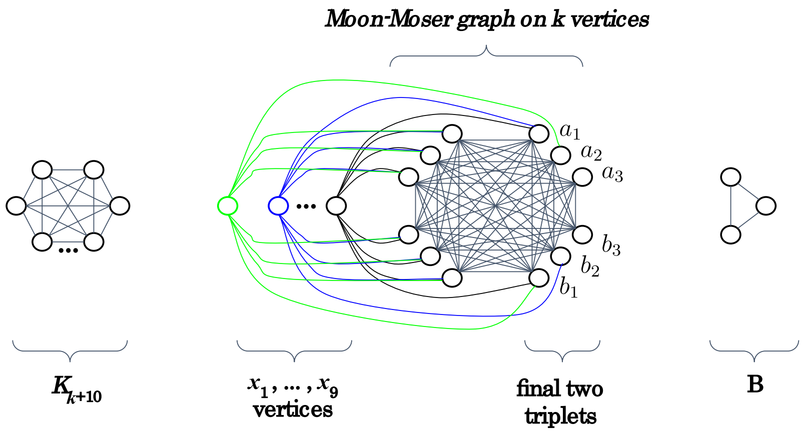

Fig. 3. Schema of a graph class where CLIQUES and BKP have exponential delay.

5. Delay of CLIQUES and Bron-Kerbosch

Next, we consider the *delay* of the algorithms, that is, the maximum time which can elapse between two consecutive outputs of a solution. In this section, we prove that both algorithms have exponential delay in the worst case.

We use Algorithm 1 as reference since it models the structure of both CLIQUES and BKP. Observe that the time taken by a recursive node is polynomial ($O(n^2)$ and $\Omega(1)$), so exponential delay incurs iff the algorithm encounters an exponentially long sequence of consecutive recursive nodes with no output. As the depth of the tree is $O(q)$, a leaf is always encountered after $O(q)$ nodes, so this sequence will also contain exponentially many *leaves*.

We build a graph G based on an integer k, which we assume to be a multiple of 3 and not smaller than 12. Build $G(V(G), E(G))$ as follows: a clique K_{k+10} with $k+10$ vertices, 9 vertices $x_1, \dots, x_9$, a Moon-Moser graph M on k vertices [21], and another clique B of constant size at the end.

Next, we connect the vertices $x_1, \dots, x_9$ to M: M is made of triplets which are independent sets but fully connected to all other vertices; let a_1, a_2, a_3 and b_1, b_2, b_3 be the final two triplets of M. We connect each of the 9 x_i vertices to the vertices of one of the 9 distinct pairs in $\{a_1, a_2, a_3\} \times \{b_1, b_2, b_3\}$. Furthermore, we connect all x_i to all vertices of the other triplets of M.

A schematic graphical representation is given in Fig. 3.

Observe that the total number of vertices in G is $n = 2k + O(1)$. Furthermore, all maximal cliques of M can be built by picking precisely one vertex from each triplet, and thus they can always be extended by a suitable x_i.

Finally, observe the degrees in G: K_{k+10} vertices have degree $k+9$, each x_i has degree $k-6+2 = k-4$, vertices in M have degree at least $k-3+3 = k$, and vertices in B have constant degree, say 2.

Consider now CLIQUES on G: in the root recursive call, $CAND = V(G)$ thus the pivot is chosen as one of the vertices from K_{k+10}, which have the highest degree. Recursion on all other vertices of K_{k+10} is prevented by the pivot rule, but all x_i vertices and vertices of M are processed (by processing we mean they are considered in the **foreach** loop). We assume they are processed in the order in which they were described, *i.e.*, as in Fig. 3. All maximal cliques involving x_i are found while processing x_i vertices, which include all cliques involving vertices of M. Once the algorithm backtracks and starts processing vertices of M, it will not find new solutions until it reaches B. It is crucial now to observe that x_i vertices will never be chosen as pivots: they are adjacent to only $k-4$ vertices of M, while each vertex of M is adjacent to $k-3$ other vertices of M, making them preferable as a pivot. This difference is preserved whenever a vertex of M is added to Q as a full triplet will disappear from $CAND$ (and the two final triplets are only considered last).

We obtain that CLIQUES runs on M, a Moon-Moser graph, without using any vertex outside M as a pivot. By [31] we know it will take $\Omega(3^{k/3}) = \Omega(3^{n/6})$, however no new solution is found, because all cliques in M can be extended with some x_i. Finally the algorithm processes vertices of B and outputs B, giving us a delay of $\Omega(3^{n/6})$-time.

The same bound holds for the BKP algorithm: as it does not specify any pivoting strategy, that of CLIQUES is a valid one. Of course, it holds also for BK, as its recursion tree is that of BKP plus additional nodes producing no output. We obtain the following:

Theorem 5.1. *The* CLIQUES, BKP, *and* BK *algorithms have* $\Omega(3^{n/6})$*-time wost-case delay.* □

Delay of Eppstein et al.'s algorithm. It is worth observing how the algorithm by Eppstein et al. [11], based on CLIQUES plus a degeneracy ordering and running in $O(n3^{d/3})$ time, was proven to have exponential delay by [9]. Note that the strategy did not apply to CLIQUES as it exploits mechanics that are present only in [11], and the delay of CLIQUES is only claimed to be $\Omega(n^3)$ in the paper.

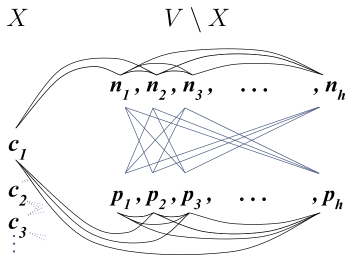

Fig. 4. Example construction of G from the $\mathcal{F}$ formula for a clause $c_1 = (v_1, \neg v_3, \neg v_h)$. Note how c_1 is connected to *all* literals that do not satisfy it, including, e.g., p_2 and n_2. The set X is composed of all c_i vertices.

6. No pivoting strategy for CLIQUES and Bron-Kerbosch has polynomial delay unless P=NP

Known pivoting strategies can have a large impact on the number of recursive nodes, this can be observed e.g., in [4] and our experiments in Section 7.

It is natural to ask: is there an ideal pivoting strategy with polynomial delay? That is, is there a strategy that guarantees a new clique —or the end of the algorithm— is always reached within polynomial time? We complete our results by showing that no such strategy can exist unless $P = NP$.[3]

To do so, we define the *extension problem* for maximal cliques, showing that it is NP-complete. Then, we show how a pivoting strategy for Algorithm 1 that guarantees polynomial delay could be used to solve this problem in polynomial time.

Hardness of the extension problem.

Problem 1 *(Extension Problem, $EXT\text{-}P(G(V,E),X)$).* Given a graph $G(V,E)$ and $X \subset V$, does G have a maximal clique Q that does not intersect X?

Looking at a recursive node of CLIQUES, with its sets Q, *CAND*, and *SUBG*, we can observe how a maximal clique will be output in its recursive subtree iff there exists a maximal clique in $G(SUBG)$ that does not intersect $SUBG \setminus CAND$. In other words, the problem answers the question “will a maximal clique be output in this recursive subtree?”.

This problem is, however, NP-complete, as we show by a reduction from CNFSAT.

Theorem 6.1. *The extension problem for maximal cliques is NP-complete.*

Proof. Let $\mathcal{F}$ be a CNF Boolean formula on h variables $v_1, \ldots, v_h$ and l clauses $c_1, \ldots, c_l$, and let the positive and negative literals of the variable v_i be represented by v_i and $\neg v_i$.

We build a graph G, with a suitable vertex set X, such that $\mathcal{F}$ can be satisfied iff G has a maximal clique not intersecting X. Let $V(G)$ and $E(G)$ be as follows:

- $V(G)$ contains a vertex p_i for each positive literal v_i in $\mathcal{F}$.
- $V(G)$ contains a vertex n_i for each negative literal $\neg v_i$ in $\mathcal{F}$.
- $V(G)$ contains a vertex c_i for each clause of $\mathcal{F}$.
- $E(G)$ contains, for all distinct i and j, (p_i, p_j), (p_i, n_j), (n_i, p_j), and (n_i, n_j), *i.e.*, all literals are connected to all others (positive and negative) except their own negation.
- $E(G)$ contains (c_i, p_i) if literal v_i does *not* appear in c_i. Similarly, $(c_i, n_i) \in E(G)$ if $\neg v_i$ does *not* appear in c_i, *i.e.*, clauses are connected to all literals that do not satisfy them.
- Finally, let X be the set of all vertices c_i corresponding to clauses. Note that $V \setminus X$ is the set of all vertices corresponding to literals.

An example is shown in Fig. 4.

Observe how a clique cannot contain both p_i and n_i, and any set $S \subseteq (V \setminus X)$ is a clique iff it does *not* contain both a literal and its negation, since all literals are adjacent to all others except their negation: thus any clique in $V \setminus X$ corresponds to valid truth assignments of the variables of $\mathcal{F}$. We can now prove that a maximal clique $S \subseteq (V \setminus X)$ exists iff $\mathcal{F}$ can be satisfied.

Firstly, if S is a maximal clique, then each c_i is *not* adjacent to some vertex $v \in S$: from the construction of the graph, this means c_i is satisfied by the literal corresponding to v, meaning the literals in S satisfy all clauses in $\mathcal{F}$. It follows that if $S \subseteq (V \setminus X)$ is a maximal clique then the set of literals it contains is a satisfying assignment of $\mathcal{F}$.

[3] Polynomial delay might be achieved by other means, what we prove here is that CLIQUES and BKP cannot guarantee polynomial delay even changing the pivot selection strategy, unless $P = NP$.

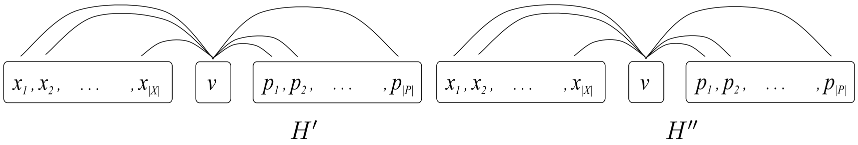

Fig. 5. Graph G^*, obtained as two copies of the graph H.

To prove the converse, assume $\mathcal{F}$ can be satisfied. Let S be the set of vertices corresponding to a truth assignment of $\mathcal{F}$ (n_i for negative and p_i for positive literals). Observe that $S \subseteq (V \setminus X)$ and that S is a clique. For each pair p_j, n_j, exactly one of them is in S, so the other cannot be added to the clique S since (by construction of G) p_j and n_j are not neighbors. Furthermore, each clause c_i must be satisfied by some literal (n_i or p_i) contained in S; from how G is built, we have that c_i is *not* adjacent to that literal, and so c_i cannot be added to S, meaning S is maximal in G. So, if $\mathcal{F}$ can be satisfied, then there exists a maximal clique $S \subseteq (V \subseteq X)$.

It follows that $EXT\text{-}P(G(V,E),X)$ has a positive answer iff $\mathcal{F}$ can be satisfied. As $EXT\text{-}P(G(V,E),X)$ is in NP, because we can test a solution by verifying the maximality of a clique, the proof is complete. □

A polynomial delay pivoting strategy implies P=NP.

Now, given $G(V,E)$ and X, we build a graph $G^*(V^*,E^*)$ such that, if Algorithm 1 runs on $G^*(V^*,E^*)$ with polynomial delay, then we can solve the NP-complete problem $EXT\text{-}P(G(V,E),X)$ in polynomial time.

$G^*(V^*,E^*)$ consists of two identical disjoint copies of a graph $H(V(H),E(H))$, built as follows. $V(H) = X \cup \{v\} \cup P$, where:

- $X = \{x_1, \ldots, x_{|X|}\}$ is the set of vertices of X from $EXT\text{-}P(G(V,E),X)$.
- $P = \{p_1, \ldots, p_{|P|}\}$ is the set of vertices of $V \setminus X$ from $EXT\text{-}P(G(V,E),X)$.

$E(H)$ is obtained by connecting all vertices from $G(V,E)$ as they are connected in G, and the vertex v to all of P and X. Fig. 5 shows a graphical example of the two copies of H.

Let the two copies of H be H' and H''. If we run Algorithm 1 on G^*, it will first choose a pivot vertex from either H' or H'': assume wlog it is H'' (the other case is identical); this means that no vertex of H' is adjacent to the pivot, so at least every vertex of H' is processed by Algorithm 1. By processing we mean it is considered in the **foreach** loop of the root recursive call of the algorithm.

As the algorithm does *not* specify in which order the vertices are processed, we will assume this is the order they appear in Fig. 5, *i.e.*, first $x_1, \ldots, x_{|X|}$, then v, then $p_1, \ldots, p_{|P|}$ (vertices of H'' are not relevant and can be disregarded). Now, take the moment when v is processed: We have that $CAND \cap \Gamma(v)$ is exactly P, and $(SUBG \setminus CAND) \cap \Gamma(v) = FINI \cap \Gamma(v)$ is X as we processed all vertices of X.

If Algorithm 1 —with any arbitrary pivoting strategy— has polynomial delay, it must either find a new maximal clique or terminate, in polynomial time: as any maximal clique containing vertices of X has already been found, this process will output a new maximal clique iff there is a maximal clique in P that cannot be extended with vertices of X, *i.e.*, since P corresponds to $V \setminus X$, there is a maximal clique in $G(V,E)$ that does not intersect X. Finally, since Algorithm 1 may spend exponential time before processing v, we want to skip this time: we do so by simply running the algorithm with $Q = \{v\}$, $CAND = P$ and $SUBG = X \cup P$.

We can thus conclude that the delay of an algorithm in this class cannot be polynomial, unless $P = NP$.

Furthermore, considering the reduction from SAT shown in Theorem 6.1, we can link the worst-case delay (*i.e.*, at least the time required to solve the extension problem) on a graph with n vertices, to the time required to solve a SAT problem of size $\Theta(n)$. This allows us to give a more formal bound on the best possible worst-case delay using the Exponential Time Hypothesis [15]. More formally:

Theorem 6.2. *No pivoting strategy for the* CLIQUES *[31] and* BKP *[3] can guarantee polynomial delay unless $P = NP$. Furthermore, if the Exponential Time Hypothesis is true, the best possible delay obtainable is $\Omega(2^{n/c})$ time for some constant c.*

7. Experimental results

While we were able to analyze the worst-case delay of CLIQUES and Bron-Kerbosch, we could not find suitable techniques to analyze its worst-case amortized cost *per solution*, which remains a compelling open problem. To give a complete picture, we attempted to analyze their amortized cost and delay *in practice*: we present an experimental evaluation of CLIQUES and Bron-Kerbosch variants on real-world networks, showing how their behavior *appears* to be output-sensitive and to have small delay on real-world networks.

Aiming to get substantial experimental evidence we ran our experiments on 138 real-world and synthetic graphs taken from the SNAP [18] and LASAGNE [17] repositories, with up to 3 million edges. We report only a subset in Table 1, while the complete list is in the Appendix (Table B.4).

Table 1
Excerpt of the graphs used in our experiments, with number of vertices (n), edges (m), maximum degree (Δ), degeneracy (d) and the number of maximal cliques (#cliques).

GRAPH	n	m	Δ	d	q	#cliques
GoogleNw	15 763	148 585	11 401	102	66	75 258
Meth	956	1 157	31	3	3	1 046
add32	4 960	9 462	31	3	4	4 519
amazon0601	403 394	2 443 408	2 752	10	11	1 023 572
auto	448 695	3 314 611	37	9	7	2 164 046
bcsstk30	28 924	1 007 284	218	58	48	6 706
brack2	62 631	366 559	32	7	5	282 557
ca-AstroPh	18 771	198 050	504	56	57	36 427
ca-HepPh	12 006	118 489	491	238	239	14 937
darwinBookInter	7 381	45 229	2 686	306	16	127 055
fe_ocean	143 437	409 593	6	4	2	409 593
forest1e4_2	10 000	153 925	1 124	101	29	96 861 484
interdom	1 706	78 983	728	129	123	3 351
Slashdot090221	82 140	500 480	2 548	54	27	854 407
soc-sign-epinions	131 827	711 209	3 558	121	94	22 226 172
spanishBookInter	11 586	44 214	3 327	342	14	66 505
ud_1e4	10 000	313 726	523	285	258	132 557
yeast_bo	1 846	2 203	56	5	6	1 940

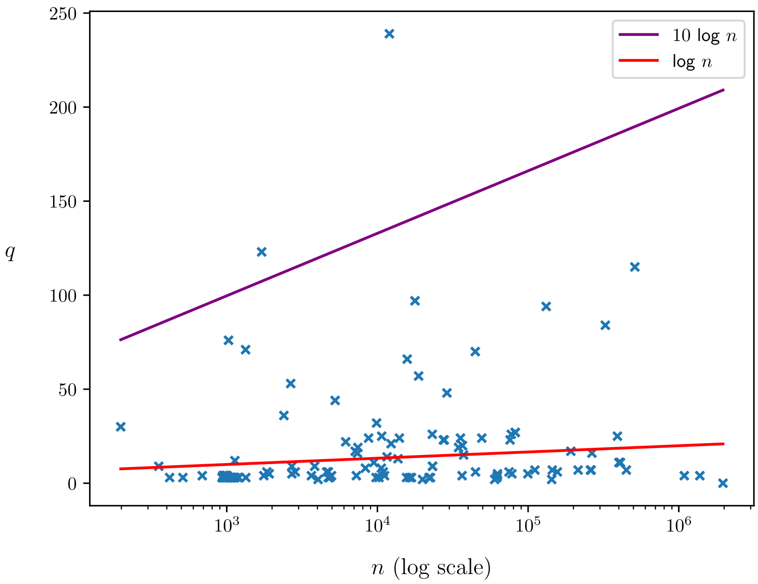

Fig. 6. Maximum clique size q against number of vertices n in 128 real-world graphs, compared to $\log_2 n$ (lower red line) and $10 \log_2 n$ (higher purple line)

7.1. Maximum clique size in real-world networks

Firstly, to gauge the scope of Corollary 4.3, we computed the maximum clique size q of real-world networks in our dataset (128 out of 138). We report this in Fig. 6 against $\log n$: indeed q is below $\log n$ on the majority of networks, and below $10 \log n$ in almost all cases. The most significant of the two outliers is the co-authorship network ca-HepPh concerning *High Energy Physics* papers on ArXiv, with 12 006 vertices and a maximum clique of size 239; this is perhaps not surprising as the graph is generated by taking each paper makes a clique out of its co-authors, and hyperauthorship is not uncommon in the Physics literature (*i.e.*, papers can have hundreds of authors). The second point above the line is the protein-protein interaction network interdom hosted at [17] and originally from http://ppi.fli-leibniz.de/, with 1 706 vertices and a maximum clique of size 123: here groups of proteins known as *protein complexes* that are involved in reactions are naturally modeled as cliques, although some studies [12] suggest that the use of hypergraphs and hyperedges instead can be more appropriate. Nonetheless, the value does not deviate dramatically from our tentative "$10 \log n$" line, as this corresponds to 107 for $n = 1\,706$.

7.2. Experimental setup

We consider the following algorithms.

- CLIQUES: the algorithm in [31], described in Algorithm 1 (with pivot u chosen as the vertex in *SUBG* maximizing $|CAND \cap \Gamma(u)|$).
- BKP_M: BKP [3] with pivot u chosen as the highest-degree vertex in *SUBG*.
- BKP_R: BKP Randomized, *i.e.*, with pivot u chosen randomly in *SUBG*.
- BK: Bron-Kerbosch, without pivoting.

Other efficient algorithms exist, but their inclusion is not meaningful, as we aim to judge pruning effectiveness of pivoting strategies and not the running time. Notable examples are the algorithm by Eppstein et al. [11], effective on sparse graphs, that uses CLIQUES as a subroutine, the algorithm by Naude [22] that provides an alternative pivot-selection strategy that preserves the worst-case optimal behavior of CLIQUES, and the algorithms by San Segundo et al. [24] which aim to quickly find a good (but not optimal) pivot candidate according to the metric of CLIQUES.

Metrics. We are not strictly interested in the running time, as a recursive node has polynomial cost. As solutions are output in leaves, we are interested in what portion of the leaves outputs a solution: the total running time is $O(poly(n))$ times the number of leaves, so the ratio $\frac{\text{CLIQUES}}{\text{LEAVES}}$ (number of maximal cliques divided by the number of leaves of the recursion tree) gives an idea of "how output-sensitive" the execution is. We also show the *delay* in terms of nodes and of just leaves, *i.e.*, the longest sequence of nodes/leaves between two consecutive outputs. For completeness, we also report the total time and delay in milliseconds.

7.3. Results

For each algorithm, on each graph, we computed the number of nodes and leaves in the recursion tree, and the metrics discussed above. Due to the large number of experiments (and the tendency of BK to time out even on small graphs) to provide a fair comparison, we set a 30 minutes time limit on all reported executions.

To allow easier reading, we include our analysis on the results obtained, as well as an excerpt of the raw data in Table 2. The complete running times data is left for completeness in Appendix (Tables B.5-B.7).

Nodes and leaves generated. We first observed how CLIQUES, thanks to its pivoting strategy, is more effective in pruning than BKP_M and BKP_R: CLIQUES often produces less half the recursive nodes of the next best algorithm, and sometimes orders of magnitude less (e.g., bccstk30, ca-AstroPh, ca-HepPh).

Same goes for the delay, that is, the highest number of consecutive recursive nodes (resp. leaves) that do *not* output a solution, which are encountered before a solution is output: CLIQUES has typically lower delay, both in terms of time (see DELAY, column *ms*) and in terms of nodes and leaves. On the other hand, BK produces a far larger amount of recursive nodes, and frequently times out.

The most relevant value to observe is the $\frac{\text{CLIQUES}}{\text{LEAVES}}$ ratio: a high value shows that the algorithm is performing in an output-sensitive way. Again we observed how CLIQUES consistently has the highest ratio, sometimes by an order of magnitude (e.g., bcsstk30, Slashdot090221, soc-sign-epinions). For completeness, it is worth observing that the $\frac{\text{CLIQUES}}{\text{LEAVES}}$ ratios of BKP_M and BKP_R, while worse than CLIQUES, are still often high. Furthermore, $\frac{\text{CLIQUES}}{\text{LEAVES}}$ for CLIQUES, BKP_M and BKP_R is seemingly independent of the size of the graph, and in most cases even close to 1. This supports the idea that $\frac{\text{CLIQUES}}{\text{LEAVES}}$ is in practice $\Omega(1/poly(n))$ for CLIQUES, BKP_M and BKP_R, and that the algorithms behave in an output-sensitive way in practice.

Running time. While a running time comparison is not the goal of this paper (and the implementations are not optimized for this purpose), it is worth observing that the following:

CLIQUES seems to perform best on graphs with highest degeneracy, denser and with more solutions; in some cases it is the only one to terminate (e.g., forest1e4_2, soc-sign-epinions, ud_1e4).

When BKP_M and BKP_R terminate, in some cases CLIQUES is still significantly faster (e.g., ca-HepPh, interdom, bcsstk30). In others, BKP_M is competitive (e.g., GoogleNW, spanishBookInter) or faster (e.g., darwinBookInter) despite generating more recursive nodes, probably due to CLIQUES having an expensive pivot computation.

On small graphs, with low degeneracy, few maximal cliques (e.g., Meth, add32, brack2, fe_ocean, yeast_bo) the differences flatten out, and performance become comparable.

For a comparison of running time of several algorithms we also refer the reader to [31,11].

8. Concluding remarks

We presented a study of the CLIQUES and Bron-Kerbosch algorithms, showing how their delay is exponential in the worst case, unless $P = NP$, settling a question unsolved for a long time. Furthermore, we have shown that the claim remains true for any pivoting strategy that can be computed in polynomial time. On the other hand, we proved that their time complexity is amortized polynomial on graphs whose largest clique has logarithmic size; we showed this condition can hold in both sparse and dense graphs, and observed experimentally that it is generally true in real-world graphs. Our experiments further support this claim as both algorithms perform well in practice on over a hundred real-world graphs. This result, summarized

Table 2
Statistics and running times. The best cliques/leaves ratio is highlighted in bold, and out of time (OOT) entries report statistics at termination time (30 minutes).

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY			TIME
					ms	nodes	leaves	
GoogleNw	CLIQUES	144 576	95 203	**0.791**	598	113	112	8 143
	BKP$_M$	396 672	163 881	0.459	451	5 857	3 097	7 712
	BKP$_R$	1 657 297	547 317	0.138	2 441	111 825	36 521	30 773
	BK	⩾1.1B	⩾599M	0.000	⩾1.7M	⩾1.1B	⩾589M	OOT
Meth	CLIQUES	2 043	1 362	**0.768**	6	26	25	60
	BKP$_M$	2 107	1 404	0.745	1	26	25	21
	BKP$_R$	2 147	1 431	0.731	1	30	29	24
	BK	2 193	1 476	0.709	1	30	29	23
add32	CLIQUES	14 259	9 857	**0.458**	18	32	31	652
	BKP$_M$	17 604	10 858	0.416	8	51	32	658
	BKP$_R$	18 456	11 618	0.389	30	51	32	610
	BK	19 916	13 094	0.345	5	51	32	570
amazon0601	CLIQUES	1 991 135	1 146 403	**0.692**	1 581	35	30	OOT
	BKP$_M$	4 054 110	2 105 612	0.373	2 801	936	493	OOT
	BKP$_R$	5 508 554	2 737 409	0.291	2 764	1 568	867	OOT
	BK	11 635 442	7 150 606	0.110	2 674	3 468	1 825	OOT
auto	CLIQUES	2 861 662	1 388 920	**0.793**	1 614	45	25	OOT
	BKP$_M$	4 094 876	1 862 742	0.586	305	133	63	OOT
	BKP$_R$	4 264 049	1 928 275	0.571	328	153	77	OOT
	BK	6 707 309	3 803 765	0.288	296	249	130	OOT
bcsstk30	CLIQUES	166 208	66 151	**0.101**	194	312	255	90 409
	BKP$_M$	10 109 420	657 434	0.010	358	95 746	7 730	107 152
	BKP$_R$	9 986 334	625 237	0.011	337	62 921	4 822	107 202
	BK	⩾1.6B	⩾815M	0.000	⩾1.7M	⩾1.6B	⩾815M	OOT
brack2	CLIQUES	670 843	332 531	**0.850**	339	367	367	88 669
	BKP$_M$	852 702	385 281	0.733	340	599	368	85 287
	BKP$_R$	875 163	389 976	0.725	354	605	368	82 944
	BK	1 299 629	699 955	0.404	346	673	382	83 387
ca-AstroPh	CLIQUES	199 293	99 233	**0.367**	143	148	134	14 901
	BKP$_M$	2 129 289	340 307	0.107	62	15 200	1 452	17 076
	BKP$_R$	3 977 870	570 566	0.064	120	30 216	3 887	23 977
	BK	>890M	>445M	0.000	>1.7M	>821M	>410M	OOT
ca-HepPh	CLIQUES	69 214	32 360	**0.462**	298	217	80	6 791
	BKP$_M$	1 056 161	90 193	0.166	140	15 079	884	11 250
	BKP$_R$	4 851 452	308 094	0.048	6 594	209 519	5 162	83 473
	BK	>634M	>317M	0.000	>17̇M	>533M	>266M	OOT
darwin BookInter	CLIQUES	258 587	124 786	**0.936**	39	13	6	2 267
	BKP$_M$	401 087	169 954	0.687	84	183	70	1 606
	BKP$_R$	6 401 693	2 763 439	0.042	70	15 920	6 533	18 404
	BK	18 404 054	12 090 465	0.010	69	43 873	27 491	38 601
fe_ocean	CLIQUES	553 025	410 454	**0.998**	230	5	4	269 666
	BKP$_M$	553 025	410 454	**0.998**	48	5	4	265 257
	BKP$_R$	553 025	410 454	**0.998**	43	5	4	263 698
	BK	553 031	410 454	**0.998**	40	5	4	263 060
forest 1e4_2	CLIQUES	238 453 305	100 171 630	**0.967**	204	41	15	1 128 622
	BKP$_M$	855 014 958	239 479 827	0.349	1 395	657 501	216 935	OOT
	BKP$_R$	433 214 072	79 685 156	0.080	6 616	1 632 740	309 377	OOT
	BK	322 216 986	161 296 155	0.000	12 128	1 829 400	919 223	OOT
interdom	CLIQUES	46 895	10 717	**0.313**	116	137	88	5 055
	BKP$_M$	5 879 885	185 950	0.018	5 007	395 562	11 187	68 271
	BKP$_R$	⩾114M	⩾2.6M	0.000	⩾217K	⩾14M	⩾360K	OOT
	BK	⩾408M	⩾204M	0.000	⩾1M	⩾408M	⩾204M	OOT
Slashdot 090221	CLIQUES	1 871 452	1 035 183	**0.825**	414	70	68	141 175
	BKP$_M$	55 634 373	15 064 515	0.057	412	190 787	56 385	226 360
	BKP$_R$	113 266 541	22 538 850	0.038	815	449 600	90 981	342 089
	BK	>1.2B	>653M	0.000	>55K	>39M	>20M	OOT

(*continued on next page*)

Table 2 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES / LEAVES	DELAY ms	DELAY nodes	DELAY leaves	TIME
soc-sign-epinions	CLIQUES	61 803 460	24 265 229	**0.916**	679	123	122	820 659
	BKP_M	495 252 038	107 973 282	0.129	15 157	2 756 897	149 240	OOT
	BKP_R	411 796 194	93 373 913	0.042	914	219 326	45 228	OOT
	BK	370 451 642	198 274 042	0.001	10 803	2 541 280	>1.2M	OOT
spanish BookInter	CLIQUES	117 784	61 773	**0.938**	107	10	7	2 710
	BKP_M	157 981	75 347	0.769	304	286	121	2 545
	BKP_R	1 722 091	761 066	0.076	253	4 879	2 427	9 476
	BK	4 159 948	2 659 625	0.022	170	9 969	6 319	13 344
ud_1e4	CLIQUES	1 137 453	203 836	**0.650**	1 037	687	444	148 183
	BKP_M	243 952 505	8 356 294	0.005	101 806	12M	490 665	OOT
	BKP_R	77 609 172	872 451	0.007	205 614	8 848 576	111 558	OOT
	BK	>1.7B	>874M	0.000	>344K	>333M	>166M	OOT
yeast_bo	CLIQUES	3 876	2 952	**0.657**	8	35	34	120
	BKP_M	4 051	3 054	0.635	14	35	34	72
	BKP_R	4 155	3 130	0.620	1	35	34	51
	BK	4 322	3 280	0.591	1	61	39	74

Table 3
Summary of the complexity of CLIQUES, BK and variants on a graph with n vertices, α maximal cliques, and largest clique size q. †: unless the Exponential Time Hypothesis is false, where c is an unspecified constant.

Algorithm	Delay	Overall time if $q = O(\log n)$	Overall time	Amortized time
BK	$\Omega(2^n n)$	$O(\alpha \cdot poly(n))$	$\Omega(2^n)$	$\Omega(2^n)$
BKP	$\Omega(3^{n/6})$	$O(\alpha \cdot poly(n))$	$\Omega(3^{n/3})$	?
CLIQUES	$\Omega(3^{n/6})$	$O(\alpha \cdot poly(n))$	$O(3^{n/3})$	?
Best possible pivoting	$\Omega(2^{\frac{n}{c}})^\dagger$	$O(\alpha \cdot poly(n))$	$O(3^{n/3})$	?

in Table 3, partially fills the long-standing gap between the theoretical worst-case exponential time complexity of CLIQUES and its practical efficiency.

The worst-case amortized cost per solution of CLIQUES and Bron-Kerbosch, and the worst-case time of Bron-Kerbosch remain open.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported in part by JSPS KAKENHI, Grant JP17K00006 and MIUR, Grant 20174LF3T8 AHeAD. The authors would like to thank L. Versari and E. Harley and G. Manoussakis who joined in useful discussions, as well as the anonymous reviewers.

Prof. Etsuji Tomita passed away during the late stages of the revision process. Editor Prof. Ryuhei Uehara and co-author Alessio Conte concurred to proceed with the publication of the manuscript, which was already in its final form. Co-author Alessio Conte thanks Prof. Uehara for his assistance with this choice, and wishes to dedicate the paper to Prof. Tomita, who he is grateful to have had the chance to work with.

Appendix A. A realistic model of dense graphs with small cliques

In Section 4 we proved the complexity of CLIQUES to be amortized polynomial in graphs with maximum clique size $O(\log n)$. We remarked how this is true for sparse graphs, and even some dense graphs such as the complete bipartite graph. In Fig. A.7 we wish to present a simple model showing that locally-clustered graphs can be dense and still satisfy the requirements of Corollary 4.3.

Appendix B. Complete experimental data

We report for the sake of completeness the statistics of the full dataset considered in our experiments, and the full result of the experimental evaluation.

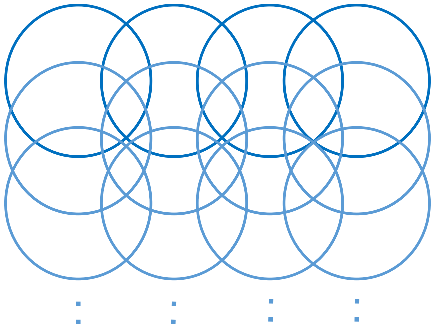

Fig. A.7. A diagram representing a graph, where each circle represents a clique with $O(\log n)$ vertices. The graph can be seen as a collection of tightly linked communities, however cliques in the graph maintain size $O(\log n)$.

Table B.4
Complete set of graphs used in our experiments, with number of vertices (n), edges (m), maximum degree (Δ), degeneracy (d) and number of maximal cliques (#cliques).

GRAPH	n	m	Δ	d	#cliques
144	144 649	1 074 393	26	9	644 735
3elt	4 720	13 722	9	4	9 000
4elt	15 606	45 878	10	4	30 269
598a	110 971	741 934	26	8	515 872
Amazon0505	410 236	2 439 436	2 760	10	1 034 135
Brady	1 116	1 330	28	4	1 210
Brady2	1 124	1 321	37	4	1 227
Burk	1 028	1 228	33	3	1 133
Caenorhabditis_el.	4 723	9 842	190	8	9 055
Chla2	1 202	1 413	27	4	1 291
Cupri	1 060	1 270	35	4	1 152
Drosophila_mel.	10 625	40 781	244	14	37 665
Erw	969	1 224	32	3	1 089
Esche2	943	1 314	43	4	1 168
Esche3	997	1 331	40	3	1 180
Gnp_1e3	1 000	3 854	17	5	3 695
Gnp_1e4	10 000	59 849	27	8	59 326
Gnp_2e3	2 000	8 994	20	6	8 800
Gnp_5e3	5 000	24 809	22	7	24 475
GoogleNw	15 763	148 585	11 401	102	75 258
HC-BIOGRID	4 039	10 321	45	14	10 321
Homo	1 027	1 166	25	3	1 056
Homo_sapiens	13 690	61 130	443	17	49 308
Mes	1 116	1 348	36	4	1 223
Meta	3 648	5 049	84	4	4 405
Meth	956	1 157	31	3	1 046
Meth2	952	1 155	29	3	1 043
Meth3	930	1 142	31	3	1 032
Meth4	936	1 153	31	3	1 043
Meth5	1 001	1 208	33	3	1 097
Meth6	1 051	1 278	30	3	1 147
Mus	1 187	1 378	24	3	1 266
Mus_musculus	4 610	5 747	183	5	5 301
Myc	1 340	1 513	34	3	1 386
Newman-Cond_m.	22 015	58 578	118	8	58 451
PGPgiantcompo	10 680	24 316	205	31	13 814
Plant	1 762	2 198	41	3	2 008
Pseudo2	977	1 206	32	4	1 085
Pseudo4	1 082	1 307	28	3	1 163
Ral	1 077	1 276	33	3	1 174
Rattus_norvegicus	1 914	2 110	75	5	1 967
Rhizo	1 071	1 323	36	3	1 176
Rhizo2	1 138	1 345	36	3	1 219
Rhodo	957	1 183	29	4	1 063
Salmo	1 006	1 323	33	4	1 168
Shigi	982	1 299	38	3	1 150
Sino	986	1 187	31	3	1 064
Yer2	956	1 147	26	3	1 035
add20	2 395	7 462	123	35	2 314
add32	4 960	9 462	31	3	4 519
advogato	7 418	42 892	821	28	48 857
alr20-MathSciNet	391 529	873 775	496	24	416 213
amazon0302	262 111	899 791	420	6	403 363
amazon0312	400 727	2 349 868	2 747	10	1 007 757
amazon0601	403 394	2 443 408	2 752	10	1 023 572
auto	448 695	3 314 611	37	9	2 164 046
bcsstk30	28 924	1 007 284	218	58	6 706
brack2	62 631	366 559	32	7	282 557
ca-AstroPh	18 771	198 050	504	56	36 427
ca-CondMat	23 133	93 439	279	25	18 502
ca-GrQc	5 241	14 484	81	43	3 905
ca-HepPh	12 006	118 489	491	238	14 937
celegans_metabol	354	1 501	186	10	493
cit-HepPh	34 546	420 876	846	30	412 491
cit-HepTh	27 770	352 284	2 468	37	464 873
citeseer	259 217	532 040	1 151	9	433 194
cnr_2000	325 557	2 738 969	18 236	83	1 425 378
coli1_1Inter	418	519	72	3	459
cora	2 708	5 278	168	4	3 563
crack	10 240	30 380	9	4	20 141
cs4	22 499	43 858	4	3	38 944
cti	16 840	48 232	6	4	47 508
darwinBookInter	7 381	45 229	2 686	306	127 055
data	2 851	15 093	17	7	11 928
dip20090126_MAX	19 928	41 202	145	8	41 202
eatRS	23 219	304 937	1 090	34	298 164
eatSR	23 218	304 934	1 090	34	298 162
email-Enron	36 691	183 830	1 383	43	226 858
email-EuAll	265 214	364 480	7 636	37	377 955
email	1 133	5 451	71	11	3 267
eva	7 253	6 723	552	3	6 611
fe_4elt2	11 143	32 818	12	4	21 655
fe_body	44 775	163 734	28	6	49 550
fe_ocean	143 437	409 593	6	4	409 593
fe_pwt	36 463	144 794	15	5	36 842
fe_rotor	99 617	662 431	125	8	554 479
fe_sphere	16 386	49 152	6	5	32 768
fe_tooth	78 136	452 591	39	7	346 120
finan512	74 752	261 120	54	6	68 608
forest1e4	10 000	49 354	572	36	73 538
forest1e4_2	10 000	153 925	1 124	101	96 861 484
forest5e4	50 000	243 441	1 900	40	242 815
forest5e4_2	50 000	1 095 697	4 953	228	>94M
frenchBookInter	8 325	23 841	1 891	17	21 027
geom	6 158	11 898	102	21	4 632
hep-th-cit_MAX	27 400	352 021	2 468	37	464 666
hprd_pp	9 465	37 039	270	14	29 404
iPfam	1 334	12 002	144	70	395
interdom	1 706	78 983	728	129	3 351
itdk0304_rlinks	192 244	609 066	1 071	32	482 045
japaneseBookInter	2 704	8 102	771	76	7 009
jazz	198	2 742	100	29	746
kron14	8 156	24 493	3 296	6	22 414
kron16	30 429	65 526	402	6	64 441
m14b	214 765	1 679 018	40	9	882 092
memplus	17 758	54 196	573	96	16 531
p2p-Gnutella31	62 586	147 891	95	6	144 481
ppi_dip_swiss	3 834	11 958	227	9	8 880
ppi_gcc	37 333	135 618	968	25	121 029
psimap	1 028	11 615	146	75	276
roadNet-PA	1 088 092	1 541 898	9	3	1 413 391
roadNet-TX	1 379 917	1 921 660	12	3	1 763 318
s838	512	819	22	2	747
Epinions1	75 879	405 739	3 044	67	1 772 879
Slashdot0811	77 360	469 180	2 539	54	823 415
Slashdot0902	82 168	504 230	2 552	55	890 041
Slashdot090221	82 140	500 480	2 548	54	854 407
sign-epinions	131 827	711 209	3 558	121	22 226 172
spanishBookInter	11 586	44 214	3 327	342	66 505
string	2 658	26 805	134	56	18 566
t60k	60 005	89 440	3	2	89 440
trust	49 288	381 036	2 598	71	4 717 941
ud_1e3	1 000	16 727	138	77	2 506
ud_1e4	10 000	313 726	523	285	132 557
ud_2e3	1 999	35 697	188	108	6 203
ud_5e3	4 998	97 027	285	165	18 946
uk	4 824	6 837	3	2	6 835
us_1e3	1 000	14 334	47	23	2 264
us_2e3	2 000	37 928	68	30	5 829
us_5e3	5 000	135 833	87	35	23 939
vibrobox	12 328	165 250	120	26	21 780
wave	156 317	1 059 331	44	8	840 081
whitaker3	9 800	28 989	8	4	19 190
wiki-Vote	7 115	100 761	1 065	53	458 988
wing	62 032	121 544	4	3	108 174
wing_nodal	10 937	75 488	28	8	52 119
yeastInter	688	1 078	71	3	991
yeast_bo	1 846	2 203	56	5	1 940

Table B.5
Performance of the clique enumeration algorithms considered on each graph. Times are in milliseconds. OOT: interrupted after 30 minutes limit (statistics at the time of interruption are reported). Continued in the next tables.

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY			TIME
					ms	nodes	leaves	
144	CLIQUES	1 850 537	898 381	0.718	25 853	7 137	7 066	572 685
	BKP_M	3 021 877	1 341 216	0.481	24 200	9 785	7 436	575 883
	BKP_R	3 015 876	1 326 267	0.486	23 930	9 793	7 445	575 210
	BK	5 008 413	2 807 609	0.230	23 827	9 928	7 578	566 055
3elt	CLIQUES	18 771	9 149	0.984	19	30	30	670
	BKP_M	22 606	9 185	0.980	5	36	30	701
	BKP_R	21 423	9 181	0.980	5	36	30	675
	BK	27 443	13 779	0.653	4	37	31	777
4elt	CLIQUES	62 832	31 058	0.975	30	99	97	5 034
	BKP_M	75 699	31 339	0.966	50	115	98	4 896
	BKP_R	71 946	31 326	0.966	28	115	98	4 744
	BK	91 754	46 706	0.648	25	116	99	4 547
598a	CLIQUES	1 336 649	668 023	0.772	20 655	8 396	8 123	302 273
	BKP_M	1 907 808	914 839	0.564	20 684	13 488	9 195	292 730
	BKP_R	1 957 923	934 408	0.552	15 840	10 944	7 236	296 297
	BK	2 917 407	1 664 177	0.310	28 538	18 063	12 908	294 071
Amazon0505	CLIQUES	1 920 786	1 100 655	0.699	1 392	24	18	OOT
	BKP_M	4 050 853	2 088 704	0.375	1 090	491	240	OOT
	BKP_R	5 302 692	2 624 145	0.296	1 117	758	355	OOT
	BK	11 867 344	7 245 230	0.108	1 103	3 969	2 234	OOT
Brady	CLIQUES	2 375	1 610	0.752	7	21	20	85
	BKP_M	2 447	1 646	0.735	1	21	20	32
	BKP_R	2 483	1 671	0.724	1	52	50	32
	BK	2 533	1 719	0.704	12	52	50	40
Brady2	CLIQUES	2 375	1 598	0.768	5	16	15	60
	BKP_M	2 428	1 623	0.756	1	17	15	30
	BKP_R	2 466	1 646	0.745	1	23	21	32
	BK	2 512	1 687	0.727	12	23	21	38
Burk	CLIQUES	2 191	1 488	0.761	5	18	17	58
	BKP_M	2 239	1 511	0.750	1	18	17	26
	BKP_R	2 283	1 546	0.733	1	19	18	27
	BK	2 325	1 584	0.715	1	19	18	26
Caenorhabditis_eleg.	CLIQUES	14 533	12 008	0.754	17	42	41	588
	BKP_M	15 331	12 639	0.716	9	42	41	598
	BKP_R	15 943	13 417	0.675	8	56	55	623
	BK	16 158	13 688	0.662	13	56	55	542
Chla2	CLIQUES	2 546	1 728	0.747	5	14	13	68
	BKP_M	2 614	1 767	0.731	1	14	13	34
	BKP_R	2 655	1 800	0.717	1	14	13	35
	BK	2 697	1 839	0.702	11	14	13	37
Cupri	CLIQUES	2 255	1 519	0.758	4	16	15	60
	BKP_M	2 311	1 547	0.745	1	17	15	25
	BKP_R	2 363	1 575	0.731	1	19	17	27
	BK	2 415	1 625	0.709	1	19	17	28
Drosophila_melanog.	CLIQUES	52 387	44 442	0.848	44	61	60	3 070
	BKP_M	55 552	46 752	0.806	10	61	60	2 259
	BKP_R	56 951	48 158	0.782	24	241	240	2 401
	BK	57 994	49 235	0.765	38	241	240	2 501
Erw	CLIQUES	2 122	1 436	0.758	5	17	16	59
	BKP_M	2 193	1 474	0.739	1	17	16	23
	BKP_R	2 240	1 502	0.725	1	19	18	23
	BK	2 293	1 552	0.702	1	28	27	22
Esche2	CLIQUES	2 168	1 475	0.792	5	11	10	55
	BKP_M	2 248	1 522	0.767	1	11	10	21
	BKP_R	2 312	1 563	0.747	1	17	16	23
	BK	2 362	1 613	0.724	1	17	16	22
Esche3	CLIQUES	2 245	1 526	0.773	5	16	15	54
	BKP_M	2 323	1 569	0.752	1	16	15	21

Table B.5 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY			TIME
					ms	nodes	leaves	
	BKP$_R$	2 383	1 609	0.733	1	21	20	25
	BK	2 437	1 660	0.711	1	21	20	24
Gnp_1e3	CLIQUES	4 764	3 829	0.965	7	8	7	67
	BKP$_M$	4 884	3 933	0.939	1	8	7	34
	BKP$_R$	4 904	3 952	0.935	1	8	7	33
	BK	4 942	3 989	0.926	1	10	9	23
Gnp_1e4	CLIQUES	69 572	60 125	0.987	41	19	18	3 571
	BKP$_M$	70 008	60 547	0.980	12	19	18	2 920
	BKP$_R$	70 045	60 586	0.979	25	23	22	2 994
	BK	70 115	60 657	0.978	14	26	26	2 957
Gnp_2e3	CLIQUES	10 887	9 025	0.975	13	14	14	201
	BKP$_M$	11 041	9 173	0.959	1	14	14	100
	BKP$_R$	11 056	9 183	0.958	2	26	26	128
	BK	11 097	9 225	0.954	2	26	26	130
Gnp_5e3	CLIQUES	29 639	24 973	0.980	23	27	26	1 012
	BKP$_M$	29 894	25 222	0.970	7	27	26	913
	BKP$_R$	29 928	25 255	0.969	15	27	27	784
	BK	29 978	25 311	0.967	7	40	40	728
GoogleNw	CLIQUES	144 576	95 203	0.791	598	113	112	8 143
	BKP$_M$	396 672	163 881	0.459	451	5 857	3 097	7 712
	BKP$_R$	1 657 297	547 317	0.138	2 441	111 825	36 521	30 773
	BK	⩾1.1B	⩾599M	0.000	⩾1.7M	⩾1.1B	⩾589M	OOT
HC-BIOGRID	CLIQUES	14 316	11 811	0.874	18	33	32	484
	BKP$_M$	14 316	11 811	0.874	5	33	32	451
	BKP$_R$	14 351	11 869	0.870	5	39	39	447
	BK	14 361	11 875	0.869	4	39	39	465
Homo	CLIQUES	2 131	1 440	0.733	5	18	17	60
	BKP$_M$	2 188	1 472	0.717	1	18	17	24
	BKP$_R$	2 222	1 492	0.708	1	20	18	27
	BK	2 270	1 541	0.685	1	20	18	26
Homo_sapiens	CLIQUES	87 129	65 167	0.757	63	51	50	4 451
	BKP$_M$	119 481	82 684	0.596	21	320	133	3 774
	BKP$_R$	134 223	92 898	0.531	21	205	113	4 119
	BK	189 011	131 902	0.374	44	1 329	743	4 346
Mes	CLIQUES	2 387	1 623	0.754	5	9	8	64
	BKP$_M$	2 455	1 657	0.738	1	10	9	29
	BKP$_R$	2 502	1 691	0.723	1	22	20	32
	BK	2 550	1 734	0.705	12	22	20	43
Meta	CLIQUES	8 471	5 788	0.761	13	19	18	331
	BKP$_M$	8 835	5 993	0.735	3	19	18	338
	BKP$_R$	8 985	6 110	0.721	5	43	40	330
	BK	9 224	6 337	0.695	5	43	40	352
Meth	CLIQUES	2 043	1 362	0.768	6	26	25	60
	BKP$_M$	2 107	1 404	0.745	1	26	25	21
	BKP$_R$	2 147	1 431	0.731	1	30	29	24
	BK	2 193	1 476	0.709	1	30	29	23
Meth2	CLIQUES	2 038	1 362	0.766	4	27	26	57
	BKP$_M$	2 104	1 405	0.742	1	27	26	20
	BKP$_R$	2 138	1 427	0.731	1	30	29	22
	BK	2 187	1 476	0.707	1	30	29	21
Meth3	CLIQUES	2 003	1 336	0.772	5	26	25	56
	BKP$_M$	2 066	1 377	0.749	1	26	25	19
	BKP$_R$	2 105	1 405	0.735	1	30	29	23
	BK	2 152	1 449	0.712	1	30	29	21
Meth4	CLIQUES	2 019	1 349	0.773	5	27	26	56
	BKP$_M$	2 086	1 393	0.749	1	27	26	19
	BKP$_R$	2 118	1 418	0.736	1	30	29	23
	BK	2 168	1 463	0.713	1	30	29	22
Meth5	CLIQUES	2 142	1 438	0.763	5	25	24	63

(*continued on next page*)

Table B.5 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY			TIME
					ms	nodes	leaves	
	BKP$_M$	2 199	1 471	0.746	1	25	24	24
	BKP$_R$	2 244	1 502	0.730	1	16	15	38
	BK	2 287	1 544	0.710	11	29	28	77
Meth6	CLIQUES	2 256	1 512	0.759	5	25	24	61
	BKP$_M$	2 332	1 557	0.737	1	25	24	28
	BKP$_R$	2 375	1 588	0.722	1	28	27	28
	BK	2 423	1 637	0.701	1	28	27	27
Mus	CLIQUES	2 506	1 718	0.737	5	30	29	71
	BKP$_M$	2 567	1 754	0.722	1	30	29	36
	BKP$_R$	2 595	1 768	0.716	1	31	30	36
	BK	2 642	1 816	0.697	13	31	30	40
Mus_musculus	CLIQUES	9 995	7 967	0.665	14	32	31	526
	BKP$_M$	10 285	8 177	0.648	5	32	31	524
	BKP$_R$	10 617	8 575	0.618	9	67	66	538
	BK	10 760	8 722	0.608	15	67	66	453
Myc	CLIQUES	2 776	1 873	0.740	5	17	16	82
	BKP$_M$	2 845	1 913	0.725	1	17	16	42
	BKP$_R$	2 893	1 950	0.711	18	26	25	57
	BK	2 940	1 994	0.695	1	26	25	26
Newman-Cond_mat	CLIQUES	80 401	64 149	0.911	72	278	277	8 261
	BKP$_M$	80 552	64 277	0.909	67	278	277	7 765
	BKP$_R$	80 672	64 362	0.908	146	515	515	7 388
	BK	80 695	64 391	0.908	127	515	515	7 798
PGP giantcompo	CLIQUES	37 679	22 207	0.622	28	46	45	2 578
	BKP$_M$	183 262	48 242	0.286	28	7 722	1 297	2 224
	BKP$_R$	305 771	61 188	0.226	27	10 668	1 405	2 335
	BK	1.0B	507M	0.000	53 081	62.9M	31.6M	775 467
Plant	CLIQUES	3 847	2 643	0.760	7	15	14	115
	BKP$_M$	3 953	2 708	0.742	14	15	14	71
	BKP$_R$	4 049	2 789	0.720	1	19	18	47
	BK	4 109	2 852	0.704	1	19	18	66
Pseudo2	CLIQUES	2 109	1 425	0.761	4	11	10	58
	BKP$_M$	2 183	1 468	0.739	1	11	10	23
	BKP$_R$	2 219	1 502	0.722	1	32	31	23
	BK	2 265	1 546	0.702	1	32	31	23
Pseudo4	CLIQUES	2 319	1 550	0.750	4	21	20	64
	BKP$_M$	2 393	1 591	0.731	1	21	20	28
	BKP$_R$	2 438	1 628	0.714	1	34	33	30
	BK	2 489	1 677	0.694	12	34	33	39
Ral	CLIQUES	2 282	1 547	0.759	4	20	19	62
	BKP$_M$	2 337	1 571	0.747	1	20	19	26
	BKP$_R$	2 374	1 604	0.732	1	34	33	30
	BK	2 421	1 648	0.712	12	34	33	37
Rattus_norvegicus	CLIQUES	3 894	3 103	0.634	6	32	31	118
	BKP$_M$	4 045	3 206	0.614	9	32	31	68
	BKP$_R$	4 143	3 328	0.591	2	33	32	60
	BK	4 199	3 387	0.581	2	33	32	82
Rhizo	CLIQUES	2 315	1 562	0.753	5	12	11	63
	BKP$_M$	2 390	1 603	0.734	1	12	11	25
	BKP$_R$	2 438	1 634	0.720	1	29	27	29
	BK	2 493	1 688	0.697	13	28	27	40
Rhizo2	CLIQUES	2 407	1 622	0.752	5	30	29	68
	BKP$_M$	2 468	1 659	0.735	1	30	29	30
	BKP$_R$	2 524	1 690	0.721	1	35	34	33
	BK	2 570	1 737	0.702	11	35	34	39
Rhodo	CLIQUES	2 075	1 395	0.762	5	15	14	57
	BKP$_M$	2 135	1 425	0.746	1	16	14	22
	BKP$_R$	2 180	1 452	0.732	1	24	23	25
	BK	2 224	1 496	0.711	1	24	23	23

Table B.5 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY ms	DELAY nodes	DELAY leaves	TIME
Salmo	CLIQUES	2 250	1 528	0.764	5	18	17	62
	BKP$_M$	2 338	1 579	0.740	1	18	17	23
	BKP$_R$	2 384	1 609	0.726	1	23	21	31
	BK	2 437	1 659	0.704	1	22	21	25
Shigi	CLIQUES	2 198	1 494	0.770	4	16	15	59
	BKP$_M$	2 279	1 539	0.747	1	16	15	25
	BKP$_R$	2 327	1 569	0.733	1	18	17	25
	BK	2 389	1 629	0.706	1	21	20	24

Table B.6
Continued from the previous table. Performance of the clique enumeration algorithms considered on each graph. Times are in milliseconds. OOT: interrupted after 30 minutes limit (statistics at the time of interruption are reported).

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY ms	DELAY nodes	DELAY leaves	TIME
Sino	CLIQUES	2 102	1 399	0.761	5	20	19	55
	BKP$_M$	2 164	1 434	0.742	1	20	19	22
	BKP$_R$	2 211	1 467	0.725	1	28	27	24
	BK	2 256	1 510	0.705	1	30	29	23
Yer2	CLIQUES	2 040	1 373	0.754	4	24	23	57
	BKP$_M$	2 104	1 408	0.735	1	24	23	21
	BKP$_R$	2 136	1 431	0.723	1	26	25	23
	BK	2 183	1 476	0.701	1	26	25	22
add20	CLIQUES	7 969	4 300	0.538	18	45	43	206
	BKP$_M$	13 557	5 280	0.438	3	187	44	133
	BKP$_R$	56 399	11 569	0.200	11	2 597	208	261
	BK	⩾2.1B	⩾1B	0.000	⩾1.7M	⩾2B	⩾1B	OOT
add32	CLIQUES	14 259	9 857	0.458	18	32	31	652
	BKP$_M$	17 604	10 858	0.416	8	51	32	658
	BKP$_R$	18 456	11 618	0.389	30	51	32	610
	BK	19 916	13 094	0.345	5	51	32	570
advogato	CLIQUES	109 561	60 547	0.807	25	20	11	2 085
	BKP$_M$	277 069	120 552	0.405	4	3 646	1 066	1 450
	BKP$_R$	530 720	235 507	0.207	16	3 247	782	2 398
	BK	6 184 403	3 414 206	0.014	429	584 264	300 869	6 493
alr20–MathSciNet	CLIQUES	1 215 919	855 145	0.487	629	134	133	1 601 291
	BKP$_M$	1 675 088	1 030 195	0.404	427	314	133	1 799 792
	BKP$_R$	1 739 115	1 063 066	0.392	373	448	133	1 766 423
	BK	⩾71M	⩾35M	0.012	⩾9.817	⩾16.7M	⩾8.3M	OOT
amazon0302	CLIQUES	1 017 803	618 591	0.652	570	34	32	1 346 318
	BKP$_M$	1 510 615	849 671	0.475	156	78	43	1 374 429
	BKP$_R$	1 639 865	902 368	0.447	171	89	54	1 406 748
	BK	2 258 284	1 410 703	0.286	158	194	119	1 310 301
amazon0312	CLIQUES	1 797 612	1 046 194	0.711	1 465	35	25	OOT
	BKP$_M$	3 523 091	1 858 961	0.397	1 411	361	180	OOT
	BKP$_R$	4 596 720	2 317 155	0.317	1 839	595	298	OOT
	BK	9 743 242	5 993 597	0.123	3 708	4 444	2 670	OOT
amazon0601	CLIQUES	1 991 135	1 146 403	0.692	1 581	35	30	OOT
	BKP$_M$	4 054 110	2 105 612	0.373	2 801	936	493	OOT
	BKP$_R$	5 508 554	2 737 409	0.291	2 764	1 568	867	OOT
	BK	11 635 442	7 150 606	0.110	2 674	3 468	1 825	OOT
auto	CLIQUES	2 861 662	1 388 920	0.793	1 614	45	25	OOT
	BKP$_M$	4 094 876	1 862 742	0.586	305	133	63	OOT
	BKP$_R$	4 264 049	1 928 275	0.571	328	153	77	OOT
	BK	6 707 309	3 803 765	0.288	296	249	130	OOT
bcsstk30	CLIQUES	166 208	66 151	0.101	194	312	255	90 409
	BKP$_M$	10 109 420	657 434	0.010	358	95 746	7 730	107 152
	BKP$_R$	9 986 334	625 237	0.011	337	62 921	4 822	107 202
	BK	⩾1.6B	⩾815M	0.000	⩾1.7M	⩾1.6B	⩾815M	OOT

(*continued on next page*)

Table B.6 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY			TIME
					ms	nodes	leaves	
brack2	CLIQUES	670 843	332 531	0.850	339	367	367	88 669
	BKP$_M$	852 702	385 281	0.733	340	599	368	85 287
	BKP$_R$	875 163	389 976	0.725	354	605	368	82 944
	BK	1 299 629	699 955	0.404	346	673	382	83 387
ca-AstroPh	CLIQUES	199 293	99 233	0.367	143	148	134	14 901
	BKP$_M$	2 129 289	340 307	0.107	62	15 200	1 452	17 076
	BKP$_R$	3 977 870	570 566	0.064	120	30 216	3 887	23 977
	BK	>890M	>445M	0.000	>1.7M	>821M	>410M	OOT
ca-CondMat	CLIQUES	90 641	51 721	0.358	74	119	113	10 194
	BKP$_M$	219 385	77 013	0.240	54	793	179	9 689
	BKP$_R$	235 381	81 732	0.226	93	655	259	9 624
	BK	83.8M	42M	0.000	19 500	31.5M	15.7M	62 624
ca-GrQc	CLIQUES	13 839	8 096	0.482	22	67	64	787
	BKP$_M$	22 458	9 788	0.399	11	135	65	796
	BKP$_R$	28 285	10 369	0.377	31	319	234	754
	BK	>1.6B	>842M	0.000	>1.7M	>1.6B	>842M	OOT
ca-HepPh	CLIQUES	69 214	32 360	0.462	298	217	80	6 791
	BKP$_M$	1 056 161	90 193	0.166	140	15 079	884	11 250
	BKP$_R$	4 851 452	308 094	0.048	6 594	209 519	5 162	83 473
	BK	>634M	>317M	0.000	>1.7M	>533M	>266M	OOT
celegans_metabol	CLIQUES	1 515	717	0.688	4	9	5	20
	BKP$_M$	2 033	835	0.590	1	61	34	5
	BKP$_R$	5 004	2 309	0.214	1	207	137	10
	BK	9 519	6 050	0.081	1	420	271	15
cit-HepPh	CLIQUES	1 072 146	548 573	0.752	263	125	123	50 291
	BKP$_M$	3 718 466	1 476 704	0.279	172	4 864	1 196	48 516
	BKP$_R$	5 590 521	1 942 933	0.212	353	27 223	8 564	54 713
	BK	66 493 864	37 774 838	0.011	1 157	1 075 312	619 415	119 771
cit-HepTh	CLIQUES	1 282 535	599 506	0.775	187	49	27	33 858
	BKP$_M$	9 822 260	3 006 490	0.155	87	93 146	20 752	37 077
	BKP$_R$	14 743 196	4 062 378	0.114	277	169 144	30 793	56 334
	BK	1.1B	608M	0.001	25 908	28.2M	14.9M	1.3M
citeseer	CLIQUES	772 939	515 362	0.841	1 388	285	284	886 448
	BKP$_M$	853 919	561 069	0.772	1 405	285	284	900 147
	BKP$_R$	863 715	568 537	0.762	1 942	415	415	939 820
	BK	919 558	625 545	0.693	1 919	415	415	938 256
cnr_2000	CLIQUES	1 787 966	935 464	0.675	71 464	4 908	4 906	OOT
	BKP$_M$	9 845 694	2 832 125	0.209	88 462	1 356 237	588 596	OOT
	BKP$_R$	12 025 587	1 984 207	0.269	146 057	760 426	88 768	OOT
	BK	>1.9B	>993M	0.000	>1.4M	>1.8B	>949M	OOT
coli1_1Inter	CLIQUES	841	582	0.789	4	10	9	19
	BKP$_M$	867	597	0.769	1	10	9	8
	BKP$_R$	961	647	0.709	1	11	9	10
	BK	980	674	0.681	1	11	9	5
cora	CLIQUES	7 353	4 869	0.732	16	16	15	205
	BKP$_M$	8 398	5 388	0.661	1	17	15	170
	BKP$_R$	8 915	5 706	0.624	1	19	15	180
	BK	9 846	6 549	0.544	2	25	15	189
crack	CLIQUES	49 606	33 670	0.598	709	3 149	2 717	2 774
	BKP$_M$	55 169	38 118	0.528	508	4 486	3 745	1 913
	BKP$_R$	55 361	37 436	0.538	315	3 581	2 971	1 893
	BK	60 762	43 585	0.462	545	4 487	3 746	2 291
cs4	CLIQUES	64 488	43 679	0.892	41	11	9	8 207
	BKP$_M$	67 197	45 460	0.857	20	14	10	7 103
	BKP$_R$	67 214	45 478	0.856	14	14	10	7 267
	BK	68 815	47 076	0.827	18	14	10	7 252
cti	CLIQUES	64 705	47 531	1.000	30	2	1	5 132
	BKP$_M$	65 266	48 091	0.988	15	3	1	5 069
	BKP$_R$	65 231	48 056	0.989	13	3	1	4 768
	BK	65 435	48 254	0.985	13	3	2	4 614

Table B.6 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY			TIME
					ms	nodes	leaves	
darwin BookInter	CLIQUES	258 587	124 786	0.936	39	13	6	2 267
	BKP$_M$	401 087	169 954	0.687	84	183	70	1 606
	BKP$_R$	6 401 693	2 763 439	0.042	70	15 920	6 533	18 404
	BK	18 404 054	12 090 465	0.010	69	43 873	27 491	38 601
data	CLIQUES	26 478	13 062	0.913	18	40	37	316
	BKP$_M$	33 539	13 799	0.864	3	76	38	277
	BKP$_R$	32 328	13 717	0.870	3	76	38	281
	BK	56 844	28 941	0.412	7	81	42	335
dip20090126_MAX	CLIQUES	60 986	50 733	0.812	37	63	62	6 687
	BKP$_M$	60 986	50 733	0.812	27	63	62	6 374
	BKP$_R$	61 130	50 877	0.810	18	62	61	5 716
	BK	61 131	50 879	0.810	13	63	62	6 052
eatRS	CLIQUES	611 406	400 161	0.745	158	26	23	24 786
	BKP$_M$	877 176	545 978	0.546	37	77	50	22 178
	BKP$_R$	973 775	602 357	0.495	434	1 405	1 403	25 510
	BK	1 219 994	805 416	0.370	2 816	8 578	8 578	25 361
eatSR	CLIQUES	613 821	417 035	0.715	216	37	35	24 994
	BKP$_M$	921 047	611 182	0.488	36	135	83	22 501
	BKP$_R$	1 051 500	702 407	0.424	33	177	101	25 247
	BK	1 219 989	869 598	0.343	35	278	195	26 465
email-Enron	CLIQUES	749 731	356 776	0.636	134	285	284	34 301
	BKP$_M$	5 601 593	2 104 096	0.108	107	7 539	2 571	40 097
	BKP$_R$	12 786 775	4 107 203	0.055	150	17 833	3 959	57 821
	BK	107 255 300	59 106 680	0.004	637	520 857	288 361	213 292
email-EuAll	CLIQUES	838 104	667 784	0.566	3 298	439	438	1 127 490
	BKP$_M$	1 923 349	1 084 273	0.349	3 167	1 132	492	982 012
	BKP$_R$	2 539 375	1 348 165	0.280	6 811	1 614	615	1 155 400
	BK	9 489 630	5 668 955	0.067	3 242	14 078	8 005	1 150 487
email	CLIQUES	7 111	4 480	0.729	10	35	35	81
	BKP$_M$	10 992	6 348	0.515	1	165	44	50
	BKP$_R$	11 908	7 009	0.466	1	63	35	43
	BK	20 473	12 529	0.261	1	981	500	44
eva	CLIQUES	13 374	11 712	0.564	47	327	326	1 301
	BKP$_M$	13 456	11 763	0.561	48	327	326	1 309
	BKP$_R$	14 029	11 786	0.560	32	327	326	867
	BK	14 044	11 804	0.559	32	327	326	904
fe_4elt2	CLIQUES	45 480	23 779	0.911	53	163	160	3 171
	BKP$_M$	54 305	24 856	0.871	40	322	161	2 599
	BKP$_R$	52 856	24 988	0.867	43	322	161	2 621
	BK	65 652	34 774	0.623	33	323	162	2 344
fe_body	CLIQUES	195 950	82 641	0.600	82	20	17	33 252
	BKP$_M$	263 969	100 373	0.494	37	37	18	31 535
	BKP$_R$	260 047	99 288	0.499	22	35	17	31 393
	BK	401 836	208 536	0.238	24	51	29	31 049
fe_ocean	CLIQUES	553 025	410 454	0.998	230	5	4	269 666
	BKP$_M$	553 025	410 454	0.998	48	5	4	265 257
	BKP$_R$	553 025	410 454	0.998	43	5	4	263 698
	BK	553 031	410 454	0.998	40	5	4	263 060
fe_pwt	CLIQUES	180 694	83 308	0.442	397	541	540	25 164
	BKP$_M$	244 614	98 415	0.374	398	1 083	551	23 935
	BKP$_R$	243 242	98 376	0.375	346	1 083	551	23 434
	BK	361 176	186 903	0.197	331	1 083	551	23 169
fe_rotor	CLIQUES	1 308 747	662 939	0.836	731	713	356	237 050
	BKP$_M$	1 704 664	783 342	0.708	672	1 775	713	233 505
	BKP$_R$	1 688 208	783 691	0.708	667	1 653	713	233 654
	BK	2 438 630	1 333 112	0.416	681	2 147	1 073	234 273
fe_sphere	CLIQUES	66 079	32 778	1.000	31	4	2	5 160
	BKP$_M$	81 913	32 783	1.000	20	4	2	4 955
	BKP$_R$	76 497	32 772	1.000	15	4	2	4 645
	BK	98 307	49 155	0.667	20	5	3	4 830

(*continued on next page*)

Table B.6 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY ms	DELAY nodes	DELAY leaves	TIME
fe_tooth	CLIQUES	823 472	408 416	0.847	423	341	338	122 442
	BKP$_M$	1 045 309	468 607	0.739	435	743	403	119 323
	BKP$_R$	1 076 393	477 319	0.725	690	784	572	119 156
	BK	1 599 742	858 275	0.403	681	829	597	119 330
finan512	CLIQUES	316 585	198 531	0.346	29 697	26 332	25 537	93 062
	BKP$_M$	467 270	224 420	0.306	29 487	51 111	26 332	89 430
	BKP$_R$	472 222	231 960	0.296	29 828	51 378	26 527	90 120
	BK	632 833	356 665	0.192	29 750	51 409	26 555	90 191
forest1e4	CLIQUES	173 966	89 201	0.824	67	44	42	3 416
	BKP$_M$	287 154	134 434	0.547	13	130	65	2 834
	BKP$_R$	831 517	374 864	0.196	26	1 371	540	5 089
	BK	1 722 318	983 520	0.075	29	3 714	2 088	7 373
forest1e4_2	CLIQUES	238 453 305	100 171 630	0.967	204	41	15	1 128 622
	BKP$_M$	855 014 958	239 479 827	0.349	1 395	657 501	216 935	OOT
	BKP$_R$	433 214 072	79 685 156	0.080	6 616	1 632 740	309 377	OOT
	BK	322 216 986	161 296 155	0.000	12 128	1 829 400	919 223	OOT
forest5e4	CLIQUES	561 624	326 434	0.744	1 360	72	70	42 698
	BKP$_M$	959 946	536 669	0.452	1 378	313	163	42 072
	BKP$_R$	1 347 420	771 182	0.315	1 320	365	350	51 576
	BK	1 775 685	1 122 978	0.216	1 318	589	350	53 185

Table B.7
Continued from the previous tables. Performance of the clique enumeration algorithms considered on each graph. Times are in milliseconds. OOT: interrupted after 30 minutes limit (statistics at the time of interruption are reported).

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY ms	DELAY nodes	DELAY leaves	TIME
forest5e4_2	CLIQUES	209 493 320	88 639 228	0.938	1 247	47	16	OOT
	BKP$_M$	391 370 869	127 318 923	0.358	2 356	79 545	24 499	OOT
	BKP$_R$	204 065 427	44 396 705	0.145	7 402	643 545	141 004	OOT
	BK	109 276 728	54 701 696	0.001	10 675	590 266	297 243	OOT
french BookInter	CLIQUES	36 352	23 502	0.895	38	12	11	1 594
	BKP$_M$	40 762	25 576	0.822	24	34	24	1 246
	BKP$_R$	67 356	47 248	0.445	25	62	61	1 760
	BK	77 037	57 647	0.365	12	62	61	1 860
geom	CLIQUES	15 972	10 179	0.455	20	25	23	1 072
	BKP$_M$	25 249	12 646	0.366	17	252	52	975
	BKP$_R$	30 437	15 219	0.304	3	253	47	827
	BK	4 445 833	2 227 505	0.002	316	524 409	262 213	3 349
hep-th-cit_MAX	CLIQUES	1 281 934	599 104	0.776	266	49	26	34 627
	BKP$_M$	9 821 630	3 006 083	0.155	109	93 146	20 752	40 633
	BKP$_R$	14 685 195	4 058 760	0.114	269	181 669	38 689	57 830
	BK	⩾1.1B	⩾608M	0.001	⩾27K	⩾28M	⩾14M	⩾1.4M
hprd_pp	CLIQUES	52 219	37 468	0.785	34	35	34	3 003
	BKP$_M$	66 818	45 352	0.648	8	50	34	2 048
	BKP$_R$	74 009	50 884	0.578	13	75	73	2 447
	BK	92 830	65 214	0.451	30	505	256	2 502
iPfam	CLIQUES	3 095	1 430	0.276	18	57	47	173
	BKP$_M$	14 672	2 060	0.192	3	779	74	72
	BKP$_R$	260 488	11 012	0.036	270	40 478	1 167	1 506
	BK	⩾1.5B	⩾782M	0.000	⩾1.7M	⩾1.5B	⩾782M	OOT
interdom	CLIQUES	46 895	10 717	0.313	116	137	88	5 055
	BKP$_M$	5 879 885	185 950	0.018	5 007	395 562	11 187	68 271
	BKP$_R$	⩾114M	⩾2.6M	0.000	⩾217K	⩾14M	⩾360K	OOT
	BK	⩾408M	⩾204M	0.000	⩾1M	⩾408M	⩾204M	OOT
itdk0304_rlinks	CLIQUES	911 737	651 753	0.740	544	286	285	624 428
	BKP$_M$	1 357 730	878 126	0.549	559	3 245	1 258	616 083
	BKP$_R$	2 119 410	1 118 460	0.431	1 308	6 604	2 258	612 965
	BK	10 096 881	6 192 157	0.078	1 364	110 913	66 222	649 734

Table B.7 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY			TIME
					ms	nodes	leaves	
japanese BookInter	CLIQUES	12 601	7 942	0.881	14	6	5	220
	BKP_M	14 232	8 645	0.809	3	18	9	159
	BKP_R	28 692	18 468	0.379	2	142	83	260
	BK	36 399	27 171	0.257	4	131	87	320
jazz	CLIQUES	3 321	1 132	0.659	6	23	11	45
	BKP_M	16 235	3 328	0.224	1	1 346	235	28
	BKP_R	37 298	7 376	0.101	4	2 127	356	57
	BK	⩾1B	⩾538M	0.000	199 206	268.2M	134.1M	800 810
kron14	CLIQUES	29 242	24 934	0.899	199	21	20	1 680
	BKP_M	30 269	25 712	0.872	138	21	20	1 438
	BKP_R	33 911	26 161	0.857	80	25	24	1 430
	BK	34 041	29 377	0.763	135	209	198	1 979
kron16	CLIQUES	95 376	78 160	0.824	99	97	96	13 607
	BKP_M	95 905	78 639	0.819	35	97	96	13 211
	BKP_R	96 507	80 107	0.804	228	756	756	12 531
	BK	96 552	80 159	0.804	311	756	756	12 981
m14b	CLIQUES	2 788 595	1 361 210	0.648	28 893	5 964	5 931	1 369 064
	BKP_M	5 037 353	2 186 619	0.403	28 124	7 460	6 072	1 280 078
	BKP_R	4 993 625	2 096 438	0.421	29 271	7 300	5 956	1 313 615
	BK	8 725 769	4 855 795	0.182	27 680	7 547	6 160	1 306 839
memplus	CLIQUES	54 421	30 428	0.543	1 001	4 099	4 097	5 130
	BKP_M	80 374	37 771	0.438	968	6 148	4 097	4 758
	BKP_R	2 255 839	135 680	0.122	1 449	182 903	5 807	21 106
	BK	⩾784M	⩾392M	0.000	⩾1.7M	⩾784M	⩾392M	OOT
p2p-Gnutella31	CLIQUES	208 988	177 302	0.815	119	18	17	56 337
	BKP_M	211 619	179 671	0.804	40	18	17	54 090
	BKP_R	211 948	180 020	0.803	35	18	17	53 762
	BK	212 518	180 608	0.800	40	18	17	54 099
ppi_dip_swiss	CLIQUES	15 373	11 311	0.785	18	26	25	420
	BKP_M	19 900	13 663	0.650	5	31	25	419
	BKP_R	20 710	14 176	0.626	8	75	73	430
	BK	26 232	18 321	0.485	7	135	73	445
ppi_gcc	CLIQUES	213 913	155 700	0.777	85	108	107	23 463
	BKP_M	416 254	234 663	0.516	52	1 508	541	22 738
	BKP_R	505 383	271 631	0.446	54	1 745	627	24 156
	BK	2 034 507	1 249 506	0.097	47	19 383	10 381	25 934
psimap	CLIQUES	2 465	1 194	0.231	14	104	103	147
	BKP_M	21 078	2 124	0.130	8	1 373	149	81
	BKP_R	58 955	3 772	0.073	44	6 966	199	297
	BK	⩾1.4B	⩾708M	0.000	⩾1.6M	⩾1.1B	⩾598M	OOT
roadNet-PA	CLIQUES	294 740	195 914	0.841	539	29	28	OOT
	BKP_M	295 447	196 423	0.819	437	29	28	OOT
	BKP_R	303 494	201 809	0.818	424	29	28	OOT
	BK	336 425	225 942	0.794	384	29	28	OOT
roadNet-TX	CLIQUES	232 282	155 725	0.808	657	31	30	OOT
	BKP_M	264 002	176 464	0.794	593	31	30	OOT
	BKP_R	225 516	151 300	0.782	552	31	30	OOT
	BK	253 876	171 661	0.766	434	31	30	OOT
s838	CLIQUES	1 270	875	0.854	4	36	35	25
	BKP_M	1 349	945	0.790	1	36	35	8
	BKP_R	1 355	947	0.789	1	25	25	7
	BK	1 372	969	0.771	1	49	48	7
Epinions1	CLIQUES	4 560 654	2 149 043	0.825	364	80	78	132 243
	BKP_M	43 956 913	12 919 690	0.137	217	41 428	12 491	194 514
	BKP_R	80 468 021	22 850 584	0.078	222	48 428	10 675	349 183
	BK	737 521 894	389 384 748	0.001	10 110	3 703 683	1 960 278	OOT
Slashdot 0811	CLIQUES	1 772 772	983 257	0.837	372	91	84	124 969
	BKP_M	48 799 044	12 621 926	0.065	311	152 753	38 260	207 689
	BKP_R	132 556 812	27 082 053	0.030	1 103	540 950	110 226	367 782
	BK	>1B	>537M	0.000	>165K	>101M	>51M	OOT

(*continued on next page*)

Table B.7 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY			TIME
					ms	nodes	leaves	
Slashdot 0902	CLIQUES	1 967 318	1 069 579	0.832	414	86	85	142 697
	BKP_M	65 289 537	16 046 787	0.055	532	249 337	54 309	260 740
	BKP_R	149 012 237	28 699 728	0.031	1 225	582 211	102 164	426 999
	BK	>928M	>556M	0.000	>89K	>47M	>29M	OOT
Slashdot 090221	CLIQUES	1 871 452	1 035 183	0.825	414	70	68	141 175
	BKP_M	55 634 373	15 064 515	0.057	412	190 787	56 385	226 360
	BKP_R	113 266 541	22 538 850	0.038	815	449 600	90 981	342 089
	BK	>1.2B	>653M	0.000	>55K	>39M	>20M	OOT
sign-epinions	CLIQUES	61 803 460	24 265 229	0.916	679	123	122	820 659
	BKP_M	495 252 038	107 973 282	0.129	15 157	2 756 897	149 240	OOT
	BKP_R	411 796 194	93 373 913	0.042	914	219 326	45 228	OOT
	BK	370 451 642	198 274 042	0.001	10 803	2 541 280	1 298 873	OOT
spanish BookInter	CLIQUES	117 784	61 773	0.938	107	10	7	2 710
	BKP_M	157 981	75 347	0.769	304	286	121	2 545
	BKP_R	1 722 091	761 066	0.076	253	4 879	2 427	9 476
	BK	4 159 948	2 659 625	0.022	170	9 969	6 319	13 344
string	CLIQUES	48 808	23 745	0.782	18	53	14	582
	BKP_M	189 526	54 749	0.339	11	3 135	525	632
	BKP_R	2 331 906	203 912	0.091	499	190 270	10 244	6 231
	BK	>1B	>544M	0.000	>1.7M	>1B	>543M	OOT
t60k	CLIQUES	149 443	89 609	0.998	72	3	2	46 286
	BKP_M	149 443	89 609	0.998	33	3	2	44 160
	BKP_R	149 443	89 609	0.998	25	3	2	42 858
	BK	149 446	89 609	0.998	29	3	2	42 896
trust	CLIQUES	12 729 403	5 317 954	0.887	293	89	57	139 868
	BKP_M	175 821 436	47 034 767	0.100	168	63 568	16 733	518 336
	BKP_R	544 873 810	139 748 484	0.033	525	155 355	40 216	OOT
	BK	664 509 527	381 627 044	0.001	10 331	3 086 333	1 936 605	OOT
ud_1e3	CLIQUES	18 914	4 590	0.546	20	153	64	662
	BKP_M	179 200	21 578	0.116	19	9 419	958	477
	BKP_R	17 518 779	677 711	0.004	10 858	2 535 801	80 467	72 135
	BK	>829M	>414M	0.000	>1.8	>829M	>414M	OOT
ud_1e4	CLIQUES	1 137 453	203 836	0.650	1 037	687	444	148 183
	BKP_M	243 952 505	8 356 294	0.005	101 806	12M	490 665	OOT
	BKP_R	77 609 172	872 451	0.007	205 614	8 848 576	111 558	OOT
	BK	>1.7B	>874M	0.000	>344K	>333M	>166M	OOT
ud_2e3	CLIQUES	44 363	10 952	0.566	34	152	72	1 512
	BKP_M	995 248	96 760	0.064	94	22 131	2 120	2 658
	BKP_R	191 326 234	4 950 645	0.001	134 418	17.8M	403 917	1 394 955
	BK	>685M	>342M	0.000	>1.7M	>637M	>318M	OOT
ud_5e3	CLIQUES	145 824	34 221	0.554	96	291	215	8 122
	BKP_M	6 635 803	535 799	0.035	598	223 119	13 583	19 394
	BKP_R	146 409 541	2 322 075	0.004	628 006	52.3M	915 131	OOT
	BK	>1.1B	>596M	0.000	>359K	>239M	>119M	OOT
uk	CLIQUES	11 658	7 224	0.946	13	8	7	597
	BKP_M	11 660	7 226	0.946	4	8	7	589
	BKP_R	11 660	7 226	0.946	4	8	7	630
	BK	11 663	7 227	0.946	10	8	7	522
us_1e3	CLIQUES	15 010	4 877	0.464	17	309	242	246
	BKP_M	129 342	24 468	0.093	16	4 434	857	232
	BKP_R	155 911	28 228	0.080	30	8 240	2 387	253
	BK	37 543 683	18 996 853	0.000	1 477	2 231 033	1 116 109	24 273
us_2e3	CLIQUES	43 093	12 778	0.456	57	484	337	738
	BKP_M	673 934	104 229	0.056	74	15 395	3 348	1 432
	BKP_R	859 300	124 491	0.047	133	36 427	8 537	1 307
	BK	>1.7B	>861M	0.000	>616K	>559M	>279M	OOT
us_5e3	CLIQUES	195 839	52 254	0.458	496	1 772	1 356	5 294
	BKP_M	6 910 270	834 062	0.029	584	92 470	14 460	11 479
	BKP_R	9 055 721	1 052 425	0.023	254	157 919	22 210	13 778
	BK	>1.7B	>884M	0.000	>483K	>444M	>222M	OOT

Table B.7 (*continued*)

GRAPH	ALGORITHM	NODES	LEAVES	CLIQUES/LEAVES	DELAY			TIME
					ms	nodes	leaves	
vibrobox	CLIQUES	180 432	97 681	0.223	991	6 592	5 223	7 493
	BKP_M	1 334 614	329 469	0.066	1 035	104 759	29 446	7 536
	BKP_R	1 834 440	379 741	0.057	1 352	153 919	41 370	7 861
	BK	172 888 265	86 564 342	0.000	4 299	6 698 084	3 351 565	117 823
wave	CLIQUES	2 003 422	985 835	0.852	3 495	1 265	1 262	608 397
	BKP_M	2 699 568	1 180 092	0.712	3 513	3 307	1 662	602 282
	BKP_R	2 700 357	1 162 683	0.723	3 671	3 308	1 662	603 944
	BK	3 991 309	2 053 999	0.409	3 475	3 731	2 085	605 496
whitaker3	CLIQUES	40 754	20 075	0.956	24	15	13	2 811
	BKP_M	48 361	20 956	0.916	9	23	13	1 912
	BKP_R	46 086	20 538	0.934	9	23	13	2 010
	BK	57 980	29 358	0.654	23	24	14	2 046
wiki-Vote	CLIQUES	1 092 582	532 791	0.861	56	27	19	8 068
	BKP_M	2 767 559	1 187 310	0.387	35	1 357	451	6 905
	BKP_R	8 239 850	2 957 681	0.155	28	10 643	4 018	22 178
	BK	41 799 504	24 265 321	0.019	207	127 743	71 000	85 294
wing	CLIQUES	178 698	124 985	0.865	3 731	4 878	4 877	49 937
	BKP_M	185 937	130 198	0.831	3 764	4 878	4 877	48 511
	BKP_R	186 128	130 387	0.830	4 358	5 604	5 602	48 207
	BK	190 262	134 522	0.804	7 468	9 702	9 702	48 360
wing_nodal	CLIQUES	136 028	66 288	0.786	57	188	158	4 360
	BKP_M	194 371	90 428	0.576	67	420	237	3 634
	BKP_R	199 945	92 114	0.566	57	359	205	3 784
	BK	304 196	171 231	0.304	55	401	248	3 636
yeastInter	CLIQUES	1 649	1 331	0.745	3	28	27	36
	BKP_M	1 718	1 370	0.723	1	28	27	10
	BKP_R	1 815	1 387	0.714	1	28	27	13
	BK	1 841	1 429	0.693	1	28	27	12
yeast_bo	CLIQUES	3 876	2 952	0.657	8	35	34	120
	BKP_M	4 051	3 054	0.635	14	35	34	72
	BKP_R	4 155	3 130	0.620	1	35	34	51
	BK	4 322	3 280	0.591	1	61	39	74

References

[1] D. Avis, K. Fukuda, Reverse search for enumeration, Discrete Appl. Math. 65 (1996) 21–46.
[2] I.M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo, The maximum clique problem, in: Handbook of Combinatorial Optimization, Springer, 1999, pp. 1–74.
[3] C. Bron, J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, Commun. ACM 16 (1973) 575–577.
[4] F. Cazals, C. Karande, Reporting maximal cliques: new insights into an old problem, Research Report RR-5615, INRIA, 2006.
[5] L. Chang, J.X. Yu, L. Qin, Fast maximal cliques enumeration in sparse graphs, Algorithmica 66 (2013) 173–186.
[6] N. Chiba, T. Nishizeki, Arboricity and subgraph listing algorithms, SIAM J. Comput. 14 (1985) 210–223.
[7] C. Comin, R. Rizzi, An improved upper bound on maximal clique listing via rectangular fast matrix multiplication, Algorithmica 80 (2018) 3525–3562.
[8] A. Conte, R. Grossi, A. Marino, L. Versari, Sublinear-space bounded-delay enumeration for massive network analytics: maximal cliques, in: ICALP 2016, 2016, 148.
[9] A. Conte, R. Grossi, A. Marino, L. Versari, Sublinear-space and bounded-delay algorithms for maximal clique enumeration in graphs, Algorithmica 82 (2020) 1547–1573.
[10] A. Conte, E. Tomita, Overall and delay complexity of the CLIQUES and Bron-Kerbosch algorithms, in: R. Uehara, S.H. Hong, S.C. Nandy (Eds.), WALCOM: Algorithms and Computation, Springer International Publishing, Cham, 2021, pp. 195–207.
[11] D. Eppstein, M. Löffler, D. Strash, Listing all maximal cliques in large sparse real-world graphs, ACM J. Exp. Algorithmics 18 (2013) 3.1:1–3.1:21.
[12] V. Fionda, Networks in biology, in: S. Ranganathan, M. Gribskov, K. Nakai, C. Schönbach (Eds.), Encyclopedia of Bioinformatics and Computational Biology, Academic Press, Oxford, 2019, pp. 915–921.
[13] S. Fortunato, Community detection in graphs, Phys. Rep. 486 (2010) 75–174.
[14] T. Fujii, E. Tomita, On efficient algorithms for finding a maximum clique, Technical Report AL81-113, IECE, 1982, pp. 25–34 (in Japanese).
[15] R. Impagliazzo, R. Paturi, On the complexity of k-SAT, J. Comput. Syst. Sci. 62 (2001) 367–375.
[16] D.S. Johnson, M. Yannakakis, C.H. Papadimitriou, On generating all maximal independent sets, Inf. Process. Lett. 27 (1988) 119–123.
[17] Laboratory of Algorithms, modelS, and Analysis of Graphs and NEtworks, https://www.pilucrescenzi.it/wp/software/lasagne/. (Accessed April 2021). Online.
[18] J. Leskovec, A. Krevl, SNAP datasets: Stanford large network dataset collection, https://snap.stanford.edu/data/.
[19] K. Makino, T. Uno, New algorithms for enumerating all maximal cliques, in: SWAT 2004, in: LNCS, vol. 3111, Springer, 2004, pp. 260–272.
[20] G. Manoussakis, A new decomposition technique for maximal clique enumeration for sparse graphs, Theor. Comput. Sci. 770 (2019) 25–33.
[21] J.W. Moon, L. Moser, On cliques in graphs, Isr. J. Math. 3 (1965) 23–28.
[22] K.A. Naudé, Refined pivot selection for maximal clique enumeration in graphs, Theor. Comput. Sci. 613 (2016) 28–37, https://doi.org/10.1016/j.tcs.2015.11.016, https://www.sciencedirect.com/science/article/pii/S0304397515010130.
[23] P.M. Pardalos, J. Xue, The maximum clique problem, J. Glob. Optim. 4 (1994) 301–328.

[24] P. San Segundo, J. Artieda, D. Strash, Efficiently enumerating all maximal cliques with bit-parallelism, Comput. Oper. Res. 92 (2018) 37–46.
[25] E. Tomita, Clique enumeration, in: Ming-Yang Kao (Ed.), Encyclopedia of Algorithms, 2nd edition, Springer US, Boston, MA, 2016, pp. 1–6.
[26] E. Tomita, Efficient algorithms for finding maximum and maximal cliques and their applications - keynote, in: WALCOM 2017, in: LNCS, vol. 10167, Springer, 2017, pp. 3–15.
[27] E. Tomita, A. Conte, Another time-complexity analysis for maximal clique enumeration algorithm CLIQUES, Technical Report COMP2020-1(2020-05), IEICE COMP., 2020, pp. 1–8, http://id.nii.ac.jp/1438/00009571/.
[28] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for finding all the cliques, Technical Report UEC-TR-C5(2), The University of Electro - Communications, 1988, http://id.nii.ac.jp/1438/00001898/.
[29] E. Tomita, A. Tanaka, H. Takahashi, An Optimal Algorithm for Finding All the Cliques, Technical Report 1989-AL-012(98), IPSJ, 1989, pp. 91–98, https://ipsj.ixsq.nii.ac.jp/ej/?action=pages_view_main&active_action=repository_view_main_item_detail&item_id=32671&item_no=1&page_id=13&block_id=8.
[30] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques, in: COCOON 2004, Springer, 2004, pp. 161–170.
[31] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques and computational experiments, Theor. Comput. Sci. 363 (2006) 28–42.
[32] S. Tsukiyama, M. Ide, H. Ariyoshi, I. Shirakawa, A new algorithm for generating all the maximal independent sets, SIAM J. Comput. 6 (1977) 505–517.