Listing All Maximal Cliques in Large Sparse Real-World Graphs in Near-Optimal Time

DAVID EPPSTEIN, University of California, Irvine
MAARTEN LÖFFLER, Utrecht University
and DARREN STRASH, University of California, Irvine

We modify an algorithm of Bron and Kerbosch [1973] for maximal clique enumeration to choose more carefully the order in which the vertices are processed, giving us a fixed-parameter tractable algorithm with running time $O(dn3^{d/3})$ on graphs with n vertices and degeneracy d. Our time bound matches a worst case bound of $(n-d)3^{d/3}$ on the number of maximal cliques when d is a multiple of 3 and $n \geq d+3$. For graphs with degeneracy d and maximum clique size κ, the algorithm satisfies a time bound of $O(d^2 n(d/\kappa)^{\kappa})$, and for K_h-minor-free graphs we obtain a time bound of $n2^{O(h \log\log h)}$, matching a bound of Fomin et al. [2010] for the number of cliques in these graphs. We implement our algorithm and provide a comparative analysis of it and a different variation of the Bron–Kerbosch algorithm by Tomita et al. [2006] on a large corpus of real-world graphs with low degeneracy. Our algorithm always performs comparably with Tomita et al. on moderately sized graphs, in some cases is much faster, and due to its more space efficient data structures is capable of being applied to significantly larger graphs.

Categories and Subject Descriptors: F.2.2 [**Nonnumerical Algorithms and Problems**]: Computations on discrete structures

General Terms: Algorithms; Design; Experimentation; Performance; Theory

Additional Key Words and Phrases: Maximal clique enumeration; Bron–Kerbosch algorithm; parameterized complexity

ACM Reference Format:
Eppstein, D., Löffler, M., and Strash, D. 2011. Listing All Maximal Cliques in Large Sparse Real-World Graphs in Near-Optimal Time. ACM J. Exp. Algor. V, N, Article A (YYYY), 20 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Cliques, complete subgraphs of a graph, are of great importance in many applications. Often, it is important to find not just one large clique, but all *maximal cliques*, sets of vertices that form a clique, but for which no superset is also a clique (Figure 1). Many algorithms are now known for this problem [Akkoyunlu 1973; Bron and Kerbosch 1973; Cazals and Karande 2008; Cheng et al. 2010; Chiba and Nishizeki 1985; Chrobak and Eppstein 1991; Du et al. 2009; Gély et al. 2009; Gerhards and Lindenberg 1979; Harary and Ross 1957; Johnston 1976; Lu et al. 2010; Makino and Uno 2004; Modani and Dey 2008; Mulligan and Corneil 1972; Pan and Santos 2008; Samatova et al. 2008; Schmidt et al. 2009; Tomita et al. 2006; Zhang et al. 2005] and

This research was supported in part by the National Science Foundation under grants 0830403 and 1217322, and by the Office of Naval Research under MURI grant N00014-08-1-1015.
Author's addresses: D. Eppstein, and D. Strash, Computer Science Department, University of California, Irvine, USA. {eppstein|dstrash}(at)uci.edu. M. Löffler, Department of Computing and Information Sciences, Utrecht University, the Netherlands. m.loffler(at)uu.nl.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

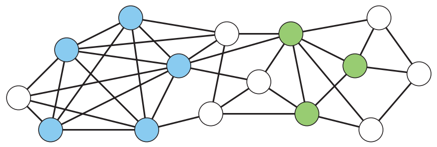
Fig. 1. A graph with two of its maximal cliques marked

for the complementary problem of finding maximal independent sets [Eppstein 2009; Johnson et al. 1988; Lawler et al. 1980; Loukakis and Tsouros 1981; Tsukiyama et al. 1977]. One of the most successful in practice is the *Bron–Kerbosch algorithm*, a simple backtracking procedure that recursively solves subproblems specified by three sets of vertices: the vertices that are required to be included in a partial clique, the vertices that are to be excluded from the clique, and some remaining vertices whose status still needs to be determined [Bron and Kerbosch 1973; Cazals and Karande 2008; Johnston 1976; Koch 2001; Tomita et al. 2006]. All maximal cliques can be listed in polynomial time per clique [Lawler et al. 1980; Tsukiyama et al. 1977] or in a total time proportional to the maximum possible number of cliques in an n-vertex graph, without additional polynomial factors [Eppstein 2003; Tomita et al. 2006]. In particular, a variant of the Bron–Kerbosch algorithm is optimal in this sense [Cazals and Karande 2008; Tomita et al. 2006]. Unfortunately this maximum possible number of cliques is exponential [Moon and Moser 1965], so that all general-purpose algorithms for listing maximal cliques necessarily take exponential time in the worst case.

1.1. Parameterized Analysis

We are faced with a dichotomy between theory, which states categorically that clique finding takes exponential time, and practice, according to which clique finding is useful and can be efficient in its areas of application. One standard way of resolving dilemmas such as this one is to apply *parameterized complexity* [Downey and Fellows 1999]: one seeks a parameter of instance complexity such that instances with small parameter values can be solved quickly. A parameterized problem is said to be *fixed-parameter tractable* if instances with size n and parameter value p can be solved in a time bound of the form $f(p)n^{O(1)}$, where f may grow exponentially or worse with p but is independent of n. With this style of analysis, instances with a small parameter value are used to model problems that can be solved quickly, while instances with a large parameter value represent a theoretical worst case that, one hopes, does not arise in practice.

The size of the largest clique does not work well as a parameter: the maximum clique problem, parameterized by clique size, is hard for $\mathbf{W}[1]$, implying that it is unlikely to have a fixed-parameter tractable algorithm [Downey and Fellows 1995], and Turán graphs $K_{\frac{n}{k},\frac{n}{k},\frac{n}{k},\ldots}$ have $(n/k)^k$ maximal cliques of size k, forcing any algorithm that lists them all to take time larger than any fixed-parameter-tractable bound. However, clique size is not the only parameter one can choose. In this paper, we parameterize maximal clique finding by *degeneracy*, a measure of the sparseness of a graph that has been used for other fixed-parameter problems [Alon and Gutner 2009; Cai et al. 2006; Golovach and Villanger 2008; Kloks and Cai 2000]. Sparse graphs often appear in practice; for instance, the World Wide Web graph, citation networks, and collaboration graphs have low arboricity [Goel and Gustedt 2006] and therefore low degeneracy. The h-index, a measure of sparsity that upper bounds degeneracy, is also low for social networks [Eppstein and Spiro 2009]. In addition, our experiments in Section 4 show that protein–protein interaction networks have low degeneracy as well. Furthermore, planar graphs have degeneracy at most five [Lick and White 1970], and the Barabási–

Albert model of preferential attachment [Barabási and Albert 1999], frequently used to model large scale-free social networks, produces graphs with bounded degeneracy.

1.2. Practice

Many different clique-finding algorithms have been implemented, and an algorithm of Tomita et al. [2006], based on the much earlier algorithm of Bron and Kerbosch [1973], has been shown through many experiments to be faster by orders of magnitude in practice than the other algorithms that have been studied to date. An unfortunate drawback of the algorithm of Tomita et al., however, is that both its theoretical analysis and implementation rely on an adjacency matrix representation of the input graph. For this reason, their algorithm has limited applicability for large sparse graphs, whose adjacency matrix may not fit into working memory. We therefore seek to have the best of both worlds: we would ideally like an algorithm that rivals the speed of the Tomita et al. result, while having linear storage cost.

In Section 4 we describe an implementation of our new algorithm that has these ideal properties. Our algorithm uses only linear space, so it is usable on very large graphs. Its running time rivals and in many cases is faster than the running time of the Tomita et al. method. And, although it is never slower than Tomita et al. by more than a small factor, it is sometimes faster by a large factor, showing that it is more reliably fast than the previous algorithm.

1.3. Our Results

In this paper, we show that:

— A very simple variant of the Bron–Kerbosch algorithm, when applied to n-vertex graphs with degeneracy d, lists all maximal cliques in time $O(dn3^{d/3})$.
— For graphs with degeneracy d and maximum clique size κ, the same algorithm takes time $O(d^2n(d/\kappa)^\kappa)$.
— For graphs that do not have the h-vertex complete graph K_h as a minor, the same algorithm takes time $n2^{O(h \log\log h)}$, matching a bound of Fomin et al. [2010] for the number of cliques in these graphs.
— Our algorithm can be implemented using $O(n)$ space.
— Every n-vertex graph with degeneracy d (where d is a multiple of three and $n \geq d+3$) has at most $(n-d)3^{d/3}$ maximal cliques, and there exists an n-vertex graph with degeneracy d that has exactly $(n-d)3^{d/3}$ maximal cliques, showing that our algorithm is close to the optimal worst-case time as a function of n and d.
— Experimental results indicate that our algorithm is fast in practice: it rivals the speed of Tomita et al. [2006] on small graphs, and because it uses linear space, it can be run on sparse graphs with millions of vertices.

Our algorithms are fixed-parameter tractable, with running times of the form $O(f(d)n)$ where $f(d) = d3^{d/3}$. Algorithms for listing all maximal cliques in graphs of constant degeneracy in time $O(n)$ were known [Chiba and Nishizeki 1985; Chrobak and Eppstein 1991], but had not been analyzed for their dependence on degeneracy. We compare the parameterized running time bounds of these other algorithms to our variant of the Bron–Kerbosch algorithm, and we show that the Bron–Kerbosch algorithm has a much smaller dependence on d.

2. PRELIMINARIES

We work with an undirected graph $G = (V, E)$, which we assume is stored in an adjacency list data structure. We let n and m be the number of vertices and edges of G, respectively. For a vertex v, we define $\Gamma(v)$ to be the set $\{w \mid (v, w) \in E\}$, which we call

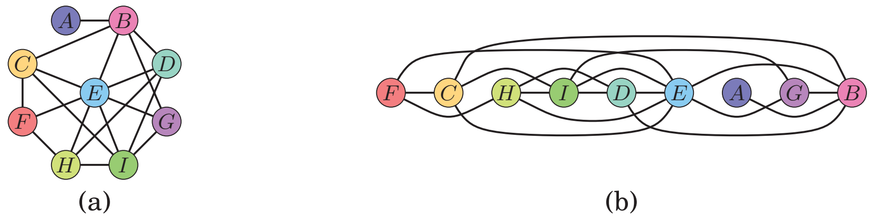

Fig. 2. (a) A graph with degeneracy 3. The clique consisting of vertices D, E, H, and I shows that it cannot have degeneracy smaller than 3. (b) A vertex ordering showing that the degeneracy is not larger than 3.

the *neighborhood* of v, and similarly for a subset $W \subset V$ we define $\Gamma(W)$ to be the set $\bigcap_{w \in W} \Gamma(w)$, which is the common neighborhood of all vertices in W.

2.1. Degeneracy

Our algorithm is parameterized by the *degeneracy* of a graph, a measure of its sparsity.

The degeneracy of a graph G is the smallest value d such that every nonempty subgraph of G contains a vertex of degree at most d [Lick and White 1970].

Figure 2(a) shows an example of a graph of degeneracy 3. Degeneracy is also known as the k-core number [Batagelj and Zaveršnik 2003], width [Freuder 1982], and linkage [Kirousis and Thilikos 1996] of a graph and is one less than the coloring number [Erdős and Hajnal 1966]. In a graph of degeneracy d, the maximum clique size can be at most $d+1$, for any larger clique would form a subgraph in which all vertices have degree higher than d.

A graph of degeneracy d has a *degeneracy ordering*, an ordering such that each vertex has d or fewer neighbors that come later in the ordering (Figure 2(b)). Such an ordering may be formed by repeatedly removing a vertex of degree d or less: by the assumption that G is d-degenerate, at least one such vertex exists at each step. Conversely, if G has an ordering with this property, then it is d-degenerate, because for any subgraph H of G, the vertex of H that comes first in the ordering has d or fewer neighbors in H. Thus, as Lick and White [1970] showed, degeneracy may equivalently be defined as the minimum d for which a degeneracy ordering exists. A third, equivalent definition is that d is the minimum value for which G has an orientation as a directed acyclic graph in which all vertices have out-degree at most d [Chrobak and Eppstein 1991]: such an orientation may be found by orienting each edge from its earlier endpoint to its later endpoint in a degeneracy ordering, and if such an orientation is given, then a degeneracy ordering may be found by topologically ordering the oriented graph.

Degeneracy is a robust measure of sparsity: it is within a constant factor of other popular measures of sparsity including arboricity and thickness. In addition, degeneracy, along with a degeneracy ordering, can be computed by a simple greedy strategy of repeatedly removing a vertex with smallest degree (and its incident edges) from the graph until it is empty [Batagelj and Zaveršnik 2003]. The degeneracy is the maximum of the degrees of the vertices at the time they are removed from the graph, and the degeneracy ordering is the order in which vertices are removed from the graph [Jensen and Toft 1995]. The easy computation of degeneracy has made it a useful tool in algorithm design and analysis [Chrobak and Eppstein 1991; Eppstein 2009].

We can implement the greedy algorithm for finding a degeneracy ordering in time $O(n+m)$ [Batagelj and Zaveršnik 2003]. To do so, maintain an array D, where $D[i]$ stores a doubly-linked list of vertices that have exactly i neighbors among the graph vertices that have not yet been included in the greedy ordering; initially $D[i]$ stores a list of the vertices with degree i. To remove a vertex of minimum degree from the graph, scan D starting at $D[0]$ until reaching the first nonempty list $D[i]$. Remove any vertex v from $D[i]$, and move each neighbor w of v from $D[j]$ to $D[j-1]$, where j is the number of unlisted neighbors of w prior to including v in the ordering. Each vertex removal step takes time proportional to the degree of the removed vertex, so the overall algorithm takes linear time.

We can also use degeneracy to bound the total number of edges in the graph. If we sum, for each vertex v, the number of neighbors of v that are later in the ordering, then each edge is counted once, so the value of the sum is just m. However, for a vertex v at position i in the degeneracy, the number of later neighbors of v can be at most $\min(d, n-i)$. Adding this quantity over all possible positions, we get the following bound on the number of edges of a d-degenerate graph:

LEMMA 2.1 (PROPOSITION 3 OF [LICK AND WHITE 1970]). *A graph $G = (V, E)$ with degeneracy d has at most $d(n - \frac{d+1}{2})$ edges.*

2.2. The Bron–Kerbosch Algorithm

The Bron–Kerbosch algorithm [Bron and Kerbosch 1973] is a recursive backtracking algorithm for listing all maximal cliques that is easy to understand, easy to code, widely used, and works well in practice. A recursive call to the Bron–Kerbosch algorithm provides three disjoint sets of vertices, R, P, and X, as arguments, where R is a (possibly non-maximal) clique and $P \cup X = \Gamma(R)$ form a partition of the vertices that are adjacent to every vertex in R. The vertices in P will be considered to be added to clique R, while those in X must be excluded from the clique; thus, within the recursive call, the algorithm lists all cliques in $P \cup R$ that are maximal within the subgraph induced by $P \cup R \cup X$. The algorithm chooses a candidate v in P to add to the clique R, and makes a recursive call in which v has been moved from R to P; in this recursive call, it restricts X to the neighbors of v, since non-neighbors cannot affect the maximality of the resulting cliques. When the recursive call returns, v is moved to X to eliminate redundant work by further calls to the algorithm. When the recursion reaches a level at which P and X are empty, R is a maximal clique and is reported (see the pseudocode in Fig. 3). To list all maximal cliques in the graph, this recursive algorithm is called with P equal to the set of all vertices in the graph and with R and X empty.

Bron and Kerbosch also describe a heuristic called *pivoting*, which limits the number of recursive calls made by their algorithm. The key observation is that for any vertex u in $P \cup X$, called a *pivot*, any maximal clique must contain one of u's non-neighbors (counting u itself as a non-neighbor). Therefore, we delay the vertices in $P \cap \Gamma(u)$ from being added to the clique until future recursive calls, with the benefit that we make fewer recursive calls. Tomita et al. [2006] show that choosing the pivot u from $P \cup X$ that maximizes $|P \cap \Gamma(u)|$ ensures that the Bron–Kerbosch algorithm has worst-case running time $O(3^{n/3})$, excluding the time to write the output. This is worst-case optimal, as there exist graphs with $3^{n/3}$ maximal cliques [Moon and Moser 1965].

3. THEORETICAL RESULTS

In this section, we show that as well as the pivoting strategy, the order in which the vertices of G are processed by the Bron–Kerbosch algorithm is also important. We develop a variant of the Bron–Kerbosch algorithm that chooses this ordering carefully and correctly lists all maximal cliques in time $O(dn3^{d/3})$. Our algorithm performs the outer level of recursion of the Bron–Kerbosch algorithm without pivoting, using a degeneracy ordering to order the sequence of recursive calls made at this level, and then switches at inner levels of recursion to the pivoting rule of Tomita et al. [2006].

We also show that this performance is optimal in terms of the degeneracy, and that the dependence on the degeneracy is better than other algorithms that are designed for clique finding in sparse graphs.

```
proc BronKerbosch($P$, $R$, $X$)
1: if $P \cup X = \emptyset$ then
2:     report $R$ as a maximal clique
3: end if
4: for each vertex $v \in P$ do
5:     BronKerbosch($P \cap \Gamma(v)$, $R \cup \{v\}$, $X \cap \Gamma(v)$)
6:     $P \leftarrow P \setminus \{v\}$
7:     $X \leftarrow X \cup \{v\}$
8: end for
```

```
proc BronKerboschPivot($P$, $R$, $X$)
1: if $P \cup X = \emptyset$ then
2:     report $R$ as a maximal clique
3: end if
4: choose a pivot $u \in P \cup X$ {Tomita et al. choose $u$ to maximize $|P \cap \Gamma(u)|$}
5: for each vertex $v \in P \setminus \Gamma(u)$ do
6:     BronKerboschPivot($P \cap \Gamma(v)$, $R \cup \{v\}$, $X \cap \Gamma(v)$)
7:     $P \leftarrow P \setminus \{v\}$
8:     $X \leftarrow X \cup \{v\}$
9: end for
```

Fig. 3. The Bron–Kerbosch algorithm without and with pivoting

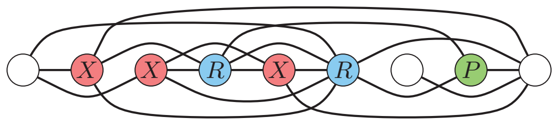

Fig. 4. Partitioning the common neighbors of a clique R into the set P of later vertices and the set X of remaining neighbors.

3.1. High-level Description of the Algorithm

Each recursive call of the original Bron–Kerbosch algorithm considers for expansion the vertices in P, one by one in an unspecified order (see line 4 of BronKerbosch in Figure 3). We first analyze what happens if we fix an order $v_1, v_2, \ldots, v_n$ on the vertices of V, and use the same order consistently to loop through the vertices of P in each recursive call of the non-pivoting version of the algorithm.

OBSERVATION 1. *When processing a clique R in the ordered variant of Bron–Kerbosch, the common neighbors of R can be partitioned into the set P of vertices that come after the last vertex of R, and the set X of remaining neighbors (Figure 4).*

PROOF. This follows easily by induction on the number of levels of recursive calls made by the algorithm and the number of steps made within each call. At each step, when we add a new vertex v to R, we restrict P and X to be intersection of its former value with the common neighbors of v, so that (in recursive calls made with v in R) the sets P and X continue to form a partition of the common neighbors of R. By the induction hypothesis, the new vertex v is the last vertex of R, and is later in the ordering than the previous last vertex which is in turn later in the ordering than all vertices in X. By the choice of ordering, the new vertex v is earlier than all remaining vertices in P. Therefore, after adding v, it remains the case that the partition of the common neighbors of R into X and P is determined by the ordering of these common neighbors with respect to the last vertex v of R. □

Our algorithm computes a degeneracy ordering of the given graph, and performs the outermost recursive calls in the ordered variant of the Bron–Kerbosch algorithm (without pivoting) for this ordering. The sets P passed to each of these recursive calls will have at most d elements in them, leading to few recursive calls within each of these outer calls. Below the top level of the recursion we switch from the ordered non-pivoting version of the Bron–Kerbosch algorithm to the pivoting algorithm (with the same choice of pivots as Tomita et al. [2006]) to further control the number of recursive calls.

```
proc BronKerboschDegeneracy($V$, $E$)
1: for each vertex $v_i$ in a degeneracy ordering $v_0, v_1, v_2, \dots$ of $(V, E)$ do
2:     $P \leftarrow \Gamma(v_i) \cap \{v_{i+1}, \dots, v_{n-1}\}$
3:     $X \leftarrow \Gamma(v_i) \cap \{v_0, \dots, v_{i-1}\}$
4:     BronKerboschPivot($P$, $\{v_i\}$, $X$)
5: end for
```

Fig. 5. Our algorithm.

3.2. Analysis of the Algorithm

LEMMA 3.1. *The Bron–Kerbosch algorithm using the Tomita et al. pivoting strategy generates all and only maximal cliques containing all vertices in R, some vertices in P, and no vertices in X, without duplication.*

PROOF. See Tomita et al. [2006]. □

THEOREM 3.2. *Algorithm* BronKerboschDegeneracy *generates all and only maximal cliques without duplication.*

PROOF. Let C be a maximal clique, and v its earliest vertex in the degeneracy order. By Lemma 3.1, C will be reported (once) when processing v. When processing any other vertex of C, v will be in X, so C will not be reported. □

To make pivot selection fast we pass as an additional argument to BronKerboschPivot a subgraph $H_{P,X}$ of G that has $P \cup X$ as its vertices; an edge (u, v) of G is kept as an edge in $H_{P,X}$ whenever at least one of u or v belongs to P and both of them belong to $P \cup X$. The pivot chosen according to the pivot rule of Tomita et al. [2006] is then just the vertex in this graph with the most neighbors in P.

LEMMA 3.3. *Whenever* BronKerboschDegeneracy *calls* BronKerboschPivot *it can form $H_{P,X}$ in time $O(d(|P| + |X|))$.*

PROOF. The vertex set of $H_{P,X}$ is known from P and X. Each edge is among the d outgoing edges from each of its vertices. Therefore, we can achieve the stated time bound by looping over each of the d outgoing edges from each vertex in $P \cup X$ and testing whether each edge meets the criterion for inclusion in $H_{P,X}$. □

The factor of d in the time bound of Lemma 3.3 makes it too slow for the recursive part of our algorithm. Instead we show that the subgraph to be passed to each recursive call can be computed quickly from the subgraph given to its parent in the recursion tree.

LEMMA 3.4. *In a recursive call to* BronKerboschPivot *that is passed the graph $H_{P,X}$ as an auxiliary argument, the sequence of graphs $H_{P\cap\Gamma(v),X\cap\Gamma(v)}$ to be passed to lower-level recursive calls can be computed in total time $O(|P|^2(|P| + |X|))$.*

PROOF. It takes $O(|P| + |X|)$ time to identify the subsets $P \cap \Gamma(v)$ and $X \cap \Gamma(v)$ by examining the neighbors of v in $H_{P,X}$. Once these sets are identified, $H_{P\cap\Gamma(v),X\cap\Gamma(v)}$ may be constructed as a subgraph of $H_{P,X}$ in time $O(|P|(|P| + |X|))$ by testing for each edge of $H_{P,X}$ whether its endpoints belong to these sets. There are $O(|P|)$ graphs to construct, one for each recursive call, hence the total time bound. $\square$

LEMMA 3.5 (THEOREM 3 OF [TOMITA ET AL. 2006]). *Let T be a function which satisfies the following recurrence relation:*

$$T(p) \leq \begin{cases} \max_k\{kT(p-k)\} + dp^2 & \textit{if } p > 0 \\ e & \textit{if } p = 0 \end{cases}$$

where p and k are integers, such that $p \geq k$, and d, e are constants greater than zero. Then, $T(p) \leq \max_k\{kT(p-k)\} + dp^2 = O(3^{p/3})$.

LEMMA 3.6. *Let v be a vertex, P_v, be v's later neighbors, and X_v be v's earlier neighbors. Then* BronKerboschPivot(P_v, $\{v\}$, X_v) *executes in time $O((d+|X_v|)3^{|P_v|/3})$, excluding the time to report the discovered maximal cliques.*

PROOF. Define $D(p,x)$ to be the running time of BronKerboschPivot(P_v, $\{v\}$, X_v), where $p = |P_v|$, and $x = |X_v|$. We show that $D(p,x) = O((d+x)3^{p/3})$. By the description of BronKerboschPivot, D satisfies the following recurrence relation:

$$D(p,x) \leq \begin{cases} \max_k\{kD(p-k,x)\} + c_1p^2(p+x) & \text{if } p > 0 \\ c_2 & \text{if } p = 0 \end{cases}$$

where c_1 and c_2 are constants greater than 0.
Since our graph has degeneracy d, the inequality $p + x \leq d + x$ always holds. Thus,

$$\begin{aligned} D(p,x) &\leq \max_k\{kD(p-k,x)\} + c_1p^2(p+x) \\ &\leq (d+x)\left(\max_k\left\{\frac{kD(p-k,x)}{d+x}\right\} + c_1p^2\right) \\ &\leq (d+x)\left(\max_k\{kT(p-k)\} + c_1p^2\right) \\ &= O((d+x)3^{p/3}) \quad \text{by letting } d = c_1, e = c_2 \text{ in Lemma 3.5} \end{aligned}$$

$\square$

THEOREM 3.7. *Given an n-vertex graph G with degeneracy d, our algorithm reports all maximal cliques of G in time $O(dn3^{d/3})$.*

PROOF. For each initial call to BronKerboschPivot for each vertex v, we first spend time $O(d(|P_v| + |X_v|))$ to set up subgraph H_{P_v,X_v}. Over the entire algorithm we spend time $\sum_v O(d(|P_v| + |X_v|)) = O(dm) = O(d^2n)$ setting up these subgraphs. The time spent performing the recursive calls is $\sum_v O((d + |X_v|)3^{|P_v|/3}) = O((dn + m)3^{d/3}) = O(dn3^{d/3})$, and the time to report all cliques is $O(d\mu)$, where μ is the number of maximal cliques. We show in the next section that $\mu = (n-d)3^{d/3}$ in the worst case, and therefore we take time $O(d(n-d)3^{d/3})$ reporting cliques in the worst case. Therefore, the algorithm executes in time $O(dn3^{d/3})$. $\square$

3.3. Worst-case Bounds on the Number of Maximal Cliques

We now show matching upper and lower bounds for the number of maximal cliques a graph with degeneracy d may have. In particular, we prove the following theorem:

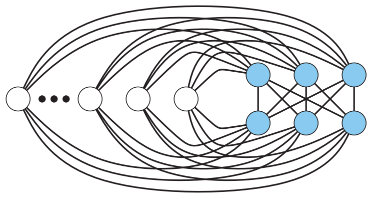

Fig. 6. The lower bound construction for $d = 6$, consisting of a Moon–Moser graph of size d on the right (blue vertices) and an independent set of $n - d$ remaining vertices that are each connected to all of the last d vertices.

THEOREM 3.8. *Let d be a multiple of 3 and $n \geq d + 3$. Then the largest possible number of maximal cliques in an n-vertex graph with degeneracy d is $(n - d)3^{d/3}$.*

PROOF. We first show that there cannot be more than $(n - d)3^{d/3}$ maximal cliques. We then show that there exists a graph that has $(n - d)3^{d/3}$ maximal cliques.

An Upper Bound. Consider a degeneracy ordering of the vertices, in which each vertex has at most d neighbors that come later in the ordering. For each vertex v that is placed among the first $n-d-3$ vertices of the degeneracy ordering, we count the number of maximal cliques such that v is the clique vertex that comes first in the ordering. Since vertex v has at most d later neighbors, the Moon–Moser bound [Moon and Moser 1965] applied to the subgraph induced by these later neighbors shows that they can form at most $3^{d/3}$ maximal cliques with each other. Since these vertices are all neighbors of v, v participates in at most $3^{d/3}$ maximal cliques with its later neighbors. Thus, the first $n - d - 3$ vertices contribute to at most $(n - d - 3)3^{d/3}$ maximal cliques total. By the Moon–Moser bound, the remaining $d + 3$ vertices in the ordering can form at most $3^{(d+3)/3}$ maximal cliques. Therefore, a graph with degeneracy d can have no more than $(n - d - 3)3^{d/3} + 3^{(d+3)/3} = (n - d)3^{d/3}$ maximal cliques.

A Lower Bound. By a simple counting argument, we can see that the graph $K_{n-d,3,3,3,3,\ldots}$ contains $(n - d)3^{d/3}$ maximal cliques: Each maximal clique must contain exactly one vertex from each disjoint independent set of vertices, and there are $(n - d)3^{d/3}$ ways of forming a maximal clique by choosing one vertex from each independent set. Figure 6 shows this construction for $d = 6$. This graph's degeneracy is at least d because, in any ordering of the vertices, the first vertex must have d or more later neighbors. Its degeneracy is at most d because, in any ordering where the $n - d$ disjoint vertices come first, these first $n-d$ vertices have exactly d later neighbors, and the last d vertices have fewer later neighbors. Thus, its degeneracy is exactly d. □

Relatedly, a bound of $(n - d + 1)2^d$ on the number of cliques (without assumption of maximality) in n-vertex d-degenerate graphs was already known [Wood 2007].

3.4. Optimal Running Time?

The $O(dn3^{d/3})$ running time of algorithm BronKerboschDegeneracy described above is nearly worst-case optimal, since there may be $\Theta((n - d)3^{d/3})$ maximal cliques, each of size d, in the worst case, so we already need to spend $O(d(n - d)3^{d/3})$ just on reporting the output. This means our algorithm is optimal as long as $n - d \in O(n)$, which is a very mild requirement as we expect in practice that $d \ll n$ on sparse graphs.

However, we will show that by a simple modification, we can in fact obtain an algorithm that runs in time $O(d(n - d)3^{d/3})$, which matches the worst-case output size for all values of d.

This new algorithm, which we'll call BronKerboschDegeneracy2, is divided into two phases. In the first phase, we run BronKerboschDegeneracy, which we allow to compute the degeneracy ordering and we stop its execution once it has processed the first $n-d$ vertices in the ordering. This takes time

$$\sum_{v_1..v_{n-d}} (d+|X_{v_i}|)3^{d/3} = O(d(n-d)3^{d/3}),$$

and gives us all maximal cliques containing the first $n-d$ vertices.

In the second phase, we compute the subgraph D of G induced by the last d vertices in the ordering. We then run the algorithm by Tomita et al. [2006] on D, obtaining at most $3^{d/3}$ cliques, which are maximal with respect to D, but not necessarily maximal with respect to the entire graph. Therefore, we check each clique C computed in this phase for maximality, by testing if any vertex v in the first $n-d$ vertices in the ordering, can be added to C to form a larger clique. If no such vertex exists, then C is maximal.

We can test if a clique is maximal in time $O(d(n-d))$ by a simple marking scheme: We first mark all the vertices in the clique C, and then we iterate over the first $n-d$ vertices and test if $|C|$ later neighbors are marked, we then unmark the vertices in C. Therefore, we can test if all cliques found in this phase are maximal in time $O(d(n-d)3^{d/3})$, and furthermore list only those cliques which are maximal. Once we have completed both phases, all maximal cliques have been found, hence, the algorithm takes $O(d(n-d)3^{d/3})$ time overall.

THEOREM 3.9. BronKerboschDegeneracy2 *lists all maximal cliques in an n-vertex graph with degeneracy d in time $O(d(n-d)3^{d/3})$.*

Though this algorithm is theoretically faster for large values of d, we caution that it is more complex to implement and will likely not lead to speedup in practice. The main difference between this algorithm and BronKerboschPivot is that, for the final d vertices in the degeneracy ordering, it avoids passing the correct set X down through the recursive calls and instead, for each generated clique, iterates through the entire graph to test for maximality of the clique. This final set of vertices is likely to constitute a negligible fraction of the total time for the algorithm, and iterating through the entire graph for each generated clique is unlikely to be cheaper than passing the correct value of X. Therefore, we consider this variant to be of theoretical interest only, and we did not test it in our experimental results.

3.5. Parameterizing by both degeneracy and clique size

As we observed in the introduction, the maximum clique problem, parameterized by clique size, is hard for $\mathbf{W}[1]$, implying that it is unlikely to have a fixed-parameter tractable algorithm [Downey and Fellows 1995]. However, we may parameterize by both degeneracy and clique size, obtaining stronger bounds than we could from degeneracy alone. In our analysis below, we let κ denote the clique size, the number of vertices in the largest clique of the given graph.

LEMMA 3.10. *Let $T(x,p,k)$ be the worst-case time for BronKerboschPivot to run, given arguments X of size x and P of size p, and given additionally a guarantee that the largest clique in P has size $k \leq p/3$. Then $T(x,p,k) = O(p(p+x)(p/k)^k)$.*

PROOF. For $k=1$, the algorithm takes time $O(p^2x)$ to compute the pivot and to set up the recursive calls to the algorithm. The assumption that the largest clique in P has size one implies that there are no edges between pairs of vertices in P, so each recursive call returns immediately, and the total time for the algorithm is $O(p^2x) = O(p(p+x)(p/k)^k)$.

For $k>1$, the time bounds for the algorithm obey a recurrence of the form

$$T(x,p,k) \leq \max_j jT(x,p-j,k-1) + O(p^2x).$$

A straightforward induction on k shows that there exists a constant c such that $T(x,p,k) \le cp(p+x)(p/k)^k$. For, applying the induction hypothesis and expanding the instances of T on the right hand side of the recurrence would give us

$$T(x,p,k) \le \max_j jc(p-j)(p-j+x)\left(\frac{p-j}{k-1}\right)^{k-1} + O(p^2x).$$

When $j = p/k$, the j and $((p-j)/(k-1))^{k-1}$ terms of this expansion combine to give $(p/k)^k$, the $p-j+x$ term can be replaced by $p+x$ while still giving a valid upper bound, and the $p-j$ term is smaller than the factor of p that we need by a factor of $(1-1/k)$, small enough to cancel the $O(p^2x)$ term in the recurrence when c is a sufficiently small constant. Similarly, when $j = \Theta(p/k)$, the j and $((p-j)/(k-1))^{k-1}$ terms still combine to give something that is at most $(p/k)^k$, and the $p-j$ is smaller than p by a factor of $(1-1/O(k))$, still small enough to cancel the $O(p^2x)$ term. When j is outside this range, the combination of the j and $((p-j)/(k-1))^{k-1}$ terms is itself enough smaller than $(p/k)^k$ to again allow us to cancel the $O(p^2x)$ term. Thus, the result follows by induction. □

THEOREM 3.11. *BronKerboschDegeneracy, given a graph G with n vertices, degeneracy d, and maximum clique size $\kappa \le d/3$, takes total time $O(d^2n(d/\kappa)^\kappa)$.*

PROOF. We sum the bound of Lemma 3.10 over all the calls to BronKerboschPivot made in the outer call of BronKerboschDegeneracy. In each call, $p \le d$, $k \le \kappa$, and $p + x \le \delta_i$ where δ_i is the degree of vertex i. Therefore, this sum is

$$\sum_i O\left(d\delta_i\left(\frac{d}{\kappa}\right)^\kappa\right) = \left(\sum \delta_i\right) O\left(d\left(\frac{d}{\kappa}\right)^\kappa\right) = O\left(d^2n\left(\frac{d}{\kappa}\right)^\kappa\right).$$

□

To show that this result leads to nontrivial improvements compared to our $O(d3^{d/3}n)$ bound, we consider the case of *minor-closed graph families*, families of graphs closed under the operations of edge contraction and edge deletion. We say that such a family $\mathcal{F}$ is *nontrivial* if it does not include all graphs; in this case, we may let h be the minimum number of vertices in a graph that does not belong to $\mathcal{F}$. The *complete graph* K_h is an *excluded minor* for $\mathcal{F}$ and $\mathcal{F} \subset \mathcal{F}_h$ where $\mathcal{F}_h$ is the family of graphs that do not have K_h as a minor.

All graphs in $\mathcal{F}_h$ have degeneracy $O(h\sqrt{\log h})$, and this is tight [Kostochka 1984; Thomason 2001; 1984]. Therefore, our time bound for BronKerboschDegeneracy that uses only the degeneracy would show that it takes time $n2^{O(h\sqrt{\log h})}$ on graphs in $\mathcal{F}_h$. However, if we use the observation that the maximum clique size κ on graphs in $\mathcal{F}_h$ is at most h, we can instead apply Theorem 3.11, giving us a tighter bound of $n2^{O(h \log\log h)}$ on the running time of BronKerboschDegeneracy. This time bound matches an upper bound of $n2^{O(h \log\log h)}$ on the number of cliques in these graphs given by Fomin et al. [2010], and provides an alternative proof for their bound.

3.6. Comparison with Other FPT Algorithms

Our algorithm is not the first to list the maximal cliques of sparse graphs, and algorithms that run in linear time on graphs of constant degeneracy were already known. Essentially, although their analysis was not necessarily phrased in these terms, these previous algorithms are fixed-parameter tractable algorithms. However, the dependence of these algorithms on the degeneracy parameter has not always been explicitly described, so we now review these existing algorithms and show that our algorithm's dependence on d is superior.

Chiba and Nishizeki. Chiba and Nishizeki [1985] describe two algorithms for finding cliques in sparse graphs. One of the two reports all maximal cliques using $O(am)$ time per clique, where a is the arboricity of the graph, and m is the number of edges in G. The *arboricity* is the minimum number of edge-disjoint spanning forests into which the graph can be decomposed [Harary 1972]. The degeneracy of a graph is closely related to arboricity, being always within a factor of two of each other: If a graph G has arboricity a, then any subgraph of G with k vertices has at most $a(k-1)$ edges, average degree $2a(k-1)/k < 2a$, and minimum degree at most $2a-1$. In the other direction, if G has degeneracy d, then we may partition it into d forests by finding a degeneracy orientation of G and, at each vertex v, assigning each of the at most d outgoing edges to a different forest.

In terms of degeneracy, Chiba and Nishizeki's algorithm uses $O(d^2n)$ time per clique. Combining this with the bound on the number of cliques derived in Section 3.3 results in a worst-case time bound of $O(d^2n(n-d)3^{d/3})$. For constant d, this is a quadratic time bound, in contrast to the linear time of our algorithm.

Another algorithm of Chiba and Nishizeki [1985] lists cliques that have l vertices in time $O(la^{l-2}m)$. It can be adapted to enumerate all maximal cliques in a graph with degeneracy d by first enumerating all cliques of order $d+1, d, \ldots$ down to 1, and then removing the cliques that are not maximal. Applying their algorithm directly to a d-degenerate graph takes time $O(ld^{l-1}n)$. Therefore, the running time to find all maximal cliques is $\sum_{1\leq i\leq d+1} O(ind^{i-1}) = O(nd^{d+1})$. Like our algorithm, this is linear when d is constant, but with a much worse dependence on the parameter d.

Chrobak and Eppstein. Chrobak and Eppstein [1991] list triangles and 4-cliques in graphs of bounded degeneracy by testing all sets of two or three later neighbors of each vertex according to a degeneracy ordering. The same idea extends to maximal cliques of size greater than four, by testing all subsets of later neighbors of each vertex. For each vertex v, there are at most 2^d subsets to test; each subset may be tested for being a clique in time $O(d^2)$, by checking whether each of its vertices has all the later vertices in the subset among its later neighbors, giving a total time of $O(nd^2 2^d)$ to list all the cliques in the graph. However, although this singly-exponential time bound is considerably faster than Chiba and Nishizeki, and is close to known bounds on the number of (possibly non-maximal) cliques in d-degenerate graphs [Wood 2007], it is slower than our algorithm by a factor that is exponential in d. Our new algorithm uses this same idea of searching among the later neighbors in a degeneracy order but achieves much greater efficiency by combining it with the Bron–Kerbosch algorithm.

Makino and Uno. Makino and Uno [2004] list all maximal cliques in graphs with maximum degree Δ in time $O(\Delta^4)$ per clique. For graphs with degeneracy d, Δ can be any value between d and $n-1$, making a meaningful comparison with our algorithm difficult. Therefore, for graphs with constant degeneracy, their time bound is a factor Δ^4 slower than our algorithm in the worst case, which may be a constant factor, or much worse. A graph with maximum degree Δ can have no more than $(n-\Delta)3^{\Delta/3}$ maximal cliques, and an analysis similar to that of the previous section shows that this algorithm has a run time of $O(\Delta^4(n-\Delta)3^{\Delta/3})$, which is fixed-parameter tractable when parameterized on Δ. However, this bound is always worse than the bound for our algorithm by a factor of at least Δ^3, and possibly by a significantly larger factor in the case that Δ is significantly larger than d.

As Abu-Khzam et al. [2006] observed, the complementary relation between cliques and vertex covers allows fixed-parameter tractable algorithms for vertex cover to be applied to the clique problem. However, it is only possible to use this approach for cliques much larger than the ones that exist in sparse graphs. Eblen et al. [2011] apply fixed-parameter analysis to algorithms for a different but closely related problem, the enumeration of maximum cliques.

4. IMPLEMENTATION AND EXPERIMENTAL ANALYSIS

In this section, we complement our theoretical results by an experimental comparison of the running time of our algorithm against the state of the art, on several data sets of real-world graphs, as well as some randomly generated graphs.

4.1. Experimental Setup

We review the algorithms we implemented, as well as the data sets we used.

4.1.1. Algorithms Tested. Many algorithms have been designed to work quickly on sparse graphs, according to their theoretical analysis; however, the algorithm of Tomita et al. [2006] is orders of magnitude faster than these algorithms in practice, as shown in the experimental results of Tomita et al.. However, there is an unfortunate drawback to this algorithm: both its theoretical analysis and implementation rely on the adjacency matrix representation of the input graph, as we noted in the introduction. For this reason, their algorithm has limited applicability for large sparse graphs, whose adjacency matrix may be too large to fit into working memory. To ensure a fair comparison, we also implemented a straightforward adaptation of their algorithm that uses an adjacency list representation instead; see below for details.

We implemented three variants of our algorithm:

— an implementation with no data structuring, which uses only the degeneracy order to compute pivots and vertex sets for recursive calls,

— an implementation of BronKerboschDegeneracy that only uses $O(m + n)$ space overhead for the data structure, and

— an alternative implementation of the data structure using bit vectors to represent the sets of neighbors of each active vertex.

However, the bit vector implementation executed no faster than the data structure implementation in our tests, so we omit its experimental timings and any detailed discussion of its implementation.

In more detail, the following four implementations appear in our tables of results:

Tomita et al. (TTT-classic). First off, we implemented the classical algorithm by Tomita, Tanaka and Takahashi, using an adjacency matrix to store the graphs.

Tomita et al. with adjacency lists (TTT-lists). The algorithm of Tomita et al. turns out to be fixed-parameter tractable when parameterized by Δ, the maximum degree, if we store the graph in the adjacency list representation[1], and if are careful with how we store the sets X and P and compute pivots. First, we note that it is possible to store the sets X and P so that we can check in constant time which set a vertex belongs to (we elaborate how this can be done in the discussion of the ELS-array algorithm). Then computing a pivot can be done in time $O(\Delta(|X| + |P|))$ by iterating over all vertices in $P \cup X$ and testing all neighbors for membership in P. Furthermore, when a vertex v is added to R for a recursive call, we can intersect the neighborhood of v with P and X in $O(\Delta)$ time, by iterating over its at most Δ neighbors and testing for membership in P or X. Thus, preparing subsets for all recursive calls takes time $O(|P|\Delta)$. Finally, we note that the pivot at the top level of recursion intersects Δ neighbors in P, and therefore, we only make $(n - \Delta)$ recursive calls at the top level.

Fitting these facts into the analysis of Tomita et al. shows that the running time of this algorithm is $O(\Delta(n - \Delta)3^{\Delta/3})$. We note that Δ may be significantly larger than

[1]In our experiments, we use an array instead of a linked list to store the neighbors of a vertex.

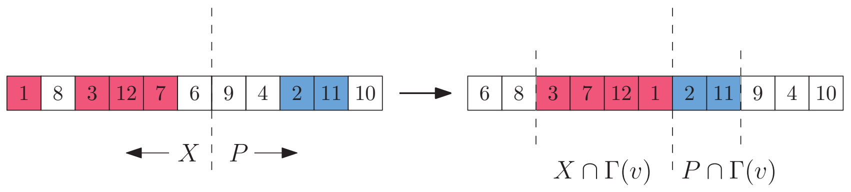

Fig. 7. When a vertex v is added to the partial clique R, its neighbors in P and X (highlighted in this example) are moved toward the dividing line in preparation for the next recursive call.

the degeneracy, so this algorithm's theoretical time bounds are not as good as those of Tomita et al. or ours; nevertheless, as we show in the next section, the simplicity of this algorithm makes it competitive in practice.

Our algorithm with no data structure (ELS-bare). It is possible to implement BronKerboschDegeneracy with only the degeneracy ordering, and no extra data structuring, only passing lists of vertices as arguments in each recursive call. The slowest part of each recursive call, and therefore the bottleneck of the algorithm, is the step in which the pivot vertex is selected. A simple strategy for determining each pivot would be to loop over all the possible pivots in $X \cup P$ and, for each one, loop over its later neighbors in the degeneracy ordering to determine how many of them are in P. The same strategy can also be used to perform the neighborhood intersection required to set up the arguments for the recursive calls. With the pivot selection and set intersection algorithms implemented in this way, the algorithm would have running time $O(d^2n3^{d/3})$, a factor of d larger than the worst-case output size and a factor of d slower than the more highly optimized version of our algorithm. However, the simplicity of this version of the algorithm may benefit it compared to the other algorithms.

Our algorithm with linear size data structure (ELS-array). To obtain our claimed $O(dn3^{d/3})$ worst-case time bound in a practical implementation, we maintain the sets of vertices P and X in a single array, the address to which is passed between recursive calls. Initially, the array contains the elements of X followed by the elements of P. We keep a reverse lookup table, so that in constant time we can look up the position of each vertex in this array. With this lookup table, we can tell whether a vertex is in P or X in constant time, by comparing its index to the position in the array where the two subarrays for X and P are separated from each other. When a vertex v is added to R in preparation for a recursive call, we reorder the array. Vertices in $\Gamma(v) \cap X$ are moved to the end of the X subarray, and vertices in $\Gamma(v) \cap P$ are moved to the beginning of the P subarray (see Figure 7). We then make a recursive call on the subarray containing the vertices $\Gamma(v) \cap (X \cup P)$. After the recursive call, we move v to X by swapping it to the beginning of the P subarray and moving the boundary so that v is in the X subarray. Of course, moving vertices between sets will affect P and X in higher recursive calls. Therefore, within a given recursive call, we maintain a separate list of the vertices that are moved from P to X, and we move these vertices back to P when the call ends.

To perform pivots quickly, our algorithm uses a modified adjacency list representation of a subgraph formed by the vertices in $X \cup P$: a set of arrays, one for each potential pivot vertex u in $P \cup X$, contains the neighbors of u in P. Whenever P changes, we reorder the elements in these arrays so that neighbors in the new set P are stored earlier than the other neighbors (see Figure 8). Computing the pivot is as simple as iterating through each array until we encounter a neighbor that is not in P. This reordering procedure allows us to maintain a single set of these arrays throughout all recursive calls, requiring linear space in total. In comparison, making a new copy of this data structure for each recursive call might require space $O(dm)$.

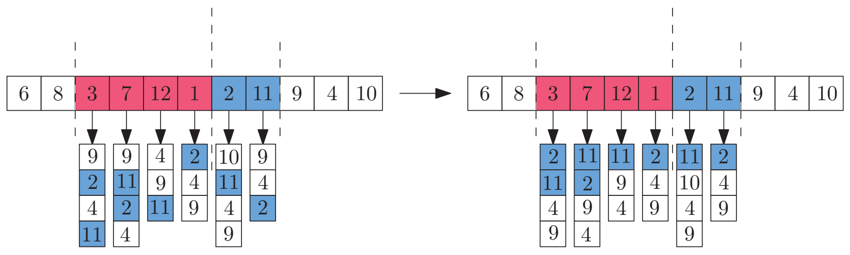

Fig. 8. For each vertex in $P \cup X$, we keep an array containing neighbors in P. We update these arrays whenever a vertex is moved from P to R, and whenever we need to intersect a neighborhood with P and X for a recursive call.

4.1.2. Data sets used. Our primary experimental data consisted of four publicly-available databases of real-world networks, including non-electronic and electronic social networks as well as networks from bioinformatics applications. As a reference point, we also ran our experimental comparisons using the two sets of graphs that Tomita et al. used in their experiments.

Mark Newman database. This is a database curated by Newman [2006] which consists primarily of social networks; it also includes word co-occurrence data and a biological neural network.

BioGRID. The Biological General Repository for Interaction Datasets (BioGRID) data [Stark et al. 2006], version 3.0.65, consists of several protein-protein interaction networks with from one to several thousand vertices, and varying sparsities.

Pajek data set. We tested six large social and bibliographic networks that appeared in the Pajek data set but were not in the other data sets [Batagelj and Mrvar 2006].

Stanford collection. We tested a representative sample of graphs from the Stanford Large Network Dataset Collection [Leskovec 2009]. These included road networks, a co-purchasing network from Amazon.com data, social networks, email networks, a citation network, and two Web graphs.

DIMACS. Tomita et al. used a data set from a DIMACS challenge [Johnson and Trick 1996], a collection of graphs that were intended as difficult examples for clique-finding algorithms, and that have been algorithmically generated. Although these graphs are more dense than our algorithm was designed for, we also tested our algorithm on the same data set.

Random graphs. Tomita et al. also generated graphs randomly with varying edge densities, most quite dense in comparison to the intended design regime for our algorithm. We did not have access to the data set they used, so to replicate their results we generated another set of random graphs with the same parameters.

4.1.3. Technical Specifications. We implemented all algorithms in the C programming language, and ran experiments on a Linux workstation running the 32-bit version of Ubuntu 10.10, with a 2.53 GHz Intel Core i5 M460 processor (with three cache levels of 128KB, 512KB, and 3,072KB respectively) and 2.6GB of memory. We compiled our code with version 4.4.5 of the gcc compiler with the `-O2` optimization flag. We have released our code under the GNU GPL 3.0 license, and we freely provide the code and data sets upon request.

4.2. Experimental Results

Tables I, II, III, IV, V, and VI list the results of our experiments. For each graph in a given data set we ran the four algorithms TTT-classic, TTT-lists, ELS-bare and ELS-array, and we provide the elapsed running times (in seconds) for each of these algorithms; an asterisk indicates that the algorithm was unable to run on that problem instance due to space limitations. In addition, we list the number of vertices n, edges m, degeneracy d, maximum degree Δ, and number of maximal cliques μ. To aid analysis, we have listed the graphs in Tables I to IV in order by degeneracy.

Table I. Experimental results for Mark Newman's data sets.

graph	n	m	d	Δ	μ	TTT-classic	TTT-lists	ELS-bare	ELS-array
zachary	34	78	4	17	39	< 0.01	< 0.01	< 0.01	< 0.01
dolphins	62	159	4	12	84	< 0.01	< 0.01	< 0.01	< 0.01
power	4,941	6,594	5	19	5,687	0.29	< 0.01	0.01	0.01
polbooks	105	441	6	25	199	< 0.01	< 0.01	< 0.01	< 0.01
adjnoun	112	425	6	49	303	< 0.01	< 0.01	< 0.01	< 0.01
football	115	613	8	12	281	< 0.01	< 0.01	< 0.01	< 0.01
lesmis	77	254	9	36	59	< 0.01	< 0.01	< 0.01	< 0.01
celegensneural	297	2,148	9	134	1,386	< 0.01	< 0.01	< 0.01	< 0.01
netscience	1,589	2,742	19	34	741	0.02	< 0.01	< 0.01	< 0.01
internet	22,963	48,436	25	2,390	39,288	6.68	0.28	0.11	0.11
condmat-2005	40,421	175,693	29	278	34,274	39.65	0.22	0.32	0.35
polblogs	1,490	16,715	36	351	49,884	0.08	0.28	0.18	0.12
astro-ph	16,706	121,251	56	360	15,794	3.44	0.19	0.22	0.23

Table II. Experimental results for BioGRID data sets (PPI Networks).

graph	n	m	d	Δ	μ	TTT-classic	TTT-lists	ELS-bare	ELS-array
mouse	1,455	1,636	6	111	1,523	0.01	< 0.01	< 0.01	< 0.01
worm	3,518	3,518	10	523	5,652	0.14	0.01	0.01	0.01
plant	1,745	3,098	12	71	2,302	0.02	< 0.01	< 0.01	< 0.01
fruitfly	7,282	24,894	12	176	21,995	0.62	0.03	0.03	0.04
human	9,527	31,182	12	308	23,863	1.06	0.03	0.05	0.05
fission-yeast	2,031	12,637	34	439	28,520	0.06	0.12	0.09	0.07
yeast	6,008	156,945	64	2557	738,613	1.74	11.37	4.22	2.17

Table III. Experimental results for Pajek data sets.

graph	n	m	d	Δ	μ	TTT-classic	TTT-lists	ELS-bare	ELS-array
foldoc	13,356	91,471	12	728	39,590	2.16	0.11	0.14	0.13
patents	240,547	560,943	24	212	482,538	*	0.56	1.22	1.65
eatRS	23,219	304,937	34	1,090	298,164	7.62	1.52	1.55	1.03
hep-th	27,240	341,923	37	2,411	446,823	12.56	3.40	2.40	1.70
days-all	13,308	148,035	73	2,265	2,173,772	5.83	62.86	9.94	5.18
ND-www	325,729	1,090,108	155	10,721	495,947	*	1.80	1.81	2.12

4.2.1. Interpretation of the results. The database by Newman [2006] (Table I) contains many graphs that were too small for us to time our algorithms accurately, but our algorithm was faster than that of Tomita et al. on all four of the largest graphs; in one case it was faster by a factor of approximately 130. On the BioGRID data (Table II), our algorithm was significantly faster than that of Tomita et al. on the worm and fruitfly networks, and matched or came close to its performance on all the other networks, even the relatively dense yeast network. Our algorithm was consistently faster on the networks in the Pajek data sets (Table III). Due to their large size, the algorithm of Tomita et al. was unable to run on two of these networks; nevertheless, our algorithm found all cliques quickly in these graphs. Finally, nearly all of the graphs in the Stanford Large Network Dataset Collection (Table IV) were too large for the Tomita et al.

Table IV. Experimental results for Stanford data sets.

graph	n	m	d	Δ	μ	TTT-classic	TTT-lists	ELS-bare	ELS-array
roadNet-CA	1,965,206	2,766,607	3	12	2,537,996	*	2.00	5.34	5.81
roadNet-PA	1,088,092	1,541,898	3	9	1,413,391	*	1.09	2.95	3.21
roadNet-TX	1,379,917	1,921,660	3	12	1,763,318	*	1.35	3.72	4.00
amazon0601	403,394	2,443,408	10	2,752	1,023,572	*	3.59	5.01	6.03
email-EuAll	265,214	364,481	37	7,636	377,956	*	4.93	1.25	1.33
email-Enron	36,692	183,831	43	1,383	226,859	31.96	2.78	1.30	0.90
web-Google	875,713	4,322,051	44	6,332	1,417,580	*	9.01	8.43	9.70
soc-wiki-Vote	7,115	100,762	53	1,065	459,002	0.96	4.21	2.10	1.14
soc-slashdot0902	82,168	504,230	55	2,552	890,041	*	7.81	4.20	2.58
cit-Patents	3,774,768	16,518,947	64	793	14,787,032	*	28.56	49.22	58.64
soc-Epinions1	75,888	405,740	67	3,044	1,775,074	*	27.87	9.24	4.78
soc-wiki-Talk	2,394,385	4,659,565	131	100,029	86,333,306	*	$> 18,000$	542.28	216.00
web-berkstan	685,231	6,649,470	201	84,230	3,405,813	*	76.90	31.81	20.87

Table V. Experimental results for Moon–Moser graphs and DIMACS benchmark graphs.

graph	n	m	d	Δ	μ	TTT-classic	TTT-lists	ELS-bare	ELS-array
M-M-30	30	405	27	27	59,049	0.04	0.04	0.06	0.04
M-M-45	45	945	42	42	14,348,907	7.50	15.11	20.36	10.21
M-M-48	48	1080	45	45	43,046,721	22.52	48.37	63.07	30.22
M-M-51	51	1224	48	48	129,140,163	67.28	150.02	198.06	91.80
MANN_a9	45	918	40	41	590,887	0.44	0.88	0.90	0.53
brock_200_2	200	9876	84	114	431,586	0.55	2.95	2.61	1.22
c-fat200-5	200	8473	83	86	7	0.01	0.01	0.01	0.01
c-fat500-10	500	46627	185	188	8	0.04	0.04	0.09	0.12
hamming6-2	64	1824	57	57	1,281,402	1.36	4.22	4.15	2.28
hamming6-4	64	704	22	22	464	< 0.01	< 0.01	< 0.01	< 0.01
johnson8-4-4	70	1855	53	53	114,690	0.13	0.35	0.40	0.24
johnson16-2-4	120	5460	91	91	2,027,025	5.97	27.05	31.04	12.17
keller4	171	9435	102	124	10,284,321	5.98	24.97	26.09	11.53
p_hat300-1	300	10933	49	132	58,176	0.07	0.29	0.25	0.15
p_hat300-2	300	21928	98	229	79,917,408	91.31	869.34	371.72	163.16

algorithm to fit into memory. For graphs which are extremely sparse, it is no surprise that the TTT-lists algorithm was faster than our algorithm, but our algorithm was consistently fast on each of these data sets, whereas the TTT-lists algorithm was orders of magnitude slower than our algorithm on the large soc-wiki-Talk network.

We also ran our comparisons using the two sets of graphs that Tomita et al. used in their experiments, the DIMACS challenge graphs (Table V) and a set of random graphs (Table VI). Our algorithm runs about 2 to 3 times slower than that of Tomita et al. on many of these graphs; this confirms that the algorithm is still competitive on graphs that are not sparse, in contrast to the competitors in Tomita et al.'s paper which ran 10 to 160 times slower on these input graphs. The largest of the random graphs in the second data set were generated with edge probabilities that made them significantly sparser than the rest of the set; for those graphs our algorithm outperformed that of Tomita et al. by a factor that was as large as 17 on the sparsest of the graphs. The TTT-lists algorithm was even faster than our algorithm in these cases, but it was significantly slower on other data.

4.2.2. A Remark About Degeneracy and Maximum Degree. Observe that the real-world graphs in Tables I to IV tend to have degeneracy that is significantly lower than the maximum degree. This experimentally verifies that degeneracy is a tighter measure of sparsity than the maximum degree, and we believe our experimental results demonstrate the power of degeneracy as a parameter when designing algorithms for such sparse real-world graphs.

Table VI. Experimental results on random graphs.

graph		d	Δ	μ	TTT-classic	TTT-lists	ELS-bare	ELS-array
n	p							
100	0.6	51	68	59,898	0.08	0.26	0.25	0.14
	0.7	59	79	439,928	0.50	2.04	1.85	0.99
	0.8	70	89	5,776,276	6.29	28.00	24.86	11.74
	0.9	81	97	240,998,654	249.15	1136.15	1028.84	425.85
300	0.1	21	41	3,663	< 0.01	0.01	0.01	< 0.01
	0.2	47	83	18,911	0.02	0.07	0.08	0.05
	0.3	74	112	86,179	0.10	0.44	0.49	0.24
	0.4	101	143	555,724	0.70	4.24	3.97	1.67
	0.5	130	173	4,151,668	5.59	42.37	36.35	13.05
	0.6	162	204	72,454,791	101.35	958.74	755.86	227.00
500	0.1	39	69	15,311	0.02	0.03	0.06	0.04
	0.2	81	127	98,875	0.11	0.46	0.61	0.27
	0.3	127	186	701,292	0.86	5.90	6.10	2.29
	0.5	225	291	103,686,974	151.67	1888.20	1521.90	375.23
700	0.1	56	97	38,139	0.04	0.10	0.19	0.09
	0.2	117	177	321,245	0.37	2.01	2.69	1.00
	0.3	184	246	3,107,208	4.06	36.13	38.12	11.47
1,000	0.1	82	132	99,561	0.11	0.34	0.70	0.28
	0.2	172	236	1,190,899	1.45	10.35	14.48	4.33
	0.3	266	349	15,671,489	21.96	262.64	280.58	66.05
2,000	0.1	170	247	750,991	1.05	5.18	11.77	3.13
3,000	0.1	263	355	2,886,628	4.23	27.51	68.52	13.62
10,000	0.001	7	24	49,716	1.19	0.04	0.07	0.07
	0.003	21	55	141,865	1.30	0.11	0.36	0.26
	0.005	38	78	215,477	1.47	0.25	1.03	0.51
	0.01	80	140	349,244	2.20	1.01	5.71	1.66
	0.03	262	360	3,733,699	9.96	20.66	133.94	20.67

5. CONCLUSION

We have presented theoretical evidence for the fast performance of the Bron–Kerbosch algorithm for finding cliques in graphs, as has been observed in practice. Our modified algorithm is fixed-parameter tractable in terms of the degeneracy of the graph, a parameter that is expected to be low in many real-world applications, and performs optimally in terms of this parameter. Furthermore, our experimental results show that our algorithm is efficient in practice for large sparse graphs. This algorithm is highly competitive with the algorithm of Tomita et al. on sparse graphs, and within a small constant factor on other graphs. The advantage of this algorithm is that it requires only linear space for storing the graph and all data structures. It does not suffer from the drawback of requiring an adjacency matrix, which may not fit into memory. Its closest competitor in this respect, the Tomita et al. algorithm modified to use adjacency lists, is sometimes faster by a small factor but is also sometimes slower by a large factor. Thus, our algorithm is a fast and reliable choice for listing maximal cliques, especially when the input graphs are large and sparse.

ACKNOWLEDGMENTS

We would like to thank Patrick Prosser for finding copyediting errors in our tables, and for finding an error with one of our data sets. Preliminary announcements of the research reported by this paper appeared as two conference papers, “Listing all maximal cliques in sparse graphs in near-optimal time”, by Eppstein, Löffler, and Strash, at the 21st International Symposium on Algorithms and Computation, Korea, 2010, and “Listing all maximal cliques in large sparse real-world graphs”, by Eppstein and Strash, at the 10th International Symposium on Experimental Algorithms, Crete, 2011.

REFERENCES

ABU-KHZAM, F. N., LANGSTON, M. A., SHANBHAG, P., AND SYMONS, C. T. 2006. Scalable parallel algorithms for FPT problems. *Algorithmica 45,* 3, 269–284.

AKKOYUNLU, E. A. 1973. The enumeration of maximal cliques of large graphs. *SIAM J. Comput. 2,* 1, 1–6.

ALON, N. AND GUTNER, S. 2009. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica 54,* 4, 544–556.

BARABÁSI, A.-L. AND ALBERT, R. 1999. Emergence of scaling in random networks. *Science 286*, 509–512.

BATAGELJ, V. AND MRVAR, A. 2006. Pajek datasets. `http://vlado.fmf.unilj.si/pub/networks/data/`.

BATAGELJ, V. AND ZAVERŠNIK, M. 2003. An $O(m)$ algorithm for cores decomposition of networks. Electronic preprint.

BRON, C. AND KERBOSCH, J. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM 16,* 9, 575–577.

CAI, L., CHAN, S. M., AND CHAN, S. O. 2006. Random separation: A new method for solving fixed-cardinality optimization problems. In *Proc. 2nd Int. Worksh. Parameterized and Exact Computation (IWPEC 2006)*. LNCS Series, vol. 4169. Springer-Verlag, 239–250.

CAZALS, F. AND KARANDE, C. 2008. A note on the problem of reporting maximal cliques. *Theor. Comput. Sci. 407,* 1-3, 564–568.

CHENG, J., KE, Y., FU, A. W.-C., YU, J. X., AND ZHU, L. 2010. Finding maximal cliques in massive networks by H*-graph. In *Proc. 2010 Int. Conf. on Management of Data (SIGMOD '10)*. 447–458.

CHIBA, N. AND NISHIZEKI, T. 1985. Arboricity and subgraph listing algorithms. *SIAM J. Comput. 14,* 1, 210–223.

CHROBAK, M. AND EPPSTEIN, D. 1991. Planar orientations with low out-degree and compaction of adjacency matrices. *Theor. Comput. Sci. 86,* 2, 243–266.

DOWNEY, R. G. AND FELLOWS, M. R. 1995. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theor. Comput. Sci. 141,* 1-2, 109–131.

DOWNEY, R. G. AND FELLOWS, M. R. 1999. *Parameterized Complexity*. Springer-Verlag.

DU, N., WU, B., XU, L., WANG, B., AND XIN, P. 2009. Parallel algorithm for enumerating maximal cliques in complex network. In *Mining Complex Data*. Studies in Computational Intelligence Series, vol. 165. Springer-Verlag, 207–221.

EBLEN, J. D., PHILLIPS, C. A., ROGERS, G. L., AND LANGSTON, M. A. 2011. The maximum clique enumeration problem: algorithms, applications and implementations. In *Proc. 7th Int. Symp. Bioinformatics Research and Applications (ISBRA 2011)*. LNCS Series, vol. 6674. Springer-Verlag, 306–319.

EPPSTEIN, D. 2003. Small maximal independent sets and faster exact graph coloring. *J. Graph Algorithms & Applications 7,* 2, 131–140.

EPPSTEIN, D. 2009. All maximal independent sets and dynamic dominance for sparse graphs. *ACM Trans. Algorithms 5,* 4, A38.

EPPSTEIN, D. AND SPIRO, E. S. 2009. The h-index of a graph and its application to dynamic subgraph statistics. In *Proc. 11th Symp. Algorithms and Data Structures (WADS 2009)*. LNCS Series, vol. 5664. Springer-Verlag, 278–289.

ERDŐS, P. AND HAJNAL, A. 1966. On chromatic number of graphs and set-systems. *Acta Mathematica Hungarica 17,* 1–2, 61–99.

FOMIN, F. V., OUM, S.-I., AND THILIKOS, D. M. 2010. Rank-width and tree-width of H-minor-free graphs. *European J. Combin. 31,* 7, 1617–1628.

FREUDER, E. C. 1982. A sufficient condition for backtrack-free search. *J. ACM 29,* 1, 24–32.

GÉLY, A., NOURINE, L., AND SADI, B. 2009. Enumeration aspects of maximal cliques and bicliques. *Discrete Appl. Math. 157,* 7, 1447–1459.

GERHARDS, L. AND LINDENBERG, W. 1979. Clique detection for nondirected graphs: Two new algorithms. *Computing 21,* 4, 295–322.

GOEL, G. AND GUSTEDT, J. 2006. Bounded arboricity to determine the local structure of sparse graphs. In *WG 2006*, F. V. Fomin, Ed. LNCS Series, vol. 4271. Springer-Verlag, 159–167.

GOLOVACH, P. A. AND VILLANGER, Y. 2008. Parameterized complexity for domination problems on degenerate graphs. *Proc. 34th Int. Worksh. Graph-Theoretic Concepts in Computer Science (WG 2008) 5344*, 195–205.

HARARY, F. 1972. *Graph Theory*. Addison-Wesley, Reading, MA.

HARARY, F. AND ROSS, I. C. 1957. A procedure for clique detection using the group matrix. *Sociometry 20,* 3, 205–215.

JENSEN, T. R. AND TOFT, B. 1995. *Graph Coloring Problems*. Wiley-Interscience, New York.

JOHNSON, D. S. AND TRICK, M. A. 1996. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*. American Mathematical Society, Boston, MA, USA.

JOHNSON, D. S., YANNAKAKIS, M., AND PAPADIMITRIOU, C. H. 1988. On generating all maximal independent sets. *Inf. Proc. Lett. 27,* 3, 119–123.

JOHNSTON, H. C. 1976. Cliques of a graph—variations on the Bron–Kerbosch algorithm. *Int. J. Parallel Programming 5,* 3, 209–238.

KIROUSIS, L. M. AND THILIKOS, D. M. 1996. The linkage of a graph. *SIAM J. Comput. 25,* 3, 626–647.

KLOKS, T. AND CAI, L. 2000. Parameterized tractability of some (efficient) Y-domination variants for planar graphs and t-degenerate graphs. In *Proc. International Computer Symposium*.

KOCH, I. 2001. Enumerating all connected maximal common subgraphs in two graphs. *Theor. Comput. Sci. 250,* 1–2, 1–30.

KOSTOCHKA, A. V. 1984. Lower bound of the Hadwiger number of graphs by their average degree. *Combinatorica 4*, 307–316.

LAWLER, E. L., LENSTRA, J. K., AND RINNOOY KAN, A. H. G. 1980. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM J. Comput. 9,* 3, 558–565.

LESKOVEC, J. 2009. Stanford Large Network Dataset Collection. `http://snap.stanford.edu/data/`.

LICK, D. R. AND WHITE, A. T. 1970. k-degenerate graphs. *Canad. J. Math. 22*, 1082–1096.

LOUKAKIS, E. AND TSOUROS, C. 1981. A depth first search algorithm to generate the family of maximal independent sets of a graph lexicographically. *Computing 27,* 4, 349–366.

LU, L., GU, Y., AND GROSSMAN, R. 2010. dMaximalCliques: A Distributed Algorithm for Enumerating All Maximal Cliques and Maximal Clique Distribution. In *Int. Conf. on Data Mining Workshops*. IEEE Computer Society, 1320–1327.

MAKINO, K. AND UNO, T. 2004. New algorithms for enumerating all maximal cliques. In *Proc. 9th Scand. Worksh. Algorithm Theory*. LNCS Series, vol. 3111. Springer-Verlag, 260–272.

MODANI, N. AND DEY, K. 2008. Large maximal cliques enumeration in sparse graphs. In *Proceeding of the 17th ACM conference on Information and knowledge management (CIKM '08)*. 1377–1378.

MOON, J. W. AND MOSER, L. 1965. On cliques in graphs. *Israel J. Math. 3,* 1, 23–28.

MULLIGAN, G. D. AND CORNEIL, D. G. 1972. Corrections to Bierstone's algorithm for generating cliques. *J. ACM 19,* 2, 244–247.

NEWMAN, M. E. J. 2006. Network data. `http://www-personal.umich.edu/~mejn/netdata/`.

PAN, L. AND SANTOS, E. E. 2008. An anytime-anywhere approach for maximal clique enumeration in social network analysis. In *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*. 3529–3535.

SAMATOVA, N. F., SCHMIDT, M. C., HENDRIX, W., BREIMYER, P., THOMAS, K., AND PARK, B.-H. 2008. Coupling graph perturbation theory with scalable parallel algorithms for large-scale enumeration of maximal cliques in biological graphs. *Journal of Physics: Conference Series 125,* 1, 012053.

SCHMIDT, M. C., SAMATOVA, N. F., THOMAS, K., AND PARK, B.-H. 2009. A scalable, parallel algorithm for maximal clique enumeration. *J. Parallel and Distributed Computing 69,* 4, 417–428.

STARK, C., BREITKREUTZ, B.-J., REGULY, T., BOUCHER, L., BREITKREUTZ, A., AND TYERS, M. 2006. BioGRID: a general repository for interaction datasets. *Nucleic Acids Res. 34*, D535–D539.

THOMASON, A. 1984. An extremal function for contractions of graphs. *Math. Proc. Camb. Phil. Soc. 95,* 2, 261–265.

THOMASON, A. 2001. The extremal function for complete minors. *J. Combinatorial Theory, Ser. B 81,* 2, 318–338.

TOMITA, E., TANAKA, A., AND TAKAHASHI, H. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci. 363,* 1, 28–42.

TSUKIYAMA, S., IDE, M., ARIYOSHI, H., AND SHIRAKAWA, I. 1977. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput. 6,* 3, 505–517.

WOOD, D. R. 2007. On the maximum number of cliques in a graph. *Graphs and Combinatorics 23,* 3, 337–352.

ZHANG, Y., ABU-KHZAM, F. N., BALDWIN, N. E., CHESLER, E. J., LANGSTON, M. A., AND SAMATOVA, N. F. 2005. Genome-scale computational approaches to memory-intensive applications in systems biology. In *Proc. 2005 ACM/IEEE Conf. on Supercomputing*.

Received Month Year; revised Month Year; accepted Month Year