PARALLEL MAXIMUM CLIQUE ALGORITHMS WITH APPLICATIONS TO NETWORK ANALYSIS

RYAN A. ROSSI*, DAVID F. GLEICH†, AND ASSEFAW H. GEBREMEDHIN‡

Abstract. We present a fast, parallel maximum clique algorithm for large sparse graphs that is designed to exploit characteristics of social and information networks. The method exhibits a roughly linear runtime scaling over real-world networks ranging from a thousand to a hundred million nodes. In a test on a social network with 1.8 billion edges, the algorithm finds the largest clique in about 20 minutes. At its heart the algorithm employs a branch-and-bound strategy with novel and aggressive pruning techniques. The pruning techniques include the combined use of core numbers of vertices along with a good initial heuristic solution to remove the vast majority of the search space. In addition, the exploration of the search tree is parallelized. During the search, processes immediately communicate changes to upper and lower bounds on the size of maximum clique. This exchange of information occasionally results in a super-linear speedup because tasks with large search spaces can be pruned by other processes. We demonstrate the impact of the algorithm on applications using two different network analysis problems: computation of temporal strong components in dynamic networks and determination of compression-friendly ordering of nodes of massive networks.

Key words. parallel maximum clique algorithms, branch-and-bound, network analysis, temporal strong components, graph compression

1. Introduction.

The maximum clique problem seeks to find a clique (complete subgraph) of the largest possible size in a given graph. The problem is, in general, NP-hard to solve, even in an approximate sense [33]. As a result one is inclined to believe that exact algorithms for finding maximum cliques will be too slow to be practical for large network analysis applications. In fact, because of this inclination, in a number of network analysis problems where maximum cliques are the natural and accurate models, practitioners frequently settle for loose, approximate models representing "dense-enough" subgraphs that can be detected fast or heuristic clique methods that generally perform well enough in practice. Yet, many real-world problems have features that do not elicit worst-case behaviors from well-designed algorithms.

In this manuscript, we present a demonstrably fast, parallel, exact algorithm for the maximum clique problem.[1] The presentation includes the design, implementation, analysis and performance evaluation of the algorithm. Further, enabled by its efficiency, we use the clique finder to achieve three goals: (i) study maximum cliques in large-scale social and information networks, (ii) find the largest temporal strong components in time-varying networks, and (iii) obtain compression-friendly orderings of vertices in graphs.

The algorithm. In its basic form, our algorithm is a branch-and-bound method with novel pruning strategies. Several key components stand out as features contributing to its efficiency and distinguishing it from existing algorithms.

First, the algorithm begins by finding a large clique using a near linear-time heuristic; the obtained solution is checked for optimality (using the bounds described in Section 3) before the algorithm proceeds any further, and the algorithm is terminated if the solution is found to be optimal. Second, we use the heuristic solution, in

*Dept of Computer Science, Purdue University, West Lafayette, IN. (rrossi@purdue.edu)

†Dept of Computer Science, Purdue University, West Lafayette, IN. (dgleich@purdue.edu)

‡School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA. (assefaw@eecs.wsu.edu)

[1] A two-page preliminary version of what is presented here has appeared in the proceedings of WWW 2014 accompanying a poster presentation at the conference [48].

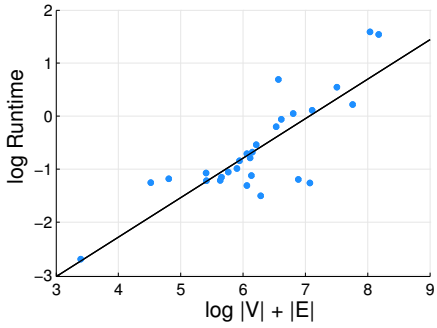


FIG. 1. *A log-log plot of the runtime of our clique finder on 32 social and information networks drawn from a variety of domains. The plot shows that the runtime scales almost linearly with network dimension.*

combination with (tight) upper bounds on the largest clique, to aggressively prune the graph. The upper bounds are computed at the level of the input graph or local neighborhoods. Third, we use implicit graph edits and periodic full graph updates in order to keep our implementation efficient. Fourth, we parallelize the search procedure. The parallel search is designed such that processes (workers) immediately communicate changes to upper and lower bounds on the size of maximum clique. As a result, vertices with especially large search spaces can be pruned by other processes, which occasionally results in a super-linear speedup. Finally, rather than a fixed algorithm, our method is a framework that can be specialized into different variants. The framework is tunable in the sense that the graph representation, data structures, and the implementations of the algorithm can be adapted based on the properties of the input graph and the target system. The framework is discussed in detail in Section 4, the bounds it makes use of in its pruning strategies are reviewed in Section 3, and the framework's overall performance is evaluated and compared against existing methods in Section 5. We have made our implementation publicly available at `https://github.com/ryanrossi/pmc`.

Cliques in large social and information networks. Our investigation on large-scale social and information networks (Section 2) reveals that finding the largest clique in such networks can in practice be done fast (Table 1, Figure 1). By way of example, using the maximum clique algorithm proposed here, we can find the maximum clique in social networks with nearly two billion edges *in about 20 minutes* on a 16-core shared memory system. More generally, our method is observed to have a roughly linear runtime (Figure 1) for these networks. As a point of comparison, our new solver significantly outperforms a recent fast maximum clique finder we developed [46] as well as an off-the-shelf clique enumerator (Section 5). Consequently, we expect the new algorithm to be useful for various tasks in which maximum cliques are needed such as analyzing large networks, evaluation of graph generators, community identification, and anomaly detection.

Applications. One motivation for this work came from a connection between the largest temporal strong component of a dynamic network and maximum cliques in an associated graph. In a network where each edge represents a contact – a phone call, an email, or physical proximity – between two entities at a specific point in time, one gets an evolving network structure [26] where a temporal path represents a sequence of contacts that obeys time. A temporal strong component is a set of vertices where all pairwise temporal paths exist, just like a strong component is a set of vertices where all pairwise paths exist.

Surprisingly, checking if an evolving network has a temporal strong component of size k is NP-complete [40, 3]. For some intuition, consider the following "wrong"

reduction from the perspective of establishing NP-hardness. A temporal strong component of size k corresponds to a clique of size k in a temporal reachability graph where each edge represents a temporal path between vertices. Finding the maximum clique, then, reveals the largest temporal strong component. At a first glance, this is no help as even approximating the largest clique in a graph is NP-hard. With a fast algorithm in place, however, the connection can be exploited. We apply our maximum clique finder for this analysis and discuss properties of temporal components we find in Twitter and phone call networks in Section 6.1.

Previous studies found bipartite cliques useful for compressing networks [10]. Here we tackle an easier problem and use cliques to compute a compression-friendly ordering that makes many edges in the graph local. We find (Section 6.2) that this ordering generates results that are nearly as good as existing heuristics tailor-designed for that problem.

Related work. Pardalos and Xue [45] provide a good review of exact algorithms for maximum clique that existed prior to 1994. Notable methods proposed since then include, among others, the works of Bomze et al. [8], Östergård [42], Tomita et al. [55], and San Segundo et al. [49]. In a recent work, Prosser [47] provides a computational study comparing various exact algorithms for maximum clique. The vast majority of existing work focuses on sequential maximum clique finders, but there is growing work on parallel algorithms as well. Recent work on parallel algorithms include the multithreaded algorithm of McCreesh and Prosser [39] and the MapReduce-based method due to Xiang et al. [61].

A related problem to maximum clique finding is maximal clique enumeration: identifying *all* the maximal cliques in G. There is a considerable body of recent work on this problem. Tomita et al. [56] look at the worst-case time complexity of generating all maximal cliques and conduct computational experiments. Eppstein et al. [24, 23] show how efficient data structures can be used to design algorithms for clique enumerations in near optimal time. Schmidt et al. [51] develop parallel algorithms for maximal clique enumeration, and Cheng et al. [13, 14] consider clique enumeration on massive graphs. Xie et al. [62] show the connection of the clique enumeration problem to frequent pattern mining. Du et al. [20, 21] find that maximal cliques in social networks are distributed according to a power-law. In particular the authors of [21] take advantage of the properties of social and information networks in order to enumerate all maximal cliques faster. In comparison, we show here how we can develop fast algorithms for solving the maximum clique problem for these networks and temporal strong components by appropriately applying pruning steps and bounds.

2. Cliques in social and information networks.

Before presenting the details of our new algorithm, we begin by demonstrating how fast it finds maximum cliques in various social and information networks and highlighting observations we make regarding the cliques obtained.

We experiment in this section with 32 networks categorized in 8 broad classes. In the appendix we report results on a more extensive collection of networks comprising 76 social and information networks and 63 dense graphs from the 1996 DIMACS Clique Challenge [58]. Table 1 lists the names and sizes of the 32 networks considered here (detailed data on the properties of these networks is provided in Tables 4–6 in the appendix). Table 1 also lists the size of the largest clique in each network and the time it took the algorithm to find each clique. We plot the runtime pictorially in Figure 1, which shows a linear scaling between a thousand and a hundred million vertices.

Below we briefly describe the networks and what cliques in them signify. For all

of the networks, we discard edge weights and self-loops when they exist. In addition, if the graph is directed, we remove non-reciprocated edges. This strategy will identify fully-directed cliques. Further, for networks with multiple components, we consider only the largest connected component (when undirected) and the largest strongly connected component (when directed).

1. Biological networks. We study a network where the nodes are proteins and the edges represent protein-protein interactions (DMELA [53]) and another where nodes are substrates and edges are metabolic reactions (CELEGANS [30]). Cliques in these networks signify biologically relevant modules.

2. Collaboration networks. These are networks in which nodes represent individuals and edges represent scientific collaborations or movie production collaborations (MATHSCINET [43]; DBLP, HOLLYWOOD [5]). Large cliques in these networks are expected because they are formed when collaborations involve many participants.

3. Interaction networks. Here, nodes represent individuals and edges represent interaction in the form of message posts (WIKI-TALK [37]). Cliques in such networks represent mutually interacting groups of individuals.

4. Retweet networks. Here, nodes are Twitter users and two users are connected by an edge if they have retweeted each other (RETWEET). We collected the network RETWEET ourselves. A clique here is a group of users that have all mutually retweeted each other; it may represent an interest cartel or an anomaly.

5. Technological networks. The nodes in these networks are routers and edges are observed communications between the entities (AS-SKITTER, RL-CAIDA [11]; WHOIS [59]). A clique represents all-to-all communication amongst entities.

6. Web link networks. Here, nodes are web-pages and edges are hyperlinks between pages (WIKIPEDIA [17]; ARABIC-2005, IT-2004, UK-2005 [4]). Large cliques represent large sets of pages where full pairwise navigation is possible.

7. Facebook networks. Nodes represent people and edges are "Facebook friendships" (CMU, MIT, STANFORD, BERKELEY, UILLINOIS, PENN, TEXAS [57]; FB-A, FB-B [60]; UCI-UNI [27]). Cliques here are groups of people with mutual friendships.

8. Social networks. Nodes are again people and edges are social relationships in the form of friendship or follower (ORKUT, LIVEJOURNAL, YOUTUBE [63]; SLASHDOT [38]; GOWALLA [15]; FLICKR [28]; TWITTER [36]; FRIENDSTER [Internet Archive]).

We summarize below our findings about cliques in these networks and the performance of our algorithm:

- We observe that the initial heuristic step of the algorithm finds the largest clique in most cases: 17 of the 32 instances considered here, and 54 of the 76 networks considered in Tables 4–6 in the appendix; see the left plot in Figure 2 for a summary. This property helps our exact maximum clique algorithm terminate quickly.
- We studied the relationship between the largest k-core (a notion to be discussed in Section 3) and the largest clique. The right part in Figure 2 shows a summary of the results we obtained on all the 76 networks. In the collaboration and most web-link networks, we find that the largest k-core coincides with a maximum clique in the graph. The social networks, in comparison, have a much larger difference between the two, which suggests a fundamental difference in the types of networks formed via collaboration relationships versus social relationships.
- We observe that technological networks have surprisingly large cliques. Given that a clique represents an overly redundant set of edges, this would suggest that these maximum cliques represent over-built technology, or critical groups of nodes.
- We observe that for the TWITTER network, the nodes in the largest clique are

TABLE 1

Properties of and results on 32 of the social and information networks studied here. The number of vertices $|V|$ and edges $|E|$ are appended by K for thousands, M for millions and B for billions. The column $K+1$ corresponds to a core number-based upper bound on maximum clique size, $\tilde{\omega}$ denotes the size of the clique obtained by the initial heuristic step, and ω denotes the actual maximum clique size. The last column shows the runtime of the exact algorithm. It can be seen that the algorithm took less than 21 minutes to solve the "largest" problem in the collection.

	graph	$\|V\|$	$\|E\|$	$K+1$	$\tilde{\omega}$	ω	Time (s.)
1.	CELEGANS	453	2.0K	11	9	9	<.01
	DMELA	7.4K	26K	12	7	7	0.06
2.	MATHSCIET	333K	821K	25	25	25	0.08
	DBLP	317K	1.0M	114	114	114	0.05
	HOLLYWOOD	1.1M	56M	2209	2209	2209	1.69
3.	WIKI-TALK	92K	361K	59	14	15	0.09
4.	RETWEET	1.1M	2.3M	19	13	13	0.58
5.	WHOIS	7.5K	57K	89	55	58	0.09
	RL-CAIDA	191K	608K	33	17	17	0.13
	AS-SKITTER	1.7M	11M	112	66	67	1.2
6.	ARABIC-2005	164K	1.7M	102	102	102	0.03
	WIKIPEDIA2	1.9M	4.5M	67	31	31	1.16
	IT-2004	509K	7.2M	432	432	432	0.12
	UK-2005	130K	12M	500	500	500	0.06
7.	CMU	6.6K	250K	70	45	45	0.09
	MIT	6.4K	251K	73	32	33	0.1
	STANFORD	12K	568K	92	51	51	0.09
	BERKELEY	23K	852K	65	42	42	0.16
	UILLINOIS	31K	1.3M	86	56	57	0.18
	PENN	42K	1.4M	63	43	44	0.24
	TEXAS	36K	1.6M	82	49	51	0.33
	FB-A	3.1M	24M	75	23	25	6.3
	FB-B	2.9M	21M	64	23	24	5.52
	UCI-UNI	59M	92M	17	6	6	33.86
8.	SLASHDOT	70K	359K	54	25	26	0.06
	GOWALLA	197K	950K	52	29	29	0.2
	YOUTUBE	1.1M	3.0M	52	16	17	0.84
	FLICKR	514K	3.2M	310	45	58	5.2
	LIVEJOURNAL	4.0M	28M	214	214	214	2.98
	ORKUT	3.0M	106M	231	44	47	48.49
	TWITTER	21M	265M	1696	174	323	598
	FRIENDSTER	66M	1.8B	304	129	129	1205

neither of the two obvious suspects, that is, (i) just spam accounts nor (ii) legitimate accounts with massive numbers of followers and following similar large numbers. Rather, it is a strange combination of the two sets. We believe that most members of this clique likely reciprocate all follower relationships.

3. Bounds on maximum clique size. As a prelude to our maximum clique algorithm, we review a few easy-to-derive upper bounds on the size of the largest clique $\omega(G)$ in a graph G. These bounds will allow us to terminate our algorithm once we have found something that hits the upper bound or stop a local search early because no larger clique exists.

A simple upper bound on the size of the largest clique is the maximum degree $\Delta(G)$ in the graph. Usually this is too simple to be useful. A stronger bound can be obtained using k-cores. A *k-core* in a graph G is a vertex induced subgraph where all vertices have degree at least k [52]. The core number of a vertex v is the *largest* k such that v is in a k-core. We denote it by $K(v)$, and we denote by $K(G)$ the largest core number in the entire graph G. Suppose that G contains a clique of size

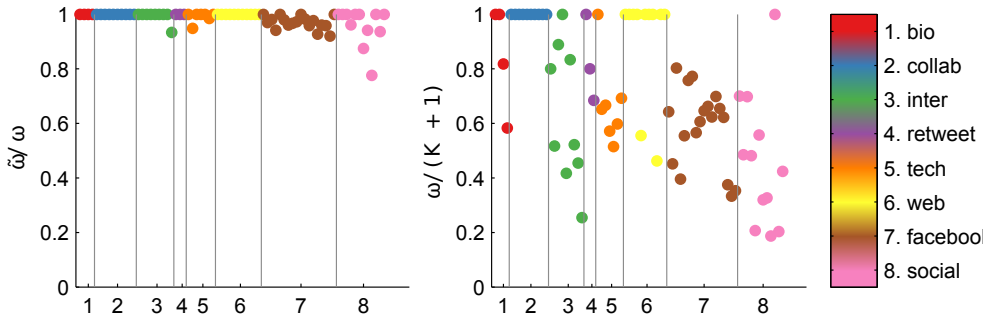


FIG. 2. *These two plots summarize the results on all the 76 social and information networks listed in Tables 4–6 in the appendix. The left figure depicts a plot of the ratio of the clique size obtained by our heuristic ($\tilde{\omega}$) to the largest clique size obtained by the entire algorithm (ω). It shows that the heuristic gives the exact solution in the biological, collaboration, and web networks in all but one case. In the right figure is shown a plot of the ratio of the maximum clique size (ω) to the largest core number plus one ($K+1$). The figures identifies the networks where the core number tightly bounds the largest clique.*

q. Then each vertex in the clique has degree $q-1$ and the entire graph must have a $(q-1)$-core. Thus $K(G)+1$ is an upper bound on the largest clique size $\omega(G)$. Note that in contrast to cliques, the core numbers of all vertices in a graph can be computed with a linear-time algorithm [2].

The value $K(G)$ is also known as the degeneracy of the graph. The quantity $K(G)+1$ is an upper bound on the number of colors used by a greedy coloring algorithm that processes vertices in order of decreasing core numbers – also known as degeneracy order [25]. The number of colors used by any greedy coloring of G is also an upper bound on the size of the largest clique because a clique of size k requires k colors. Let $L(G)$ be the number of colors used by a greedy coloring algorithm that uses the degeneracy order. Then $L(G) \leq K(G)+1$ and we get a potentially tighter bound on the size of the largest clique. The bound $L(G)$ can be computed in linear time with some care on the implementation of the greedy coloring scheme. We summarize the bounds we have at this point:

FACT 3.1. $\omega(G) \leq L(G) \leq K(G)+1 \leq \Delta(G)+1$.

We can further improve the bounds in Fact 3.1 by using one additional fact about a maximum clique in a graph. Define the *neighborhood graph* of a vertex v to be the graph induced by v and its neighbors. Then any neighborhood graph of a vertex within the largest clique has a clique of the same size within the neighborhood graph as well. The way our algorithm proceeds is by iteratively removing vertices from the graph that cannot be in the largest clique. Let $N_R(v)$, the *reduced* neighborhood graph of v, be the vertex-induced subgraph of G corresponding to v and all neighbors of v that have *not* been removed from the graph yet. All the bounds in Fact 3.1 apply to finding the largest clique in each of these neighborhood subgraphs. We can therefore state:

FACT 3.2.

$$\omega(G) \leq \max_v L(N_R(v)) \tag{3.1}$$

$$\leq \max_v K(N_R(v)) + 1 \tag{3.2}$$

$$\leq \max_v \Delta(N_R(v)) + 1. \tag{3.3}$$

Computing the tighter bounds in Fact 3.2 requires slightly more than linear work. For each vertex, we need to form the neighborhood graph. If we look at the union of all of these neighborhood graphs, there is a vertex in some neighborhood graph for each edge in G. Thus there are a total of $O(|E|)$ vertices in all neighborhoods. By the same argument, there are $O(|E| + |T|)$ edges where $|T|$ is the total number of triangles in the graph. Consequently, we can make the following statement:

FACT 3.3. *The total work involved in computing the bounds in Fact 3.2 is bounded by* $O(|E| + |T|)$.

4. A maximum clique algorithms framework. Given an undirected graph $G = (V, E)$, let C_v denote a clique of the largest size containing the vertex v. A maximum clique in G can be found by computing C_v for every vertex v in V and then picking the largest among these. This clearly is wasteful. Most branch-and-bound type algorithms for maximum clique speed up the process by keeping around the size of the largest clique computed at any point in the course of the algorithm (`maxSoFar`) and avoiding computation of every C_u, $u \in V$, that would eventually be smaller than `maxSoFar`, a process generically referred to as *pruning* [45, 42, 55, 46, 61]. The algorithms differ chiefly in the way the pruning is done. The algorithm we developed in a recent work [46] uses a hierarchical pruning strategy that relies primarily on comparisons of degrees of vertices in the original input graph with `maxSoFar`, effectively using the weakest bound in Fact 3.1. In comparison, the new method presented here uses the tightest bound in Fact 3.2. Furthermore, it contains a variety of new algorithmic and performance optimization ingredients that result in significantly superior performance.

For reference throughout the discussion in this section, we outline our algorithm in the psuedocodes in Algorithm 1 and Algorithm 2.

4.1. The fast heuristic clique finder. Our exact maximum clique algorithm begins by calling a fast heuristic clique finder that makes use of core numbers of vertices, which in turn is computed in a prior auxiliary step (see Lines 2 and 3 in the procedure MAXCLIQUE in Algorithm 2). The goal of the initial heuristic step is to find a large clique in the graph quickly. The heuristic is similar to the maximum-degree based heuristic described in [46], which, in exploring for a maximum clique in which a vertex v participates, simply picks a vertex of the highest degree in the neighborhood of v. The heuristic search described here differs as we use core numbers of vertices to guide the search instead. The inspiration for this change is the relationship between core numbers, the degeneracy order, and a simple 2-approximation algorithm for the densest subgraph problem [35, 12].

The heuristic, outlined in Algorithm 1, builds a clique by searching around each vertex in the graph and greedily adding vertices from the neighborhood as long as they form a clique. The order of vertices is the degeneracy order (the input parameter **K** contains the needed core numbers of the vertices; we write it in **boldface** to indicate

Algorithm 1 Our greedy heuristic to find a large clique. This is used as the first step in the exact algorithm, outlined in Algorithm 2. The input array **K** holds core numbers of vertices. The output of the algorithm is a large clique H.

```
1  procedure HeuristicClique(G = (V, E), K)
2      Set H = {}, Set max = 0
3      for each v ∈ V in decreasing core number order do
4          if v's core number is ≥ max then
5              Let S be the neighbors of v with core numbers ≥ max
6              Set C = {}
7              for each vertex u ∈ S by decreasing core number do
8                  if C ∪ {u} is a clique then
9                      Add u to C
10             if |C| > max then
11                 Set H = C, Set max = |H|
12     return H, a large clique in G
```

that it is a vector (an array)). Because the core numbers are also a lower bound on the size of the largest clique a vertex participates in, we can efficiently prune the greedy exploration.

As mentioned in Section 2, this heuristic step in itself finds the *largest clique* in the graph in over half of the social networks we consider. It can therefore be used as a stand-alone procedure. All steps in Algorithm 1, except for the statements in Lines 7–9, can be performed using work proportional to the degree of a vertex. Those statements in turn require work proportional to the size of the subgraph induced by the neighborhood of a vertex. The overall runtime can therefore be (loosely) upper-bounded by $O(|E| \cdot \Delta(G))$.

4.2. Initial pruning. After our exact algorithm finds a heuristic clique H in the input graph G using the core numbers of the vertices, it puts those numbers to another strategic use. Suppose we find a clique in G of size $\tilde{\omega} = |H|$. Then we can eliminate all vertices with core numbers strictly less than $\tilde{\omega}$ from our search (Line 4 in MaxClique). This pruning operation works because a clique of size $\tilde{\omega} + 1$ or larger must have vertices with core numbers at least $\tilde{\omega}$. In a few cases, we observed this step suffices to certify that H is the maximum clique as we remove all of the graph. This happens, for instance, with the LiveJournal network. Moreover, this pruning procedure reduces memory requirements significantly for most networks.

In our implementation, for this initial pruning, vertices are explicitly removed from the graph. This step often removes a substantial fraction of the total vertices and reduces the total memory required to store the graph.

4.3. Searching. After we reduce the size of the graph via the initial pruning, we then run a search strategy over all the remaining vertex neighborhoods in the graph (the **while**-loop in MaxClique). The algorithm we run is similar to a standard (Bron-Kerbosch) branch-and-bound scheme for maximal clique enumeration [9]. However, we unroll the first two levels of branching and apply our clique bounds in order to find only the largest clique.

At this point, we wish to introduce a bit of terminology. Recall (from Section 3) that $N_R(v)$ is the reduced neighborhood graph of v. Let $d_R(v)$ denote the reduced degree of v. The reduced neighborhood graphs exclude vertices that have been removed from the graph due to changes in the lower bound on the clique size caused by k-cores

Algorithm 2 Our exact maximum clique algorithm. See Section 4.5 for details about how to parallelize it.

```
1  procedure MaxClique(G = (V, E))
2      Set K = CoreNumbers(G)                          ▷ K is a vertex-indexed array
3      Set H = HeuristicClique(G, K)                   ▷ H is global (is updated in Branch)
4      Remove (explicitly) vertices with K(v) < |H|
5      while |G| > 0 do
6          Let u be the vertex with smallest reduced degree
7          InitialBranch(u)                            ▷ the routine grows H
8          Remove u from G
9          Periodically, explicitly remove vertices from G
10     Return H, the largest clique in G

11 procedure InitialBranch(u)
12     Set P = N_R(u)
13     if |P| ≤ |H| then return
14     Set K_N = CoreNumbers(P)
15     Set K(P) = max_{v∈P} K_N(v)
16     if K(P) + 1 < |H| then return
17     Remove any vertex with K_N(v) < |H| from P
18     Set L = Color(P, K_N) in degeneracy order       ▷ L is nr of colors
19     if  L ≤ |H| then return
20     Branch(P, {})

21 procedure Branch(P, C)
22     while  |P| > 0 and |P| + |C| > |H| do
23         Select a vertex w from P and remove w from P
24         Set C' = C ∪ {w}
25         Set P' = P ∩ {N_R(w)}
26         if  |P'| > 0 then
27             Set L = Color(P') in natural (any) order
28             if  |C'| + L > |H| then
29                 Branch(P', C')
30         else if |C'| > |H| then                     ▷ C' is maximal
31             Set H = C'                              ▷ new max clique
32             Remove any v with K(v) < |H| from G     ▷ implicitly
```

and vertices whose local searches have terminated. At the risk of being overly formal, let $\tilde{\omega}$ be the current best lower bound on the clique size, and let X be a set of vertices removed via searching. Then:

$$N_R(v) = G(\{v\} \cup \{u : (u, v) \in E, K(u) \geq \tilde{\omega}, u \notin X\}).$$

We explore the remaining vertices in order of the smallest to largest reduced degree. For each vertex, we explore its neighborhood using the function InitialBranch. When InitialBranch returns, we have found the largest clique involving that vertex, and so we can remove it from the graph. This is done by marking it as removed in an array and then checking that array before using information about the vertex in the future. This implementation provides constant time deletion operations, albeit with

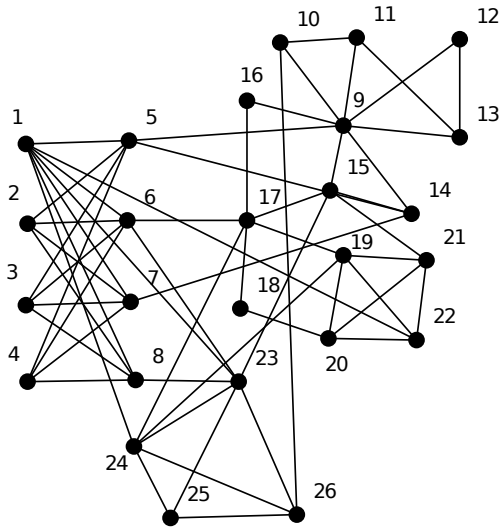


FIG. 3. *An example used to illustrate the workings of Algorithm 2. See discussion at the end of Section 4.3.*

an additional check on use. To eliminate these checks, we find it advantageous to periodically recreate the graph data structure in light of all the deletions and recompute k-cores. This reduces the cost of the intersection operations. In addition, we believe that this step aggregates memory access to a more compact region thereby improving caching on the processor. We do this every four seconds of wall clock time. The four second interval worked well in our experiments, but the choice is rather arbitrary, and we did not investigate other choices of intervals in any detail. Here, incorporating the streaming algorithm proposed in [50] may help make the recomputation of k-cores more efficient.

The first step of INITIALBRANCH is a set of tests to check if any of the bounds from Fact 3.2 rule out finding a bigger clique in the neighborhood of the vertex u being explored. The first test (Line 13) essentially corresponds to the weakest bound, Equation (3.3), in Fact 3.2. To check against the bound given by Equation (3.2), we compute the core numbers for each vertex in the neighborhood subgraph. If the largest core number in the neighborhood subgraph is no better than the current lower bound, we immediately return and add the vertex to the list of searched vertices (Line 16). If it isn't, then we compute a greedy coloring of the subgraph using the degeneracy order in order to obtain the coloring bound from Fact 3.2 (Equation (3.1)). We check against this bound, and we immediately return if the comparison suggests no larger clique is present (Line 19). If none of these checks pass, then we enter into a recursive procedure that examines all subsets of the neighborhood in a search for cliques (BRANCH).

The procedure BRANCH maintains a reduced neighborhood subgraph P and a clique C. The invariant shared by these sets is that we can add a vertex from P to C and get a clique one vertex larger. We pick a vertex and do this. To be precise, we pick the vertex with the most-recently introduced color (in our implementation, this is the largest color where colors are positive integer numbers), as this is a weak clique indicator. We then check if the clique C' is maximal by testing if there exists any set P' that satisfies the invariant. If it is not, then we test if it is possible that C' and P' have a large clique. The largest clique possible is $|C'| + \omega(P') \leq |C'| + L(P')$, and so using the function COLOR, we compute a new greedy coloring to get the upper bound $L(P')$. Unlike the greedy coloring in INITIALBRANCH, here we do not use the

degeneracy ordering as it was not worth the extra work in our investigations. If C' and P' pass these tests, we recurse on C' and P'. If C' is maximal, then we compare it against the current best clique H, and update H if C' is larger.

Illustration (Figure 3). We use the example in Figure 3 to illustrate several of the points we have been discussing thus far. The core number $K(G)$ of this graph is 4, which yields the upper bound of 5 on the maximum clique size. The clique detected by our heuristic is $\{1, 8, 23\}$; the graph has two maximum cliques: $\{19, 20, 21, 22\}$ and $\{23, 24, 25, 26\}$. Our algorithm removes vertices $10, 11, 12, 13$ and $16, 17, 18$ in the initial pruning. Subsequently, our method will explore vertex 9 and remove it based on the maximum neighborhood core of 3. It explores vertex 15 next and removes it due to the neighborhood core bound. It then removes vertex 14 due to an insufficient degree. Subsequently, it finds the clique around vertex 19, then prunes all vertices except 1 through 8 due to core number bounds. Finally, it eliminates vertex 1 due to the neighborhood core bound; all other vertices are then iteratively removed via degree bounds.

4.4. Performance optimization. To keep the presentation simple, we have left out several details on performance enhancement that we have in our implementation. (The code is available online for interested readers). To give an example, we use an adjacency matrix structure for *small* graphs in order to facilitate constant-time edge queries, and we use a fast procedure for neighborhood set intersection that runs in time proportional to the size of the output set.

In the overall algorithm, we identify the following elements as the most important for attaining high-performance:

- finding a good initial solution via the fast heuristic clique finder,
- using the smallest to largest ordering in the main loop; this helps ensure that neighborhoods of high degree vertices are as small as possible,
- using efficient data structures for all the operations and graph updates, and
- aggressively using k-core bounds and coloring bounds to remove vertices early.

4.5. Parallelization. We have parallelized the search procedure in the algorithm. Our own implementation uses shared memory, but we describe the parallelization at a high-level such that it could be used with a distributed memory architecture as well. The focus of our discussion is on the general scheme and not on the particular details.

The parallel constructs we use are a worker task-queue and a global broadcast channel. In fact, the basic algorithm remains the same. We compute the majority of the preprocessing work in serial with the exception of a parallel search for the clique in the initial heuristic step. Here, we assume that each worker has a copy of the graph and distribute vertices to workers to find the largest heuristic clique in the neighborhood. In serial, we reduce the graph in light of the bounds, and then re-distribute a copy of the graph to all workers. At this point, we view the main while loop as a task generator and farm the current vertex out to a worker to find the largest clique in that neighborhood. Workers cooperate by communicating improved bounds between each other whenever they find a clique and whenever they remove a vertex from the graph using the shared broadcast channel. When a worker receives an updated bound, we have found that it is often possible for that worker to terminate its own search at once. Unlike most previous algorithms, the speedup from our parallel maximum clique algorithms can be *super linear* since we are less dependent on the precise order of vertices explored. In our own shared memory implementation, we avoid some of the communications by using global arrays and locked updates.

5. Performance evaluation. As demonstrated in Table 1 and Figure 1, our clique finder runs fast on social and information networks and it exhibits roughly linear runtime scaling as the problem size is increased. We used a two processor, Intel E5-2670 system with 16 cores and 256 GB of memory for those tests and the additional tests presented in this section. None of the experiments came close to using all the memory. In this section we look at four additional questions regarding performance:

a) How does the runtime of our algorithm breakdown into time spent in the initial heuristic and the rest of the algorithm?
b) How scalable is our parallel algorithm?
c) How does our method compare to other clique finders on social and information networks?
d) Is the tighter upper bound that results from using neighborhood cores (as opposed to cores in the original graph) worth the additional effort?

In what follows, we will refer to the new algorithm we propose here (outlined in Algorithm 1 and 2) as "pmc" (short for parallel maximum clique). For detailed performance analysis purposes we will consider several variants of pmc.

5.1. Dataset. For the results reported in this section, we additionally use problems from the DIMACS Clique Challenge [58]. We do so in order to evaluate the performance of our clique finder on an established benchmark of difficult problems. These problems represent particularly hard instances; some of them remain unsolvable in reasonable time by the best, tailor-designed state-of-the-art maximum clique algorithms. Detailed data on the properties of these graphs is given in Table 8 and Table 7 in the appendix. We restate here a few of the more general features. These graphs are all small, ranging between 45 to 1500 vertices. They contain, however, an enormous number of edges and triangles in comparison with the social and information networks. The number of triangles range between 34 thousand and 520 million. We divide the 63 graphs in the collection our method was able to solve into an "*easy*" set of 27 graphs (where our algorithm finds a solution in less than a second) and a "*hard*" set of 31 graphs (where the solution time could vary from a second to an hour). Tables 8 and 7 are grouped according to these two categories.

5.2. Runtime breakdown (Question (a)). As mentioned earlier, our maximum clique algorithm (Algorithm 2) begins by computing core numbers of vertices and subsequently invoking a fast heuristic clique finding step (Algorithm 1). With the help of the core numbers, the solution obtained by the heuristic step (clique H) is used to prune out portions of the input graph that *cannot* result in a larger clique (Line 4 of Algorithm 2). If the heuristic solution is in fact optimal, the remaining graph would necessarily be empty and the algorithm would terminate immediately, returning H as the solution. In that case, the runtime of the overall algorithm would simply be the runtime of the heuristic. Tables 4 through 7 in the appendix list the runtime of just the heuristic ($t_{\tilde{\omega}}$) and of the overall algorithm (t_{ω}) for all of the graphs in the testbed. It can be seen that for a vast majority of the social and information networks, the runtimes t_{ω} and $t_{\tilde{\omega}}$ within an order of magnitude, while for some of the DIMACS graphs the ratio $t_{\tilde{\omega}}/t_{\omega}$ can be many order of magnitude. Figure 4 provides a pictorial summary: it shows a plot of the ratio just mentioned for the test graphs in the different categories with non-trivial runtimes of over 0.01 seconds.

5.3. Parallel speedup (Question (b)). At the left in Figure 5 we show the speedup obtained, as more processors are employed, by our pmc method for three social networks. At the right in the same figure we show speedup results of pmc

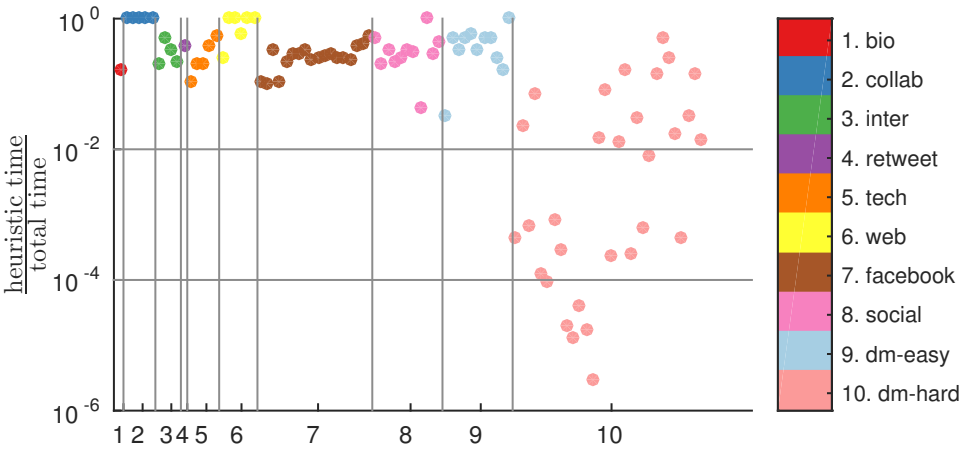


FIG. 4. *The ratio of time taken by the heuristic step compared to the total time for all graphs where clique-finding took more than* 0.01 *seconds; note that dm-easy and dm-hard are the DIMACS "easy" and "hard" sets listed in Tables 7 and 8 in the appendix.*

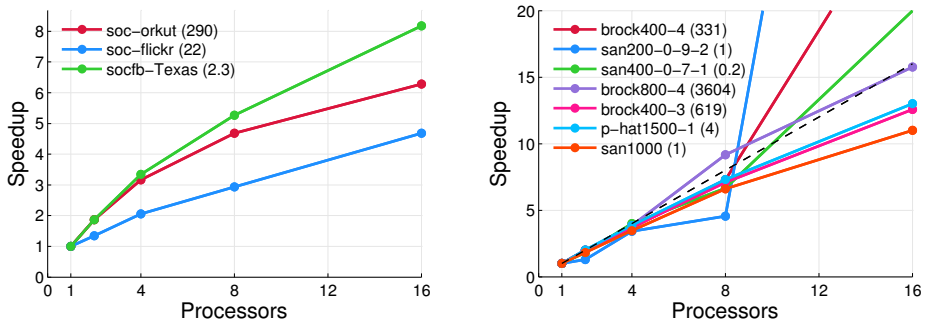


FIG. 5. *Speedup of our parallel maximum clique algorithm on social and information networks (left) and DIMACS graphs (right). Single processor runtimes in seconds are shown in parentheses.*

for seven of the DIMACS graphs. The runtime for both includes all the serialized preprocessing work, such as computing the core numbers initially. The figures show two different behaviors. For social networks, we only get mild speedups on 16-cores, the best result being for the largest problem SOC-ORKUT. For the DIMACS graphs on the other hand, we observe roughly linear and, sometimes, super-linear speedup as we increase the number of processes. The super-linear speedup is due to fast returns from unfruitful branches as a result of the parallel exploration of the search space. These results indicate that our parallelization strategy is promising and helps reduce the runtime for difficult problems. In future work, we plan to investigate the performance of the same strategy on implementations for distributed-memory and other emerging architectures.

5.4. Performance profile plots. To help address the two remaining questions c) and d), we use performance profile plots to compare algorithms [19]. Performance profile plots compare the performance of multiple algorithms on a set of problems (test set). The essential underlying idea is the use of a cumulative *distribution function* for a performance metric (in our case runtime), instead of, for example, taking averages

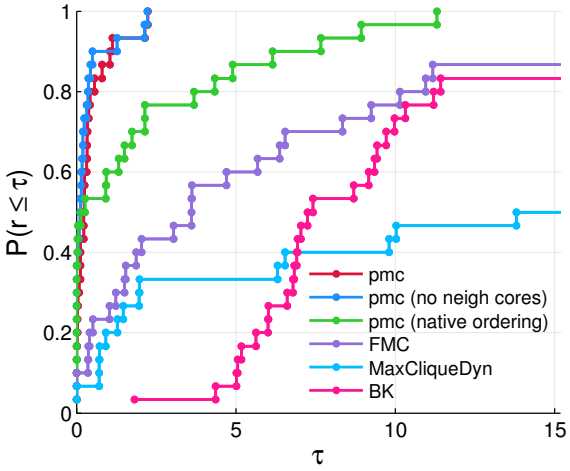


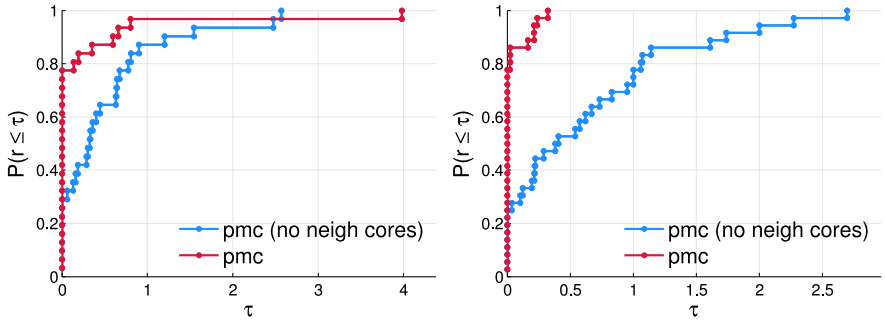
FIG. 6. *Performance profile plots comparing a serial version of pmc and its variants against three existing maximum clique algorithms on 30 social and information networks.*

or sum-totals over all the test cases. Performance profile plots are similar to ROC curves in that the best results are curves that lie towards the upper left. For a quick intuition, suppose we have N problems (test cases) in total and that an algorithm solves M of them within 4 times the speed of the best solver for each problem. Then we would have a point $(\tau, r) = (\log_2 4, M/N)$ in the plot. Note that the horizontal axes reflects a speed difference factor of 2^τ. The fraction of problems that an algorithm solves successfully is given by the left-most highest point on the curve. In Figure 6, for instance, the method labeled BK only solves around 80% of the problems in the test set, and it does so at a factor of 2^{10} times the runtime of the fastest algorithm in the set (pmc).

5.5. Comparison with other methods on social networks (Question (c)). The test we conduct here begins with a "self-comparison". In particular, we consider several variants of pmc in order to assess the effects of the various components on the method's performance. We then compare these variants against a set of existing methods. The variants of pmc we consider are: a serial version with neighborhood cores exploited (pmc), the same version but without exploiting neighborhood cores (pmc no neigh cores), and a version that uses only the k-core pruning steps and searches vertices in their native order, the order in which they were read from disk, rather than degeneracy order (pmc native ordering).

We compare these three variants of pmc against each other and against three state-of-the-art maximum clique finders: the recent method FMC (for fast maximum clique) from [46], the method MaxCliqueDyn [34] which dynamically adapts a greedy color sort, and a recent implementation of the Bron-Kerbosch (BK) algorithm in the `igraph` package [18]. Figure 6 shows the results of such comparisons on a set of 30 social and information networks.

From the performance profile plots in Figure 6, for these types of networks, we find little difference between using and not using the neighborhood cores within our own framework, and somewhat more pronounced difference between using degeneracy ordering versus native ordering. Relative to the alternative algorithms in the compari-



(a) DIMACS-Hard (Serial) (b) DIMACS-Hard (16 Threads)

FIG. 7. *Performance profile plots comparing two versions of pmc (with and without neighborhood cores) on DIMACS-Hard graphs. The left figure shows comparison of serial versions of the two variants, the right figure shows similar comparison of the parallel versions.*

son, we see that the most optimized version of the method proposed here (pmc) offers a dramatic performance improvement. Compared to the BK algorithm, pmc is over 1000 times faster for some problems and solves all of the instances. Compared to the FMC algorithm, pmc is about 50 times faster. This illustrates that our algorithm uses properties of the social and information networks to quickly hone in on the largest clique.

5.6. Assessment of using neighborhood cores (Question (d)). We already saw from the tests discussed in the previous paragraphs that using neighborhood cores in our maximum clique algorithm makes little difference for social and information networks. How about for the DIMACS (and similarly-structured) graphs? The plot in Figures 7(a) and 7(b) show results on the 30 *hard* instances of the DIMACS problems for a test assessing the impact of using and not using neighborhood cores. The plots show results for two cases: pmc run in serial (a) and pmc run in parallel on 16 threads (b). It can be seen that, in the serial case, the neighborhood cores greatly help reduce the work in the majority of cases. In a few cases, not using them entails a large increase in work (the point furthest to the right in the serial figure). All of the work involved in computing these cores is parallelized, and we observe that, in parallel, using them is never any worse than about $2^{0.5} \approx 144\%$ the speed of the fastest method.

5.7. Summary of performance results. In summary, we observe that a) the overall runtime of our algorithm is of the same order of magnitude as the time spent in the initial heuristic for social and information networks, and easy DIMACS graphs, whereas the variation is more significance for the denser DIMACS graphs; b) our parallelization strategy is effective; c) our algorithm outperforms existing algorithms dramatically; and d) neighborhood core bounds are of great help for solving challenging problems. We recommend using neighborhood cores as they help the algorithm terminate faster with challenging problems and almost never take more than twice the time for easy ones.

6. Applications. Although the maximum clique problem is generally NP-hard, as we saw in previous sections, our procedure runs in nearly linear time on many real-world networks. This makes it plausible for the procedure to be used as a part in a fast method for another, encompassing real-world problem, opening up an opportunity

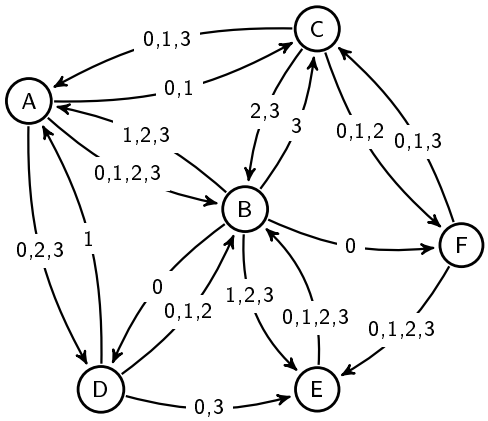


FIG. 8. *An example, adopted from [3], illustrating a temporal network. The reader can verify that there exist temporal paths from node A to all other nodes, from B to all other nodes, from C to all other nodes, from D to all nodes but F, from E to all nodes but F, and from F to all other nodes.*

for potentially great impact. In this section, we illustrate this potential by using our method as a subroutine in algorithms for two applied problems: finding the largest temporal strong component in a dynamic network and finding a compression-friendly order of the nodes of a network.

6.1. Temporal strong components. Temporal strong components were recently proposed by Bhadra et al. and Nicosia et al. to extend the idea of a strong component in a (static) network to a temporal (dynamic) network [3, 40]. Let V be a set of vertices, and $E_T \subseteq V \times V \times \mathbb{R}^+$ be the set of temporal edges between vertices in V. Each edge (u, v, t) has a unique time $t \in \mathbb{R}^+$. For such a temporal network, a path represents a sequence of edges that must be traversed in increasing order of edge times. That is, if each edge represents a contact between two entities, then a path is a feasible route for information. See Figure 8 for an illustration.

Temporal paths are inherently asymmetric because of the directionality of time. Two vertices (u, w) are strongly connected if there exists a temporal path $\mathcal{P}$ from u to w and from w to u. A temporal strongly connected component (temporal SCC) is defined as a maximal set of vertices $C \subseteq V$ such that any pair of vertices in C are strongly connected. Note that this is exactly the same definition as a strong component in a graph where we replaced the notion of a path with a temporal path. In the example in Figure 8, the set $\{A, B, C, D, E\}$ forms a temporal SCC.

As previously mentioned, checking if a graph has a k-node temporal SCC is NP-complete [3, 40]. Nonetheless, we can compute the largest such strong component using a maximum clique algorithm. Let us briefly explain how (see Algorithm 3 for an outline). The first step is to transform the temporal graph into what is called a strong-reachability graph. For each pair of vertices in V, we place an edge in the strong reachability graph if there is a temporal path between them. This is easy to do by using a method developed by [44]. With this reachability graph, the second step of the computation is to remove any non-reciprocated edges and then find a maximum clique. That maximum clique is the largest set of nodes where all pairwise temporal paths exist, and hence, is the largest temporal strong component [40]. Note that the removal of non-reciprocated edges can result in some vertices being singletons, which are in turn removed prior to the computation of the temporal SCC. Thus the vertex set V_R in the reachability graph could be a strict subset of the vertex set V in the temporal graph.

Dataset. We study three types of temporal networks. In each, the nodes

Algorithm 3 Largest Temporal Strong Component.
Input: Temporal Graph $\mathcal{G} = (V, E_T)$

procedure MAX-TSCC($\mathcal{G} = (V, E_T)$)
  E_R = REACH($\mathcal{G}$)
  Remove non-reciprocal edges from E_R
  Obtain V_R by discarding singleton vertices from V
  Compute the MAX-CLIQUE C in the reachability graph (V_R, E_R)
procedure REACH($\mathcal{G} = (V, E_T)$)
  Sort edges to be in reverse time order
  Set E_R to be the set of all self-loops
  for $(i, j, t) \in E_T$ **do**
    Add (i, k) to E_R for all k where $(j, k) \in E_R$
  Return E_R

TABLE 2
For each temporal network, we list the number of temporal edges, the number of vertices and edges in the reachability graph, the size ω of the temporal strong component and the runtime of our maximum clique algorithm.

graph	$\lvert E_T \rvert$	$\lvert V_R \rvert$	$\lvert E_R \rvert$	ω	Time (s.)
INFECT-DUBLIN	415K	11K	176K	84	<.01
INFECT-HYPER	20K	113	6.2K	106	<.01
FB-MESSAGES	61K	1.9K	532K	707	0.05
REALITY	52K	6.8K	4.7M	1236	0.19
RETWEET-ELECT	61K	18K	66K	166	0.02
RETWEET-COPEN	45K	8.6K	474K	581	0.22

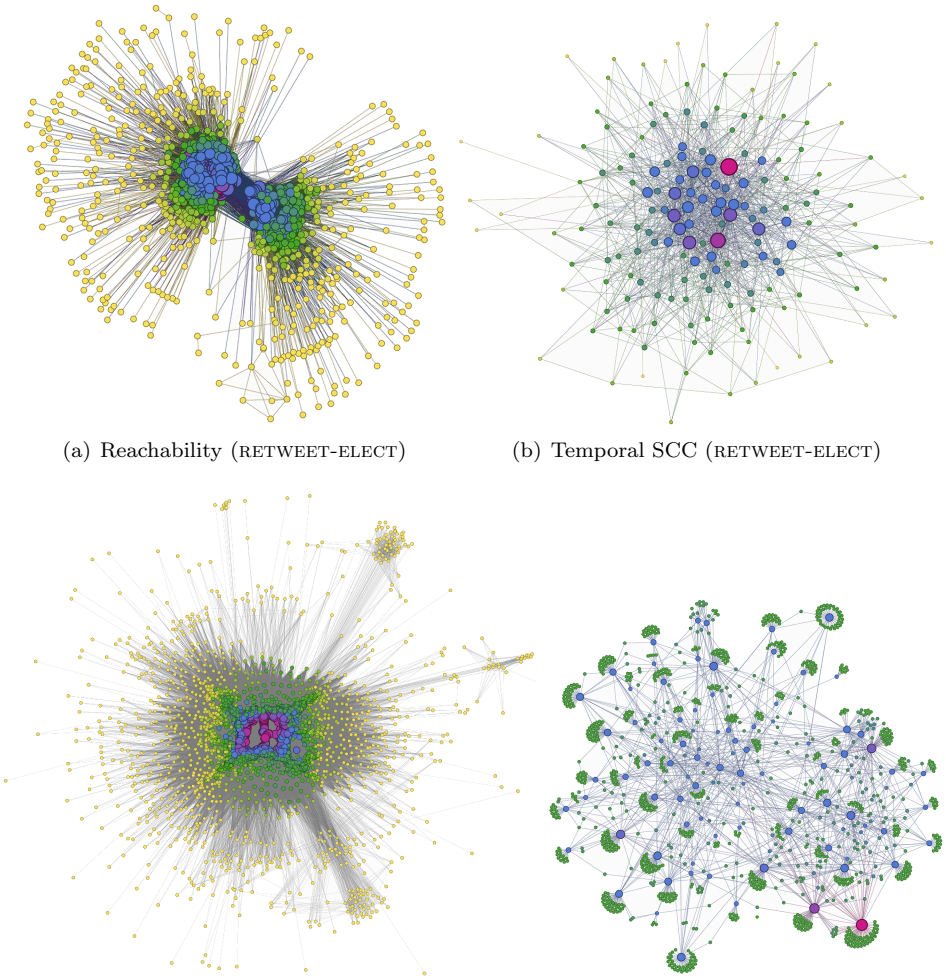
represent people.

1. Contact networks: The edges here represent face-to-face contacts in a social experiment designed to simulate epidemic spreading of a contagious agent (INFECT-DUBLIN, INFECT-HYPER [29]). See ref. [54] for more details about these data.

2. Interaction networks: The edges represent private Facebook messages (FB-MESSAGES [41]) or cellular telephone calls in data gathered at the Reality Mining project at MIT (REALITY [22]).

3. Retweet networks: The edges here are retweets. We analyzed a network of political retweets centered around the November 2010 election in the US (RETWEET-ELECT [16]), and a similar network of retweets about a UN conference held in Copenhagen (RETWEET-COPEN [1]). The latter data was collected over a two week period.

Results and analysis. Figure 9 shows the reachability and largest temporal strong component for the RETWEET-ELECT and REALITY networks. It took the maximum clique finder less than a second to identify these components. We summarize the remaining experiments on the temporal strong components in Table 2. For all of these networks, we were able to identify the largest temporal strong component in less than a second after we computed the reachability network. There are two reasons for this performance. First, in all of the networks except for the interaction networks, the largest clique is the set of vertices with highest core numbers. Second, our heuristic computes the largest clique in all of these networks, and we are able to quickly reduce the remaining search space when it isn't the largest k-core as well.

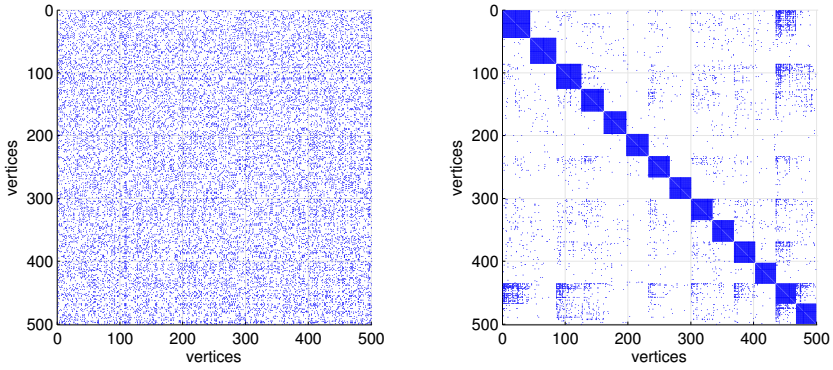


(a) Reachability (RETWEET-ELECT) (b) Temporal SCC (RETWEET-ELECT)

(c) Reachability (REALITY) (d) Temporal SCC (REALITY)

FIG. 9. *Results on computation of temporal strong connected components in the* RETWEET-ELECT *and* REALITY *networks. In order to compute the largest temporal SCC, we first compute the strong reachability network (a, c). These networks are rather dense and often reveal clear community structure. In the* RETWEET-ELECT *network, we see clear communities for the political left and right. We find that the largest temporal SCC in* RETWEET-ELECT *(b) consists of 166 twitter users classified as politically "right" according to the original data with only a single exception. In the largest temporal SCC in the* REALITY *network (d), we see a small group of core users maintaining connectivity among various groups.*

Observations. We observe several interesting properties in these temporal strong components. In the two contact networks (INFECT-HYPER and INFECT-DUBLIN), both of the largest strong components had about 100 vertices, despite the drastically different sizes of the initial dataset. We suspect this is a consequence of the data collection methodology since the INFECT-DUBLIN data were collected over months whereas the INFECT-HYPER data were collected over days. In the interaction networks, the components contain a significant fraction of the total vertices, roughly 20-30%. In the retweet networks, the components are a much smaller fraction of the vertices.



(a) Before (b) After

FIG. 10. *Plots of the nonzero entries of the adjacency matrix of a graph before and after the clique-based ordering of the columns is applied.*

TABLE 3
Size in bytes required to store two Facebook graphs using the bvgraph *compression scheme in three different orders.*

Graph	Vertices	Edges	Native	LLP	PMC
FB-PENN	42K	1.4M	4237507	2740801	3104286
FB-TEXAS	36K	1.6M	4605427	3232909	3508224

Given the strong communication pattern between the groups, the components are good candidates for centers of communities in the networks.

Together these results show that temporal strong components are a strict requirement on a group of nodes in a network. For instance, there is a considerable difference in the size of temporal strong components between networks with asymmetry in the relations (RETWEET-ELECT) compared with networks with symmetric relationships (FB-MESSAGES and REALITY). This finding may be important for those interested in designing seeded viral campaigns on these networks.

6.2. Ordering for network compression. In this application, we consider using the maximum cliques of a network to produce an ordering of the vertices that should be useful for *compression*, reducing the space needed to store the network structure. Compression has two important benefits. First, it reduces the amount of IO traffic involved in using the graph. Second, good compression schemes may reduce the amount of work involved in running an algorithm on the graph [32, 31]. State-of-the-art network compression techniques heavily exploit locality of links within the adjacency list representation of a graph to reduce the number of bits required to store each edge [6, 7, 5]. Cliques are the densest local feature of a graph, and in this application, we order the vertices of a network such that every vertex is in a large clique. This ensures that there are many local edges within the graph. We then evaluate how well the *bvgraph* [6, 7] compression method reduces the graph size using this ordering.

The specific ordering we use is the result of the following process. Given a graph G, we find a maximum clique C in G, remove C from G, and repeat the process until

all vertices are removed. To improve the runtime, we ran our heuristic method to find large cliques. We then order the vertices according to the cliques, C_1, C_2, ..., C_I, where I denotes the number of iterations needed. Internally within each C_i we order the vertices by their degrees. We then permute the graph to use this ordering and use the *bvgraph* compression scheme with all default settings to compress the networks. Table 3 shows the results we get on two Facebook networks (and Figure 10 illustrates the application of the clique-based reordering of the columns of the adjacency matrix of a portion of the FB-PENN network). We compare the compression obtained by reporting the size of each graph in bytes after compressing. We evaluate three orderings of the vertices: the native order, the Layered Label Propagation (LLP) order proposed to help improve compression with the *bvgraph* algorithm [5], and our clique-based order computed using PMC. We find that our ordering results in better compression than using the native ordering of the data and it is comparable to the LLP order although slightly worse. Previous research found that identifying and compressing large bicliques with a linear number of edges helped to improve upon methods that use the adjacency list [10]. Given the success of this simple ordering, we plan to evaluate these more involved schemes in future work.

7. Conclusions. We presented a new fast algorithm that finds the maximum clique on billion-edge social networks in minutes. The algorithm exhibits linear runtime scaling over graphs from a thousand vertices to a hundred million vertices and has good parallelization potential. We applied the algorithm to compute the largest temporal strong components of a dynamic network, which involves finding the largest clique in a static reachability graph, and to obtain an ordering friendly for graph compression. Our hope is that maximum clique will become a standard network analysis measure. Towards that end, we make our software package and related information available online for others to use: the original codes used to generate the results for this paper can be found at `http://www.cs.purdue.edu/~dgleich/codes/maxcliques` and an updated repository is available at `https://github.com/ryanrossi/pmc`.

REFERENCES

[1] N. Ahmed, F. Berchmans, J. Neville, and R. Kompella. Time-based sampling of social network activity graphs. In *SIGKDD MLG*, pages 1–9, 2010.

[2] V. Batagelj and M. Zaversnik. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[3] S. Bhadra and A. Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. *ADHOC-NOW*, pages 259–270, 2003.

[4] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. UbiCrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[5] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*, pages 587–596. ACM, 2011.

[6] P. Boldi and S. Vigna. The Webgraph Framework I: Compression techniques. In *Proceedings of the 13th international conference on the World Wide Web*, pages 595–602, New York, NY, USA, 2004. ACM Press.

[7] P. Boldi and S. Vigna. Codes for the world wide web. *Internet Mathematics*, 2(4):407–429, 2005.

[8] I. Bomze, M. Budinich, P. Pardalos, M. Pelillo, et al. The maximum clique problem. *Handbook of combinatorial optimization*, 4(1):1–74, 1999.

[9] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Comm. ACM*, 16(9):575–577, 1973.

[10] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the international conference on Web search and web data mining (WSDM2008)*, pages 95–106, New York, NY, USA, 2008. ACM.

[11] CAIDA. Skitter. http://caida.org/tools/measurement/skitter/.
[12] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '00, pages 84–95, London, UK, UK, 2000. Springer-Verlag.
[13] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks. *ACM Transactions on Database Systems*, 36(4), 2011.
[14] J. Cheng, L. Zhu, Y. Ke, and S. Chu. Fast algorithms for maximal clique enumeration with limited memory. In *SIGKDD*, pages 1240–1248, 2012.
[15] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: User movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1082–1090, New York, NY, USA, 2011. ACM.
[16] M. Conover, J. Ratkiewicz, M. Francisco, B. Gonçalves, A. Flammini, and F. Menczer. Political polarization on twitter. In *ICWSM*, 2011.
[17] P. G. Constantine and D. F. Gleich. Using polynomial chaos to compute the influence of multiple random surfers in the PageRank model. In A. Bonato and F. C. Graham, editors, *Proceedings of the 5th Workshop on Algorithms and Models for the Web Graph (WAW2007)*, volume 4863 of *Lecture Notes in Computer Science*, pages 82–95. Springer, 2007.
[18] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.
[19] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.
[20] N. Du, C. Faloutsos, B. Wang, and L. Akoglu. Large human communication networks: patterns and a utility-driven generator. In *SIGKDD*, pages 269–278, 2009.
[21] N. Du, B. Wu, L. Xu, B. Wang, and P. Xin. Parallel algorithm for enumerating maximal cliques in complex network. In *Mining Complex Data*, volume 165, pages 207–221. Springer, 2009.
[22] N. Eagle and A. Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.
[23] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. *Algorithms and Computation*, pages 403–414, 2010.
[24] D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. *Experimental Algorithms*, pages 364–375, 2011.
[25] P. Erdös and A. Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Academiae Scientiarum Hungarica*, 17:61–99, 1966.
[26] A. Ferreira. On models and algorithms for dynamic communication networks: The case for evolving graphs. In *ALGOTEL*, 2002.
[27] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *INFOCOM*, pages 1 –9, 2010.
[28] D. F. Gleich. Graph of flickr photo-sharing social network. In *DOI: 10.4231/ D39P2W550*, 2012.
[29] L. Isella, J. Stehlé, A. Barrat, C. Cattuto, J. Pinton, and W. Van den Broeck. What's in a crowd? analysis of face-to-face behavioral networks. *Journal of theoretical biology*, 271(1):166–180, 2011.
[30] H. Jeong, B. Tombor, R. Albert, Z. Oltvai, and A. Barabasi. The large-scale organization of metabolic networks. *Nature*, 407:651–654, 2000.
[31] U. Kang and C. Faloutsos. Beyond'caveman communities': Hubs and spokes for graph compression and mining. In *ICDM*, pages 300–309. IEEE, 2011.
[32] C. Karande, K. Chellapilla, and R. Andersen. Speeding up algorithms on compressed web graphs. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 272–281, New York, NY, USA, 2009. ACM.
[33] S. Khot. Improved inapproximability results for maxclique, chromatic number and approximate graph coloring. In *FOCS*, pages 600–, 2001.
[34] J. Konc and D. Janezic. An improved branch and bound algorithm for the maximum clique problem. *proteins*, 4:5, 2007.
[35] G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222 – 236, 1994.
[36] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.
[37] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *WWW*, pages 641–650, 2010.
[38] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community structure in large networks:

Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[39] C. McCreesh and P. Prosser. Multi-threading a state-of-the-art maximum clique algorithm. *Algorithms*, 6(4):618–635, 2013.

[40] V. Nicosia, J. Tang, M. Musolesi, G. Russo, C. Mascolo, and V. Latora. Components in time-varying graphs. *Chaos*, 22(2):023101, 2012.

[41] T. Opsahl and P. Panzarasa. Clustering in weighted networks. *Social networks*, 31(2):155–163, 2009.

[42] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Disc. Appl. Math.*, 120:197–207, 2002.

[43] G. Palla, I. Farkas, P. Pollner, I. Derényi, and T. Vicsek. Fundamental statistical features and self-similar properties of tagged networks. *New Journal of Physics*, 10(12):123026, 2008.

[44] R. K. Pan and J. Saramäki. Path lengths, correlations, and centrality in temporal networks. *Phys. Rev. E*, 84:016105, Jul 2011.

[45] P. M. Pardalos and J. Xue. The maximum clique problem. *J. of Global Opt.*, 4(3):301–328, 1994.

[46] B. Pattabiraman, M. M. A. Patwary, A. H. Gebremedhin, W. Liao, and A. Choudhary. Fast algorithms for the maximum clique problem on massive sparse graphs. In *WAW 2013, Algorithms and Models for the Web Graph. LNCS 8305.*, pages 156–169, 2013.

[47] P. Prosser. Exact algorithms for maximum clique: A computational study. *arXiv:1207.4616v1*, 2012.

[48] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, and M. M. A. Patwary. Fast maximum clique algorithms for large graphs. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, WWW Companion '14, pages 365–366, Republic and Canton of Geneva, Switzerland, 2014. International World Wide Web Conferences Steering Committee.

[49] P. San Segundo, D. Rodríguez-Losada, and A. Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Comput. Oper. Res.*, 38:571–581, 2011.

[50] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and U. V. Çatalyürek. Streaming algorithms for k-core decomposition. *Proc. VLDB Endow.*, 6(6):433–444, Apr. 2013.

[51] M. Schmidt, N. Samatova, K. Thomas, and B. Park. A scalable, parallel algorithm for maximal clique enumeration. *Journal of Parallel and Distributed Computing*, 69(4):417–428, 2009.

[52] S. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[53] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *PNAS*, 105(35):12763–12768, 2008.

[54] SocioPatterns. Infectious contact networks. http://www.sociopatterns.org/datasets/. Accessed 09/12/12.

[55] E. Tomita and T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. of Global Optimization*, 37(1):95–111, 2007.

[56] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.

[57] A. Traud, P. Mucha, and M. Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 2011.

[58] M. A. Trick and D. S. Johnson, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. AMS, 1996.

[59] WHOIS. Internet routing registries. http://www.irr.net/.

[60] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *EuroSys*, pages 205–218, 2009.

[61] J. Xiang, C. Guo, and A. Aboulnaga. Scalable maximum clique computation using mapreduce. In *Conference on Data Engineering (ICDE)*, pages 74–85. IEEE, 2013.

[62] Y. Xie and P. S. Yu. Max-clique: A top-down graph-based approach to frequent pattern mining. In *ICDM*, pages 1139–1144, 2010.

[63] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 745–754, Dec 2012.

Appendix A. Dataset properties.

All of the tables in this appendix share the same columns. The headers indicate:

$\|V\|$	number of vertices
$\|E\|$	number of edges
$\|T\|$	number of triangles
Δ	maximum degree
d_{avg}	average degree
κ	mean clustering coefficient
T_{max}	maximum number of triangles incident on a vertex
T_{avg}	average number of triangles incident on a vertex
K	core number of the graph
ω	size of maximum clique
t_ω	run time of our exact maximum clique algorithm
$\tilde{\omega}$	size of approximate maximum clique computed by our heuristic
$t_{\tilde{\omega}}$	run time of our heuristic maximum clique algorithm

graph	$\|V\|$	$\|E\|$	$\|T\|$	Δ	d_{avg}	κ	T_{max}	T_{avg}	K	ω	t_ω (sec)	$\tilde{\omega}$	$t_{\tilde{\omega}}$ (sec)
diseasome	516	1188	4080	50	4	0.43	152	7.9	11	11	0.01	11	0.01
yeast	1458	1948	618	56	2	0.05	18	0.4	6	6	0.01	6	0.01
celegans	453	2025	9852	237	8	0.12	870	21.7	11	9	0.01	9	0.01
dmela	7393	25569	8697	190	6	0.01	225	1.2	12	7	0.06	7	0.01
netscience	379	914	2763	34	4	0.43	75	7.3	9	9	0.01	9	0.01
GrQc	4158	13422	143337	81	6	0.63	1179	34.5	44	44	0.01	44	0.01
CondMat	21363	91286	513153	279	8	0.26	1615	24	26	26	0.01	26	0.01
HepPh	11204	117619	10073670	491	20	0.66	39633	899.1	239	239	0.01	239	0.01
AstroPh	17903	196972	4050042	504	22	0.32	11269	226.2	57	57	0.01	57	0.01
dblp2010	226413	716460	4793937	238	6	0.38	5947	21.2	75	75	0.05	75	0.05
citeseer	227320	814134	8139894	1372	7	0.46	5398	35.8	87	87	0.06	87	0.06
MathSciNet	332689	820644	1730334	496	4	0.14	1564	5.2	25	25	0.08	25	0.08
dblp2012	317080	1049866	6673155	343	6	0.31	8345	21	114	114	0.05	114	0.05
hollywood2009	1069126	56306653	14748661845	11467	105	0.31	3977656	13795.1	2209	2209	1.69	2209	1.69
enrononly	143	623	2667	42	8	0.36	125	18.7	10	8	0.01	8	0.01
infecthyper	113	2196	50601	98	38	0.5	1731	447.8	29	15	0.01	15	0.01
infectdublin	410	2765	21342	50	13	0.44	280	52.1	18	16	0.01	16	0.01
emailuniv	1133	5451	16029	71	9	0.17	261	14.1	12	12	0.01	12	0.01
fbmessages	1266	6451	7473	112	10	0.04	242	5.9	12	5	0.01	5	0.01
reality	6809	7680	1200	261	2	0	52	0.2	6	5	0.05	5	0.01
emailEU	32430	54397	146976	623	3	0.03	1615	4.5	23	12	0.02	12	0.01
enronlarge	33696	180811	2175933	1383	10	0.09	17744	64.6	44	20	0.03	20	0.01
wikiTalk	92117	360767	2509401	1220	7	0.05	17699	27.2	59	15	0.09	14	0.02
roadNetPA	1087562	1541514	201336	9	2	0.06	8	0.2	4	3	0.5	3	0.21
roadNetCA	1957027	2760388	361476	12	2	0.06	7	0.2	4	4	0.25	4	0.25
retweet	96	117	36	17	2	0.07	6	0.4	4	4	0.01	4	0.01
twittercopen	761	1029	447	37	2	0.06	27	0.6	5	4	0.01	4	0.01
retweetcrawl	1112702	2278852	525939	5070	4	0	1555	0.5	19	13	0.58	13	0.23

TABLE 4
Biological, Collaboration, Interaction, Road, and Retweet networks

graph	$\|V\|$	$\|E\|$	$\|T\|$	Δ	d_{avg}	κ	T_{max}	T_{avg}	K	ω	t_ω (sec)	$\tilde{\omega}$	$t_{\tilde{\omega}}$ (sec)
routersrf	2113	6632	31212	109	6	0.23	588	14.8	16	16	0.01	16	0.01
WHOIS	7476	56943	2347482	1079	15	0.31	22271	314	89	58	0.09	55	0.01
internetas	40164	85123	189840	3370	4	0.01	8513	4.7	24	16	0.05	16	0.01
p2pgnutella	62561	147878	6072	95	4	0	17	0.1	7	4	0.05	4	0.01
RLcaida	190914	607610	1364412	1071	6	0.06	6044	7.1	33	17	0.13	17	0.05
asskitter	1694616	11094209	86309526	35455	13	0.01	564609	50.9	112	67	1.2	66	0.67
wikiVote	889	2914	6357	102	6	0.13	251	7.2	10	7	0.01	7	0.01
epinions	26588	100120	479100	443	7	0.09	5151	18	33	16	0.02	16	0.01
brightkite	56739	212945	1483224	1134	7	0.11	11517	26.1	53	37	0.05	37	0.01
slashdot	70068	358647	1205643	2507	10	0.03	13382	17.2	54	26	0.06	25	0.02
twitterfollow	404719	713319	88617	626	3	0	1687	0.2	29	6	0.23	6	0.05
gowalla	196591	950327	6819414	14730	9	0.02	93817	34.7	52	29	0.2	29	0.05
youtube	495957	1936748	7331658	25409	7	0.01	151081	14.8	50	16	0.55	14	0.18
youtubesnap	1134890	2987624	9169158	28754	5	0.01	180820	8.1	52	17	0.84	16	0.27
flickr	513969	3190452	176313864	4369	12	0.15	524525	343	310	58	5.2	45	0.22
livejournal	4033137	27933062	250658109	2651	13	0.14	79740	62.1	214	214	2.98	214	2.98
orkut	2997166	106349209	1573931856	27466	70	0.04	1313133	525.1	231	47	48.49	44	14.39
friendster	65608366	1806067135	12521172426	5214	55	0.02	158150	190.8	304	129	1205.47	129	548.04

TABLE 5
Technological and Social networks

graph	$\|V\|$	$\|E\|$	$\|T\|$	Δ	d_{avg}	κ	T_{max}	T_{avg}	K	ω	t_ω (sec)	$\tilde{\omega}$	$t_{\tilde{\omega}}$ (sec)
CMU	6621	249959	6930159	840	75	0.19	24065	1046.7	70	45	0.09	45	0.01
MIT	6402	251230	7111758	708	78	0.18	27788	1110.9	73	33	0.1	32	0.01
UCSB37	14917	482215	9229971	810	64	0.16	16140	618.8	66	53	0.03	52	0.01
Duke14	9885	506437	15427674	1887	102	0.17	41982	1560.7	86	34	0.19	32	0.02
Stanford3	11586	568309	17507610	1172	98	0.16	33177	1511.1	92	51	0.09	51	0.02
UConn	17206	604867	10252326	1709	70	0.13	21515	595.9	66	50	0.07	49	0.02
UCLA	20453	747604	15339651	1180	73	0.14	17534	750	66	51	0.1	49	0.03
OR	63392	816886	10504602	1098	25	0.15	19467	165.7	53	30	0.09	29	0.03
Wisconsin87	23831	835946	14586621	3484	70	0.12	46791	612.1	61	37	0.17	36	0.04
Berkeley13	22900	852419	16108779	3434	74	0.11	69511	703.4	65	42	0.16	42	0.04
UIllinois	30795	1264421	28052775	4632	82	0.14	66178	911	86	57	0.18	56	0.05
Indiana	29732	1305757	28173249	1358	87	0.14	37292	947.6	77	48	0.21	46	0.06
Penn94	41536	1362220	21623388	4410	65	0.1	68097	520.6	63	44	0.24	43	0.06
UF	35111	1465654	36440373	8246	83	0.12	159087	1037.9	84	55	0.28	51	0.07
Texas84	36364	1590651	33535656	6312	87	0.1	141050	922.2	82	51	0.33	49	0.08
Banon	2937612	20959854	155972349	4356	14	0.05	36861	53.1	64	24	5.52	23	2.1
Aanon	3097165	23667394	166819284	4915	15	0.05	50241	53.9	75	25	6.3	23	2.56
uciuni	58790782	92208195	19126557	4960	3	0	7781	0.3	17	6	33.86	6	18.49
polblogs	643	2280	9012	165	7	0.16	392	14	13	9	0.01	9	0.01
google	1299	2773	15204	59	4	0.53	189	11.7	18	18	0.01	18	0.01
edu	3031	6474	30174	104	4	0.27	523	10	30	30	0.01	30	0.01
BerkStan	12305	19500	30915	59	3	0.28	384	2.5	29	29	0.01	29	0.01
webbase2001	16062	25593	63345	1679	3	0.02	1362	3.9	33	33	0.01	33	0.01
spam	4767	37375	387051	477	15	0.15	6273	81.2	36	20	0.04	20	0.01
indochina2004	11358	47606	630234	199	8	0.57	1475	55.5	50	50	0.01	50	0.01
sk2005	121422	334419	2980263	590	5	0.47	3481	24.5	82	82	0.02	82	0.02
arabic2005	163598	1747269	65011059	1102	21	0.95	5884	397.4	102	102	0.03	102	0.03
wikipedia2009	1864433	4507315	6653808	2624	4	0.05	12404	3.6	67	31	1.16	31	0.69
it2004	509338	7178413	1016652636	469	28	0.95	93368	1996	432	432	0.12	432	0.12
uk2005	129632	11744049	2513657160	850	181	1	124251	19390.7	500	500	0.06	500	0.06

TABLE 6
Facebook and Web networks

graph	$\|V\|$	$\|E\|$	$\|T\|$	Δ	d_{avg}	κ	T_{max}	T_{avg}	K	ω	t_ω (sec)	$\tilde{\omega}$	$t_{\tilde{\omega}}$ (sec)
DSJC500-5	500	62624	7856550	286	250	0.5	20527	15713.1	226	13	0.3	12	0.01
MANN-a9	45	918	33732	41	40	0.92	757	749.6	41	16	0.01	16	0.01
brock200-2	200	9876	479688	114	98	0.49	3163	2398.4	85	12	0.01	9	0.01
brock200-3	200	12048	873387	134	120	0.61	5404	4366.9	106	15	0.01	12	0.01
brock200-4	200	13089	1120308	147	130	0.66	7031	5601.5	118	17	0.02	14	0.01
c-fat200-1	200	1534	16230	17	15	0.73	100	81.2	15	12	0.01	12	0.01
c-fat200-2	200	3235	76758	34	32	0.76	429	383.8	33	24	0.01	24	0.01
c-fat500-1	500	4459	55704	20	17	0.74	141	111.4	18	14	0.01	14	0.01
c-fat200-5	200	8473	546252	86	84	0.77	2814	2731.3	84	58	0.01	58	0.01
c-fat500-2	500	9139	247008	38	36	0.76	534	494	36	26	0.03	26	0.01
c-fat500-5	500	23191	1639962	95	92	0.77	3441	3279.9	93	64	0.02	64	0.01
c-fat500-10	500	46627	6696744	188	186	0.77	13609	13393.5	186	126	0.05	126	0.03
gen200-p0-9-55	200	17910	2874231	190	179	0.9	16144	14371.2	167	55	0.03	36	0.01
hamming6-4	64	704	2880	22	22	0.19	45	45	23	4	0.01	4	0.01
hamming6-2	64	1824	92160	57	57	0.9	1440	1440	58	32	0.01	32	0.01
hamming8-4	256	20864	2016000	163	163	0.6	7875	7875	164	16	0.02	16	0.01
hamming8-2	256	31616	7531776	247	247	0.97	29421	29421	248	128	0.01	128	0.01
hamming10-2	1024	518656	519739392	1013	1013	0.99	507558	507558	1014	512	0.36	512	0.19
johnson8-2-4	28	210	1260	15	15	0.43	45	45	16	4	0.01	4	0.01
johnson8-4-4	70	1855	71820	53	53	0.74	1026	1026	54	14	0.01	14	0.01
johnson16-2-4	120	5460	360360	91	91	0.73	3003	3003	92	8	0.04	8	0.01
keller4	171	9435	649791	124	110	0.63	4770	3799.9	103	11	0.01	11	0.01
sanr200-0-7	200	13868	1332702	161	138	0.7	8899	6663.5	125	18	0.06	15	0.01
san200-0-7-1	200	13930	1400736	155	139	0.73	8577	7003.7	126	30	0.01	17	0.01
san200-0-7-2	200	13930	1454424	164	139	0.74	9785	7272.1	123	18	0.02	13	0.02
san200-0-9-1	200	17910	2880723	191	179	0.9	16354	14403.6	163	70	0.01	46	0.01
san400-0-5-1	400	39900	5225496	225	199	0.66	15898	13063.7	184	13	0.01	8	0.01

TABLE 7
DIMACS "Easy" graphs

graph	$\|V\|$	$\|E\|$	$\|T\|$	Δ	d_{avg}	κ	T_{max}	T_{avg}	K	ω	t_ω (sec)	$\tilde{\omega}$	$t_{\tilde{\omega}}$ (sec)
DSJC1000-5	1000	249826	62378964	551	499	0.5	75982	62379	460	15	22.4	13	0.01
MANN-a27	378	70551	26008398	374	373	0.99	69067	68805.3	365	126	0.44	125	0.01
MANN-a45	1035	533115	546656220	1031	1030	1	529012	528170	1013	345	189.49	340	0.13
brock200-1	200	14834	1631100	165	148	0.75	10123	8155.5	135	21	0.14	16	0.01
brock400-3	400	59681	13281468	322	298	0.75	38587	33203.7	279	31	78.18	20	0.01
brock400-1	400	59723	13311033	320	298	0.75	38113	33277.6	278	27	105.82	19	0.01
brock400-4	400	59765	13337715	326	298	0.75	39654	33344.3	278	33	11.91	20	0.01
brock400-2	400	59786	13350273	328	298	0.75	40148	33375.7	279	29	35.2	21	0.01
brock800-3	800	207333	69627666	558	518	0.65	100803	87034.6	484	25	517.39	17	0.01
brock800-1	800	207505	69786183	560	518	0.65	101485	87232.7	488	23	785.4	17	0.01
brock800-4	800	207643	69940425	565	519	0.65	103277	87425.5	486	26	251.93	17	0.01
brock800-2	800	208166	70468920	566	520	0.65	104016	88086.1	487	24	583.42	17	0.01
gen200-p0-9-44	200	17910	2874090	190	179	0.9	16160	14370.5	168	44	0.71	35	0.01
keller5	776	225990	98043630	638	582	0.75	151289	126345	561	27	3362.05	23	0.01
p-hat300-3	300	33390	5664621	267	222	0.76	26733	18882.1	181	36	0.68	27	0.01
p-hat500-2	500	62946	9957924	389	251	0.58	40865	19915.8	171	36	0.12	25	0.01
p-hat500-3	500	93800	27160053	452	375	0.77	77235	54320.1	304	50	41.07	38	0.01
p-hat700-2	700	121728	26656629	539	347	0.58	79243	38080.9	236	44	0.77	36	0.01
p-hat1000-1	1000	122253	9216417	408	244	0.28	22689	9216.4	164	10	0.06	9	0.01
p-hat1000-2	1000	244799	73970454	766	489	0.57	157604	73970.5	328	46	40.65	35	0.01
p-hat1500-1	1500	284923	34315488	614	379	0.29	53239	22877	253	12	0.33	10	0.01
sanr200-0-9	200	17863	2850288	189	178	0.9	15942	14251.4	167	42	16.08	36	0.01
san200-0-9-2	200	17910	2873712	188	179	0.9	15819	14368.6	170	60	0.01	40	0.01
san200-0-9-3	200	17910	2871009	187	179	0.9	15645	14355	170	44	1.29	33	0.01
sanr400-0-5	400	39984	3996342	233	199	0.5	13587	9990.9	178	13	0.07	11	0.01
san400-0-7-1	400	55860	11388942	301	279	0.73	32312	28472.4	262	40	0.02	31	0.01
san400-0-7-2	400	55860	11280099	304	279	0.73	32886	28200.2	260	30	0.04	17	0.01
san400-0-7-3	400	55860	11171766	307	279	0.72	33668	27929.4	254	22	0.58	16	0.01
sanr400-0-7	400	55869	10901049	310	279	0.7	33547	27252.6	259	21	22.9	18	0.01
san400-0-9-1	400	71820	23169036	374	359	0.9	62776	57922.6	345	100	0.31	58	0.01
san1000	1000	250500	86356584	550	501	0.69	100736	86356.6	465	15	0.07	9	0.01

TABLE 8
DIMACS "Hard" Graphs