Fast Maximum Clique Algorithms for Large Graphs

Ryan A. Rossi, David F. Gleich,
Assefaw H. Gebremedhin
Purdue University
Computer Science
{rrossi,dgleich,agebreme}@purdue.edu

Md. Mostofa Ali Patwary
Northwestern University
Electrical Engineering and
Computer Science
mpatwary@eecs.northwestern.edu

ABSTRACT

We propose a fast, parallel maximum clique algorithm for large sparse graphs that is designed to exploit characteristics of social and information networks. Despite clique's status as an NP-hard problem with poor approximation guarantees, our method exhibits nearly linear runtime scaling over real-world networks ranging from 1000 to 100 million nodes. In a test on a social network with 1.8 billion edges, the algorithm finds the largest clique in about 20 minutes. Key to the efficiency of our algorithm are an initial heuristic procedure that finds a large clique quickly and a parallelized branch and bound strategy with aggressive pruning and ordering techniques. We use the algorithm to compute the largest temporal strong components of temporal contact networks.

Categories and Subject Descriptors

G.2.2 [**Graph theory**]: Graph algorithms; H.2.8 [**Database Applications**]: Data Mining

1. DISCUSSION

Our algorithm is a branch and bound method with novel and aggressive pruning strategies. Branch and bound type algorithms for maximum clique explore all maximal cliques that cannot be pruned via search tree optimizations [5, 3, 8, 6, 9]. They differ chiefly in the way the pruning is done. Our algorithm has several distinguishing features. First, it begins by finding a large clique using a near linear-time heuristic; the obtained solution is checked for optimality before the algorithm proceeds any further, and the algorithm is terminated if the solution is found to be optimal. Second, we use this heuristic clique, in combination with (tight) upper bounds on the largest clique, to aggressively prune. The upper bounds are computed at the level of the input graph or individual neighborhoods. Third, we use implicit graph edits and periodic full graph updates in order to keep our implementation efficient. Our algorithm also incorporates other features discussed in the full version of this paper [7].

WWW 2014 Companion, April 7-11, 2014. Seoul, Korea
ACM 0-12345-67-8/90/01.

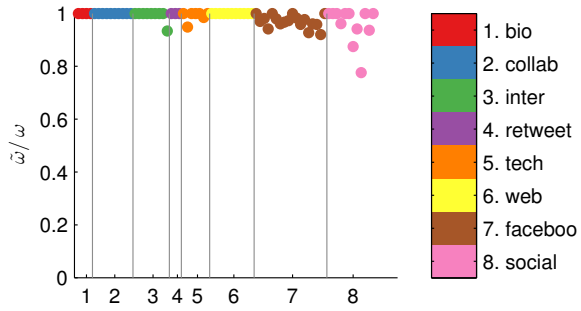

Figure 1: This figure shows the results of the heuristic compared with the largest clique over the 74 networks studied in the full version that come from 8 types of data. We find that our heuristic ($\tilde{\omega}$) finds the largest clique (ω) in biological, collaboration, and web networks in all but one case.

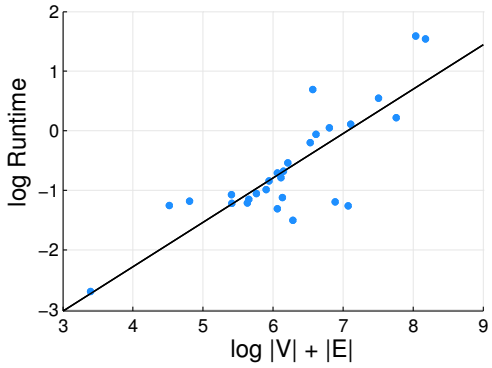

Figure 2: The empirical runtime of our exact clique finder in social and information networks scales almost linearly with the network dimension.

Heuristic. Our 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 larger clique. The order of vertices is the degeneracy order. This heuristic step finds the *largest clique* in the graph in over half of the social networks we consider (see Figure 1). It can therefore be used as a stand-alone procedure (if exact maximum clique is not required).

Bounds. Our branch and bound procedure, Algorithm 2, heavily uses several bounds to prune the search space. A simple upper bound on the size of the largest clique can

be obtained using k-cores, which can be computed in linear time. Let the core-number of vertex v be denoted by $K(v)$. If $K(G)$ is the largest core number of any vertex in G, then $K(G)+1$ is an upper bound on the clique size. Well-known relationships between core numbers, degeneracy order, and coloring allow us to further tighten this bound. Let $L(G)$ be the number of colors used by a greedy coloring algorithm in degeneracy order. Then $L(G) \leq K(G)+1$ and we get a potentially tighter bound on the size of the largest clique, and we have:

FACT 1.1. $\omega(G) \leq L(G) \leq K(G)+1$.

We can further improve the bounds in Fact 1.1 by exploiting the fact that the largest clique must also be in a vertex neighborhood. 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 pruned from the graph yet. All the bounds in Fact 1.1 apply to finding the largest clique in each of these neighborhood subgraphs:

FACT 1.2. $\omega(G) \leq \max_v L(N_R(v)) \leq \max_v K(N_R(v)) + 1$

Our algorithm uses the bounds in Fact 1.2 in its search procedure. Computing these bounds requires slightly more than linear work. In particular, the total work involved in computing the bounds is bounded by $O(|E| + |T|)$, where $|T|$ is the total number of triangles in the graph.

Runtime. We plot the runtime of the exact algorithm (Algorithm 2) pictorially in Figure 2 for a representative subset of 32 of the 74 networks. The figure demonstrates linear scaling between 1000 vertices and 100M vertices. The runtime for the friendster graph with 1.8 billion edges (from the SNAP collection) is 20 minutes. The algorithm is parallelized as described in the full version of this paper [7]. Our source code and additional data on a more extensive collections of networks is available in an online appendix.[1]

Application. To demonstrate the impact on an application, we used our max-clique finder to identify temporal strong connected components (SCC). Temporal SCCs were recently proposed to extend the idea of a strong component to a temporal graph [1, 2]. But, checking if a graph has a k-node temporal SCC is NP-complete. Nonetheless, we can compute the largest such strong component using a maximum clique algorithm. The first step is to transform the temporal graph into what is called a strong-reachability graph. For each pair of vertices in the temporal graph, we place an edge in the strong reachability graph if there is a temporal path between them using a method by [4]. The second step is find a maximum clique in the reachability graph with non-reciprocated edges removed. That maximum clique is the largest temporal strong component [2]. When we try this on reachability graphs with millions of edges, our clique finder takes less than a second.

Algorithm 1 Our greedy heuristic to find a large clique. The input array K holds core numbers of vertices.

```
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 neighs. of v with core numbers ≥ max
6              Set C = {}
7              for each vertex u ∈ S by decreasing core num. do
8                  if C ∪ {u} is a clique then Add u to C
9              if |C| > max then Set H = C, Set max = |H|
10     return H, a large clique in G
```

Algorithm 2 Our exact maximum clique algorithm.

```
1  procedure MaxClique(G = (V, E))
2      Set K = CoreNumbers(G) ▷ K is a vertex-indexed array
3      Set H = HeuristicClique(G, K)
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 degen. order ▷ L is nr of colors
19     if  L ≤ |H| then return
20     Branch({}, P)

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

2. REFERENCES

[1] 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.

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

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

[4] R. K. Pan and J. Saramäki. Path lengths, correlations, and centrality in temporal networks. *arXiv*, page 1101.5913v2, 2011.

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

[6] 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.

[7] R. Rossi, D. Gleich, A. Gebremedhin, M. Patwary, and M. Ali. Parallel maximum clique algorithms with applications to network analysis and storage. *arXiv preprint arXiv:1302.6256*, pages 1–10, 2013.

[8] 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.

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

[1] The appendix is at https://www.cs.purdue.edu/homes/dgleich/codes/maxcliques/ and the code is at http://ryanrossi.com/pmc.