Fast Maximum Clique Algorithms for Large Graphs

Ryan A. Rossi
Purdue University
rrossi@purdue.edu

David F. Gleich
Purdue University
dgleich@purdue.edu

Assefaw H. Gebremedhin
Purdue University
agebreme@purdue.edu

Md. Mostofa Ali Patwary
Northwestern University
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. ALGORITHMS AND APPLICATIONS

We propose a fast, parallel maximum clique finder well-suited for applications involving large sparse graphs. 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 [3, 7, 5, 8]. They differ chiefly in the way the pruning is done. Our algorithm is distinguished by several 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. Additional features the algorithm incorporates are discussed in the full version of this paper [6].

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2577283.

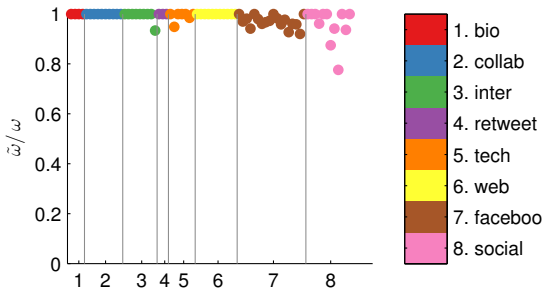

Figure 1: Results of our heuristic on over 74 networks originating from 8 types of data. We find that our heuristic ($\tilde{\omega}$) finds the largest clique (ω) in biological, collaboration, and web networks in all but one case.

Heuristic step. 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 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.

Bounds. Our branch and bound procedure, Algorithm 2, heavily uses several bounds to prune the search space. 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:

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.

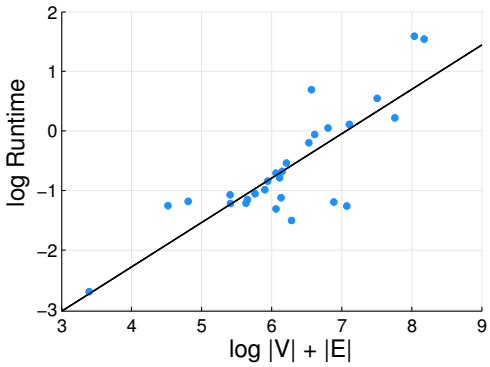

Figure 2: The empirical runtime of our exact clique finder in social and information networks scales almost linearly with the network dimension.

Runtime. We plot the runtime of the algorithm 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 in a shared memory setting using a worker task-queue and a global broadcast [6]. Our source code and additional data on an extensive collections of networks is available in an online appendix.[1]

Application. As a demonstrative application, we use our maximum clique finder to identify temporal strong components (tSCC), a recently introduced notion [1, 2]. A temporal network is defined by a set of vertices V and a temporal set of edges $E_T \subseteq V \times V \times \mathbb{R}^+$ between the vertices. Specifically, each edge (u, v, t) in a temporal network has a unique time $t \in \mathbb{R}^+$. In such a network, a path represents a sequence of edges that must be traversed in increasing order of edge times. If each edge represents a contact between two entities, for example, then a path is a feasible route for information. Two vertices (u, w) are strongly connected if there exists a temporal path $\mathcal{P}$ from u to w and from w to u. And a tSCC is a maximal set of vertices $C \subseteq V$ such that any pair of vertices in C are strongly connected [1, 2]. Checking if a graph has a k-node tSCC is NP-complete. Nonetheless, we can compute the largest tSCC by finding a maximum clique in a derived graph called a *strong-reachability graph* [2]. A strong-reachability graph is obtained by inserting an edge between every pair of vertices in the temporal graph whenever there is a temporal path between them [4]. A maximum clique in the reachability graph, after non-reciprocated edges have been removed, is then the largest temporal strong component [2]. When we apply our maximum clique finder to compute tSCC in reachability graphs with millions of edges, it takes less than a second.

Acknowledgements. This work is supported in part by NSF grants CCF-1149756 and CCF-1218916.

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

procedure HeuristicClique($G = (V, E)$, K)
  Set $H = \{\}$, Set max $= 0$
  for each $v \in V$ in decreasing core number order **do**
    if v's core number is $\geq$ max **then**
      Let S be the neighs. of v with core numbers $\geq$ max
      Set $C = \{\}$
      for each vertex $u \in S$ by decreasing core num. **do**
        if $C \cup \{u\}$ is a clique **then** Add u to C
      if $|C| >$ max **then** Set $H = C$, Set max $= |H|$
  return H, a large clique in G

Algorithm 2 Our exact maximum clique algorithm.

procedure MaxClique($G = (V, E)$)
  Set $K =$ CoreNumbers(G) ▷ K is a vertex-indexed array
  Set $H =$ HeuristicClique(G, K)
  Remove (explicitly) vertices with $K(v) < |H|$
  while $|G| > 0$ **do**
    Let u be the vertex with smallest reduced degree
    InitialBranch(u) ▷ the routine grows H
    Remove u from G
    Periodically, explicitly remove vertices from G
  Return H, the largest clique in G

procedure InitialBranch(u)
  Set $P = N_R(u)$
  if $|P| \leq |H|$ **then return**
  Set $K_N =$ CoreNumbers(P)
  Set $K(P) = \max_{v \in P} K_N(v)$
  if $K(P) + 1 < |H|$ **then return**
  Remove any vertex with $K_N(v) < |H|$ from P
  Set $L =$ Color(P, K_N) in degen. order ▷ L is nr of colors
  if $L \leq |H|$ **then return**
  Branch({}, P)

procedure Branch(C, P)
  while $|P| > 0$ and $|P| + |C| > |H|$ **do**
    Select a vertex u from P and remove u from P
    Set $C' = C \cup \{u\}$
    Set $P' = P \cap \{N_R(u)\}$
    if $|P'| > 0$ **then**
      Set $L =$ Color(P') in natural (any) order
      if $|C'| + L > |H|$ **then** Branch(C', P')
    else if $|C'| > |H|$ **then** ▷ C' is maximal
      Set $H = C'$ ▷ new max clique
      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] 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.

[6] 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.

[7] 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.

[8] 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.