Find Cliques Fast with our Parallel Max-Clique Algorithms for Billion Edge Graphs

Ryan A. Rossi
Purdue University
rrossi@purdue.edu

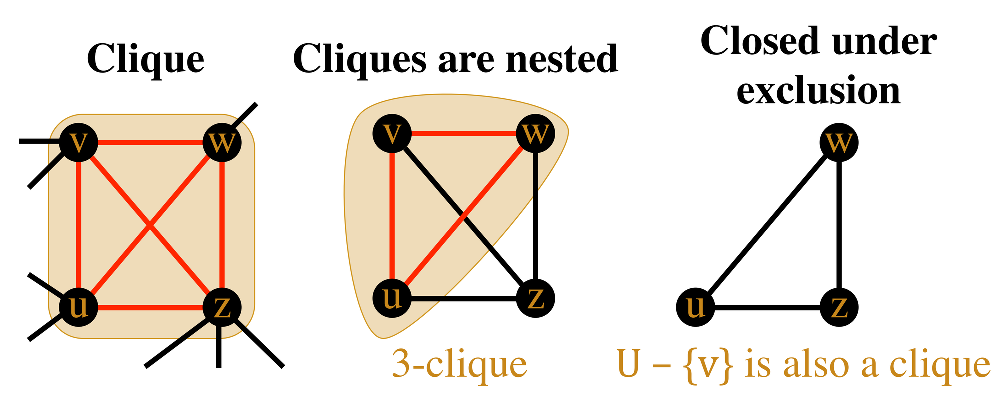
David F. Gleich
Purdue University
dgleich@purdue.edu

Assefaw H. Gebremedhin
Purdue University
agebreme@purdue.edu

Md. Mostofa Ali Patwary
Northwestern University
mpatwary@eecs.northwestern.edu

1. What if CLIQUE were Fast?

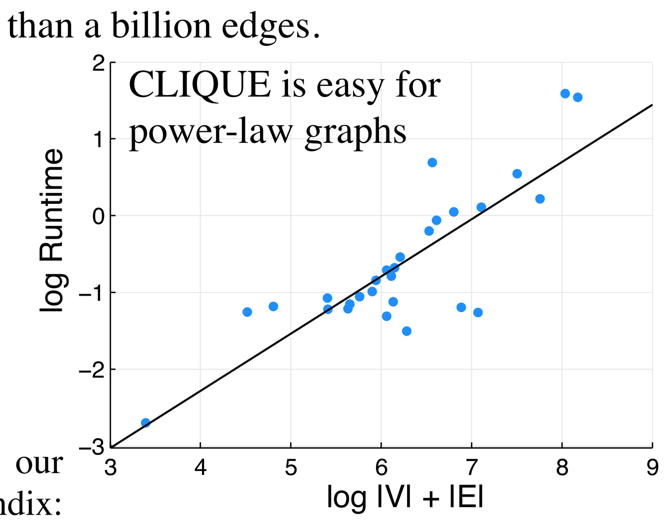
Consider a simple undirected graph G. A clique of size k is a subset of k vertices that forms a complete subgraph. The *maximum clique problem* is to find the largest such k contained in G.



CLIQUE in general is NP-hard, even to approximate it. In this work, we propose a fast, parallel, maximum clique algorithm for large social and information networks. The runtime of our algorithm is shown to be linear in the size of the graph. This holds even for big graphs with more than a billion edges.

This now makes it possible for CLIQUE to be used in tasks such as:

- Analyzing massive networks
- Evaluating graph generation
- Community detection
- Anomaly identification



The CLIQUE problem can be solved in polynomial time for planar and perfect graphs. In this work, we demonstrate that CLIQUE is also easy for **power-law graphs**.

In this spirit, we have released our codes and an online appendix: http://www.cs.purdue.edu/homes/dgleich/codes/maxcliques/

2. Social & Info Networks

Collaboration and web networks: We find that the largest k-core is the clique number, and can be verified by our heuristic!

Technological networks: Surprisingly large maximum cliques given that it indicates an overly large set of redundant edges, suggesting over-built technology, or critical groups of nodes.

Social & FB networks: These networks have the largest difference between the actual clique number and the largest k-core (harder to verify using only our heuristic).

Twitter: The maximum clique is a strange set of spammers and legitimate users (whom likely reciprocate all followers)

Friendster: Our fast heuristic finds the exact clique number, of this large 1.8 billion edge network in only ~500 seconds! Also the exact clique finder only takes 1205 seconds!

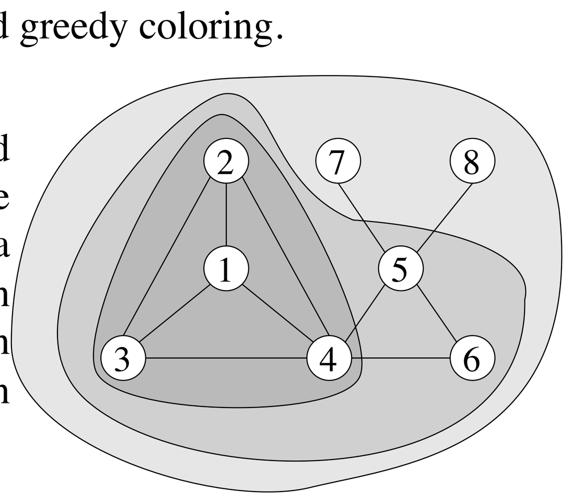
graph	$\lvert V\rvert$	$\lvert E\rvert$	$\tilde{\omega}$	ω	Time	
CELEGANS	453	2.0k	9	9	<.01	bio
DMELA	7.4k	26k	7	7	0.06	
MATHSCIET	333k	821k	25	25	0.08	collab
DBLP	317k	1.0M	114	114	0.05	
HOLLYWOOD	1.1M	56M	2209	2209	1.69	
WIKI-TALK	92k	361k	14	15	0.09	
RETWEET	1.1M	2.3M	13	13	0.58	
WHOIS	7.5k	57k	55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

3. BOUNDS ON CLIQUE SIZE

Our algorithm uses novel bounds for social and information networks, namely, the core numbers and greedy coloring.

K-core bounds

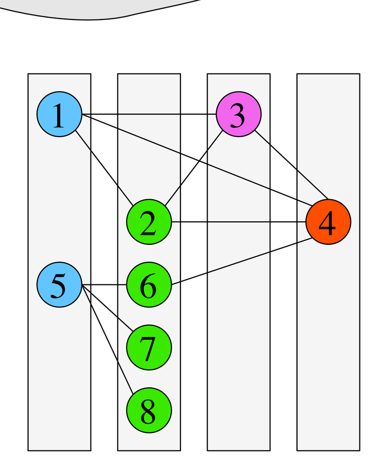
A k-core in G is a vertex induced subgraph where all vertices have degree at least degree k. The core number of a vertex v is the largest k such that v is in a k-core. Let K(G) be the largest core in G, then K(G)+1 is an upper bound on the clique size



Greedy Coloring

Color vertices in order of decreasing core numbers, assigning to each vertex v, the smallest possible integer not yet assigned to one of its neighbors. Let L(G) be the number of colors:

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



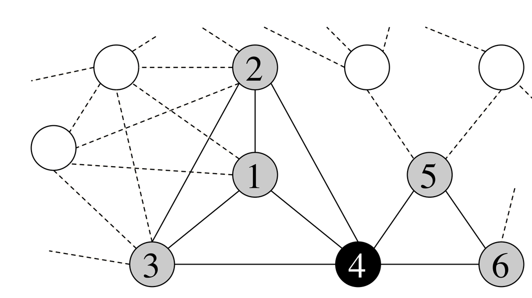
Neighborhood bounds

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

4. Our Maximum Clique Finder

The algorithms search over vertex-induced neighborhoods:

- After searching a vertex it is removed from the graph.
- Clique computations are "independent"

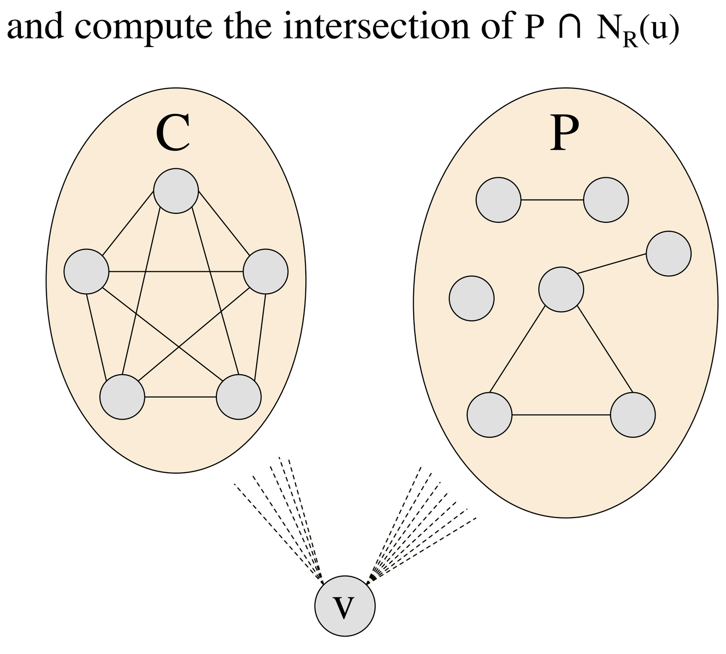


N(4) = {1,2,3,5,6}

1. **Use a fast heuristic to approximate the size of the maximum clique.**
   - **Search ordering.** Our fast heuristic searches vertices by decreasing core number.
   - **Greedy strategy.** For each vertex and its induced neighborhood, we build a clique by greedily adding, at each step, the vertex with largest core number
   - **Pruning.** Since the core numbers are also a lower bound on the size of the largest clique a vertex participates, we can efficiently prune the search space.
2. **Initial pruning.** Once we have a large clique H, we may remove all vertices (and their edges) that have K(v) < |H|.
   - This pruning procedure reduces the memory requirements quite significantly for most networks.
   - In some cases, we find that K(v)+1 = |H| and simply return H.
3. **Order the remaining vertices so that they're searched from smallest to largest degree.**
4. **Compute and prune vertex neighborhood.** While computing each vertex neighborhood, we systematically prune using core numbers and a pruned vertex array X.
   $N_R(v) = G(\{v\} \cup \{u : (u,v) \in E, K(u) \geq \tilde{\omega}, u \notin X\}).$
5. **Compute core numbers of vertex neighborhood.** Afterwards, we set P = $N_R(v)$ and compute core numbers on the reduced neighborhood.
   - Vertices with insufficient neighborhood core numbers are again removed from P.
   - P is also ordered by neighborhood cores.
6. **Greedy coloring.** Using the degeneracy ordering from the neighborhood k-cores, we compute a greedy coloring to obtain an upper bound on the clique size of the neighborhood, which is guaranteed to be at least as tight as the upper bound given by neighborhood cores
7. **Recursively search pruned vertex-neighborhood P**

```
Branch(C,P):
while |P| > 0,
    If |C| + L > |H|,
       Select u from P, remove it, and add it to C
       Set P' to be P ∩ N_R(u)
       If |P'| > 0,
            recolor P' and update coloring number L
            Branch(C,P')
       Else if |C| > |H|, Set H to be C (new max)
       Remove last vertex from C (backtrack)
```

C is the clique being built, whereas P is the set of potential vertices that could be added to C to form a clique of |C|+1. After a vertex u from P is added to C, we must remove it from P and compute the intersection of P ∩ N_R(u)



8. **Explicitly reduce the graph periodically.** This operation reduces the cost of the intersections in the clique search procedure, and also has caching benefits.
9. **Repeat steps 4-8 until all vertex neighborhoods are searched**

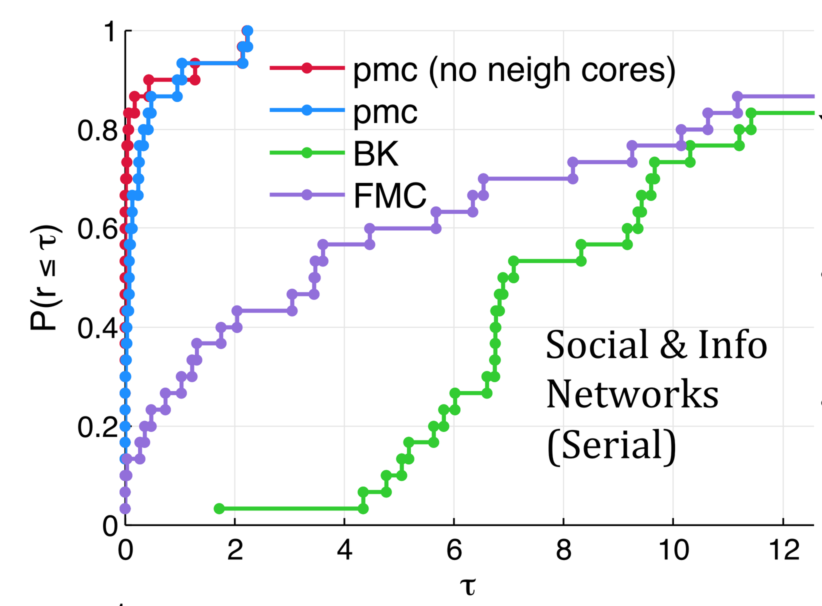
We believe the most important steps are:

- finding a good approximation via the fast heuristic
- searching vertices -- smallest to last ordering
- efficient data structures for all operations and graph updates
- aggressively using k-core bounds and coloring bounds to remove vertices early

5. Performance Results

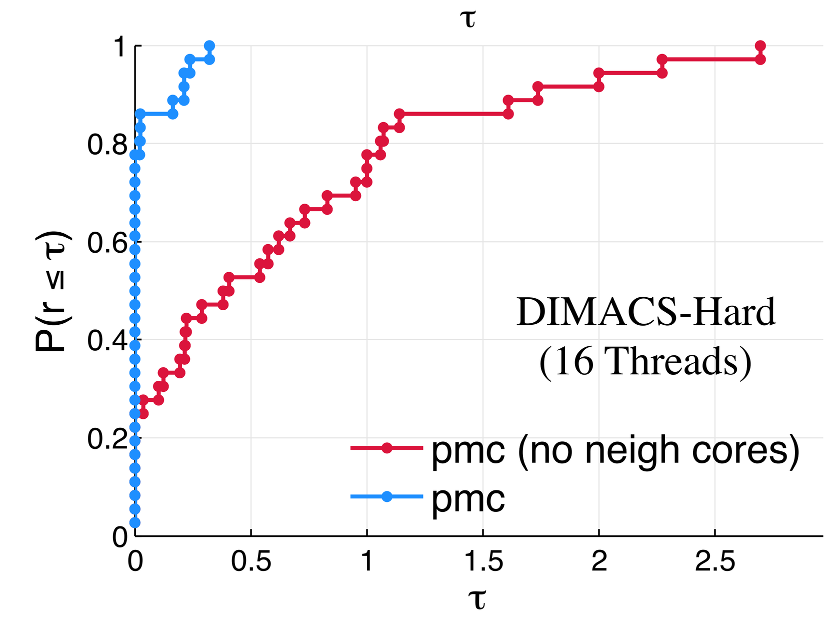
Performance Profiles

Suppose we have N problems, and M of them are solved within 4 times of the best solver, then we'd have a point: $(\tau, p) = (\log_2 4, M/N)$



BK solves only 80% of the problems

- FMC is much better than BK, but not comparable to PMC.
- For most of these networks, PMC with neighborhood cores is only marginally faster.



For the hard DIMAC problems, neighborhood cores improve performance quite significantly.

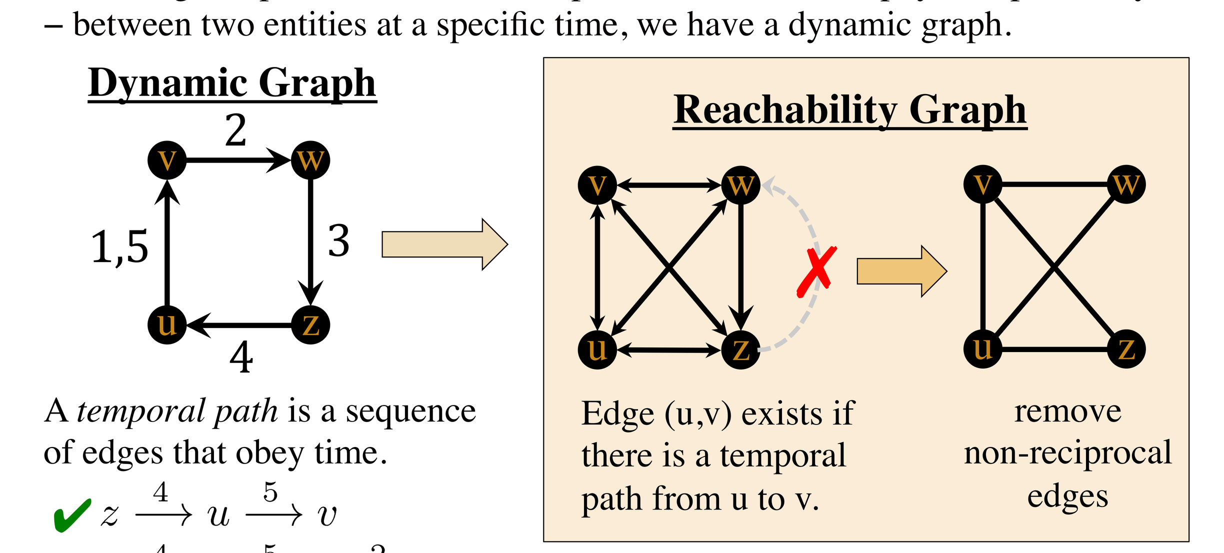
The performance of neighborhood cores in our parallel algorithm is shown to increase compared to the serial version.

Main findings. Our algorithm outperforms the competition dramatically. The neighborhood core bounds help with challenging problems and almost never take more than twice the time.

6. Temporal SCC

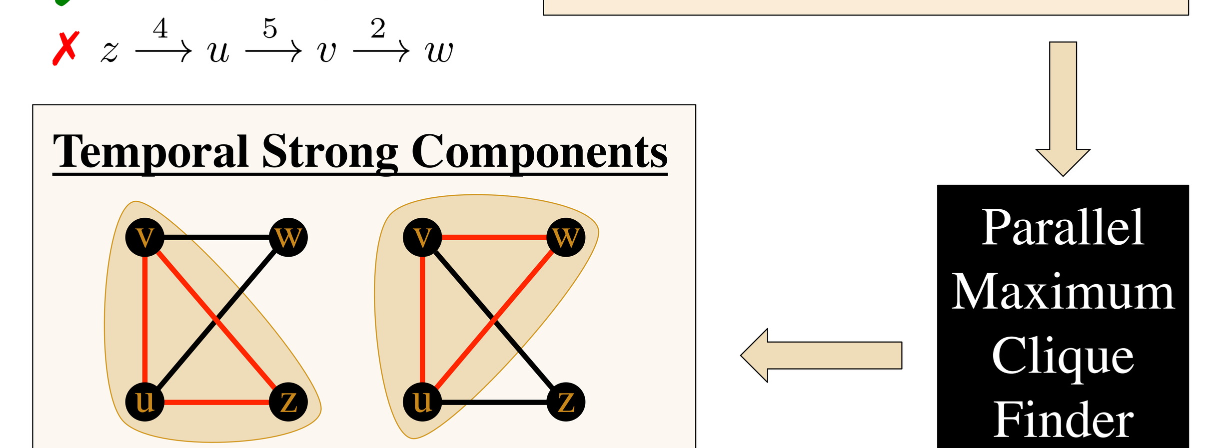
We use our fast maximum clique finder to compute the *largest temporal strong component*, which is known to be an NP-hard problem.

When edges represent a contact – a phone call, email, or physical proximity – between two entities at a specific time, we have a dynamic graph.



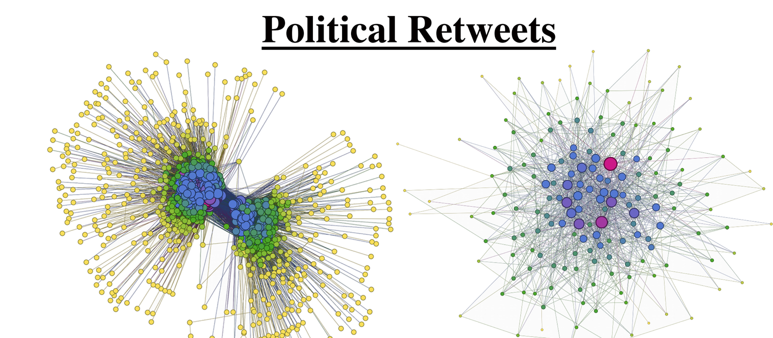
A *temporal path* is a sequence of edges that obey time.

✔ $z \xrightarrow{4} u \xrightarrow{5} v$
✗ $z \xrightarrow{4} u \xrightarrow{5} v \xrightarrow{2} w$



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
ENRON	50k	151	9.8k	120	<.01
FB-FORUM	33k	897	71k	266	0.02
FB-MESSAGES	61k	1.9k	532k	707	0.05
REALITY	52k	6.8k	4.7M	1236	0.19
RETWEET	61k	18k	66k	166	0.02
TWITTER-COP	45k	8.6k	474k	581	0.22
MITTROMNEY	8.5k	7.8k	108	5	<.01
BAHRAIN	8k	4.7k	129	8	<.01
BARACKOBAM	9.8k	9.6k	226	10	<.01

Political Retweets



Reachability graph shows clear communities of the political left and right

Largest Temporal-SCC consists of politically right users.

- In all networks, our algorithm computes the **largest temporal-SCC** in *less than a second*.
- Our fast heuristic finds the largest clique in all these networks

MAIN FINDINGS

- Our algorithm is fast and shown to be effective for many types of graphs, outperforming the competition
- CLIQUE is easy for powerlaw graphs; linear in the number of edges and vertices
- Temporal SCC's are easy to compute in practice
- K-core pruning reduces the search space for sparse networks
- Our parallel algorithms reduces the dependency on the initial ordering of vertices, sometimes giving superlinear speedups