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 $G$.

2. Social & Info Networks

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

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

Social & Info 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 number only takes 1205 seconds!

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 $k$. The $k$-core number of a vertex $v$ is the largest such that $v$ is in a $k$-core. Let $K(G)$ be the largest $k$-core in $G$, then $K(G)$ is an upper bound on the clique size.

Greedy Coloring

Color vertices in ordering of decreasing core numbers, assigning to each vertex $v$, the smallest possible integer not yet assigned to any of its neighbors. Let $L(G)$ be the color number of $G$.

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

4. Our Maximum Clique Finder

The algorithms search over vertices induced by k-cliques.

- After searching a vertex 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 $|H| < |V| - |H|$. This pruning procedure reduces the memory requirements quite significantly for most networks.

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 $X$.

5. Compute core numbers of vertex neighborhood. Afterwards, we save $X$ and compute core numbers on the reduced neighborhood.

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 $P$.

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.
- Greedy coloring - 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

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.

6. Temporal SCC

Branch $(C,P)$:

- If $|C| = |P|$, then $C = P$.
- If $|C| > |P|$, $C = P$, and $|P| = |H|$.
- If $|C| < |P|$, then $P = P'$ and update coloring number $L$.

Branch(C,P):

- Else if $|C| > |B|$, Set $H$ to $C$ (new max)
- Remove last vertex from $C$ (backtrack)
- $L = L(P'') - 1$

C is the clique being built, whereas $P$ is the set of potential vertices that could have been added to $C$ to form a clique of $|C|$. After a vertex $u$ from $P$ is added to $C$, we must remove from $P$ and compute the intersection of $P \cap N[u]$.

7. Temporal SCC Results

- We believe the most important steps are:
- Finding a good approximation via the fast heuristic
- Greedy coloring - 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

- The neighborhood core bounds help with challenging problems and almost never take more than twice the time.

- Our parallel algorithms reduces the dependency on the initial ordering of vertices, sometimes giving superlinear speedups.

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

- 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 the neighborhood cores in our parallel algorithm is shown to increase compared to the serial version.

8. Conclusion

- Our algorithm is fast and shown to be effective for many types of graphs, outperforming the competition.
- CLIQUE is easy for power-law graphs.
- PMC is much better than BK, but not comparable to PMC.
- For most of these networks, PMC with neighborhood cores is only marginally faster.
- Our parallel algorithms reduces the dependency on the initial ordering of vertices, sometimes giving superlinear speedups.

- In all these 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 outperforms the competition dramatically.

The neighborhood core bounds help with challenging problems and almost never take more than twice the time.

- Our parallel algorithms reduces the dependency on the initial ordering of vertices, sometimes giving superlinear speedups.

- We use our 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.

- We believe the most important steps are:
- Finding a good approximation via the fast heuristic
- Greedy coloring - 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

- The neighborhood core bounds help with challenging problems and almost never take more than twice the time.