A Fast Parallel Maximum Clique Algorithm for Large Sparse Graphs and Temporal Strong Components

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. We observe roughly linear runtime scaling over graphs between 1000 vertices and 100M vertices. In a test with a 1.8 billion-edge social network, the algorithm finds the largest clique in about 20 minutes. For social networks, in particular, we found that using the core number of a vertex in combination with a good heuristic clique finder efficiently removes the vast majority of the search space. In addition, we parallelize the exploration of the search tree. In the algorithm, processes immediately communicate changes to upper and lower bounds on the size of maximum clique, which occasionally results in a super-linear speedup because vertices with especially large search spaces can be pruned by other processes. We use this clique finder to investigate the size of the largest temporal strong components in dynamic networks, which requires finding the largest clique in a particular temporal reachability graph.

Categories and Subject Descriptors

G.2.2 [**Graph theory**]: Graph algorithms; [**Theory of computation**]: Dynamic graph algorithms

General Terms

Algorithms, Experimentation

Keywords

maximum clique, parallel algorithms, temporal strong components, social networks, information networks

1. INTRODUCTION

We began studying how to compute cliques quickly in order to compute the largest temporal strong component of a dynamic network [40, 5]. When each edge represents a contact – a phone call, an email, or physical proximity – between two entities at a specific point in time, we have 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 in a graph is a set of vertices where all pairwise paths exist. We present a formal treatment and discuss properties of the temporal strong components we find in Twitter and phone call networks towards the end of the manuscript (Section 6).

Surprisingly, checking if an evolving network has a temporal strong component of size k is NP-complete [40, 5]. For some intuition, we present a "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 is hard [34]. Yet, many real-world problems do not elicit worst-cast behavior from well-designed algorithms.

We propose a state-of-the-art *exact maximum clique finder* and use it to investigate cliques in temporal reachability networks as well as social and information networks. We demonstrate that finding the largest clique in big social and information networks is fast (Table 1). By way of example, we can find the maximum clique in social networks with nearly two billion edges *in about 20 minutes* with a 16-processor shared memory system. In fact, for most social networks, the majority of time is spent doing serialized preprocessing work. Empirically, 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 one of our prior clique finders [47] as well as an off the shelf clique enumerator (Section 5). Consequently, we expect our maximum clique algorithms to be useful for tasks such as analyzing large networks, evaluation of graph generators, community detection, and anomaly detection.

Our algorithm is a branch and bound method with novel and aggressive pruning strategies. Four key components stand out as features contributing to its efficiency. First, the algorithm begins by finding a large clique using a fast heuristic; the obtained solution is checked for optimality, and in fact, in many cases turns out to be optimal. Second, the algorithm uses the heuristic solution, in combination with bounds on the largest clique, in order to guide the pruning strategy. Third, we use implicit graph edits and periodic full graph updates in order to keep our implementation efficient.

Fourth, we parallelize the search procedure in a manner useful for shared memory and distributed computing.

We make our implementation and further experiments available in an online appendix.

http://www.cs.purdue.edu/~dgleich/codes/maxcliques

2. MAXIMUM CLIQUES IN SOCIAL AND INFORMATION NETWORKS

We experiment with 32 networks in 8 broad classes of social and information data. Later, we also consider temporal reachability networks (Section 6). In the online appendix, we present a more extensive collection of around 80 networks. Table 1 describes the properties of the data, shows the size of the largest clique, and states the time taken to find it. We plot the runtime pictorially in Figure 1, which demonstrates linear scaling between 1000 vertices and 100M vertices. Below, we briefly explain the source of the data, what cliques in the data signify, and some interesting observations about cliques in social and information networks.

For all of the following networks, we discard any edge weights, self-loops, and only consider the largest strongly connected component. In contrast to the temporal components we describe later, in this section we mean the standard strong components. If the graph is directed, we then remove non-reciprocated edges. This strategy will identify fully-directed cliques.

1. Biological networks. We study a network where the nodes are proteins and the edges represent protein-protein interactions (dmela [52]) and another where the nodes are substrates and the edges are metabolic reactions (celegans) [20]. Cliques in these networks are biologically relevant modules.

2. Collaboration networks. These are networks in which nodes represent individuals and edges represent scientific collaborations or movie production collaborations (mathscinet [44], dblp, hollywood [7]). Large cliques in these networks are expected because they are formed when collaborations have many participants.

3. Interaction networks. Here, nodes represent individuals and edges represent interaction in the form of message posts (wiki-talk [36]).

4. Retweet networks. In retweet networks, nodes are Twitter users and edges represent whether the users have retweeted each other. We collected this network ourself. Cliques are groups of users that have all mutually retweeted each other and may represent an interest cartel or anomaly.

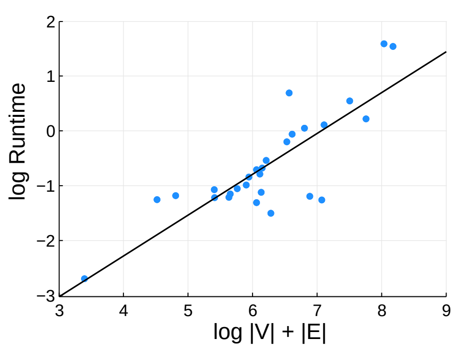


Figure 1: The empirical runtime of our clique finder in social and information networks scales almost linearly with the network dimension.

Table 1: For each of the social and information networks in this study we find the largest clique in less than 21 minutes. The size of an initial heuristic clique is $\tilde{\omega}$ and the actual maximum clique is ω.

graph	$\|V\|$	$\|E\|$	$\tilde{\omega}$	ω	Time (s.)
CELEGANS	453	2.0k	9	9	<.01
DMELA	7.4k	26k	7	7	0.06
MATHSCIET	333k	821k	25	25	0.08
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
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
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
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
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

5. Technological networks. The nodes in these networks are routers (as-skitter, rl-caida [10], whois [61]), and edges are observed communications between the entities.

6. Web link networks. Here, nodes are web-pages and edges are hyperlinks between pages (wikipedia [28], arabic-2005, it-2004, uk-2005 [6]). The largest clique represents the largest set of pages where full pairwise navigation is possible.

7. Facebook networks. The nodes are people and edges represent "Facebook friendships" (CMU, MIT, Stanford, Berkeley, UIllinois, Penn, Texas [58], fb-a, fb-b [62], uci-uni [27]).

8. Social networks. Nodes are again people and edges represent social relationships in terms of friendship or follower (orkut [38], LiveJournal [3], flickr [29], gowalla [13], slashdot [37], youtube [39], twitter [35], friendster [Internet Archive]).

In our investigation of these maximum cliques, we found:

- For the twitter network, the nodes in the largest clique are a strange set of spam accounts and legitimate accounts with thousands of followers and following thousands. We believe that most members of this clique likely reciprocate all follower relationships.
- Technological networks have surprisingly large cliques. Given that a clique represents an overly redundant set of edges, this would suggest that these cliques represent over-built technology, or critical groups of nodes.
- In the online appendix we study the relationship between the largest k-core and the largest clique. In collaboration

networks we find that the largest k-core is a clique for every graph. 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.

- Our heuristic (Section 4.1) found the largest clique in 17 of the 32 instances. In the larger set considered online, it finds the largest in 52 of 72 networks. This property helps the branch and bound algorithm terminate quickly.

3. BOUNDS ON THE CLIQUE SIZE

As a prelude to our algorithm, we review a few easy to derive upper bounds on the size of the largest clique $\omega(G)$. These bounds will allow us to terminate our algorithm once we've found something that hits the upper-bound or stop a local search early because there is no larger clique present.

A simple upper bound on the size of the largest clique is the maximum degree in the graph. Usually, this is too simple to be useful. A stronger bound on the size of the largest clique comes from the k-cores of the network. A k-core in a graph G is a vertex induced subgraph where all vertices have degree at least k [51]. The core number of a vertex v is the largest k such that v is in a k-core. Suppose that G contains a clique of size k, then each vertex in the clique has degree $k-1$ and the entire graph must have a $k-1$-core. Consequently, 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. In contrast to cliques, the core numbers of all vertices in a graph can be computed with a linear time algorithm due to Batagelj et al. [4].

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]. Note that 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. At this point, we have the bounds:

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

We can further improve these bounds by using one additional fact about the largest clique. 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 $N_R(v)$ be the vertex-induced subgraph of G corresponding to v and all neighbors of v that haven't been removed from the graph yet. All the bounds above apply to finding the largest clique in each of these neighborhood subgraphs. We then have:

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

Computing these bounds requires slightly more than linear work. For each vertex, we must 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, the total work in computing these bounds is $O(|E|+|T|)$.

4. OUR MAXIMUM CLIQUE FINDER

The method we employ is a branch and bound strategy. It begins by finding a large clique using a fast heuristic method (Section 4.1). Using the lower bound from the heuristic, we conduct an initial pruning of the graph by utilizing bounds on the largest clique possible at each vertex (Section 4.2). Next, we run the main search loop, which we describe in Section 4.3. We focus our description on how we use the neighborhood information to further prune the search space based on core numbers and coloring. We also mention the order of vertices explored in the search – others found this to be particularly important in clique finders [24]. Finally, we discuss how to parallelize the procedure in Section 4.4 and its performance characteristics in Section 5. For reference throughout this discussion, we give a pseudocode for the method in Figures 2 and 3.

4.1 A fast heuristic

The goal of our heuristic is to find a large clique in the graph quickly. It is similar to previous clique heuristics [47] that explore a neighborhood and pick a clique among high degree vertices. Our heuristic search differs as we use the core numbers of each vertex to guide the search instead. Starting at each vertex in the graph, the algorithm greedily builds a clique by testing vertices in order of their core numbers. 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 the previous section, this heuristic finds the *largest clique* in the graph in over half of the social networks we consider.

4.2 Initial pruning

After our algorithm finds a heuristic clique H 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 elminate all vertices with core numbers strictly less than $\tilde{\omega}$ from our search. 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, this step suffices to certify that H is the maximum clique as we remove all of the graph. This happens, for instance, with LiveJournal. Moreover, this pruning procedure reduces the memory requirements quite significantly for most networks.

Note that in this initial pruning, vertices are explicitly removed from the graph. However, vertices removed in future pruning steps are simply marked as deleted in an index array. Future graph operations, such as neighborhood queries, check this array before returning their contents.

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 algorithm we run is similar to a standard branch and bound scheme for maximal clique enumeration [9]. However, we unroll the first two

Figure 2: Our greedy heuristic to find a large clique.

```
Heuristic(G): returns H, a large clique in G
H = {}, max = 0
for each vertex v in decreasing core number order
  Return if v's core number is less than max
  Let S be the neighs. of v with core numbers > max
  Set C = {}
  for each vertex u in S by decreasing core numbers
    if C ∪ u is a clique, add u to C
  if |C| > max, H = C and max = |H|
```

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. Let $N_R(v)$ (recall this from Section 3) and $d_R(v)$ be the reduced neighborhood graph of v and the reduced degree of v. These sets do not contain any vertices that have been removed from the graph due to changes in the lower-bound on the clique size due to k-cores and any vertices whose local searches have terminated. With a 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:

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

We explore the remaining vertices in order of the smallest to largest reduced degree (see Figure 3, `Clique`, main loop). For each vertex, we explore it's neighborhood using the function `InitialBranch`. After initial branch returns, we have found the largest clique involving that vertex, and so we can remove it from the graph. Again, this is done by marking it as removed in an array. We did, however, find it advantageous to periodically recreate the graph data structure in light of all the edits 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 first step of `InitialBranch` is a test to check if any of our bounds rule out finding a bigger clique in the neighborhood of u. To do so, we compute the core numbers for each vertex in the neighborhood subgraph. If the largest core number in the neighborhood graph is no better than the current lower bound, we immediately return and add the vertex to the list of searched vertices. If it isn't, then we compute a greedy coloring of the subgraph using the degeneracy order. Using the coloring bound from Section 3, we can immediately return if there is no large clique present. If none of these checks pass, then we enter into a recursive search procedure that examines all subsets of the neighborhood in a search for cliques via the `Branch` function.

The `Branch` function maintains a subgraph P and a clique C. The invariant shared by these sets is that we can add any 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 largest color but we do not believe this choice is critical.) We then check if the clique C' is maximal by testing if there is any set P' that exists that satisfies our invariant. If it's a maximal clique, then we check against our current best clique H, and update it if we found a larger clique. If it is not, then we test if it's possible that C' and P' have a big clique. The biggest clique possible is $|C'| + \omega(P') \leq |C'| + L(P')$, and so the function `Recolor` computes a new greedy coloring to get the upper bound

Figure 3: Our maximum clique algorithm. See Section 4.4 for details about how to parallelize it.

```
Clique(G): returns the largest clique in G
K = CoreNumbers(G)
H = Heuristic(G, K)
Remove vertices with K(v) < |H| Explicitly
while |G| > 0
  Let u be the vertex with smallest reduced degree
  InitialBranch(u,G)
  Remove u from G
  Periodically, explicitly remove vertices from G.
```

```
InitialBranch(u, G): Check bounds and start at u
Set P = N_R(u) and return if |P| ≤ |H|.
K_N = CoreNumbers(P) set K(P) = max_{v∈P} K_N(v)
If K(P) + 1 ≤ |H|, return
Remove any vertex with K_N(v) ≤ |H| from P
L = GreedyColoring(P, K_N)
if L ≤ |H|, return
Branch({}, P)
```

```
Branch(C, P): Explore cliques with C and P
while |P| > 0 and |P| + |C| > |H|
  Select a vertex u from P and remove u from P
  C' = C ∪ {u}
  P' = P ∩ {N_R(u)}
  if |P'| > 0
    Recolor(P') and set L to the coloring number
    if |C'| + L > |H|, Branch(C',P')
  else if |C'| > |H| C' is maximal
    Set H to C' new max clique!
    Remove any v with K(v) < |H| from G. Implicitly
```

$L(P')$. Unlike the greedy coloring above, 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'.

We stress that it becomes tedious to detail every relevant performance enhancement inside our implementation and refer interested readers to the code itself. To give a small example, we use an adjacency matrix structure for small graphs in order to facilitate $O(1)$ edge checks. We use a fast $O(d)$ neighborhood set intersection procedure, and have many other optimizations throughout the code.

In the overall procedure, we believe the following steps are most important:

- finding a good initial clique via our heuristic,
- 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.

4.4 Parallelization

As mentioned before, we have parallelized the clique search procedure. Our own implementation uses shared memory, but we'll describe the parallelization procedure such that it could be used with a distributed memory architecture as well. The parallel constructs we utilize 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 heuristic clique. Here, we assume that each worker has a copy of the graph and distribute vertices to workers to find the best 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's often possible for that worker to terminate its own search at once. In this manner, the speedup from our parallel algorithm can be *super linear* since we are less dependent on the precise order of vertices explored – something that is known to affect clique finders greatly. In our own shared memory implementation, we avoid some of the communications by using global arrays and locked updates.

5. PERFORMANCE RESULTS

As we illustrated in Table 1 and Figure 1, the runtime of our clique finder on social and information networks is fast, and it exhibits roughly linear scaling as we increase the problem size. We used a two processor, Intel E5-2760 system with 16 cores and 256 GB of memory for those tests and the remaining tests. None of the experiments came close to using all the memory.

In this section, we will be concerned with three questions. Does our parallelization scheme work? How does our method compare to other clique finders on social and information networks? And finally, is the tighter upper bound that results from using neighborhood cores worth the additional expense? In the following, we call our own algorithm from Figures 2 and 3 "pmc." The version without neighborhood cores uses the same strategy, but eliminates lines 3–6 of the `InitialBranch` function. Instead of using the degeneracy ordering to compute the `GreedyColoring`, we use a largest degree-first ordering.

For these results, we will use problems from the 20 year old DIMACS clique challenge [59] to study the performance of our clique finder on an established benchmark of difficult problems. In the interest of space, we do not present individual data on them. These graphs are all small: 45—1500 vertices. However, they contain an enormous number of edges and triangles compared with social networks. The number of triangles ranges between 34,000 and 520 million. Of the 57 graphs our method was able to solve, we divide them into an easy set of 26 graphs, where our algorithm terminates in less than a second and a hard set of 32 graphs which take between one second and an hour.

Parallel Speedup. In Figure 4 we show the speedup as we add processes to our pmc method for three social networks. In Figure 5 we show speedup results of pmc for 7 of the DIMACS networks. The runtime for both includes all the serialized preprocessing work, such as computing the core numbers initially. The figures illustrate two different behaviors. For social networks, we only get mild speedups on 16-cores for the largest problem (soc-orkut). For the DIMACS graphs, we observe roughly linear and, occasionally, super-linear performance increases as we increase the number of processes. These results indicate our parallelization works well and helps reduce the runtime for difficult problems.

Performance Profile Plots. For the two remaining questions, we use a performance profile plot to compare algorithms [17]. These plots compare the performance of multiple algorithm on a range of problems. They are similar to ROC curves in that the best results are curves that lie towards the upper left. Suppose we have N total problem 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 $(\tau, p) = (\log_2 4, M/N)$. Note that the horizontal axes reflects a speed difference factor of 2^τ. The fraction of problems that an algorithm cannot solve is given by the right-most point on the curve. In Figure 6(a), for instance, the Bron-Kerbosch (BK) algorithm only solves around 80% of the problems in the test set.

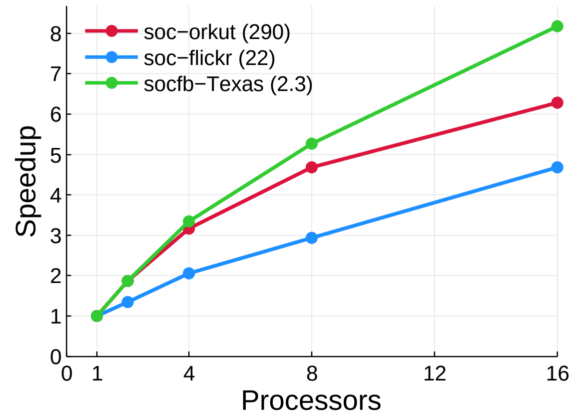


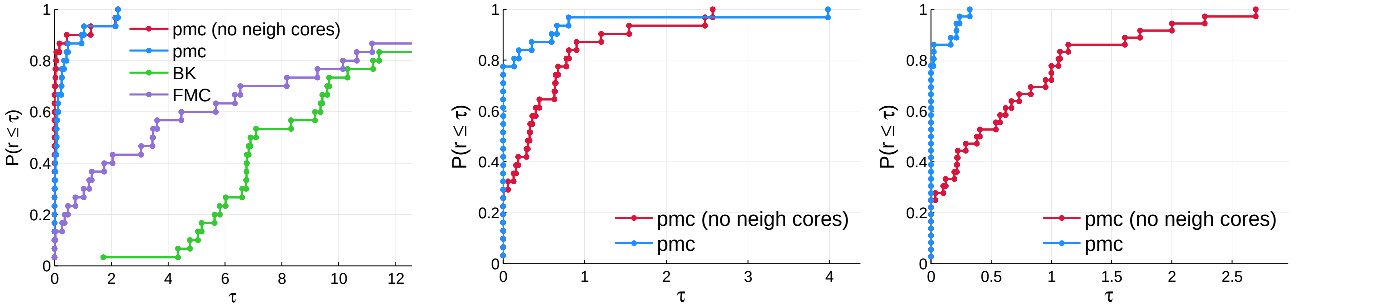
Figure 4: Speedup of our parallel maximum clique algorithm on social and information networks. We only see mild speedup because the majority can be solvedwithin seconds using the sequential algorithm. Single process runtime in parentheses.

Figure 6(a), social and information network. We compare pmc, with and without neighborhood cores, to an implementation of the Bron-Kerbosch (BK) algorithm in the `igraph` network analysis package [16] and to one of our prior algorithms (fmc) [47]. Based on the interpretation of the performance profile plot, we find little difference between using the neighborhood cores and eliminating them – although it is a bit faster without them. However, we find a big difference between our clique finder and the alternatives. Compared to the BK algorithm, we are over 1000 times faster for some problems, and we solve all of the instances. Compared to the fmc algorithm, we are about 50 times faster. This illustrates that our algorithm uses properties of the social and information networks to hone in on the largest clique quickly.

Figures 6(b) and 6(c), DIMACS networks. On the 32 hard instances of DIMACS problems, in the serial case, the neighborhood cores greatly help reduce the work in the majority of cases. In a few cases, they resulted in 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.

Summary of results.

· our parallelization strategy is effective,



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

Figure 6: Performance Profiles. Comparing PMC to a version without the neighborhood core pruning and ordering.

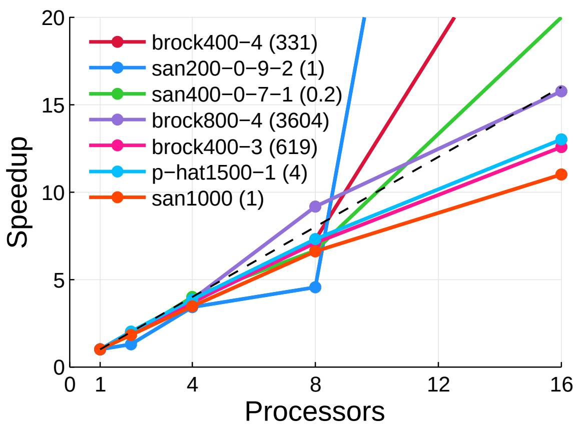


Figure 5: Speedup of our parallel maximum clique finder compared to a sequential version. Notably, superlinear performance arises in some cases. For example, the maximum clique of san200-0-9-2 is computed 82x faster using our parallel algorithm. The superlinear performance arises in our parallel maximum clique algorithm by reducing the dependence on the ordering of vertices by exploring multiple vertices simultaneously while cooperating by updating bounds and sharing other information. Single process runtime in parentheses.

· our algorithm outperforms the competition dramatically,
· neighborhood core bounds help with 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.

6. TEMPORAL STRONG COMPONENTS

In this section, we use the maximum clique method as a subroutine to compute temporal strong components [40]. Since this area is somewhat new, we review some of the basic definitions in order to motivate the relationship between cliques and temporal strong components.

Definition 6.1 (temporal network). *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.

Definition 6.2 (temporal paths). *A temporal path from u to w in $G = (V, E_T)$ is a sequence of edges $e_1, \ldots, e_k$ such that the $e_1 = (u, v_1, t_1), \ldots, e_k = (u_k, w, t_k)$ where $v_j = u_{j+1}$ and $t_j < t_{j+1}$ for all $j = 1$ to k. We say that u is temporally connected to w if there is such a temporal path.*

This definition echoes the standard definition of a path, but adds the additional constraint that paths must follow the directionality of time. Temporal paths are inherently asymmetric because of the directionality of time.

Definition 6.3 (temporal strong component). *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 (tSCC) is defined as a maximal set of vertices $C \subseteq V$ such that any pair of vertices in C are strongly connected.*

While the vertices of a strong component in a graph define an equivalency class, and hence, we can partition a network into components, the same fact is not true of temporal strong components. The vertices in a temporal strong component can overlap with those in another temporal strong component.

Note that a temporal weak component is always equal to the connected component in the static graph [54]. We conclude by mentioning that stronger definitions of temporal components exist. For example, the temporal paths used to define a strong component can be further restricted to only use vertices from the component C itself.

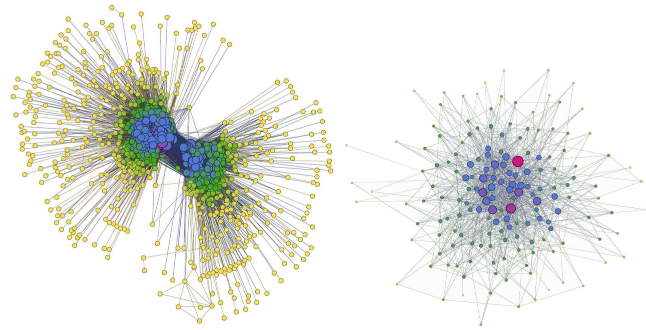
As previously mentioned, checking if a graph has a k-node temporal SCC is NP-complete [5, 40]. Nonetheless, we can compute the largest such strong component using a maximum clique algorithm. Let us briefly explain how.

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. In Algorithm 1, this corresponds to the REACH subroutine. That method uses the temporal ordering proposed by [45] and builds up temporal paths backwards in time. 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].

Algorithm 1 Largest Temporal Strong Component

```
procedure MAX-TSCC(𝒢 = (V, E_T))
    E_R = REACH(𝒢)
    Remove non-reciprocal edges from E_R
    Compute the MAX-CLIQUE in the graph (V, E_R)
    Return the subgraph of G induced by C

procedure REACH(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) ∈ E_T do
        Add (i, k) to E_R for all k where (j, k) ∈ E_R
    return E_R
```



(a) Reachability (retweet) (b) Temporal SCC (retweet)

Figure 7: In order to compute the largest temporal strong component, we first compute the strong reachability network (a). These networks are rather dense and often reveal clear community structure. Here we see a clear communities for the political left and right. We find that the largest temporal strong component in the retweet network *(b)* consists of 166 twitter users classified as politically "right" according to the original data with only a single exception.

6.1 Temporal Graph Data.

We use three types of temporal graph data. For all networks, we discard self-loops and any edge weights. In all cases, the nodes represent people. And in all cases, the largest temporal strong components reflect groups of people that meet, interact, or retweet with each other sufficiently often to transmit any message or meme.

Contact networks. The edges are face-to-face contacts (infect-dublin, infect-hyper[32]). See ref. [53] for more details about these data.

Interaction networks. The edges are observed interactions such as forum posts (fb-forum [41]), private messages (fb-messages [42]), or emails (enron [14]). We also investigate a cellular telephone call network where the edges are calls (reality [21]).

Retweet networks. Here, the edges are retweets. We analyzed a network of political retweets centered around the November 2010 election in the US (retweet [15]). A similar dataset is a retweet and mentions network from the UN conference held in Copenhagen. The data was collected over a two week period (twitter-copen [1]).

Hashtab networks. Again the edges are retweets associated with a hashtag. We collected these from Truthy [60] on September 20th, 2012. Of these, only a few had non-trivial temporal strong components which we discuss below.

6.2 Results and analysis

Figure 7 shows the reachability and largest temporal strong component from the retweet network about politics. It took the maximum clique finder less than a second to identify this clique. 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 quickly able to reduce the remaining search space when it isn't the largest k-core as well.

There are a few interesting properties of these temporal strong components. In the 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 property of the data collection methodology because the infect-dublin data were collected over months, instead of days for the infect-hyper. In the interaction networks, the temporal components contain a significant fraction of the total vertices, roughly 20-30%. In the retweet networks, these temporal strong components are a much smaller fraction of the total vertices in the graph. Given the strong communication pat-

Table 2: For each temporal network, we list the number of temporal edges, number of vertices and edges in the reachability graph, the size of the temporal strong component, and the compute time for the max clique.

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
ALWEFAQ	7.1k	4.2k	355	16	<.01
JUSTINBIEB	9.6k	9.4k	442	17	<.01
OCCUPYWALL	3.9k	3.6k	931	18	<.01
GMANEWS	8.8k	8.3k	1.1k	22	<.01
LOLGOP	10k	9.7k	4.5k	42	<.01

tern between these groups, they are good candidates for the centers communities in these networks. Finally, the hashtag networks have small temporal strong components. Only a tiny fraction of the total users participate in them.

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 (retweets and hashtags) compared with networks with symmetric relationships (fb-forum, fb-messages, and reality). This finding may be important for those interesting in designing seeded viral campaigns on these networks.

7. RELATED WORK

A related problem to maximum clique is maximal clique enumeration: identifying *all* the maximal cliques in G. This problem tends to get more attention in data mining literature [63, 12]. For instance maximal cliques in social networks are distributed according to a power-law [18]. There is a considerable body of work of recent work on this problem [57, 24, 23, 50, 11]. In particular, Du et al. takes advantage of the properties of social and information networks in order to enumerate all maximal cliques faster [19]. In comparison, we wish to highlight how fast we can solve the maximum clique problem for these networks and temporal strong components by appropriately applying pruning steps and bounds. Many of our bounds take advantage of egonet properties [2, 30].

In terms of other maximum clique algorithm, Pardalos and Xue [46] provide a good review of exact algorithms prior to 1994. Notable methods proposed later include: among others, the works of Bomze *et al.* [8], Östergård [43], Tomita *et al.* [56], and San Segundo *et al.* [49]. In a very recent work, Prosser [48] provides a computational study comparing various exact algorithms for maximum clique.

Finding a maximum clique in G is equivalent to finding maximum independent set in the complement of G [55, 33]. Thus, known approximation results on the independent set problem [31] can be related to the maximum clique problem. Greedy strategies – similar in spirit to the fast heuristic we use here – are also effective on those problems too. However, the complement graph for the majority of these networks will be incredibly dense as the original graphs are very sparse.

8. CONCLUSIONS

We propose a new fast algorithm that finds the maximum clique on billion-edge social networks in minutes. It exhibits linear runtime scaling over graphs from 1000 vertices to 100 million vertices and has good parallelization potential. We created it in order to compute the largest temporal strong components of a dynamic network, which involves finding the largest clique in a static reachability graph. Our hope is that that maximum clique will now become a standard network analysis measure. Towards that end, we make our software package available for others to use:

http://www.cs.purdue.edu/~dgleich/codes/maxcliques

We are also investigating a modification of our code to enumerate maximum cliques (not maximal!). This problem has only recently been studied [22]. We find that this can be done almost as efficiently as finding the largest maximum clique and believe that the set of maximum cliques should prove useful for community detection and anomaly detection on networks.

9. 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] L. Akoglu, M. McGlohon, and C. Faloutsos. oddball: Spotting anomalies in weighted graphs. In *Advances in Knowledge Discovery and Data Mining*, volume 6119, pages 410–421. Springer, 2010.

[3] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *SIGKDD*, pages 44–54, 2006.

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

[5] 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.

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

[7] 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, 2011.

[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] CAIDA. Skitter. http://caida.org/tools/measurement/skitter/.

[11] 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.

[12] J. Cheng, L. Zhu, Y. Ke, and S. Chu. Fast algorithms for maximal clique enumeration with limited memory. In *SIGKDD*, pages 1240–1248, 2012.

[13] E. Cho, S. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *SIGKDD*, pages 1082–1090, 2011.

[14] W. Cohen. Enron email dataset. http://www.cs.cmu.edu/~enron/. Accessed in 2009.

[15] M. Conover, J. Ratkiewicz, M. Francisco, B. Gonçalves, A. Flammini, and F. Menczer. Political polarization on twitter. In *ICWSM*, 2011.

[16] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[17] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[18] 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.

[19] 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.

[20] J. Duch and A. Arenas. Community identification using extremal optimization phys. *Rev. E*, 72:027104, 2005.

[21] N. Eagle and A. Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.

[22] J. D. Eblen, C. A. Phillips, G. L. Rogers, and M. A. Langston. The maximum clique enumeration problem: algorithms, applications and implementations. In *ISBRA*, pages 306–319, 2011.

[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. Gleich, P. Constantine, A. Flaxman, and A. Gunawardana. Tracking the random surfer: empirically measured teleportation parameters in PageRank. In *WWW*, pages 381–390, 2010.

[29] D. F. Gleich. Graph of flickr photo-sharing social network crawled in may 2006. In *DOI: 10.4231/D39P2W550*, 2012.

[30] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *SIGKDD*, pages 597–605, 2012.

[31] M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18:145–163, 1997.

[32] 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.

[33] R. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *JACM*, 32(4):762–773, 1985.

[34] S. Khot. Improved inapproximability results for maxclique, chromatic number and approximate graph coloring. In *FOCS*, pages 600–, 2001.

[35] 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.

[36] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *WWW*, pages 641–650, 2010.

[37] 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.

[38] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *SIGCOMM*, pages 29–42, 2007.

[39] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *IMC*, 2007.

[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. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 2011.

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

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

[44] 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.

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

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

[47] B. Pattabiraman, M. M. A. Patwary, A. H. Gebremedhin, W. keng Liao, and A. Choudhary. Fast algorithms for the maximum clique problem on massive sparse graphs. *arXiv:1209.5818v2*, 2012.

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

[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] 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.

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

[52] 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.

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

[54] J. Tang, M. Musolesi, C. Mascolo, and V. Latora. Characterising temporal distance and reachability in mobile and online social networks. *SIGCOMM Computer Communication Review*, 40(1):118–124, 2010.

[55] R. Tarjan and A. Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, 6(3):537–546, 1977.

[56] 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.

[57] 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.

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

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

[60] Truthy. Information diffusion research at indiana university. http://truthy.indiana.edu/. Accessed 10/20/12.

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

[62] 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.

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