Toward an Efficient, Highly Scalable Maximum Clique Solver for Massive Graphs

Ronald D. Hagan[1], Charles A. Phillips[1], Kai Wang[1], Gary L. Rogers[2], Michael A. Langston[1]

[1]Department of Electrical Engineering and Computer Science,
University of Tennessee, Knoxville, TN, USA.

[2]National Institute for Computational Sciences,
University of Tennessee, Oak Ridge, TN, USA.

***Abstract*— As the size of available data sets grows, so too does the demand for efficient parallel algorithms that will yield the solution to complex combinatorial problems on graphs that may be too large to fit entirely in memory. In previous work, we have provided a set of out-of-core algorithms to solve one of the central examples of such a problem, maximum clique. In this paper, we review the algorithms and report on our ongoing work to use them as a starting point for an optimized, highly scalable implementation of a maximum clique solver.**

Keywords—big data; parallel graph algorithms; out-of-core; maximum clique

I. INTRODUCTION

Ongoing advances in data collection technologies have led to an explosive growth in the size of data sets available to researchers in various domains. One need only look to transcriptomic data analysis, where the development of array platforms capable of targeting sequences at the exon level has pushed the size of the associated correlation graphs from around 30 thousand vertices and millions of edges to the order of 1.2 million vertices and trillions of edges. Further complicating the analysis of truly big data graphs, many of the problems of interest are difficult to solve on graphs of even moderate size.

One of the central such problems is the much-studied maximum clique problem. A clique in a graph is a complete subgraph, i.e., a subgraph which is missing none of its possible edges. The problem of determining if a graph has a clique of size at least k is well-known to be $\mathcal{NP}$-complete. We are concerned here with the $\mathcal{NP}$-hard optimization version. That is, we seek the largest k for which the decision version returns "yes."

A collection of parallel algorithms for solving the maximum clique problem on graphs too large to fit into core memory was presented in [1]. At the time, the primary focus was on providing a practical approach to handling the immense size of the graphs. These algorithms used the existing serialized code, Maximum Clique Solver (MCS) [2], as the basis to solve the maximum clique problem on datasets that were too large to store in core memory as a bit adjacency matrix.

Although the algorithms were written for distributed memory parallel systems, the initial versions focused on writing parallel wrappers around serialized code, rather than optimizing performance via parallel examination of the search space. While other authors have recently introduced high-performance parallel algorithms for maximum clique, we believe that there is significant room for improvement to existing techniques. A MapReduce-based implementation is presented in [3], but it suffers from quadratic space overhead, since it divides the problem into subgraphs, each pair of which requires an overlap graph. It also suffers reduced flexibility compared to being written using MPI. The algorithms presented in [4] provide good scalability to large graphs, but are tuned to leverage aggressive pruning techniques targeted specifically to structural characteristics common to social and information networks. In previous work on big data clique solutions, little attention has been devoted to problem reduction through out-of-core preprocessing.

In section II, the existing parallel algorithms are reviewed. To illustrate the potential of our algorithms to operate on big data graphs, the results of using the EOC algorithms on two large datasets are presented in section III. Finally, in section IV, the ongoing research towards creating an optimized, highly scalable implementation of a parallel maximum clique solver is presented.

II. PARALLEL ALGORITHMS

The first of our parallel algorithms, termed Edge-in-Core (EIC), is capable of handling a graph too large to be held as an adjacency matrix in core memory, but whose edge list can be stored in core memory. The second algorithm, termed Edge-Out-of-Core (EOC), can deal with graphs so large that the edges cannot be stored as an adjacency list in main memory (although the node list must still be able to fit in core). Both algorithms were implemented with MPI using a master-worker paradigm.

The two algorithms operate in the same basic manner, taking a graph too large to store in memory and

978-1-4799-5666-1/14/$31.00 ©2014 IEEE

carving it into smaller subgraphs. The subgraphs are then farmed out to worker nodes where the size of their local maximum cliques is determined. It is essential that the dissection of the graph be done in such a way as to ensure that a clique in the original graph will be maintained in some subgraph sent to a worker node. Doing so guarantees that the size of the largest local maximum clique reported for a subgraph will be the global maximum clique size for the original graph.

Both the EIC and the EOC algorithms preprocess the graph to eliminate vertices that cannot be members of a clique larger than the current maximum clique size (CMCS) by removing all vertices that have a degree less than CMCS-1. By default, the CMCS is initially set at 2 and increases when a new maximum clique size is returned to the master node. The master node is responsible for preprocessing the graph and constructing workloads to send to the worker nodes. Given N worker nodes in our system, each with M bytes of memory, there will be N bins, each capable of containing a subgraph of the original graph that is less than M bytes in size. Each bin may contain a mixture of connected components and any number of vertices (along with their neighborhoods), as long as the memory constraints are not violated.

A. *Edge-In-Core*

The EIC algorithm begins by reading in the graph and storing it as an adjacency list. The preprocessing step involves recursively removing all vertices of degree CMCS-1 and any isolated vertices. The workloads are then constructed by first computing the connected components and identifying their size.

Any connected components that are less than M bytes in size are automatically placed into bins to be sent to a worker node. No further analyses are needed on the vertices that make up these connected components. If, however, a connected component is too large to fit into core memory on the worker node, it must be split up further. The dissection of individual connected components is done by iteratively removing the subgraph consisting of a root vertex r and its neighborhood and adding it to the current work bin. The algorithm first seeks to remove all cut vertices (vertices whose removal disconnects a connected component) as a root vertex. After the cut vertices are exhausted, if there is still at least one connected component that is too large to fit into a work bin, then the vertex of highest degree can be chosen as r, so as long as $|N_G(r)| < mB$, where mB is the size of available memory in the work bin B. Once a root vertex has been selected, the search space is expanded via a breadth-first search. Good candidate neighborhoods are those that either overlap significantly with the neighborhood of the root vertex or those that form totally disjoint neighborhoods.

As the EIC algorithm has access to the degree structure of the graph, the expansion of the search space can be done in a manner contingent on the structure of the graph being decomposed. In order to expand the search space to include overlapping neighborhoods, EIC begins by inserting the neighborhood of the root vertex r into the work bin along with every $N_G(w)$, where w is a neighbor of r and $|N_G(w)| < mB$. Note that if $N_G(w)$ is contained in $N_G(r)$, then we can remove both vertices r and w from the search space. Therefore, if the

```
Input: Graph G = (V, E)
Output: Maxmum clique size of G

Master Code
Read Graph G into memory and store as adjacency list
Run Preprocessing
Run Connected Component and Find Degree Structure
while Unprocessed vertices exist do
    foreach Request for work from processor i do
        Insert as many connected components (or
        neighborhoods of vertex r) as possible
        Eliminate all possible vertices from search space
        Send bin to worker node i
        if Worker node i returns a maximum clique size
        that exceeds CMCS then
            Update CMCS
            Run Preprocessing
            Run Connected Components
        end
    end
    if All workers have had at least one work segment
    or each connected component has had cut vertices
    removed then
        Run Preprocessing
        Run Connected Components
    end
end

Worker Code
while Available work do
    Send request for work to master node
    foreach Job received from master node do
        Run Maximum Clique Solver on subgraph
        Send job results to master node
    end
end
```

Algorithm 1 from [1]. The EIC algorithm is used for the case in which edge information can be stored in core memory.

induced neighborhood of the root vertex is dense, expanding the search space to include the induced neighborhoods of the low degree neighbors of the root vertex is preferable, as the number of vertices that can be eliminated from the search space in a single step is increased. On the other hand, if the induced neighborhood of the root vetex is sparse, then expanding the search space to include disjoint neighborhoods is preferable. This eliminates at least one vertex per disjoint neighborhood that is selected, in addition to the root vertex. Disjoint neighborhoods are selected by performing a breadth-first search with the root vertex r as the source node. Any vertex w that has a distance of three from vertex r is a candidate vertex. The $N_G(w)$ is inserted into the worker bin as long as $|N_G(w)| < mB$.

Once the worker bin is filled to capacity, or the vertices of G are exhaused, then the worker bin is sent to any available worker node. The worker node executes the MCS algorithm on its workload and generates a local maximum clique size. The results are returned to the master node and a request for more work is submitted. After each worker node has processed at

least one workload, or each connected component has had at least one cut vertex removed, then the master node interleaves the preprocessing step and recomputes the connected component structure. The process continues until all vertices in *G* have been eliminated from the search space. Pseudocode for this algorithm is from [1] and is listed in Algorithm 1.

B. *Edge-Out-of-Core*

While the EOC algorithm is similar in nature to the EIC algorithm, some necessary adjustments must be made to process graphs that are too large for their edge list to be stored in core memory on the worker node. The EOC algorithm requires that only the vertex list and graph metadata be stored in core memory. In order to process graph metadata, the master node must make multiple external passes over the graph in order to compute degree structure and connected component information. These graphs are generally stored on hard disks, which are orders of magnitude slower than core memory. This time penalty must be taken into account when balancing the gains of preprocessing against the accrued I/O costs. Also, without quick access to the edge information of a particular vertex, the ability to tune the workload construction based on significantly overlapping neighborhoods, or disjoint neighborhoods, is no longer a viable option.

The EIC algorithm requires that only the master node needs access to the external file. The EOC algorithm requires that every worker node will need access to the file since the master node does not have the graph stored in core memory. On smaller clusters, each worker node typically has some type of local storage on which a copy of the graph can be stored. On larger systems, it is common to find a single shared high performance file system, such as Lustre, to which all of the worker nodes must share access. Regardless of the storage solution chosen, the amount of time spent on disk I/O is the limiting performance factor for the EOC algorithm.

The roles of the master and worker nodes in the EOC algorithm are slightly changed compared to EIC, as the worker nodes take on increased responsibility. The master node now sends only the root vertices of neighborhoods in the worker bins. The size of the bins are decreased by the degree of the root vertex plus the root vertex itself. Given that the master node does not have any edge information, the two root vertices that are selected can be neighbors, but must have no other neighbors in common.

To track changes in the search space, the master node maintains a list of vertices that have been eliminated, called the do-not-read (DNR) list. The DNR list is passed to each worker node when it is sent a new workload. The worker nodes use both their respective work bins and the DNR list to filter out edges from the graph when they parse the file on disk. This reduces the number of vertices that each worker node needs to store in core memory. Once the worker node has completed its workload, the local maximum clique size is returned to the master node and a request for more work is submitted. Pseudocode for the EOC algorithm from [1] appears in Algorithm 2.

Input: Graph $G = (V, E)$
Output: Maximum clique size of G

Master Code
while *Unprocessed vertices exist* **do**
  Read Graph G and find degree structure and connected components information
  foreach *Request for work from processor* i **do**
    Insert as many connected components (or neighborhoods of vertex r) as possible
    Send DNR to processor i
    Send bin to processor i
    Add all possible vertex elements in bin to DNR
    if *Worker node* i *returns a clique size exceeding CMCS* **then**
      update CMCS
    end
    if *Worker node* i *returns an updated list of vertices to eliminate* **then**
      add set of vertices to DNR
    end
  end
end

Worker Code
while *Available work* **do**
  Send request for work to master node
  foreach *Job received from master node* **do**
    Read in subgraph G' from original graph file
    Run Maximum Clique Solver on subgraph of G'
    Send job results to master node Send set of vertices to master node to eliminate
  end
end

Algorithm 2 from [1]. The EOC algorithm is used for the case in which edge information will not fit into core memory.

III. Results

Tests were performed on Darter, a Cray XC30, that is housed and operated by the National Institute for Computational Sciences (NICS). Each compute node contains two 8-core Intel Xeon E5-2600 processors, 32GB of main memory, and is connected via Cray's Aries interconnect. The MCS algorithm that is used as the main compute engine requires that the subgraph be stored on a worker node in an efficient bitmatrix, at most a 500K square matrix.

Two different datasets were examined. The first dataset is a California road network with 1.9 million nodes, 2.7 million edges, and a maximum clique size of 4[5]. Nodes in this dataset represent intersections and destinations. Edges represent the roads that connect the intersections and/or destinations[6]. The second dataset is derived from the Youtube social network. It has 1.1 million nodes, 2.9 million edges, and a maximum clique size of 17. Nodes in this dataset represent users, while edges connect users who are identified as friends or are members of the same user-defined group. The run times of the EOC algorithm on both datasets are listed in Table 1.

Dataset|	2 nodes	5 nodes	25 nodes	75 nodes
California Road	10980	3823	256	153
Youtube	13860	4451	878	417

Table 1. Execution times (in seconds) of the EOC algorithm. Individual execution runs were conducted using 2, 5, 25, and 75 processor nodes for each graph.

IV. Ongoing Work

The primary focus of the development of the original OOC algorithms was to provide a practical set of tools to work with graphs too large to fit into core memory. With the algorithms described in the previous sections in hand, focus is now turned to their optimization. There are three areas of ongoing research that, when completed, will provide a state-of-the-art exact maximum clique solver for truly enormous graphs.

A. *Improved Processor Utilization*

The first area to undergo optimization is the underlying MCS compute engine. Once a workload is sent from the master node to the compute node, given that these problems are typically memory bound and not CPU bound, a single core is used by MCS to solve the local maximum clique problem. A hybrid OpenMP/MPI approach is currently being investigated to determine if multiple cores on a single node can work concurently on the same data, without needing multiple copies of the data in memory at the same time.

B. *Improved Preprocessing*

In the parlance of fixed-parameter tractibility, the maximum clique problem is W[1]-hard. On the other hand, it's complementary dual, minimum vertex cover, is fixed parameter tractable, or FPT. Our original out-of-core algorithms used the custom compute engine, MCS, based on algorithms for vertex cover derived from work reported in [7,8]. Two tenets of FPT are kernelization, in which an input of size *n* is reduced to a compute core with size depending only on the parameter, and branching, by which an efficient tree structure is used to explore the solution space. The kernelization process in MCS occurs during preprocessing.

Recent work has investigated the role of kernelization vs. branching in a parallel implementation of FPT vertex cover using an expanded set of reduction rules for kernelization [9]. During testing, it was discovered that some classes of our test graphs tended to kernelize exceedingly well, reducing to the point that the time needed in branching was negligible. In several cases, the graph was solved completely through kernelization alone. See Table 2.

The Road graph is derived from the California road network, as reported in section III. The Airport graph is based on direct flight connectivity in the United States. The Power graph was obtained from the high voltage power grid for the western United States. The arXiv graph is a high energy physics collaboration network. All of the graphs were obtained from the Stanford Large Network Dataset Collection [10].

C. *Targeted Solvers*

The third area of optimization is in the efficiency of the MCS compute engine. Currently, MCS has the same type of approach for all graphs. However, recent findings have suggested that a dynamic approach to analyzing different types of graphs is needed in order to get the best use out of MCS. In addition to the FPT vertex cover based codes, three other implementations have been investigated to solve maximum clique.

Graph	Road	Airport	Power	arXiv
Nodes	30000	1858	4941	12008
Edges	87628	17215	6594	118521
Kernel Size	147	126	7	240
Clique Size	3	56	6	239
Branching	0.00363	0.00598	-	-

Table 1. Graphs for which kernelization excelled. Run times are reported in seconds. Both the Power and arXiv graphs were solved completely through kernelization.

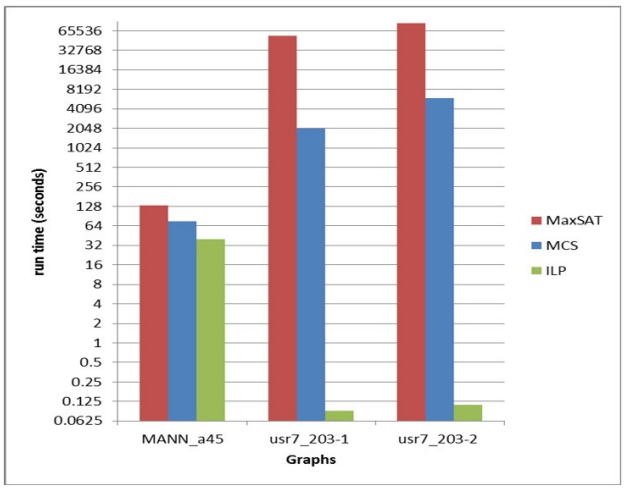

Figure 1. Graph instances solved most efficiently by ILP. Run times are reported in seconds. MANN_a45 is a DIMACS challenge graph while usr7_203-1 and usr7_203-2 are two regular graphs (constructed by union of strongly regular graphs) downloaded from http://pallini.di.uniroma1.it/library/conauto_dim/usr.zip

- A package based on Tomita's MCS algorithm. This is a branch and bound algorithm with improved approximate coloring for clique upper bounds [11].
- Akmaxsat, A MaxSAT solver. Akmaxsat is a branch and bound based propagation algorithm featuring a lazy deletion data structure [12]. Instances of the maximum clique problem were first transformed to partial MaxSAT instances using an encoding based on [13].

- Via Integer Linear Programming using IBM ILOG CPLEX.

Extensive experimentation using the algorithms above has been conducted on a myriad of both real-world and synthetic graphs. The full results will appear in an upcoming paper. The ultimate goal is to identify a metric based on structural characteristics of a graph that can be used to accurately predict the best algorithm to use for its solution.

Although in general we have seen that Tomita's MCS algorithm is the fastest, in testing we have encountered groups of graphs for which the MaxSAT and ILP approaches outperform the others. See Figures 1 and 2.

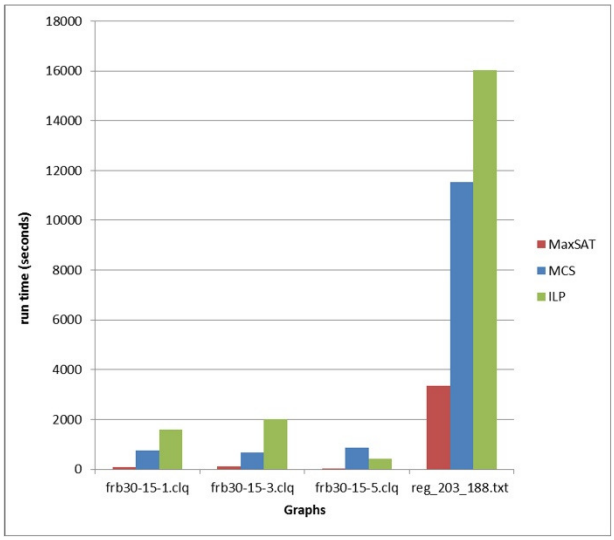

Figure 2. Graph instances solved most efficiently by MaxSAT. Run times are reported in seconds. frb30-15-1.clq, frb30-15-3.clq, and frb30-15-5.clq are BHOSLIB benchmarks downloaded from http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm. reg_203_188.txt is a regular graph generated by sampling a random graph realizing a given arbitrary degree sequence. The generator is available for download from http://www2.warwick.ac.uk/fac/sci/maths/people/staff/charo_delgenio/

References

[1] G. L. Rogers, A. D. Perkins, C. A. Phillips, J. D. Eblen, F. N. Abu-Khzam and M. A. Langston. Using out-of-core techniques to produce exact solutions to the maximum clique problem on extremely large graphs. *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications*, Rabat, Morocco, May 2009.

[2] J. D. Eblen. The maximum clique problem: algorithms, applications, and implementations. PhD dissertation., University of Tennessee, 2010.

[3] J. Xiang, C. Guo, A. Aboulnaga. Scalable maximum clique computation using MapReduce. *Proceedings of 29th IEEE International Conference on Data Engineering* (ICDE) 74-85, 2013.

[4] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, and M. A. Patwary. Fast maximum clique algorithms for large graphs. *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion* (WWW Companion) 365-366, 2014.

[5] J. Leskovec, K. Lang, A. Dasgupta, 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.

[6] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Proceedings of 2012 IEEE International Conference on Data Mining* (ICDM), 2012.

[7] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. *Proceedings, Workshop on Algorithm Engineering and Experiments*, pages 62–69, 2004.

[8] F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C. T. Symons. Scalable parallel algorithms for fpt problems. *Algorithmica*, 45:269–284, 2006.

[9] R. D. Hagan, C. Lowcay, C. A. Phillips, G. L. Rogers, K. Wang and M. A. Langston. On the relative significance of kernelization versus branching for parallel FPT implementations. *Proceedings, International Conference on Parallel and Distributed Computing and Networks,* 2013

[10] SNAP: Stanford Network Analysis Platform. http://snap.stanford.edu.

[11] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In Md.Saidur Rahman and Satoshi Fujita, editors, *WALCOM: Algorithms and Computation*, volume 5942 of *Lecture Notes in Computer Science*, pages 191–203. Springer Berlin Heidelberg, 2010.

[12] A. Kugel. Improved exact solver for the weighted max-sat problem. In Daniel Le Berre, editor, *POS-10*, volume 8 of *EPiC Series*, pages 15–27. EasyChair, 2012.

[13] C. M. Li and Z. Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *AAAI'10*. AAAI Press, 2010.