Enumerating Maximal Bicliques from a Large Graph using MapReduce

Arko Provo Mukherjee  
Department of Electrical and Computer Engineering  
Iowa State University  
2215 Coover Hall, Ames, IA, 50010, USA  
arco@iastate.edu

Srikanta Tirthapura  
Department of Electrical and Computer Engineering  
Iowa State University  
2215 Coover Hall, Ames, IA, 50010, USA  
snt@iastate.edu

Abstract—We consider enumeration of maximal bipartite cliques (bicliques) from a large graph, a task central to many practical data mining problems in social network analysis and bioinformatics. We present novel parallel algorithms for the MapReduce platform, and an experimental evaluation using Hadoop MapReduce.

Our algorithm is based on clustering the input graph into per-vertex clusters, followed by processing each cluster in parallel. The algorithm incorporates two ideas which help it scale to large graphs: (1) the redundancy between different reduce tasks is minimized through a careful pruning of the search space, and (2) the load on different reducers is balanced through the use of an appropriate total order among the vertices. Our evaluation shows that the algorithm scales to large graphs with millions of maximal bicliques. To our knowledge, this is the first work on maximal biclique enumeration for graphs of this scale.

Keywords—Graph mining; Maximal Biclique Enumeration; Enumeration algorithm; MapReduce; Hadoop; Parallel algorithm; Biclique; Load Balancing;

I. INTRODUCTION

A graph is a natural abstraction to model rich relationships in data, and massive graphs are ubiquitous in applications such as online social networks [28, 33], information retrieval from the web [5], citation networks [4], and physical simulation and modeling [44], to name a few. Finding information from such data can often be reduced to a problem of mining features from massive graphs. We consider scalable methods for discovering densely connected subgraphs within a large graph. Mining dense substructures such as cliques, quasicliques, bicliques, quasi-bicliques etc. is an important area of study [3, 17, 1, 39].

A fundamental dense substructure of interest is a biclique. A biclique in a graph \( G = (V,E) \) is a pair of subsets of vertices \( L \subseteq V \) and \( R \subseteq V \) such that (1) \( L \) and \( R \) are disjoint and (2) there is an edge \((u,v) \in E\) for every \( u \in L \) and \( v \in R \). For instance, consider the following graph relevant to an online social network, where there are two types of vertices, users and web pages. There is an edge between a user and every webpage that the user “likes”

The research reported in this papers is partially supported by the HPC equipment purchased through NSF MRI grant number CNS 1229081 and NSF CRI grant number 1205413.

on the social network. A biclique in such a graph consists of a set of users \( U \) and a set of webpages \( W \) such that every user in \( U \) has liked every page in \( W \). Such a biclique indicates a set of users who share a common interest, and is valuable for understanding the actions of users on this social network. Often, it is useful to identify only maximal bicliques in a graph, which are those bicliques that are not contained within any other larger bicliques. We consider the problem of enumerating all maximal bicliques from a graph (henceforth referred to as MBE).

Many graph mining tasks have relied on enumerating bicliques to identify significant substructures within the graph. For instance the analysis of web search queries [47] considered the “click-through” graph, where there are two types of vertices, web search queries and web pages. There is an edge from a search query to every page that a user has clicked in response to the search query. MBE was used in clustering queries using the click through graph. MBE has been used in social network analysis, in detection of communities in social networks [20], and in finding antagonistic communities in trust-distrust networks [25]. It has also been applied in detecting communities in the web graph [19, 35].

In bioinformatics, MBE has been used widely e.g. in construction of the phylogenetic tree of life [11, 36, 46, 31], structure discovery and analysis of protein-protein interactions [6, 37], analysis of gene-phenotype relationships [45], prediction of miRNA regulatory modules [48], modeling of hot spots at protein interfaces [23], and in analysis of relationships between genotypes, lifestyles, and diseases [30]. Other applications include Learning Context Free Grammars [49], finding correlations in databases [18], for data compression [2], role mining in role based access control [7], and process operation scheduling [29].

While it is easy to find a single maximal biclique in a graph, enumerating all maximal bicliques is an NP-hard problem (Peters [34]). This does not however mean that typical cases are unsolvable. In fact, there are output-polynomial time algorithms whose theoretical runtime is bounded by a polynomial in the number of vertices in the graph, and the number of maximal bicliques that are output [3]. Thus it is reasonable to expect to be able to devise algorithms for
works are also similar.

1 vertices, and about 140,000 maximal bicliques, and other presents experimental data only on graphs of up to 2,000 vertices and a few hundred thousand maximal bicliques, and have not been shown to scale to large graphs. For instance, the popular “consensus” method for biclique enumeration [3] have not been evaluated only on small graphs of a few thousands of vertices, which reduces enumeration load on subgraphs that are dense, and increases the load on subgraphs that are sparse, leading to a better parallel efficiency. We provide some basic statistical analysis of the Reducer runtimes with and without the load balancing to justify our claim.

We give a detailed analysis of the communication costs of the clustering based MapReduce Algorithms described.

We present a direct parallelization of the consensus sequential algorithm [3], using an approach different from clustering. We found that while this approach may use a smaller memory per node that the clustering approach, it requires substantially greater runtime.

Finally, we present detailed experimental results on real-world and synthetic graphs. Overall, the clustering approach (using depth first search), when combined with load balancing and reduction of redundant work, performs the best on large graphs. Our algorithms can process graphs having millions of edges, and tens of millions of maximal bicliques, and can scale out with the cluster size. To our knowledge, these are the largest reported graph instances where bicliques have been successfully enumerated.

We also provide experimental evidence showing that our parallel Algorithm based on depth first search effectively generates only large maximal bicliques. For this we show that the runtime of our parallel algorithm decreases with increase in the size threshold of the generated maximal bicliques.

B. Prior and Related Work

Makino et. al [26] describes methods to enumerate all maximal bicliques in a bipartite graph, with the delay between outputting two bicliques bounded by a polynomial in the maximum degree of the graph. Zhang et. al [50] describe a branch-and-bound algorithm for the same problem. However, these approaches do not work for general graphs, as we consider here.

There is a variant of MBE where we only seek induced maximal bicliques in a graph. An induced maximal biclique is a maximal biclique which is also an induced subgraph; i.e. a maximal biclique \((L, R)\) in graph \(G\) is an induced maximal biclique if \(L\) and \(R\) are themselves independent...
sets in $G$. We consider the non-induced version, where edges are allowed in the graph between two vertices that are both in $L$, or both in $R$ (such edges are of course, not a part of the biclique). The set of maximal bicliques that we output will also contain the set of induced maximal bicliques, which can be obtained by post-processing the output of our algorithm. Note that for a bipartite graph, every maximal biclique is also an induced maximal biclique. Algorithms for Induced MBE include work by Eppstein [12], Dias et. al [10], and Gasperm et. al [14].

Alexe et. al [3] present an iterative algorithm for non-induced MBE using the “consensus” method, which we briefly review in Section II-B. Another technique for MBE is based on a recursive depth first search (DFS) [22, 24]. [22] presents an approach based on a connection with mining closed patterns in a transactional database, and apply the algorithm from [42], which is based on depth first search. [24] present a more direct algorithm for biclique enumeration based on depth first search, which we use in our work. This is described in more detail in Section II-B.

Another approach to MBE is through a reduction to the problem of enumerating maximal cliques, as described by Gély et. al [15]. Given a graph $G$ on which we need to enumerate maximal bicliques, a new graph $G'$ is derived such that through enumerating maximal cliques in $G'$ using an algorithm such as [40, 41], it is possible to derive the maximal bicliques in $G$. However, this approach is not practical for large graphs since in going from $G$ to $G'$, the number of edges in the graph increases significantly.

To our knowledge, the only prior work on parallel algorithms for MBE is by Nataraj and Selvan [32], who use the correspondence between maximal bicliques and closed patterns [22] to derive a parallel method for enumerating maximal bicliques. A significant issue is that [32] assumes that the input graph is presented as an adjacency matrix, which is then converted into a transactional database and distributed among the processors. In contrast, we do not assume an adjacency matrix, but assume that the graph is presented as a list of edges. Thus we are able to work on much larger graphs than [32]; the largest graph that they consider has 500 vertices and about 9000 edges.

MBE is related to, but different from the problem of finding the largest sized biclique within a graph (maximum biclique). There are a few variants of the maximum biclique problem, including maximum edge biclique, which seeks the biclique in the graph with the largest number of edges, and maximum vertex biclique, which seeks a biclique with the largest number of vertices; for further details and variants, see Dawande et al. [8]. MBE is harder than finding a maximum biclique, since it enumerates all maximal bicliques, including all maximum bicliques.

II. Preliminaries

We present a formal problem definition, review prior sequential algorithms, and then briefly review the MapReduce parallel programming model that we use.

A. Problem Definition

We consider a simple undirected graph $G = (V,E)$ without self-loops or multiple edges, where $V$ is the set of all vertices and $E$ is the set of all edges of the graph. Let $n = |V|$ and $m = |E|$. Graph $H = (V_1,E_1)$ is said to be a sub-graph of graph $G = (V,E)$ if $V_1 \subset V$ and $E_1 \subset E$. $H$ is known as an induced subgraph if $E_1$ consists of all edges of $G$ that connect two vertices in $V_1$. For vertex $u \in V$, let $\eta(u)$ denote the vertices adjacent to $u$. For a set of vertices $U \subseteq V$, let $\eta(U) = \bigcup_{u \in U} \eta(u)$. For vertex $u \in V$ and $k > 0$, let $\eta^k(u)$ denote all vertices that can be reached from $u$ in $k$ hops. For $U \subseteq V$, let $\eta^k(U) = \bigcup_{u \in U} \eta^k(u)$. We call $\eta^k(U)$ as the $k$-neighborhood of $U$. For a set of vertices $U \subseteq V$, let $\Gamma(U) = \bigcap_{u \in U} \eta(u)$.

Definition 1. A biclique $B = \langle L,R \rangle$ is a subgraph of $G$ containing two non-empty and disjoint vertex sets, $L$ and $R$ such that for any two vertices $u \in L$ and $v \in R$, there is an edge $(u,v) \in E$.

Note that the definition on $B = \langle L,R \rangle$ does not impose any restriction on the existence of edges among the vertices within $R$ or within $L$, i.e., we consider non-induced bicliques.

Definition 2. A biclique $M = \langle L,R \rangle$ in $G$ is said to be a maximal biclique if there is no other biclique $M' = \langle L',R' \rangle \neq \langle L,R \rangle$ such that $L \subset L'$ and $R \subset R'$.

The Maximal Biclique Enumeration Problem (MBE) is to enumerate the set of all maximal bicliques in graph $G = (V,E)$.

In our algorithms, we assume that vertex identifiers are unique and are chosen from a totally ordered set. This is usually not a limiting assumption, since vertex identifiers are usually strings which can be ordered using the lexicographic ordering.

B. Sequential Algorithms

We describe the two approaches to sequential algorithms for MBE that we consider, one based on a “consensus algorithm” [3], and the other based on depth first search [24].

Consensus Algorithm: Alexe et. al [3] present an iterative approach to MBE. This type of algorithm starts off with a set of simple “seed” bicliques. In each iteration, it performs a “consensus” operation, which involves performing a cross-product on the set of current candidates bicliques with the seed bicliques, to generate a new set of candidates, and the process continues until the set of candidates does not
change anymore. Due to lack of space, we do not present the details here, and refer the reader to [3]. It is proved that these algorithms exactly enumerate the set of maximal bicliques in the input graph.

The consensus approach has a good theoretical guarantee, since its runtime depends on the number of maximal cliques that are output. In particular, the runtime of the MICA version of the algorithm is proved to be bounded by $O(n^3 \cdot N)$ where $n$ is the number of vertices and $N$ total number of maximal bicliques in $G$. The consensus algorithm has been found to be adequate for many applications and is quite popular.

We use the consensus algorithm in two ways. One as a candidate method for a sequential algorithm within each cluster. In another, we consider a direct parallelization of the consensus algorithm without using the clustering method.

**Sequential DFS Algorithm:** The basic sequential depth first approach (DFS) that we use is described in Algorithm 1, based on [24]. It attempts to expand an existing maximal biclique into a larger one by including additional vertices that qualify, and declares a biclique as maximal if it cannot be expanded any further. The algorithm takes the following inputs: (1) the graph $G = (V, E)$, (2) the current vertex set being processed, $X$, (3) $T$, the tail vertices of $X$, i.e. all vertices that come after $X$ in lexicographical ordering and (4) $s$, the minimum size threshold below which a maximal biclique is not enumerated. $s$ can be set to 1 so as to enumerate all maximal bicliques in the input graph. However, we can set $s$ to a larger value to enumerate only large maximal bicliques such that for $B =< L, R >$, we have $|L| \geq s$ and $|R| \geq s$. The size threshold $s$ is provided as user input. The other inputs are initialized as follows: $X = \emptyset$, $T = V$.

The algorithm recursively searches for maximal bicliques. It increases the size of $X$ by recursively adding vertices from the tail set $T$, and pruning away those vertices from $T$ which along with $X$ do not have any common vertices in their neighborhood. From the expanded $X$, the algorithm outputs the maximal biclique $(\Gamma(\Gamma(X)), \Gamma(X))$.

**C. MapReduce**

MapReduce [9] is a popular framework for processing large data sets on a cluster of commodity hardware. A MapReduce program is written through specifying map and reduce functions. The map function takes as input a key-value pair $(k, v)$ and emits zero, one, or more new key-value pairs $(k', v')$. All tuples with the same value of the key are grouped together and passed to a reduce function, which processes a particular key $k$ and all values that are associated with $k$, and outputs a final list of key-value pairs. The outputs of one MapReduce round can be the input to the next round. Communication happens only when the outputs from the map methods are retrieved by the different reduce methods based on the key, i.e. when data is grouped by keys. Further details are available in [9, 16]. We used Hadoop [43, 38], an open source implementation of MapReduce, on top of a distributed file system HDFS. While we consider the MapReduce framework for this work, our algorithms are generic and can be used with other distributed frameworks like Pregel [27].

**Algorithm 1: Depth First Search: PA(G,X,T,s)**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>forall the vertex $v \in T$ do</td>
</tr>
<tr>
<td>2</td>
<td>if $</td>
</tr>
<tr>
<td>3</td>
<td>${v} \leftarrow T \setminus {v}$</td>
</tr>
<tr>
<td>4</td>
<td>if $</td>
</tr>
<tr>
<td>5</td>
<td>return</td>
</tr>
<tr>
<td>6</td>
<td>Sort vertices in $T$ as per ascending order of $</td>
</tr>
<tr>
<td>7</td>
<td>forall the vertex $v \in T$ do</td>
</tr>
<tr>
<td>8</td>
<td>$T \leftarrow T \setminus {v}$</td>
</tr>
<tr>
<td>9</td>
<td>if $</td>
</tr>
<tr>
<td>10</td>
<td>$N \leftarrow \Gamma(X \cup {v})$</td>
</tr>
<tr>
<td>11</td>
<td>$Y \leftarrow \Gamma(N)$</td>
</tr>
<tr>
<td>12</td>
<td>Biclique $B \leftarrow (Y,N)$</td>
</tr>
<tr>
<td>13</td>
<td>if $(Y \setminus (X \cup {v})) \subseteq T$ then</td>
</tr>
<tr>
<td>14</td>
<td>if $</td>
</tr>
<tr>
<td>15</td>
<td>${v} \leftarrow T \setminus {v}$</td>
</tr>
<tr>
<td>16</td>
<td>Emit $B$ as a maximal biclique</td>
</tr>
</tbody>
</table>

### III. MapReduce Algorithms for MBE

In this section, we describe algorithms for MBE using MapReduce. We first present the basic clustering approach, which can be used with any sequential algorithm for MBE, followed by enhancements to the basic clustering approach, and finally the parallel consensus approach.

**A. Basic Clustering Approach**

We first present the basic clustering framework for parallel MBE. The approach is to cluster the input graph into several overlapping sub-graphs (clusters) and then run the sequential DFS algorithm in parallel for each cluster.

For each $v \in V$, the cluster $C(v)$ consists of the induced subgraph on all vertices in $\eta^2(v)$ (i.e. the 2-neighborhood of $v$ in $G$). The different clusters $C(v)$ are processed in parallel, and a sequential MBE algorithm is used to enumerate the maximal bicliques from each cluster. While all maximal bicliques in $G$ are indeed output by this approach, the same biclique maybe enumerated multiple times. To suppress duplicates, the following strategy is used: a maximal biclique $B$ arising from cluster $C(v)$ is emitted only if $v$ is the smallest vertex in $B$ according to the total order of the vertices. The basic clustering framework is generic and can be used with any sequential algorithm for MBE. We have
considered the clustering algorithm using the DFS and the consensus algorithms for MBE.

**Lemma 1.** The basic clustering approach enumerates all maximal bicliques in graph $G = (V, E)$.

**Proof:** We show the following two properties. First, every maximal biclique in $G$ must be output as a maximal biclique from cluster $C(v)$ for some $v \in V$. Second, every maximal biclique output from each cluster must be a maximal biclique in $G$. To prove the first direction, consider a maximal biclique $M = \langle L, R \rangle$ in $G$. Let $v$ be the smallest vertex in $M$ in lexicographic order, and without loss of generality suppose that $v \in L$. By the definition of a biclique, for each $u \in R$, $u$ is a neighbor of $v$. Similarly, every vertex $w \in L$ is a neighbor of $v$, and is hence in $\eta^2(v)$. Hence $M$ is completely contained in $C(v)$. Note that $M$ is also a maximal biclique in $C(v)$. To see this, note that if $M$ is not maximal biclique in $C(v)$, then $M$ is not maximal in $G$ either.

We prove by contradiction that every maximal biclique in each cluster $C(v)$ is also a maximal biclique in $G$. Consider a biclique $M$ emitted as maximal from cluster $C(v)$ such that it is not maximal in $G$. Then, there exists a maximal biclique $M'$ that can be generated by extending $M$. However, it is easy to see that every vertex in $M'$ must also be contained in $\eta^2(v)$, and hence $M'$ is also contained in $C(v)$, contradicting our assumption that $M$ is a maximal biclique in $C(v)$.

There are two problems with the basic clustering approach described above. First is redundant work. Although each maximal biclique in $G$ is emitted only once, it may still be generated multiple times, in different clusters. This redundant work significantly adds to the runtime of the algorithm. Second is an uneven distribution of load among different subproblems. The load on a subproblem $C(v)$ depends on two factors, the complexity of cluster $C(v)$ (i.e., the number and size of maximal bicliques within $C(v)$) and the position of $v$ in the total order of the vertices. The earlier $v$ appears in the total order, the greater is the likelihood that a maximum biclique in $C(v)$ has $v$ has its smallest vertex, and hence the greater the responsibility for emitting bicliques that are maximal within $C(v)$. Using a lexicographic ordering of the vertices may lead to a significantly increased workload for clusters of lower numbered vertices and a correspondingly low workload for clusters of higher numbered vertices.

**B. Reducing Redundant Work**

In order to reduce redundant work done at different clusters, we modify the sequential DFS algorithm for MBE that is executed at each reducer. We first observe that in cluster $C(v)$, the only maximal bicliques that matter are those with $v$ as the smallest vertex; the remaining maximal bicliques in $C(v)$ will not be emitted by this reducer, and need not be searched for here. We use this to prune the search space of the sequential DFS algorithm used at the reducer.

All search paths in the algorithm which lead to a maximal biclique having a vertex less than $v$ can be pruned away. Hence, before starting the DFS, we prune away all vertices in the Tail set that are less than $v$, as described in Algorithm 6. Also, in DFS Algorithm 7, we prune the search path in Line 12 if the generated neighborhood contains a vertex less than $v$; maximal bicliques along this search path will not have $v$ as the smallest vertex. Finally in Line 19 of Algorithm 7, we emit a maximal biclique only if the smallest vertex is the same as the key of the reducer in Algorithm 6.

The above algorithm, the “optimized DFS clustering algorithm”, or “CD0” for short, is described in Algorithm 2. This takes two rounds of MapReduce. The first round, described in Algorithms 3 (map) and 4 (reduce), is responsible for generating the 1-neighborhood for each vertex. The second round, described in Algorithms 5 (map) and 6 (reduce) first constructs the clusters $C(v)$ and runs the optimized sequential pruning algorithm at the reducer. Note that Algorithm 6 passes the size threshold $s$ while calling the optimized DFS Algorithm 7. The size threshold is an user input and can be passed on to Reducer (Algorithm 6) by using the Configuration parameters of Hadoop. Like the sequential algorithm, this parameter can be set to 1 to enumerate all maximal bicliques and to a larger value to enumerate only large maximal bicliques.

Algorithm 2: Algorithm CD0

**Input:** Edge List of $G = (V, E)$

1. Generate Adjacency List (Map) – Algorithm 3
2. Generate Adjacency List (Reduce) – Algorithm 4
3. Create Two Neighborhood (Map) – Algorithm 5
4. Create Two Neighborhood (Reduce) – DFS – Algorithm 6

Algorithm 3: Generate Adjacency List – Map

**Input:** Edge $(x, y)$

1. Emit $(key \leftarrow x, value \leftarrow y)$
2. Emit $(key \leftarrow y, value \leftarrow x)$

Algorithm 4: Generate Adjacency List – Reduce

**Input:** $key \leftarrow v, value \leftarrow \{Neighbors of v\}$

1. $neighborhood \leftarrow \emptyset$
2. $forall\ val \in value$ do
3. $neighborhood \leftarrow neighborhood \cup val$
4. $N \leftarrow \langle v, neighborhood\rangle$
5. Emit $(key \leftarrow \emptyset, value \leftarrow N)$
This condition is satisfied by the reducer with key \( a \). Lemma 2. No maximal biclique in \( G \) is emitted by more than one reducer in Algorithm 2.

**Proof:** Without the loss in generality, consider any maximal biclique \( M = \langle L, R \rangle \). Let \( a \in \{L \cup R\} \) be the smallest vertex in \( \{L \cup R\} \). Consider the reducer with key \( a \). In Line 20 of Algorithm 2, a maximal biclique is emitted only if the condition in line 19 is satisfied. This condition is satisfied by the reducer with key \( a \). However, this condition is not satisfied for any reducer such that key \( a \neq a \). Thus maximal biclique \( M \) is emitted only for the reducer with key \( a \).

**Lemma 3.** Algorithm 2 generates all maximal bicliques in a graph.

**Proof:** The correctness of this Lemma can be proved from Lemmas 1 and 2. Algorithm 2 generates the 2-neighborhood induced subgraph for each vertex in \( G \). It then runs the Sequential DFS algorithm with the optimizations explained above.

The correctness relies on the following two observations: Firstly, from Lemma 2, a maximal biclique is emitted from a reducer only if the smallest vertex in the biclique is the same as the reducer key. Secondly, no vertex is ever removed from the set \( X \). The set \( X \) thus always grows in size and never gets smaller in the course of the depth-first search. This is because the set \( Y \) is generated from set \( X \) in line 11 of Algorithm 7 and the set \( Y \) is passed as the new set \( X \) for the next level of recursion. The set \( Y \) is generated from the set \( X \) by taking the neighborhood of neighborhood of set \( X \). \( \eta(X) \) contains the set of all vertices connected to all vertices in \( X \). Then \( \Gamma(\Gamma(X)) \) contains all vertices connected to all vertices in \( \eta(X) \). This must include \( X \). Hence \( Y \supseteq X \).

From the above two observations we can prove the Lemma. Since we emit only those maximal bicliques for which the smallest vertices is the same as the reducer key \( k \), we do not need to search the paths that produce maximal bicliques with smallest vertex less than \( k \). Also, since no vertex is ever removed from the set \( X \) through the recursion path, we can be sure that at no point in the execution of the algorithm we will have \( v \in X \) such that \( v < k \). Now the

<table>
<thead>
<tr>
<th>Label</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD0</td>
<td>CDFS + Reducing Redundant Work, without Load Balancing</td>
</tr>
<tr>
<td>CD1</td>
<td>CDFS + Reducing Redundant Work + Load Balancing using Degree</td>
</tr>
<tr>
<td>CD2</td>
<td>CDFS + Reducing Redundant Work + Load Balancing using Size of 2-neighborhood</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm 5:** Create Two Neighborhood – Map

1. **Input:** \( N \leftarrow (v, \text{neighborhood}) \)
2. Emit \((\text{key} \leftarrow v, \text{value} \leftarrow N)\)
3. **forall the** \( y \in \text{neighborhood do} \)
   4. Emit \((\text{key} \leftarrow y, \text{value} \leftarrow N)\)

**Algorithm 6:** Create Two Neighborhood (DFS) – Reduce

1. **Input:** \( \text{key} \leftarrow v, \text{value} \leftarrow \{2\text{-neighborhood of } v\} \)
2. \( G' = (V', E') \leftarrow \text{Induced subgraph on } \eta^2(v) \)
3. \( X \leftarrow \emptyset \)
4. \( T \leftarrow V' \)
5. **forall the** \( t \in T \) **do**
   6. if \( t < \text{key} \) then
   7. \( T \leftarrow T \setminus \{t\} \)
8. CD0\_Seq\((G', X, T, \text{key}, s)\)

**Algorithm 7:** Optimized DFS – CD0\_Seq

1. **Input:** \( G', X, T, \text{key}, s \)
2. **forall the** \( v \in T \) **do**
   3. if \( |\Gamma(X \cup \{v\})| < s \) then
   4. \( T \leftarrow T \setminus \{v\} \)
   5. if \( |X| + |T| < s \) then
      6. return
   7. **forall the** \( v \in T \) **do**
   8. if \( |X \cup \{v\}| + |T| \geq s \) then
   9. \( N \leftarrow \Gamma(X \cup \{v\}) \)
   10. \( Y \leftarrow \Gamma(N) \)
   11. if \( Y \) contains vertices smaller than key \( s \) then
       12. continue
   13. if \( (Y \setminus (X \cup \{v\})) \subseteq T \) then
       14. if \( |Y| \geq s \) then
           15. \( v_s \leftarrow \text{Smallest vertex in } B \)
           16. if \( v_s = \text{key} \) then
               17. Emit \((\text{key} \leftarrow \emptyset, \text{value} \leftarrow B)\)
               18. CD0\_Seq\((G', Y, T \setminus Y, \text{key}, s)\)
               20. Emit \((\text{key} \leftarrow \emptyset, \text{value} \leftarrow B)\)
               21. CD0\_Seq\((G', Y, T \setminus Y, \text{key}, s)\)
set \( T \) can be considered as the candidate set as we always add elements to set \( X \) from set \( T \). Thus in Algorithm 6 we remove all vertices from the set \( T \) that are less than \( k \). Further in Algorithm 7, if we generate a maximal biclique in Line 12 with minimum vertex less than \( k \) then we prune the search tree through that path as all further maximal bicliques found in that search path will contain that vertex less than \( k \).

C. Load Balancing

Algorithm 8: Algorithms CD1 and CD2

**Input:** Edge List of \( G = (V, E) \)
1. Generate Adjacency List (Map) – Algorithm 3
2. Generate Adjacency List (Reduce) – Algorithm 4
3. Send vertex property (Map) – Algorithm 5
4. Send vertex property (Reduce) – Algorithm 9
5. Create Two Neighborhood (Map) – Algorithm 10
6. Create Two Neighborhood (Reduce) – Algorithm 11

Algorithm 9: Send vertex property – Reduce

**Input:** key \( \leftarrow v \), value \( \leftarrow \{2\text{-neighbors of } v\} \)
1. \( S \leftarrow 2\text{-neighbors of } v \)
2. neighborhood \( \leftarrow \text{Compute neighborhood of } v \text{ from } S \)
3. \( N \leftarrow \langle v, \text{neighborhood} \rangle \)
4. // Need to pass neighborhood for Round 3
5. Emit \((key \leftarrow \emptyset, \text{value} \leftarrow N)\)
6. // Need to send vertex property to all 2-neighbor
7. \( p \leftarrow \text{Value of vertex property of } v \text{ from } S \)
8. for all the vertices \( s \in S \) do
9. \[ \text{Emit} \langle key \leftarrow \emptyset, \text{value} \leftarrow \{s, v, p\} \rangle \]

Algorithm 10: Create Two Neighborhood with vertex property – Map

**Input:** \( N \leftarrow \langle v, \text{neighborhood} \rangle \text{ OR } \{s, v, p\} \)
1. if \( \text{input} = N \) then
   2. Emit \((key \leftarrow v, \text{value} \leftarrow N)\)
   3. for all the \( y \in \text{neighborhood} \) do
      4. Emit \((\text{key} \leftarrow y, \text{value} \leftarrow N)\)
3. else if \( \text{Value} = \{s, v, p\} \) then
4. Emit \((\text{key} \leftarrow s, \text{value} \leftarrow \{v, p\})\)

In Algorithm 2, lexicographical ordering was used to order the vertices, which is agnostic of the properties of the cluster \( C(v) \). The way the optimized DFS works (Algorithm 7), a reducer processing a vertex that is earlier in the

Algorithm 11: Create Two Neighborhood with vertex property (DFS) – Reduce

**Input:** key \( \leftarrow v \), value \( \leftarrow \\{\eta^2(v) \text{ along with vertex properties}\} \)
1. \( G' = (V', E') \leftarrow \text{Induced subgraph on } \eta^2(v) \)
2. Map \( \leftarrow \text{HashMap of vertex and vertex property created from } value \text{ required to compute the new ordering} \)
3. \( X \leftarrow \emptyset \)
4. \( T \leftarrow V' \)
5. for all the vertex \( t \in T \) do
6. \[ \text{if } t < \text{key in the new ordering then} \]
7. \[ \text{T} \leftarrow T \setminus \{t\} \]
8. CDL_Seq\((G', X, T, \text{key,Map, s})\)

Algorithm 12: Load Balanced DFS – CDL_Seq

**Input:** \( G', X, T, \text{key,Map, s} \)
1. for all the vertex \( v \in T \) do
2. \[ \text{if } |\Gamma(X \cup\{v\})| < s \text{ then} \]
3. \[ \text{T} \leftarrow T \setminus \{v\} \]
4. if \( |X| + |T| < s \) then
5. return
6. Sort vertices in \( T \) as per ascending order of \( |\Gamma(X \cup \{v\})| \)
7. for all the vertex \( v \in T \) do
8. \[ \text{T} \leftarrow T \setminus \{v\} \]
9. if \( |X \cup \{v\}| + |T| \geq s \) then
10. \[ \text{N} \leftarrow \Gamma(X \cup \{v\}) \]
11. \[ \text{Y} \leftarrow \Gamma(N) \]
12. if \( Y \) contains vertices smaller than key in the new ordering then
13. continue
14. \( \text{Biclique } B \leftarrow \langle Y, N \rangle \)
15. if \( (Y \setminus (X \cup \{v\})) \subseteq T \) then
16. if \( |Y| \geq s \) then
17. \( v_s \leftarrow \text{Smallest vertex in } B \text{ in the new ordering} \)
18. if \( v_s = \text{key} \) then
19. \[ \text{/} \text{Maximal biclique found} \]
20. \[ \text{Emit} \langle \text{key} \leftarrow \emptyset, \text{value} \leftarrow B \rangle \]
21. CDL_Seq\((G', Y, T \setminus Y, \text{key,Map, s})\)
total order is responsible for emitting more of the maximal bicliques within its cluster.

For improving load balance, we adjust the position of vertex $v$ in the total order according to the properties of its cluster $C(v)$. Intuitively, the more complex cluster $C(v)$ is (i.e., more and larger the maximal bicliques), the higher should be position of $v$ in the total order, so that the burden on the reducer handling $C(v)$ is reduced. While it is hard to compute (or estimate) the number of maximal bicliques in $C(v)$, we consider two properties of vertex $v$ that are simpler to estimate, to determine the relative ordering of $v$ in the total order: (1) Size of 1-neighborhood of $v$ (Degree), and (2) Size of 2-neighborhood of $v$.

In case of a tie, the vertex ID is used as a tiebreaker. These approaches were considered since vertices with higher degree are potentially part of a denser part of the graph and are contained within a greater number of maximal bicliques. The size of the 2-neighborhood gives the number of vertices in $C(v)$ and may provide a better estimate of the complexity of handling $C(v)$, but this is more expensive to compute than the size of the 1-neighborhood of the vertex.

The discussion below is generic and holds for both approaches to load balancing. To run the load balanced version of DFS, the reducer running the sequential algorithm must now have the following information for the vertex (key of the reducer): (1) 2-neighborhood induced subgraph, and (2) vertex property for every vertex in the 2-neighborhood induced subgraph, where “vertex property” is the property used to determine the total order, be it the degree of the vertex or the size of the 2-neighborhood of the vertex.

The discussion below is generic and holds for both approaches to load balancing. To run the load balanced version of DFS, the reducer running the sequential algorithm must now have the following information for the vertex (key of the reducer): (1) 2-neighborhood induced subgraph, and (2) vertex property for every vertex in the 2-neighborhood induced subgraph, where “vertex property” is the property used to determine the total order, be it the degree of the vertex or the size of the 2-neighborhood of the vertex.

The discussion below is generic and holds for both approaches to load balancing. To run the load balanced version of DFS, the reducer running the sequential algorithm must now have the following information for the vertex (key of the reducer): (1) 2-neighborhood induced subgraph, and (2) vertex property for every vertex in the 2-neighborhood induced subgraph, where “vertex property” is the property used to determine the total order, be it the degree of the vertex or the size of the 2-neighborhood of the vertex.

### Definition 3. Communication complexity of a MapReduce Algorithm $A$ for Round $r$ is denoted by $C^r_A$ and is defined as the sum of the total number of bytes emitted by all Mappers and the total number of bytes emitted by all the Reducers. We consider the output size for Reducers contributing to $C^r_A$ as each Reducer writes into the distributed file system incurring communication.

### Definition 4. Let $C_A$ denote the total communication complexity for a MapReduce Algorithm $A$ having $R$ rounds. We define $C_A = \sum_{r=1}^{R} C^r_A$.

#### Lemma 4. Total communication complexity of Algorithm CD0 is $O(m \cdot \Delta + \beta)$.

**Proof:** Algorithm CD0 has two rounds of MapReduce. For the first round, Algorithm 3, which is the Map method emits each edge twice, resulting in a communication complexity of $O(m)$. Similarly, Algorithm 4, which is the reducer emits each adjacency list once. This also results in a communication complexity of $O(m)$. Hence total communication complexity of the first round is $O(m)$.

Now let us consider the second round of MapReduce. The total communication between the Map and Reduce methods (Algorithms 5 and 6 respectively) can be computed by analyzing how much data is received by all Reducers. Each reducer receives the adjacency list of all the neighbors of the key. Let $d_i$ be the degree of vertex $v_i$, for $v_i \in V$, $i = 1, \ldots, n$.

Total communication is thus $\sum_{i=1}^{n} d_i^2 = d_1^2 + d_2^2 + \cdots + d_n^2$. This is $O((d_1 + d_2 + \cdots + d_n) \cdot \Delta) = O(n \cdot d \cdot \Delta)$. Since $\bar{d} = m/n$, total communication becomes $O(m \cdot \Delta)$. The output from the final Reducer (Algorithm 6) is the collection of all maximal bicliques and hence the resulting communication cost is $O(\beta)$. Combining two rounds, total communication complexity becomes $O(m + m \cdot \Delta + \beta) = O(m \cdot \Delta + \beta)$.

#### Lemma 5. Total communication complexity of Algorithm CD1 / CD2 is $O(m \cdot \Delta + \beta)$.

**Proof:** First, note that both Algorithms CD1 and CD2 have the same communication complexity and observe that the first round uses the same Map and Reduce methods as CD0. Thus communication for Round 1 is $O(m)$. Again, note that Map method for Round 2 is same as CD0 and hence by Lemma 4, communication for Round 2 is $O(m \cdot \Delta)$.

The Reducer (Algorithm 9) of Round 2 sends the vertex property information to all its 2-neighbors. Thus every reducer receives information about all of its 2-neighbors. This makes the total output size of Reducer to be $O(m \cdot \Delta)$. The Map method of Round 3 (Algorithm 10) sends out the 2-neighborhood information as well as the vertex information to all vertices in 2-neighborhood. Thus communication cost becomes $O(m \cdot \Delta)$. The Reducer (Algorithm 11) emits all maximal bicliques and hence the resulting communication cost is $O(\beta)$. Thus total communication cost for Algorithms CD1 and CD2 is $O(m \cdot \Delta + \beta)$.

### E. Parallel Consensus

We briefly describe another approach which directly parallelizes the consensus sequential algorithm of [3], in a manner different from the clustering approach. The motivation for
this approach is as follows. The clustering approach has the following potential drawback, it requires each cluster \( C(v) \) to have the entire 2-neighborhood of \( v \). For dense graphs, the size of the 2-neighborhood of a vertex can be large, so that the complexity of each reduce task can be substantial. With the nature of the MapReduce model, dynamic load balancing among the reducers is not (easily) possible, so that load balancing will always be an issue for non-uniform, irregular computations.

Unlike the parallel DFS algorithm which works on subgraphs of \( G \), the consensus algorithm is always directly dealing with bicliques within graph \( G \). At a high level, it performs two operations repeatedly (1) a “consensus” operation, which creates new bicliques by considering the combination of existing bicliques, and (2) an “extension” operation, which extends existing bicliques to form new maximal bicliques. There is also a need for eliminating duplicates after each iteration, and also a step needed for detecting convergence, which happens when the set of maximal bicliques is stable and does not change further.

We developed a parallel version of each of these operations, by performing the consensus, extension, duplicate removal, and convergence test using MapReduce. We omit the details due to lack of space, but present experimental results from our implementations.

**IV. Experimental Results**

We implemented the parallel algorithms on a Hadoop cluster, including the DFS-based clustering algorithms (CDFS, CD0, CD1, CD2), consensus-based clustering algorithm, and the parallel consensus algorithm. The Hadoop cluster has 28 nodes, each with a quad-core AMD Opteron processor with 8GB of RAM. All programs were written using Java and run with 2GB of heap space. The Java version on all systems was 1.5.0, and the Hadoop version used was 0.20.203.

In addition, we implemented the sequential DFS algorithm [24] (without the optimizations that we introduced in our work), and the sequential consensus algorithm (MICA) [3]. The sequential algorithms were not implemented on top of Hadoop and hence had no associated Hadoop overhead in their runtime. But on the real-world graphs that we considered, these algorithms did not complete within 12 hours, except for the p2p-Gnutella09 graph.

We used both synthetic and real-world graphs for our experiments. A summary of all the graphs used is shown in Table II. The real-world graphs were obtained from the SNAP collection of large networks [21] and were drawn from social networks, collaboration networks, communication networks, product co-purchasing networks, and internet peer-to-peer networks. Some of the real world networks were so large and dense that no algorithm was able to process them. For such graphs, we thinned them down by deleting edges with a certain probability. This makes the graphs less dense, yet preserves some of the structure of the real-world graph. We show the edge deletion probability in the name of the network. For example, graph “ca-GrQc-0.4” is obtained from “ca-GrQc” by deleting each edge with probability 0.4.

Synthetic graphs are either random graphs obtained by the Erdos-Renyi model [13], or random bipartite graphs obtained using a similar model. To generate a bipartite graph with \( n_1 \) and \( n_2 \) vertices in the two partitions, we randomly assign an edge between each vertex in the left partition to each vertex in the right partition. A random Erdos-Renyi graph on \( n \) vertices is named “ER-\(< n \>”, and a random bipartite graph with \( n_1 \) and \( n_2 \) vertices in the bipartitions is called “Bipartite-\(< n_1 \>\,< n_2 \>”.

We seek to answer the following questions from the experiments.

1) What is the relative performance of the different methods for MBE?
2) How do these methods scale with increasing number of reducers?
3) How does the runtime depend on the input size and the output size?

Figure 2 plots the runtime data for the algorithms mentioned in Table II. All data used for these plots was generated with 100 reducers.

**Impact of Reducing Redundant Work**: From Figure 2, we can see that the optimizations to basic DFS clustering through eliminating redundant work make a significant impact to the runtime for all input graphs. For instance, in Figure 2d, on input graph email–EuAll–0.6 CD0, which incorporates these optimizations, runs 9 times faster than CDFS, the basic clustering approach without reducing redundant work.

**Impact of Load Balancing**: From Figure 2, we also observe that for graphs on which the algorithms do not finish very quickly (within 200 seconds), load balancing helps significantly. In Figure 2d, for graph email–EuAll–0.6, the Load Balancing approaches (CD1 and CD2) are 7 – 7.4 times faster than the Algorithm without Load Balancing, but optimized to reduce redundant work (CD0). Other examples include Figures 2e, where for input graph loc–BrightKite–0.6, CD1 was 4.5 times faster than CD0 and CD2 was about 3.8 times faster. For some graphs, such as email-EuAll-0.4 and web-NotreDame-0.8, CD0 failed to complete even after 12 hours, but CD1 and CD2 were successful in processing them within 2 hours. This shows that there must be significant imbalance in the load, with subproblems on some dense clusters being much more expensive to process than others. This shows that for most input graphs, the versions optimized through both load balancing and reducing redundant work worked the best overall.

However, for graphs that are quickly processed, the load balancing performs slightly slower than CD0 (see Figure 2a). This can be explained by the additional overhead of load...
Table II: Various properties of the input graphs used, and runtime of different algorithms using 100 reducers. DNF means that the algorithm did not finish in 12 hours. The size threshold was set as 1 to enumerate all maximal bicliques.

<table>
<thead>
<tr>
<th>Label</th>
<th>Input Graph</th>
<th>#vertices</th>
<th>#edges</th>
<th>#max–bicliques</th>
<th>Output Size</th>
<th>CDFS</th>
<th>CD0</th>
<th>CD1</th>
<th>CD2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p2p-Gnutella09</td>
<td>8114</td>
<td>26013</td>
<td>20332</td>
<td>407558</td>
<td>113</td>
<td>92</td>
<td>132</td>
<td>130</td>
</tr>
<tr>
<td>2</td>
<td>email-EuAll-0.6</td>
<td>125551</td>
<td>168087</td>
<td>292008</td>
<td>9161154</td>
<td>42023</td>
<td>4640</td>
<td>683</td>
<td>626</td>
</tr>
<tr>
<td>3</td>
<td>com-Amazon</td>
<td>334863</td>
<td>925872</td>
<td>706854</td>
<td>12739908</td>
<td>186</td>
<td>113</td>
<td>185</td>
<td>221</td>
</tr>
<tr>
<td>4</td>
<td>com-DBLP-0.6</td>
<td>175944</td>
<td>419573</td>
<td>1875185</td>
<td>82814962</td>
<td>1659</td>
<td>409</td>
<td>374</td>
<td>478</td>
</tr>
<tr>
<td>5</td>
<td>email-EuAll-0.4</td>
<td>7928</td>
<td>35397</td>
<td>6597716</td>
<td>315555360</td>
<td>8657</td>
<td>3858</td>
<td>1512</td>
<td>2943</td>
</tr>
<tr>
<td>6</td>
<td>loc-BrightKite-0.6</td>
<td>40142</td>
<td>171421</td>
<td>10075745</td>
<td>77419528</td>
<td>28558</td>
<td>11451</td>
<td>2506</td>
<td>2998</td>
</tr>
<tr>
<td>7</td>
<td>ego-Facebook-0.6</td>
<td>3928</td>
<td>35397</td>
<td>6597716</td>
<td>315555360</td>
<td>8657</td>
<td>3858</td>
<td>1512</td>
<td>2943</td>
</tr>
<tr>
<td>8</td>
<td>web-NotreDame-0.8</td>
<td>30561</td>
<td>300398</td>
<td>19941634</td>
<td>942300172</td>
<td>DNF</td>
<td>DNF</td>
<td>1688</td>
<td>2327</td>
</tr>
<tr>
<td>9</td>
<td>ca-GrQc-0.4</td>
<td>5021</td>
<td>17409</td>
<td>16133368</td>
<td>3101214314</td>
<td>37279</td>
<td>6895</td>
<td>5790</td>
<td>6374</td>
</tr>
<tr>
<td>10</td>
<td>ER-50K</td>
<td>50000</td>
<td>275659</td>
<td>51756</td>
<td>1116752</td>
<td>96</td>
<td>89</td>
<td>133</td>
<td>136</td>
</tr>
<tr>
<td>11</td>
<td>ER-60K</td>
<td>60000</td>
<td>330015</td>
<td>61821</td>
<td>1334716</td>
<td>98</td>
<td>89</td>
<td>135</td>
<td>135</td>
</tr>
<tr>
<td>12</td>
<td>ER-70K</td>
<td>70000</td>
<td>419573</td>
<td>1875185</td>
<td>82814962</td>
<td>1659</td>
<td>409</td>
<td>374</td>
<td>478</td>
</tr>
<tr>
<td>13</td>
<td>ER-80K</td>
<td>80000</td>
<td>419573</td>
<td>1875185</td>
<td>82814962</td>
<td>1659</td>
<td>409</td>
<td>374</td>
<td>478</td>
</tr>
<tr>
<td>14</td>
<td>ER-90K</td>
<td>90000</td>
<td>419573</td>
<td>1875185</td>
<td>82814962</td>
<td>1659</td>
<td>409</td>
<td>374</td>
<td>478</td>
</tr>
<tr>
<td>15</td>
<td>ER-100K</td>
<td>100000</td>
<td>419573</td>
<td>1875185</td>
<td>82814962</td>
<td>1659</td>
<td>409</td>
<td>374</td>
<td>478</td>
</tr>
<tr>
<td>16</td>
<td>ER-250K</td>
<td>250000</td>
<td>1562707</td>
<td>252996</td>
<td>6274864</td>
<td>167</td>
<td>114</td>
<td>165</td>
<td>162</td>
</tr>
<tr>
<td>17</td>
<td>ER-500K</td>
<td>500000</td>
<td>3751823</td>
<td>506319</td>
<td>15057870</td>
<td>374</td>
<td>183</td>
<td>227</td>
<td>252</td>
</tr>
<tr>
<td>18</td>
<td>Bipartite-50K-100K</td>
<td>150000</td>
<td>1999002</td>
<td>306874</td>
<td>9256056</td>
<td>873</td>
<td>183</td>
<td>227</td>
<td>252</td>
</tr>
</tbody>
</table>

Table III: Mean and Standard Deviation computation of all 100 reducer runtimes for Algorithms CD0, CD1 and CD2. The analysis is done for the Reducer of the last MapReduce round as it performs the actual Depth First Search computation.

<table>
<thead>
<tr>
<th>Input Graph</th>
<th>CDFS</th>
<th>CD0</th>
<th>CD1</th>
<th>CD2</th>
</tr>
</thead>
<tbody>
<tr>
<td>loc-BrightKite-0.6</td>
<td>1005.94</td>
<td>631.92</td>
<td>625.14</td>
<td>1688.92</td>
</tr>
<tr>
<td>Variance</td>
<td>3470135.82</td>
<td>256859.25</td>
<td>302764.97</td>
<td>125639.12</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>1862.83</td>
<td>506.81</td>
<td>550.24</td>
<td>747.83</td>
</tr>
</tbody>
</table>

Observe that the Load Balanced Algorithms CD1 and CD2 have a much less standard deviation for reducer runtimes than CD0.

balancing (an extra round of MapReduce), which does not payoff unless the work done at the DFS step is significant.

We tried two Load Balancing approaches, one based on the vertex degree and the other on the size of the 2-neighborhood of the vertex. From Figure 2 we can observe no one approach was consistently better than the other, and the performance of the two were close to each other. For some input graphs, like Email-EuAll-0.4, the 2-neighborhood approach (CD2) fared better than the degree approach (CD1), whereas for some other input graphs like web-NotreDame-0.8, the degree approach fared better.

Finally, to further analyze the impact of Load Balancing, we also calculated the Mean and Standard Deviation of the run time of each of the 100 reducers of the Reducer Algorithm 6 for CD0 and Reducer Algorithm 11 for Algorithms CD1 and CD2 respectively. We present results of this analysis for input graphs loc-BrightKite-0.6 and ego-Facebook-0.6. Table III shows the mean and standard deviation analysis for the above-mentioned input graphs.

Consensus versus Depth First Search: Both clustering consensus as well as parallel consensus were much slower than the clustering DFS approaches. In all instances except for very small input graphs, clustering consensus did not finish within 12 hours while CD1 and CD2 finished within 1-2 hours. Further, direct parallel consensus, which uses a different parallelization strategy as explained in Section III-E, was 13 to 100 times slower than clustering consensus. These numbers are not plotted in the figures. This shows that Clustering based on DFS is the method of choice for parallel MBE.

Scaling with Number of Reducers: We measured how Algorithms CD1 and CD2 scaled with the number of reducers. In Figure 3 we plot the runtime of CD1 and CD2 with and increase in the number of reducers. In Figure 4, we also plot the speedup, defined as the ratio of the time taken with 1 reducer to the time taken with r reducers, as a function of the number of reducers r. We observe that the runtime decreases with increasing number of reducers, and further, the algorithms achieve near-linear speedup with the number of reducers. This data shows that the algorithms are scalable and may be used with larger clusters as well.

Relationship to Output Size:: We also measured the runtime of the algorithms with respect to the output size. We define the output size of the problem as the sum of the number of edges for all enumerated maximal bicliques. Figure 5 shows the runtime of algorithms CD0, CD1, and CD2 as a function of the output size. This data is only constructed for random graphs, where the different graphs considered are generated using the same model, and hence have very similar structure. We observe that the runtime...
increases almost linearly with the output size for all three algorithms CD0, CD1, and CD2.

With real world graphs, this comparison does not seem as appropriate, since the different real worlds graphs have completely different structures; however, we observed that Algorithms CD1 and CD2 have a near-linear relationship with the output size, even on real world graphs.

Large Maximal Bicliques: Finally we measured the runtime of Algorithm CD1 with respect to the size threshold $s$. Figure 6 shows the runtime with respect to various size thresholds ranging from 1 to 5. We can observe that the runtime decreases with increase in the size threshold $s$. For instance, in Figure 6a, we can observe for the graph email-EuAll-0.4, the time taken to enumerate maximal bicliques larger than size 5, is about 5 times less and the time taken for size threshold 3 is about 2.1 times less than that to enumerate all maximal bicliques. Similarly, we can observe from Figure 6b, that enumeration times are 3.8 and 1.6 times less for size thresholds 5 and 3 respectively for graph web-NotreDame-0.8.

V. Conclusion

Mining maximal bicliques is a fundamental tool that is useful in uncovering dense relationships within data that can
be represented as a graph. We presented scalable parallel methods for mining maximal bicliques from a large graph. We presented a basic clustering framework for parallel enumeration of bicliques. On top of this, we presented two optimizations, one for reducing redundant work, and the other for improving load balance, both of which significantly improved the observed performance. These algorithms scale well with increasing numbers of reducers. To our knowledge, our work is the first to successfully enumerate bicliques from graphs of this size; previous reported results were mostly sequential methods that worked on much smaller graphs.

The following directions are interesting for exploration: (1) How does this approach perform on even larger clusters, and consequently, larger input graphs? What are the bottlenecks here? and (2) Are there further improvements in search space pruning, and load balancing?
Figure 6: Runtime vs the size threshold for the emitted maximal bicliques. All experiments were performed using Algorithm CD1 and with 100 reducers.

REFERENCES


