

15-826 Project Phase 1 Report

Jiajun Wang

`jiajunwa@andrew.cmu.edu`

San-Chuan Hung

`sanchuah@andrew.cmu.edu`

October 9, 2014

1 Literature Survey

Next we list the papers that each member read, along with their summary and critique.

1.1 Papers read by Jiajun Wang

The first paper was by Kang et al. [?]

- *Main idea:* This paper proposes a graph mining package named PEGASUS, which performs multiple graph mining tasks based on Hadoop platform. The core innovation of PEGASUS improves graph mining performance by using MapReduce. In order to achieve this, the paper firstly summarizes different mining operations and models these operations into a matrix-vector multiplication expression, which is called "Generalized Iterative Matrix-Vector multiplication" (GIM-V). GIM-V focuses on typical datasets consistent with node and edge data. It requires 3 processing steps: "combine2", "combineAll", "assign". On one side, these 3 steps are mapped to matrix-vector multiplication. On the other side, these 3 steps can be mapped to different MapReduce stages. According to the definition, we can find that GIM-V steps are strongly related with map-reduce processes. Particularly, "combine2" can be done using a map-reduce process and then the output is sent to another map-reduce process, which implement "combineAll"/"assign" functions. The basic version of this implementation is called "GIM-V Base". Secondly, the paper discusses how to improve the basic version of GIM-V. Following methods are discussed: 1. Block Multiplication (Perform GIM-V on non-zero blocks, which forces co-locating and shorten sorting time). 2. Clustered Edges (Preprocessing for clustering edge files for better performance). 3. Diagonal Block Iteration (Preprocessing diagonal blocks to reduce iteration). 4. Node Renumbering (Make the minimum node located at the center of the graph to reduce iteration). Thirdly, the paper analyzes and evaluates the performance of GIM-V with different improving methods. The experiments compared different scenarios from 2 aspects. The first one is scalability. We can see from the chart that the run time of GIM-V

is decreased while the number of machines is increased. The second one is comparing between different GIM-V versions. We can see that basically, GIM-V BL-CL outperforms other methods. Finally, GIM-V is applied into real world tasks to check usability.

- *Shortcomings:* For future research, I think besides finding more ways to improve the algorithm, this package can also be enhanced at the system layer. For handling very large graph data more efficiently, we can customize Hadoop system so as to optimize disk IO and cache.

The second paper was by Kang et al. [?]

- *Main idea:* This paper introduces GBASE, which supports both storing and querying for large-scale graph data. The main focuses of GBASE are faster query while using less storage space. Basically, GBASE reaches this goal by using parallel processing and distributed compressed storage based on MapReduce framework. Firstly, in order to save graph data more efficiently, GBASE requires the data to be compressed. By assuming the data set has homogeneous block, the paper proposes block based compression algorithms which ignore empty blocks and encode informative blocks. The interesting thing here is that although compressing and uncompressing data cost extra time, this method does not affect overall query response time. Because compressed data is much smaller than original one. Another fact that enhances query is the way GBASE stores data. After compression, the data is stored following "Grid placement". Which maximizes data localization and minimize the accessed files during query. Secondly, the paper analyzes nature of supported graph queries. By leveraging MapReduce framework, GBASE implements efficient scalable graph queries. Finally, GBASE is evaluated using large graph data. The results of the experiments show two things. Initially, compression plays very important role in both storage saving and query performance enhancement. We see the data storage consumption is reduced 43x. Next, MapReduce framework does provide scalability. Big data set is processed faster when increase machines.
- *Shortcomings:* One problem of this paper may be the assumption of homogeneous block. For a random graph data, block compression may not work well. But on the other hand, this fact means GBASE can handle graph that has homogeneous blocks very effectively. Another point for future work in my opinion is using cache to accelerate compression/decompression process to further improve the query response time.

The third paper was by Kang et al. [?]

- *Main idea:* This paper introduces a new graph mining algorithm package named HADI, which is designed to estimate diameter and radius of large graphs. The key points of HADI are 1. approximation instead of calculate the exact value, 2. The algorithm is scalable and could be applied to MapReduce framework. These innovations optimize the algorithm performance. So HADI is able to handle massive data in reasonable time. In the first part, the author defines effective radius and diameter for approximation. Basically, the algorithm will take 90% as a threshold. And the result can still

keep the important information about the graph. Then a scalable algorithm based on Flajolet-Martin algorithm is proposed. This algorithm requires $O(n \log n)$ space so it cannot be executed in a single machine for big data input. Subsequently, the author introduces HADI algorithm. It is a disk-based algorithm so as to deal with large graph. Initially, a 2 stages algorithm HADI-naive is illustrated. Naive method divides previous algorithm in to a 2 phases MapReduce process. But it is not efficient enough. Then improved version of HADI algorithm is proposed. The basic idea is using an extra MapReduce phase to illuminate the copying unnecessary bitstrings. Moreover, 2 more optimizations are applied for handling block operations and compress the input size. As a result, HADI achieves $O(d(m+n)/M \log (m+n)/M)$ time complexity and $O((m+n) \log n)$ space complexity (n for nodes number, m for edges number and d for diameter). Thirdly, the paper reports lot of experiment results to prove the stability and efficiency of HADI. We can see that using more nodes will shorten the running time of the algorithm. Although it is not linear related due to the overhead of the framework, the chart is persuasive to show the scalability. On the other hand, comparing between different optimized versions show that the final version works much better than the naive one. Finally, the author applied HADI on real world data and found interesting result. It demonstrates the power of HADI and validates effective radius and diameter definition.

- *Shortcomings:* For the future work, besides including more graph mining operations and supporting larger data size, I think the author can also try to leverage some in-memory distributed database for overcoming the shortage of disk-based.

1.2 Papers read by San-Chuan Hung

The first paper was by Alvarez-Hamelina et al. [?]

- *Main idea:* The work proposed a large-scale graph visualization algorithm based on k-core. K-core means a sub-graph where the degrees of the nodes in the sub-graph are equal or higher than k . The visualization algorithm plots the points in the highest k-core to the center of the graph, and then plots the points in the second-highest k-core around previous highest k-core points, and so on.
- *Shortcomings:*
 - The visualization algorithm may not be able to present the importance of "weak links," because the degree of the nodes in weak links may be not large enough to be shown in k-core algorithm. For example, for a bridge node whose degree is 2 but linking two large components, the node will not be focused because it's coreness is low.
 - The links are not weighted in the proposed algorithm. In some cases, the weights on links matter (for example, the intersect holding networks of companies); however, the proposed visualization algorithm dose not utilize link weights.
 - In the proposed algorithm, it just random sampled the links to present. As it was mentioned in the paper, edge reduction can be more sophisticated to empha-

size the connections between groups or nodes. For example, it can just present important (high betweenness) edges.

The second paper was by Kang et al. [?]

- *Main idea:* The work proposed a framework for graph mining called HEIGEN to utilize Hadoop to do spectral analysis on large-scale data. Especially, HEIGEN accelerates the efficiency by choosing right part of operations to parallelize and reducing network traffic by dividing adjacency matrix into blocks. HEIGEN is useful for finding clusters and calculating triangles in large graphs.
- *Shortcomings:*
 - Maybe HEIGEN can be shifted to memory-based map-reduce framework, like Spark, which is more efficient than Hadoop.

The third paper was by Koutra et al. [?]

- *Main idea:* The work compares different label propagation models, predicting the label of nodes in a graph by propagating known node labels through links, like Random Walk with Restarts, Semi-Supervised Learning and Belief Propagation(BP). It also proposed FABP model, implementing modified BP model in Hadoop framework, which is scalable, as accurate as BP model, and faster than BP model.
- *Shortcomings:*
 - The experiments of FABP in this paper are mainly two-classes label classification (AI papers vs. NOT AI papers, Educational websites vs. Adult websites). I think future experiments can test FABP on multiple-classes label cases, like predicting websites is Adult/Educational/Financial.

2 K-cores Algorithm Implementation

We modified some origin code in gm_main.py, making some methods can be assigned specific source table and dest table, so our k_core function can utilize existing functions. The main code of k-cores algorithm is shown below.

```
def get_row_count ( table ) :
    cur = db_conn.cursor ()
    cur.execute ("SELECT COUNT(*) from %s" % table)
    count = cur.fetchone () [0]
    cur.close ()
    return count

def k_core (k=5):
    cur = db_conn.cursor ()

    isFinished = False
```

```

last_node_num = -1

temp_link_table = "TMP_GM_TABLE_UNDIRECT"
temp_link_table_2 = "TMP_GM_TABLE_UNDIRECT_2"
temp_degree_table = "TMP_GM_NODE_DEGREES"

# copy links from GM_TABLE_UNDIRECT to temp_link_table
gm_to_undirected(
    gm_table_name=GM_TABLE,
    gm_table_undirect_name=temp_link_table
)

gm_sql_table_drop_create(
    db_conn,
    temp_link_table_2,
    "src_id integer, dst_id integer, weight real"
)

while not isFinished:
    # calculate current degree for each node
    gm_node_degrees(
        gm_table_name=temp_link_table,
        dest_table_name=temp_degree_table)

    # remove the node whose degree is under k
    print "before delete nodes: %d " % get_row_count(temp_degree_table)
    cur.execute("DELETE FROM %s" %(temp_degree_table) +
        " WHERE in_degree < %d" % k
    )
    db_conn.commit()
    print "after delete nodes: %d " % get_row_count(temp_degree_table)

    print "before delete links: %d " % get_row_count(temp_link_table)
    gm_sql_table_drop_create(
        db_conn,
        temp_link_table_2,
        "src_id integer, dst_id integer, weight real"
    )

    cur.execute("INSERT INTO %s (src_id , dst_id , weight) "

```

```

                                %(temp_link_table_2) +
                                " SELECT src_id , dst_id , weight FROM %s "
                                %(temp_link_table) +
                                " LEFT JOIN %s as ANode ON %s.src_id = ANode.node_id"
                                %(temp_degree_table , temp_link_table) +
                                " LEFT JOIN %s as BNode ON %s.dst_id = BNode.node_id"
                                %(temp_degree_table , temp_link_table) +
                                " WHERE ANode.node_id is NOT NULL AND
                                BNode.node_id is NOT NULL"
                                )

gm_sql_table_drop(db_conn , temp_link_table)

gm_sql_create_and_insert(
    db_conn ,
    temp_link_table ,
    temp_link_table_2 ,
    "src_id integer , dst_id integer , weight real",
    "src_id , dst_id , weight",
    "src_id , dst_id , weight"
)

db_conn.commit()
print "after delete links: %d " % get_row_count(temp_link_table)

# check if the number of nodes is changed. If no, break
current_node_num = get_row_count(temp_degree_table)
print "current_node_num = %d " % current_node_num

if(current_node_num == last_node_num):
    isFinished = True
    break
else:
    last_node_num = current_node_num

if current_node_num > 0:
    # component detection
    gm_connected_components(
        num_nodes=current_node_num ,
        con_comp_table_name=GMK_CORE,
        node_table_name=temp_degree_table ,
        link_table_name=temp_link_table

```

```

    )
else:
    gm_sql_table_drop_create(
        db_conn=db_conn,
        table_name=GMK_CORE,
        create_sql_cols="node_id integer, component_id integer"
    )

cur.execute("DROP TABLE %s" % temp_link_table_2)
cur.execute("DROP TABLE %s" % temp_link_table)
cur.execute("DROP TABLE %s" % temp_degree_table)

db_conn.commit()
cur.close()

```

3 Unit Test

Please execute "make test" to run unit test.

3.1 Connected Component

- Case 1 All separate graph.

Input:

1, 2

3, 4

5, 6

7, 8

Output:

8 7

4 3

1 1

5 5

3 3

6 5

2 1

7 7

- Case 2. Bridge connected.

Input:

1, 2

2, 3

1, 3

2, 4

4, 5
4, 6
5, 6
Output:

4 1
1 1
5 1
3 1
6 1
2 1

- Case 3. clique.

Input:

1, 2
1, 3
1, 4
1, 5
2, 3
2, 4
2, 5
3, 4
3, 5
4, 5

Output:

4 1
1 1
5 1
3 1
2 1

- Case 4. Twin nodes.

Input:

1, 2

Output:

1 1
2 1

- Case 5. Circle.

Input:

1, 2
2, 3
3, 4
4, 5
5, 6
6, 7

7, 8
8, 9
9, 10
10, 1
Output:
8 1
4 1
1 1
5 1
3 1
10 1
9 1
6 1
2 1
7 1

3.2 Degree Distribution

- Case 1. Circle. Input:

1, 2
2, 3
3, 4
4, 5
5, 6
6, 7
7, 8
8, 9
9, 10
10, 1
Output:
2 10

- Case 2. clique.

Input:
1, 2
1, 3
1, 4
1, 5
2, 3
2, 4
2, 5
3, 4
3, 5
4, 5

Output:

4 5

- Case 3. 10 nodes undirect graph

Input:

1, 2

1, 3

1, 4

1, 5

1, 6

1, 7

2, 3

2, 7

2, 8

3, 4

4, 5

4, 10

5, 6

6, 7

6, 9

8, 9

8, 10

Output:

4 3

3 6

6 1

- Case 4. direct star graph

Input:

1, 2

1, 3

1, 4

1, 5

2, 1

3, 1

4, 1

5, 1

Output:

(In-Degree Distribution)

4 1

1 4

(Out-Degree Distribution)

4 1

1 4

- Case 5. Twin nodes

Input:

1, 2

Output:

1 2

3.3 K-cores algorithm

- Case 1. 10 nodes with $k = 5$

Input:

1, 2

1, 3

1, 4

1, 5

1, 6

1, 7

2, 3

2, 7

2, 8

3, 4

4, 5

4, 10

5, 6

6, 7

6, 9

8, 9

8, 10

9, 10

Output:

None

- Case 2. 6 all connected nodes, $k = 4$

Input:

1, 2

1, 3

1, 4

1, 5

1, 6

2, 3

2, 4

2, 5

2, 6

3, 4

3, 5

- 3, 6
 4, 5
 4, 6
 5, 6
 Output:
 4 1
 1 1
 5 1
 3 1
 6 1
 2 1
- Case 3. Twin nodes, $k = 1$
 Input:
 1, 2
 Output:
 1 1
 2 1
 - Case 4. Circle, $k = 2$
 Input:
 1, 2
 2, 3
 3, 4
 4, 5
 5, 6
 6, 7
 7, 8
 8, 9
 9, 10
 10, 1
 Output:
 8 1
 4 1
 1 1
 5 1
 3 1
 10 1
 9 1
 6 1
 2 1
 7 1
 - Case 5. 5 nodes clique, $k = 2$
 Input:

```

1, 2
1, 3
1, 4
1, 5
2, 3
2, 4
2, 5
3, 4
3, 5
4, 5
Output:
4 1
1 1
5 1
3 1
2 1

```

4 graphMiner Result

4.1 Connected Component

For cit-Patents dataset, we found that there are 3627 components in the graph, and the size of the biggest component is 3764117.

For soc-Epinions1 dataset, we found that there are 2 components in the graph, and the sizes of components are 75877 and 2.

4.2 Eigenvalue

```

cit-patent:
1  53.64209492
2  10.32982762
3  -5.606214201
soc-Epinions1:
1  241.4042149
2  17.96681342
3  -14.38048991

```

4.3 K core

For cit-Patents dataset, we found that there are three 5-cores components in the graph, and the sizes of components are 1847127, 9 and 8.

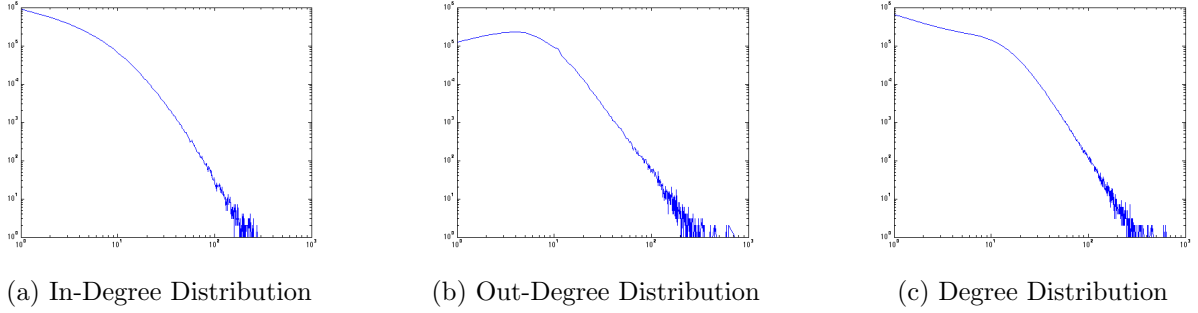


Figure 1: Degree Distributions of cit-Patents Network

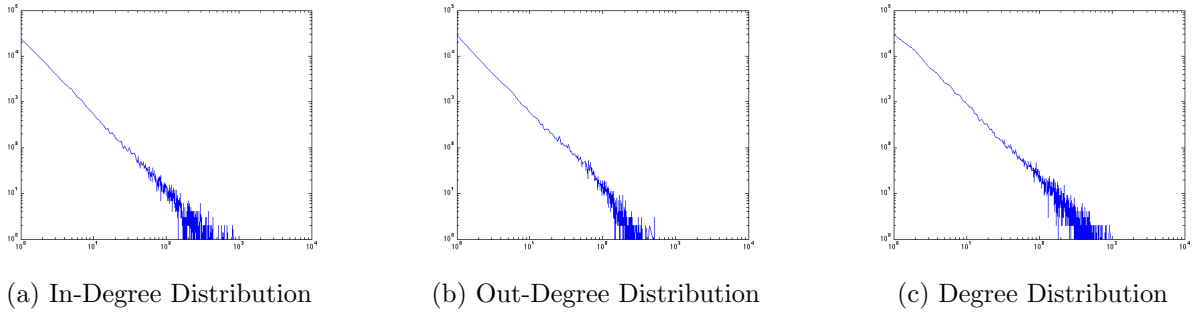


Figure 2: Degree Distributions of soc-Epinions Network

For soc-Epinions1 dataset, we found that there are only one 5-cores components in the graph, and the sizes of components is 16226.

4.4 Degree Distrubutions

According to Figure 1 and Figure 2, we found that the distributions of two networks follow power law.

4.5 PageRank and Eigenvector

See Figure 3 and Figure 4. The values of PageRank and Eigenvector are highly skewed.

4.6 Radius

See Figure 5 and Figure 6. The radius of most of nodes in two networks is 6, which shows that small-word phenomenon exists in two networks.

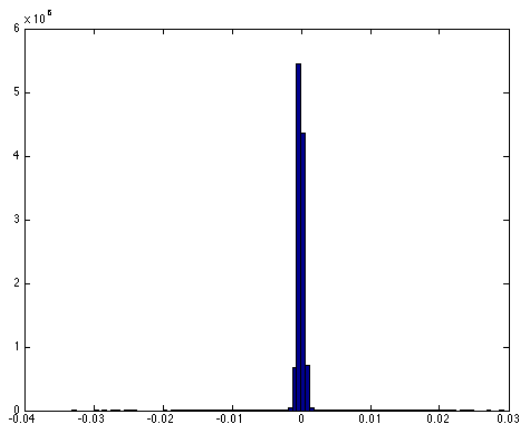
5 Division of Labour and Future Plan

5.1 Division of Tasks in Phase 1

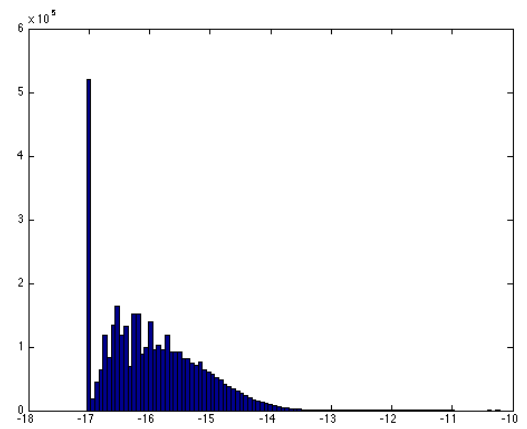
- Jiajung Wang:
 - Survey Literatures
 - Propose unit test examples
 - Summarize the results of two graphs
 - Assist to organize latex files
- San-Chuan Hung:
 - Survey Literatures
 - Implement k-cores algorithms
 - Assist to summarize the results of two graphs
 - Organize latex files

5.2 Future Plan

- Jiajung Wang:
 - Experiment different indexing procedures
 - Write the report
- San-Chuan Hung:
 - Experiment different indexing procedures
 - Write the report

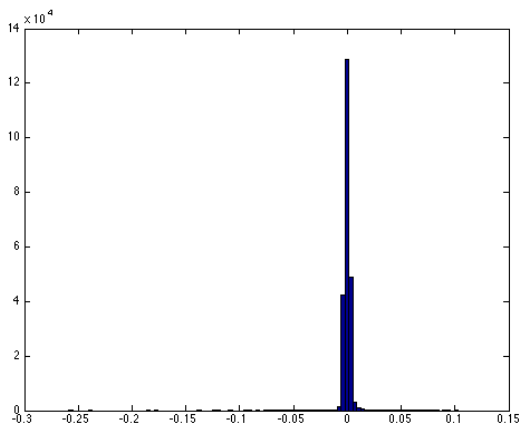


(a)

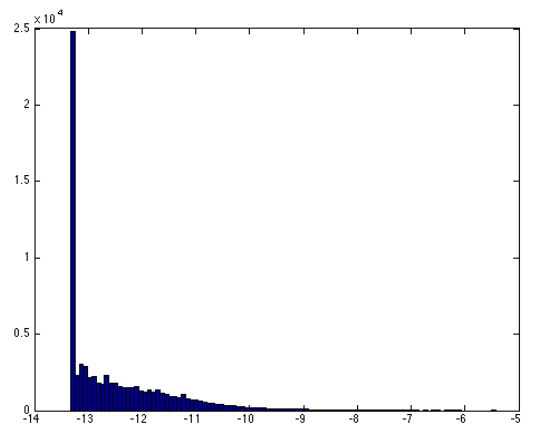


(b)

Figure 3: eigvec (a) and pagerank (b) of cit-Patents



(a)



(b)

Figure 4: eigvec (a) and pagerank (b) of soc-Epinions1

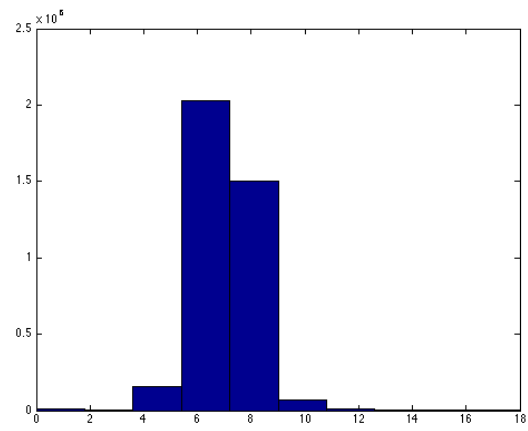


Figure 5: radius of cit-Patents

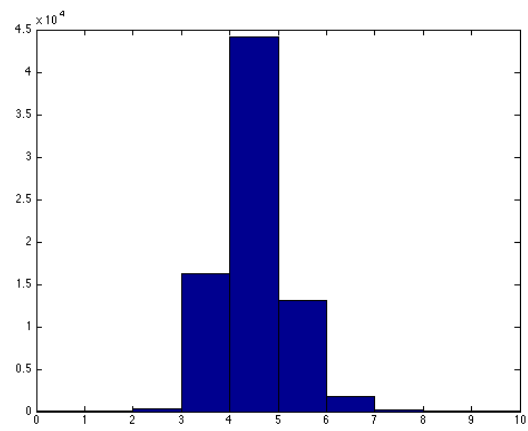


Figure 6: radius of soc-Epinions1