Exploiting Efficient Densest Subgraph Discovering Methods for Big Data

Bo Wu and Haiying Shen

Heuristic Dense Subgraph Discovery Algorithm

High-level description

- This method partitions the intractable big dataset to many small components, which still contain the dense subgraphs, but can be processed easily by Goldberg's algorithm:
  - 1. Partition the graphs into overlapping small components, which contain the dense subgraphs
  - 2. Use Goldberg's algorithm to discover the dense subgraphs in the small components
  - 3. Measure the densities of the dense subgraphs which have been found in each small component to discover the densest subgraph and top dense subgraphs

High-level description

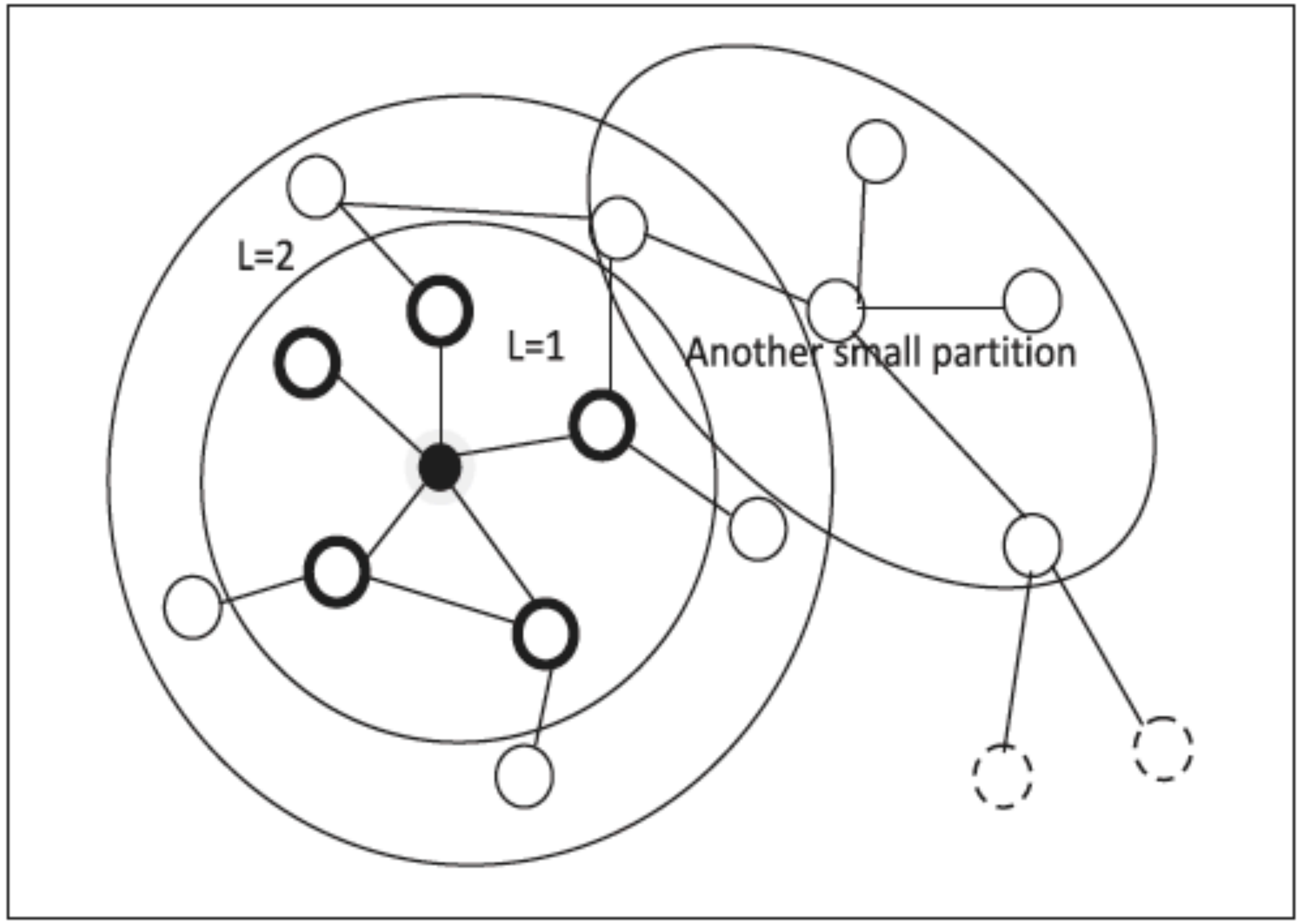


Fig. 2. An example of components when L = 1 and L = 2, respectively.

Parameter determination

- **The Number of Traversing Steps (Denoted by L)**
  - The quality of a subgraph is measured by the density of the subgraph
  - If L is too large, then the size of each component will be too large
  - If L is too small, then we may not discover high-quality dense subgraphs
  - Theorem 3.1 shows that the densest subgraph of a natural graph should have a very small diameter, so $L = 1 \; or \; L = 2$

Parameter determination

- **Criteria of Seed Selection**
    - The selected seeds are the main factor for the effectiveness of the algorithm in discovering dense subgraphs
    - An ideal method is to select the most highly connected vertex from each dense subgraph
    - However, this task is non-trivial since we do not know the places of the dense subgraphs in the graph, or whether two vertices are in the same dense subgraph

Parameter determination

- **Criteria of Seed Selection (continued)**
  - Therefore, we use a heuristic method which chooses vertices with higher degrees as seeds
  - However, for a large graph, choosing a vertex with the highest degree may lead to a memory overload
  - Therefore, we select the seeds which have the suitable degrees as Theorem 3.1 indicated
  - For the degree power law exponent γ for each natural graph, it can be estimated by a Kolmogorov-Smirnov (K-S) test

Parameter determination

- **The Number of Seeds (Denoted by m)**
  - Algorithm chooses more than one vertex, and compute the densest subgraph from multiple component candidates
  - For a graph in which the degree follows a power law, Theorem 3.1 guarantees there are a limited number of qualified seed
  - Experiment found that a floor level of parameters, i.e., setting $L = 1$, $m = 0.01$ percent of all the vertices, can guarantee a good performance

Process of the Algorithm

- **The algorithm has two main phases:**
  - **Graph partition**
    1. We measure the degree of each vertex, and select the top m suitable degree vertices as seeds
    2. We partition the large graph to the small components, which are generated by traversing the graph from the seeds for L steps
  - **Densest subgraph discovery**
    - We can apply any traditional algorithms to discover the densest subgraph contained in the small components

Process of the Algorithm

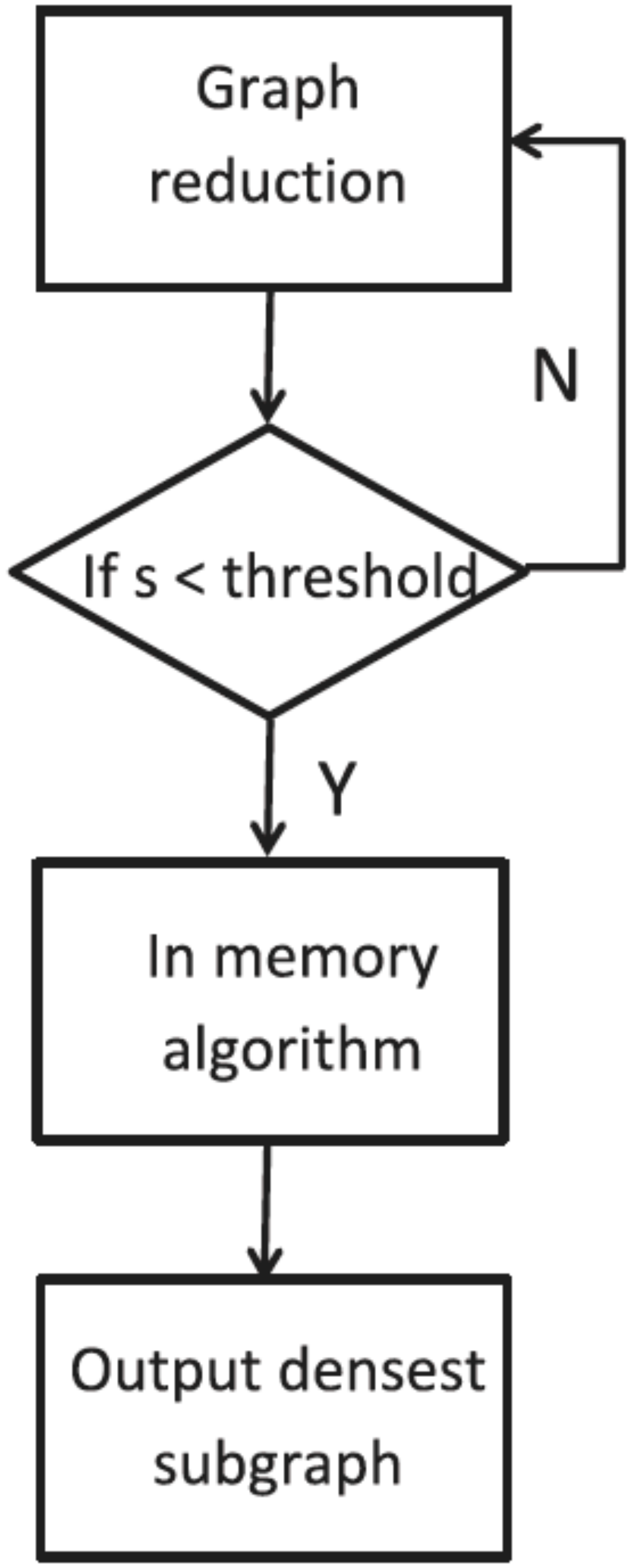


Fig. 3. Flowchart of the M-O algorithm.

Graph Reduction Algorithm 1. Seed selection

- Mapper Input: $< u, v >$ is the edge list of vertices
- Mapper Output: $< u, v >$
- Reducer Input: $< u, neighborlist >$
- Reducer Output: If length of $neighborlist$ larger than $threshold$, output $< u, \$ >$

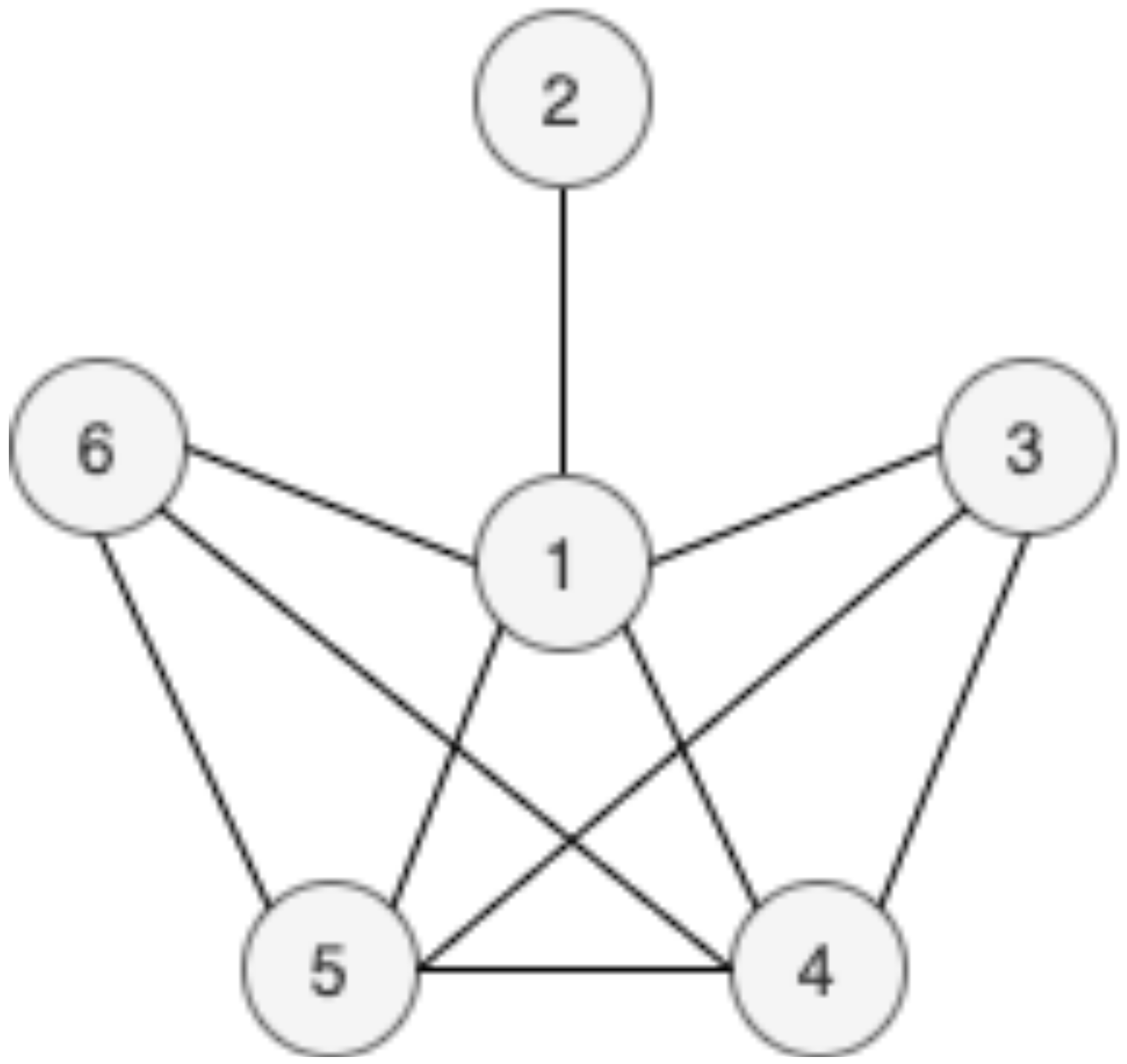
```
# Nodes: 7115 Edges: 103689
# FromNodeId    ToNodeId
30      1412
30      3352
30      5254
30      5543
30      7478
3       28
3       30
3       39
3       54
3       108
```

Algorithm 1. Seed Selection in a MapReduce Program

1: **Mapper**
  Input: $< u, v >$
  emit $< u, v >$;
end
Reducer
  Input: $< u, neighbor\_list >$
  if $|neighbor\_list| > threshold$ **then**
    emit $< u, \$ >$;
  end
end

Graph Reduction Algorithm 1. Seed selection

Graph



Edges

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

$threshold = 4$

$degree_i = \{d_1 : 5, d_2 : 1\ , d_3 : 3\ , d_4 : 4\ , d_5 : 3\}$

$output :< 1,\$ >$

Algorithm 1. Seed Selection in a MapReduce Program

```
Mapper
  Input: < u, v >
  emit < u, v >;
end
Reducer
  Input: < u, neighbor_list >
  if |neighbor_list| > threshold then
      emit < u, $ >;
  end
end
```

Demo1

Graph Reduction Algorithm 2. Tag Seeds in Each Edge

- Reducer Input:

  $< u, neighborlist >$

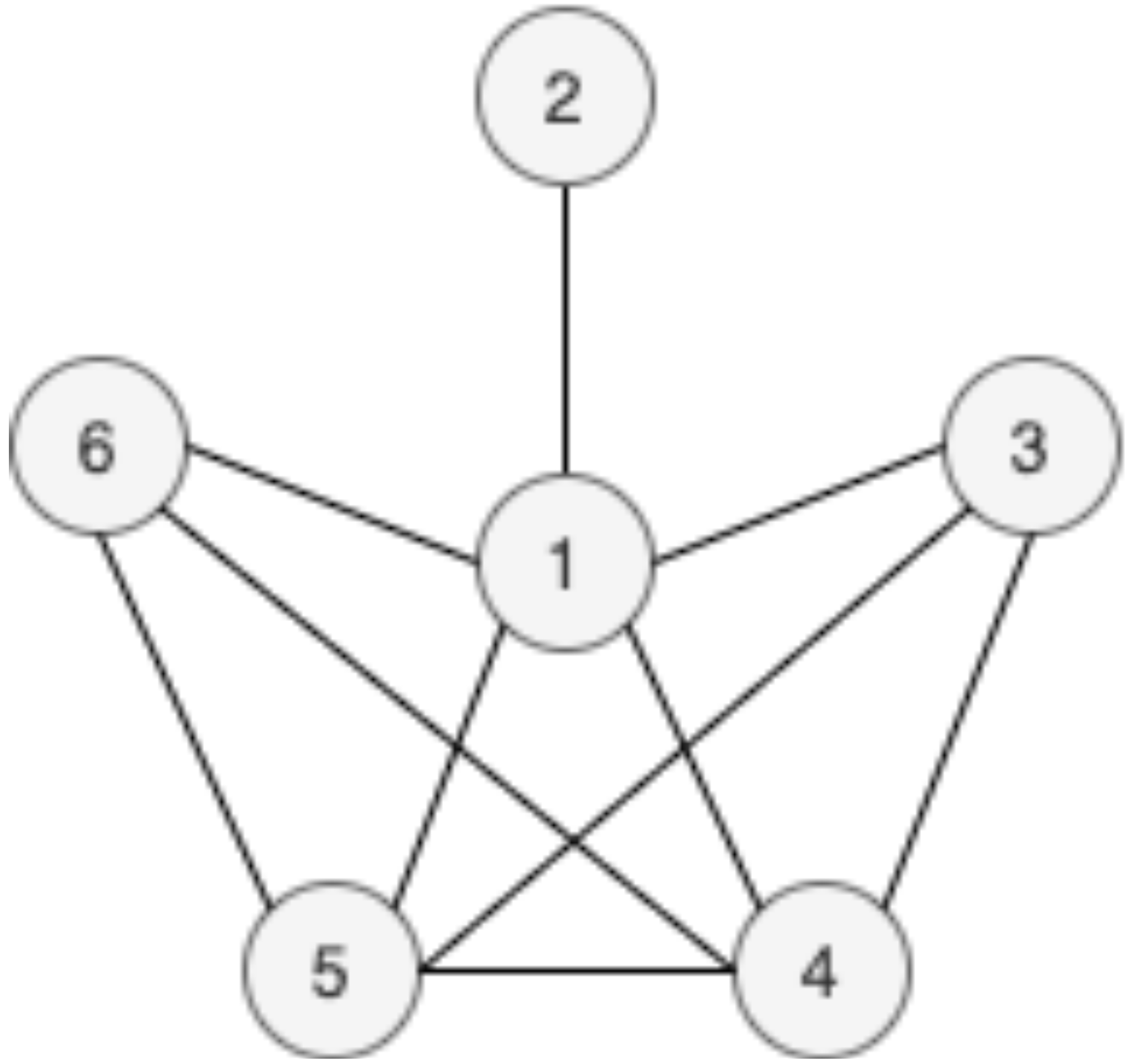
- Reducer Output: If $\$$ belongs $neighborlist$, then iterate $neighborlist$ and tag vertex u to $u*$

Algorithm 2. Tag Seeds in Each Edge

Mapper
  Input: $< u, v >$ and $< u, \$ >$
  emit Input;
end
Reducer
  Input: $< u, neighbor\_list >$
  if $\$ \in |neighbor\_list|$ **then**
    $neighbor\_list = neighbor\_list \backslash \$$;
    foreach v in $neighbor\_list$ **do**
      emit $< u^*, v >$;
    end
  else
    foreach v in $neighbor\_list$ **do**
      emit $< u, v >$;
    end
  end
end

Graph Reduction Algorithm 2. Tag Seeds in Each Edge

Graph



Edges

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

Algorithm 2. Tag Seeds in Each Edge

Mapper
  Input: $< u, v >$ and $< u, \$ >$
  emit Input;
end
Reducer
  Input: $< u, neighbor\_list >$
  if $\$ \in |neighbor\_list|$ **then**
    $neighbor\_list = neighbor\_list \backslash \$;$
    foreach v in $neighbor\_list$ **do**
      emit $< u^*, v >$;
    end
  else
    foreach v in $neighbor\_list$ **do**
      emit $< u, v >$;
    end
  end
end

Demo2

Graph Reduction Algorithm 3. Traverse 1 More Hop Based on Current Component

- Algorithm3 uses two MapReduce rounds to traverse one more hop from the seeds, and tag all the edges whose two endpoints both belong to the vertex set in the traversal

Algorithm 3. Traverse 1 More Hop Based on Current Component

1: **Mapper**
   Tag the edges in the traversal path.
   end
   Reducer
   Generate the edge list.
   end
2: **Mapper**
   Tag the remaining edges whose two endpoints both belong the vertex set in the traversal.
   end
   Reducer
   Generate the edge list.
   end

Demo3

In memory Algorithm

- **Densest subgraph discovery**
    - Author uses $ExactAlg$ as in memory algorithm which derives from Goldberg's Algorithm
    - $ExactAlg$ transforms a graph to a network and make densest subgraph into min-cut problem

Heuristic Dense Subgraph Discovery Algorithm (M-O Algorithm)

Algorithm 4. The Pseudocode of the M-O Algorithm

Graph Reduction Algorithm

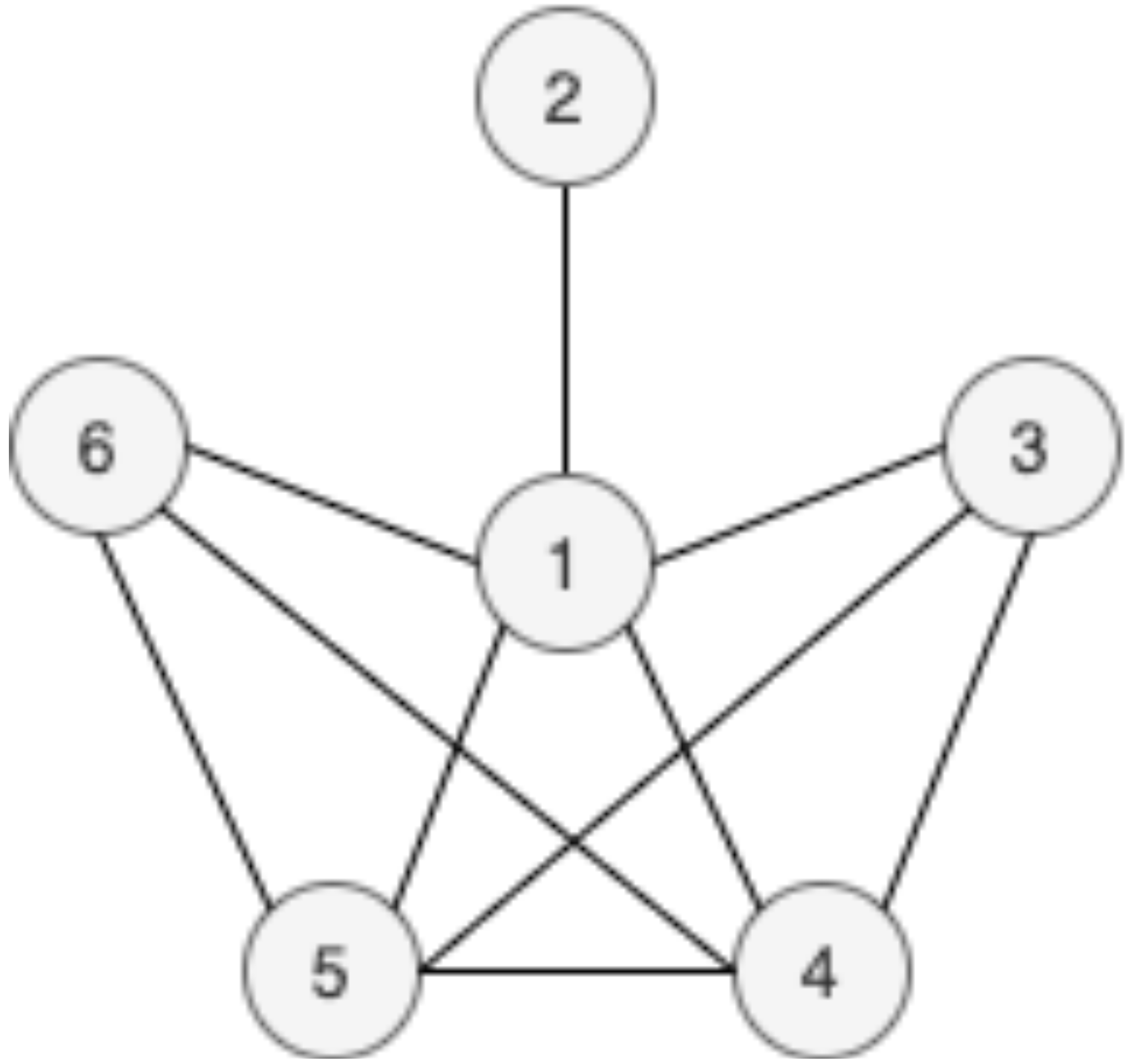
1: Given: $G = (V, E)$;
2: $S \leftarrow V$, $\rho_{max} \leftarrow \rho(S)$;
3: **while** $S > threshold$ **do**
  $S_c \leftarrow \{v_i \in S | deg_S(v_i) \leq \rho_{max}\}$;
  $S \leftarrow S \backslash S_c$;
  if $\rho(S) > \rho_{max}$ **then**
    $\rho_{max} \leftarrow \rho(S)$;
  end
end
4: $G_0 = (S_0, E(S_0)) \leftarrow G_S = (S, E(S))$;

In memory Algorithm

5: Given: $G = (V, E)$;
6: $l \leftarrow 0, u \leftarrow n$;
7: **while** $(l - u) \geq \frac{1}{n(n-1)}$ **do**
  $g \leftarrow \frac{l+u}{2}$;
  Construct $N = (V_N, E(V_N))$;
  Find min-cut (S, T);
  if $S = \{s\}$ **then**
    $u \leftarrow g$
  end
  if $S \neq \{s\}$ **then**
    $l \leftarrow g$;
    $V_1 \leftarrow S - \{s\}$;
  end
end
8: **return** subgraph of G derived by $c(S, T)$;

Heuristic Dense Subgraph Discovery Algorithm (M-O Algorithm)

Graph

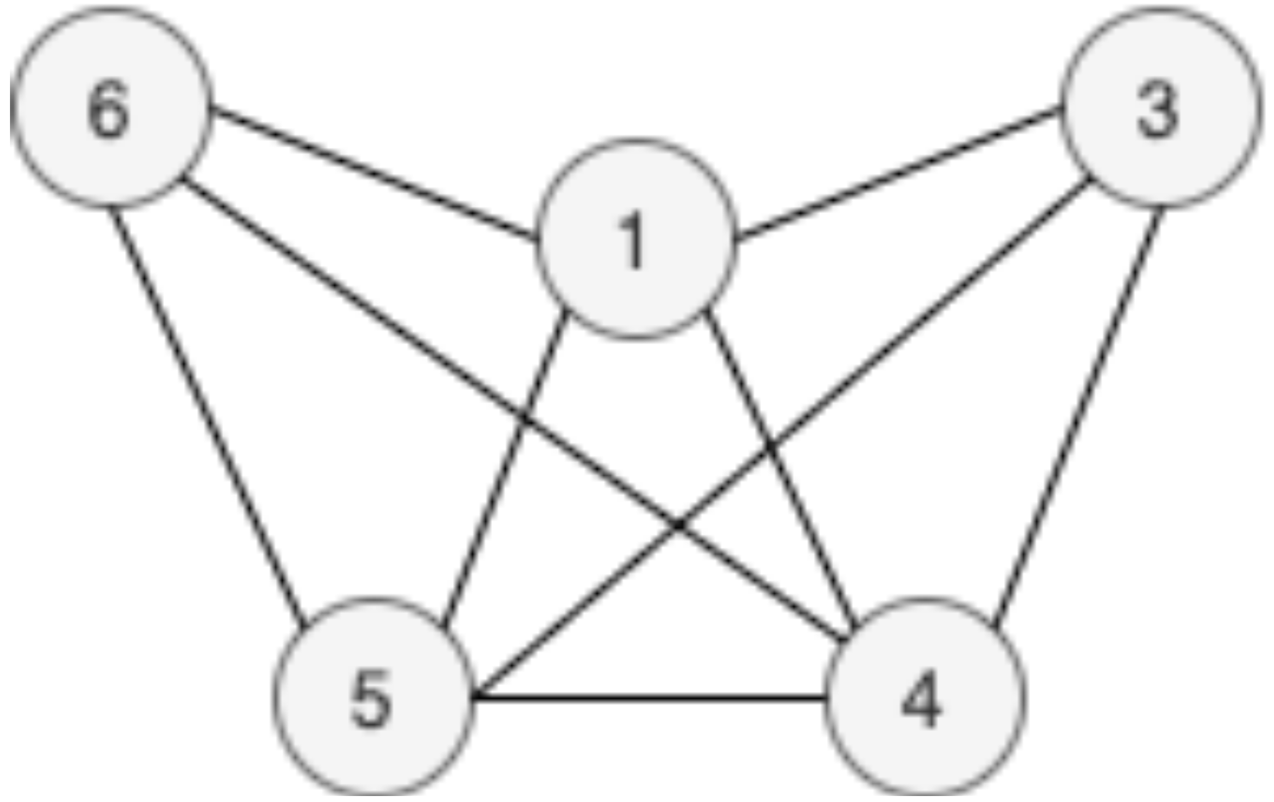


Edges

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

Density = |E|/|V|=10/6=1.666667

Graph



Edges

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

Density = |E|/|V|=9/5=1.8

Demo4

Experiment on real world dataset

Datasets: Wiki-Vote

TABLE 1
Description of Real-World Datasets

ID	Name	Description	$\|V\|$	$\|E\|$	Type
Dataset 1	Wiki-Vote [34]	Wikipedia who votes on whom network	7,115	207,378	small
Dataset 2	CA-GrQc [35]	Collaboration network of Arxiv General Relativity	12,008	237,010	small
Dataset 3	Email-Enron [36]	Enron company email list	36,692	367,662	small
Dataset 4	CA-HepPh [35]	Arxiv High Energy Physics paper citation network	34,546	421,578	small
Dataset 5	slash [16]	Slashdot social network from November 2008	77,360	905,468	small
Dataset 6	com-youtube [37]	Youtube online social network	1,134,890	2,987,624	large
Dataset 7	com-lj [37]	LiveJournal online social network	3,997,962	34,681,189	large
Dataset 8	com-orkut [37]	Orkut online social network	3,072,441	117,185,083	large

- **Vertices denote user, edges denote votes**

Parameters setting

- **According to the Theorem 3.1:**
  - $L = 1$
  - $m = |V| \times 0.01 = 7115 \times 0.01 = 71$

Experiment found that when $L = 1$ $m = 0.01$ had best performance

Graph Reduction

```
❯ ls
1098_out.txt  122_out.txt   1608_out.txt  2328_out.txt  2688_out.txt  310_out.txt   3453_out.txt  3787_out.txt  4948_out.txt  5524_out.txt  68_out.txt   789_out.txt  put.sh
1098.txt      122.txt       1608.txt      2328.txt      2688.txt      310.txt       3453.txt      3787.txt      4948.txt      5524.txt      68.txt       789.txt
1133_out.txt  1305_out.txt  173_out.txt   2485_out.txt  2871_out.txt  311_out.txt   3456_out.txt  4045_out.txt  4967_out.txt  5531_out.txt  6_out.txt    813_out.txt
1133.txt      1305.txt      173.txt       2485.txt      2871.txt      311.txt       3456.txt      4045.txt      4967.txt      5531.txt      6.txt        813.txt
1151_out.txt  1374_out.txt  1922_out.txt  24_out.txt    2967_out.txt  312_out.txt   3614_out.txt  4310_out.txt  5079_out.txt  5697_out.txt  722_out.txt  826_out.txt
1151.txt      1374.txt      1922.txt      24.txt        2967.txt      312.txt       3614.txt      4310.txt      5079.txt      5697.txt      722.txt      826.txt
1166_out.txt  1496_out.txt  2237_out.txt  2565_out.txt  2972_out.txt  3352_out.txt  3641_out.txt  4441_out.txt  5179_out.txt  5800_out.txt  737_out.txt  988_out.txt
1166.txt      1496.txt      2237.txt      2565.txt      2972.txt      3352.txt      3641.txt      4441.txt      5179.txt      5800.txt      737.txt      988.txt
11_out.txt    1542_out.txt  2256_out.txt  2651_out.txt  3032_out.txt  3447_out.txt  3642_out.txt  457_out.txt   5188_out.txt  5802_out.txt  766_out.txt  993_out.txt
11.txt        1542.txt      2256.txt      2651.txt      3032.txt      3447.txt      3642.txt      457.txt       5188.txt      5802.txt      766.txt      993.txt
1210_out.txt  1549_out.txt  2326_out.txt  2658_out.txt  306_out.txt   3449_out.txt  36_out.txt    4632_out.txt  5189_out.txt  600_out.txt   784_out.txt  996_out.txt
1210.txt      1549.txt      2326.txt      2658.txt      306.txt       3449.txt      36.txt        4632.txt      5189.txt      600.txt       784.txt      996.txt
❯ ls | wc -l
145
❯ expr 145 / 2
72
```

After Graph Reduction, we get 72 components

In memory Algorithm

```
❯ cat DSG_density_sorted.txt
graph/766 25.123880597014924
graph/2565 24.774687065368568
graph/2688 23.970920840064622
graph/1549 23.38095238095238
graph/457 21.444633730834752
graph/2967 20.761658031088082
graph/1166 20.598333333333333
graph/5524 20.26153846153846
graph/11 19.80757097791798
graph/3453 19.177777777777777
graph/4967 18.175084175084177
graph/2237 18.15702479338843
graph/3352 17.912408759124087
graph/311 17.877622377622377
graph/2485 17.80743243243243
graph/1151 17.79492600422833
graph/988 17.50853242320819
graph/5802 17.495114006514658
graph/3642 17.45762711864407
graph/3449 16.96
```

Then we use ExactAlgo calculate
density for each components