Graph Clustering for Keyword Search

Rose Catherine K.
Roll no: 07305010

M. Tech. Project Stage 3
under the guidance of

Prof. S. Sudarshan

Computer Science and Engineering
Indian Institute of Technology Bombay
Keyword searching - important paradigm of searching.  
Keyword search on external memory datagraphs could perform better if the nodes that are connected to each other are retrieved together.

**Clustering**: finding a grouping of graph nodes such that, connections within it are dense; inter-cluster edges are low.

**Community**: set of real-world entities that form a closely knit group

**Objective function**: distance-based measures, cut-size, community-related measures: modularity, conductance

**Graph Conductance**:

\[
\Phi(S) = \frac{|\partial(S)|}{\min(Vol(S), Vol(\bar{S}))}
\]

For \( S \subseteq V \):

* \( Vol(S) \): sum of node-degrees in \( S \)
* \( \partial(S) \): edges from \( S \) to \( \bar{S} \)
input: \( k \) - desired number of partitions

**Objective**

- group the nodes into \( k \) clusters, such that, all clusters are of roughly the same size.
- minimize the number of cut edges.

**Metis**

1. Coarsen the graph, by collapsing edges and grouping nodes.
2. Create a good partition on the smallest graph.
3. Project this partition back onto the original graph, by refining the partition in the intermediate levels.
4. Recursively partition the two clusters obtained, to get \( k \) partitions.

**Shortcomings:**

- Cannot find communities of varying sizes.
- Since it creates multiple versions of the graph, requires lot of memory.
Finding Communities using Random Walks on Graphs

Random walks:

- a graph traversal technique.
- Probability distribution of a walk: probability of a random walk of $k$ steps, started at a particular $startNode$, to be at a particular node at the instant/step of inspection ($nodeProbability$).

Clustering using Random walks:

- Objective: find the cluster to which a particular node belongs, or the enclosing cluster of a seed set.
- Intuition:
  - Walk started from a node in the cluster will remain within it, with a large probability.
  - Probability distribution of the random walk gives a rough ranking of the nodes of the graph.
  - A good cluster can be obtained by considering the highest ranking nodes, and by using conductance to choose the best.
Sudden drop in probability, outside the cluster boundary

\[ \Phi(S) = \frac{2}{22} = 0.09 \]

Cut 1: \( \Phi(S - n_1) = \frac{4}{22-2} = 0.2 \)

Cut 2: \( \Phi(S + n_2) = \frac{3}{22+3} = 0.12 \)

- Dip in conductance at cluster boundary
Clustering using Nibble Algorithm [ST04]

**Objective:** find the cluster to which the seed node belongs

**Nibble Algorithm:**

input: Start node \( v \), Graph \( G \), Max Conductance \( \theta_0 \)

1. Compute the bound on maxIterations, \( t_0 \), and threshold, \( \epsilon \).
2. Start spreading probabilities from \( v \).
3. Truncate the walk by setting \( \text{nodeProbability} \) to 0 where it is \( < \epsilon \).
4. Sort the nodes in the decreasing order of their probabilities.
5. Check if a \( j \) exists such that:
   - Conductance of the first \( j \) nodes \( \leq \theta_0 \)
   - The above set of nodes satisfy predefined requirements on its volume.
6. If a \( j \) was found, then return the first \( j \) nodes of the sorted set.
7. Otherwise, do the next step of spreading probabilities and repeat from Step (3).
Partitioning using Nibble:

1. Merge the clusters returned by Nibble.
2. Stop merging when the volume exceeds a predetermined fraction of $G$.
3. Shortcoming: processes the graph in top-down manner - difficult for large graphs.

Clustering using Nibble with seed set [AL06]

- Objective: find the enclosing community for a ‘seed set’ of nodes
- Modification to Nibble: assign equal probabilities to all nodes in the seed set, and spread from all seed nodes.
- Shortcoming: Seed set is chosen manually.
Shortcomings of the Nibble algorithm

- Specify the conductance of the clusters, apriori.
- May terminate at larger conductance, before finding the best.
- User cannot control the cluster size.
- No control over the spread of the walk.
Clustering using Modified-Nibble algorithm: Outline

Overall clustering algorithm

1. Choose a start node.
2. Nibble out a cluster for the start node, and remove it from the graph.
3. Repeat from step (1), until the entire graph is processed.

- Proceed by removing one cluster at a time, rather than processing the entire graph at once.
- Beneficial for clustering massive graphs.
Modified Nibble Algorithm

1. Set the initial probability of the start node to 1 and start spreading probability from it, for a specific number of steps (batch).
2. Find the best cluster for the currently active nodes, using Find Best Cluster algorithm.
3. If the cluster obtained has same or higher conductance than the best cluster of the previous iteration, stop and return the latter.
4. Else, if the conductance has reduced, continue spreading of probabilities from all the active nodes (next batch), and repeat from step (2).

- The conductance of clusters are not taken as input from the user.
- The algorithm finds the cluster of best conductance.
Find Best Cluster Algorithm

1. Consider the nodes in the decreasing order of probabilities.
2. The candidate clusters \( C^i \) contain nodes from 1 to \( i \) of the sorted set.
3. Compute the conductance of all the candidates.
4. Return the one with smallest conductance as the best cluster.

- The algorithm always finds a cluster, unlike the Nibble algorithm, which will return a cluster only if it satisfies some specific requirements.
Sample execution of Modified Nibble clustering algorithm

Fig: Prob. distrn. after 1 step

**Batch 1**

\[ \Phi(\text{best cluster}) = \frac{4}{12} = 0.33 \]

Preferred cluster S, not found yet.
**Batch 2**

\[ \Phi(S) = \frac{2}{22} = 0.09 \]

\[ \Phi(Cut1) = \frac{4}{22-2} = 0.2 \]

\[ \Phi(Cut2) = \frac{3}{22+3} = 0.12 \]

Best Cluster = S

---

Fig: Prob. distrn. after 3 steps
Batch 3
\[ \Phi(\text{Cut3}) = \frac{4}{28} = 0.14 \]
\[ \Phi(\text{Cut4}) = \frac{6}{32} = 0.18 \]
Best Cluster = S

Fig: Prob. distrn. after 5 steps
Parameters and Heuristics

H1. Start node
Ideal setting (communities are known beforehand): choose the node which is most ‘central’ to the cluster.

(a) Max degree
(b) Min degree

- High-degree nodes are mostly hub nodes.
- Could create many short-cut paths; random walk could spread to a large proportion of the graph, in a few steps.
- Nodes with lower out-degree are usually towards the periphery of the graph.
- Removing clusters from the periphery could make the processing of the core, easier.
H2. Nodes spreading in each step

(a) Spread from all active nodes

(b) Only a single node spreads in each step

- $\delta$: amount of probability received by a node, which is yet to be spread to its neighbors.
- A node spreads $\text{spreadProbability}$ fraction of only its $\delta$; remaining gets added to its $\text{nodeProbability}$ (not transferable).
- Node to spread next in each step, is the one with largest value for $\delta$.
- Number of iterations in a batch: $m \times \text{maxClusterSize}$.
- $m$ controls the amount of spreading in the graph, prior to testing for best cluster.
H3. Self-transition probability of a random walk

- Determined by spreadProbability.
- Lower values tend to over-emphasize proximity to the start node.
- Higher values can blur the cluster boundary rapidly.
- spreadProbability set to 0.5 for most experiments.
Parameters and Heuristics

H4. Number of iterations in a Batch

- Each invocation of FindBestCluster involves sorting - slow down the clustering process considerably.
- Concept of Batch of random walks:
  - Use a series to decide the number of steps in a batch.
  - Invoke FindBestCluster only after the batch of steps.

Arithmetic Plus Geometric Progression (APGP)

\[ t_i^{apgp} = (a + id) + (a r^i), \quad i = 0, 1, 2, \ldots \]

- Choose smaller values for \( r \) and larger values for \( d \).
- For larger values of \( i \), terms of GP will surpass those of AP.
- Number of times sorting is done: \( O(\log \text{totalNumSteps}) \)
H5. Upper bound on total number of random walk steps

- If the conductance of the best cluster found in a batch has lowered, the spreading of probabilities is continued.
- Upper bound: maxClusterSize
  - Ensures that, all nodes of a cluster whose diameter is maxClusterSize, are touched before spreading of probabilities is discontinued.
H6. Upper bound on number of active nodes

- The random walk can spread to the entire graph, if left unchecked.
- Intuition for random walk based clustering - it is possible to extract a cluster by exploring only a local neighborhood of the start node.
- Restrict the size of this neighborhood to \( \text{maxActiveNodeBound} \).

\[
\text{maxActiveNodeBound} = f \times \text{maxClusterSize}
\]
H7. **Behavior on** `maxActiveNodeBound`  
If the number of active nodes is restricted, options when the number of active nodes reach the bound:

(a) Stop processing and output the best cluster obtained so far.  
(b) Continue with spreading, but propagate to only those nodes that are already active.

- Bound might be reached rapidly, due to hub nodes.  
- Identifying a good cluster in a very few steps of the walk, becomes difficult.  
- Terminating the walk as soon as the bound is reached (option (a)) can hurt the overall quality of the clustering.  
- Disadvantage: increases the processing time.
H8. Compaction procedure

- Modified Nibble procedure may return clusters of sizes much smaller than MaxClusterSize.
- Large number of supernodes in the graph.
- Bundle together, multiple clusters.

**CP1.** Blind and greedy compaction of all clusters

**CP2.** Edge aware compaction of all clusters

**CP3.** Naïve compaction of tiny clusters

  - Both CP1 and CP2 improve edge compression, but create dense graphs.
  - Combine only tiny clusters that don’t have any cut edges.
  - Applying CP3 compaction will not make the supernode graph denser.
Co-citation heuristic

- Co-citation of $A_1$ and $A_2$ occurs, when $C$ links to both $A_1$ and $A_2$.
- If all co-cited nodes were in a single cluster, all edges to them will be condensed to a very few superedges.

H9. Remove hub nodes

- Select nodes of indegree at least $\text{maxClusterSize}$.
- Choose the top $t \times \text{maxClusterSize}$ and create $t$ clusters of size, $\text{maxClusterSize}$.
- Execute the clustering procedure on the remainder graph.
In FindBestCluster, candidate clusters were generated by considering the graph nodes in the order of their increasing probabilities.

Straightforward implementation leads to some interesting formations in the supernode graph - observed from experiments conducted on sample datasets.
**Umbrella formation**

- *abandoned* nodes: nodes that are separated from all its neighbors.
- Many reasons for occurrence of formations:
  - e.g. $n_c$ is a hub which connects to many authoritative nodes. Each neighbor gets absorbed into the cluster for its domain, leaving out $n_c$.
- Results in more cache misses during search.
Graph formation heuristics

H10. Graph formation heuristic

(a) Post-process
- After the best cluster is found, add the abandoned nodes to it.
- Can increase the size of the cluster beyond maxClusterSize.

(b) Abandoned node awareness
- Prevent the occurrence of formations right from the creation of candidate clusters.
- Add all abandoned nodes to the candidate clusters.
- Discard candidates whose size goes beyond maxClusterSize.
Final Modified-Nibble algorithm

**input:** Graph $G$, maxClusterSize

**Overall clustering algorithm**

1. If $H_9$ (co-citation) is used, remove hub nodes from graph.
2. Choose a start node, using $H_1$.
3. Nibble out a cluster for the start node, and remove it from the graph.
4. Repeat from step (2), until the entire graph is processed.
5. Use $H_8$ to compact the clusters obtained.
Modified Nibble Algorithm

1. Set the initial probability of the start node to 1.
2. Batch \(i\):
   - spread probabilities from all active nodes or a single node (\(H2\)).
   - amount spread is decided by \(H3\).
   - number of iterations in this batch is decided by \(H4\).
   - if maxActiveNodeBound is used (\(H6\), according to \(H7\):
     - (a) stop this batch and proceed to step 6
     - (b) continue, but spread only to already active nodes.
3. Find the best cluster \(C_i\) for Batch \(i\), using Modified FindBestCluster algorithm.
4. If \(C_i\) has same or higher conductance than \(C_{i-1}\), stop and set \(C_{best}\) as \(C_{i-1}\), and go to step 6.
5. Else, \(C_{best}\) is \(C_i\) and start next batch. But, if number of iterations have reached the bound set using \(H5\), then go to step 6.
6. If graph heuristic \(H10\) is used and is set to (a)-post process, add the abandoned nodes of \(C_{best}\) to it.
7. Return \(C_{best}\) as the best cluster of start node.
Modified FindBestCluster Algorithm

1. Consider the nodes in the decreasing order of probabilities.
2. The candidate clusters $C^i$ contain nodes from 1 to $i$ of the sorted set.
3. If graph heuristic $H10$ is used, and is set to (b) - abandoned node awareness, for all candidates, add the abandoned nodes; and discard larger ones.
4. Compute the conductance of all remaining candidates.
5. Return the one with smallest conductance as the best cluster.
Experiments and Analysis
Digital Bibliography Library Project (dblp) (2003 version)

- Tables: author, cites, paper, writes
- Number of nodes: 1,771,381
- Number of edges: 2,124,938
- max degree = 784

Wikipedia (2008 version)

- Tables: document, links
- Number of nodes: 2,648,581
- Number of undirected edges: 39,864,569
- max degree = 267,884
### Heuristic / Parameter | Choice / Value
--- | ---
H1 - start node | max degree
H2 - nodes spreading in each step | all active nodes
H3 - self-transition probability | 0.5
H4 - number of steps in a batch | APGP \((a=2, d=7, r=1.5)\)
H5 - maximum number of steps | maxClusterSize
H6 - maxActiveNodeBound | \(f = 500\)
H7 - behavior on H6 | stop on maxActiveNodeBound
H8 - compaction | CP1 - blind & greedy compaction
H9 - co-citation | no

- Doesn’t take care of the graph formations.

BI - for short
Node and edge compression

**Node Compression** = \frac{\text{number of nodes in the original graph}}{\text{number of clusters}}

**Edge Compression** = \frac{\text{number of edges in the original graph}}{\text{number of inter-cluster edges}}

- Node compression is easier to obtain.
- Edge compression - main indicator of quality of clustering.
- Higher the edge compression, better the clustering.
### Compression on dblp3

<table>
<thead>
<tr>
<th>maxClusterSize</th>
<th># clusters</th>
<th>edge compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>24,113</td>
<td>10.31</td>
</tr>
<tr>
<td>200</td>
<td>12,698</td>
<td>12.78</td>
</tr>
<tr>
<td>400</td>
<td>6,709</td>
<td>15.53</td>
</tr>
<tr>
<td>800</td>
<td>3,505</td>
<td>18.55</td>
</tr>
<tr>
<td>1500</td>
<td>1,909</td>
<td>23.46</td>
</tr>
</tbody>
</table>

- By increasing `maxClusterSize` from 100 to 1500, compression improves 2 times.
Chart of cluster size vs. frequency of dblp3

- Indicates that the inherent clusters of dblp3, are mostly of size 100 to 400.
Compression on wiki

<table>
<thead>
<tr>
<th>maxClusterSize</th>
<th># clusters</th>
<th>edge compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>16,208</td>
<td>3.203</td>
</tr>
<tr>
<td>400</td>
<td>8,052</td>
<td>5.031</td>
</tr>
<tr>
<td>1500</td>
<td>2,205</td>
<td>21.299</td>
</tr>
</tbody>
</table>

- By increasing maxClusterSize from 200 to 1500, compression improves by more than 6 times.
- There are many communities in wikipedia of large size.
- The last entry indicates that there are communities of even larger size.
Analysis of the effect of heuristics and parameters on compression
H2 - nodes spreading in each step

(a) all active nodes spread in each step of the walk
(b) only a single node spreads in each step

<table>
<thead>
<tr>
<th>H2</th>
<th># clusters</th>
<th># inter-cluster edges</th>
<th>edge compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>61,633</td>
<td>96,101</td>
<td>22.115</td>
</tr>
<tr>
<td>(b)</td>
<td>73,839</td>
<td>118,406</td>
<td>17.946</td>
</tr>
</tbody>
</table>

Edge compression on dblp3. (*settings: maxClusterSize = 1500, no compaction*)

- Higher compression with H2(a).
maxActiveNodeBound = f × maxClusterSize

Effect of \( f \) on edge compression in dblp3 (\( mcs = 1500 \))
Edge compression improves with increase in $f$.

Compression improves to about 27 when number of active nodes are not bound.

With $f = 500$, compression obtained is 23.4.

For an improvement in compression by a factor of 1.14, we incur 2.5 times the processing cost.

We upper bound the number of active nodes, with $f = 500$. 
Following options when the number of active nodes reach the bound:

(a) terminate the search
(b) continue spreading, but only to current active nodes

<table>
<thead>
<tr>
<th></th>
<th># clusters</th>
<th>edge compression</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>H7(a)</td>
<td>77,462</td>
<td>14.39</td>
<td>1.5 hrs</td>
</tr>
<tr>
<td>H7(b)</td>
<td>65,883</td>
<td>16.54</td>
<td>4 days</td>
</tr>
</tbody>
</table>

Edge compression on dblp3 (settings: startnode - minDegree, no compaction)

- Edge compression improves when the search for clusters is continued on reaching the bound.
- But, processing time shoots up, to 4 days.
- We use option H7(a) - stop on maxActiveNodeBound.
H9 - co-citation heuristic for wikipedia

- H9 heuristic - remove hub nodes, prior to clustering.
- Number of hub nodes removed = $t \times \text{maxClusterSize}$.

<table>
<thead>
<tr>
<th>t</th>
<th># clusters</th>
<th>edge compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2,350</td>
<td>22.431</td>
</tr>
<tr>
<td>1</td>
<td>2,294</td>
<td>29.867</td>
</tr>
<tr>
<td>2</td>
<td>2,290</td>
<td>30.554</td>
</tr>
</tbody>
</table>

Edge compression on wiki. (settings: \textit{minDegree start}, \textit{H7(b)-continue on maxActiveNodeBound})

- When top indegree nodes are removed, edge compression increases from 22.4 ($t=0$) to 29.8 ($t=1$).
- Degree of co-citation of these nodes are high.
- But, by removing twice the number of top indegree nodes, improvement is negligible - co-citation drops with decreasing degree.
- H9 could create many short-cut paths in the supernode graph.
### H10 - heuristics for graph formations

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>maxClusterSize</th>
<th>Bridge</th>
<th>V</th>
<th>Umbrella</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI</td>
<td>200</td>
<td>480</td>
<td>148</td>
<td>3,466</td>
</tr>
<tr>
<td>BI</td>
<td>400</td>
<td>412</td>
<td>126</td>
<td>3,014</td>
</tr>
<tr>
<td>BI + H1(b)</td>
<td>400</td>
<td>584</td>
<td>95</td>
<td>4,588</td>
</tr>
<tr>
<td>BI + H1(b) + H7(b)</td>
<td>400</td>
<td>327</td>
<td>22</td>
<td>1,058</td>
</tr>
</tbody>
</table>

Graph formations on dblp3 (*settings: no compaction*)

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>maxClusterSize</th>
<th>Umbrella</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI</td>
<td>1500</td>
<td>180,725</td>
</tr>
<tr>
<td>BI + H1(b) + H7(b)</td>
<td>1500</td>
<td>291,068</td>
</tr>
<tr>
<td>BI + H1(b) + H7(b) + H9</td>
<td>1500</td>
<td>246,864</td>
</tr>
</tbody>
</table>

Graph formations on wiki (*settings: no compaction*)

- Graph formations are prevalent.
H10 - heuristics for graph formations

(a) Post-process

<table>
<thead>
<tr>
<th>Dataset</th>
<th>maxClusterSize</th>
<th>Final maxClusterSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp3</td>
<td>200</td>
<td>323</td>
</tr>
<tr>
<td>wiki</td>
<td>1500</td>
<td>5627</td>
</tr>
</tbody>
</table>

Increase in the final cluster size using H10(a)

- Using H10(a), increase in the final cluster size for wiki is not within acceptable limits.
- H10(b): Abandoned node awareness - will produce formation-free clusters of size within the maxClusterSize parameter.
- We will use H10(b).
## Final settings for Modified Nibble clustering

<table>
<thead>
<tr>
<th>Heuristic / Parameter</th>
<th>Choice / Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1 - start node</td>
<td>max degree</td>
</tr>
<tr>
<td>H2 - nodes spreading</td>
<td>all active nodes</td>
</tr>
<tr>
<td>H3 - self-transition probability</td>
<td>0.5</td>
</tr>
<tr>
<td>H4 - number of batch steps</td>
<td>APGP with $a=2$, $d=7$, $r=1.5$</td>
</tr>
<tr>
<td>H5 - max number of steps</td>
<td>maxClusterSize</td>
</tr>
<tr>
<td>H6 - maxActiveNodeBound</td>
<td>$f = 500$</td>
</tr>
<tr>
<td>H7 - behavior on H6</td>
<td>stop on maxActiveNodeBound</td>
</tr>
<tr>
<td>H8 - compaction</td>
<td>CP3-naïve compaction of tiny clusters</td>
</tr>
<tr>
<td>H9 - co-citation</td>
<td>no</td>
</tr>
<tr>
<td>H10 - graph formation</td>
<td>abandoned node awareness</td>
</tr>
</tbody>
</table>
Comparison with Other Clustering Algorithms

Final Implementation of Modified Nibble clustering algorithm (FI), compared with:

- EBFS
- Metis

Comparison metrics:

- edge compression on dblp3 and wiki datasets.
- connection query performance, using the Incremental Expansion Backward search algorithm on dblp3
e.g. krishnamurthy parametric query optimization
- near query performance on dblp3
e.g. author (near data mining)
- time and space requirements for clustering.
EBFS: edge compression

Edge compression on dblp3 of FI and EBFS

- FI is able to achieve better edge compression than EBFS, on the dblp3 dataset.
EBFS: performance on connection queries

CPU + IO time (sec) : connection query on dblp3

- Final implementation of modified nibble is out-performing ebfs by a very large margin.
EBFS: performance on near queries

CPU + IO time (sec) : near queries on dblp3

- FI is able to beat EBFS on all queries considered.
Difficulty in comparing FI with Metis: parameters and objectives are much different.

For comparison purposes, we use clusterings whose maxClusterSize and average cluster sizes are comparable.

**FI clustering used for dblp3**
- maxClusterSize = 400
- number of clusters = 31,215

**Metis clustering used for dblp3**
- k (number of clusters) = 30,000
- maximum cluster size = 335
Metis: Edge compression

Edge compression on dblp3

<table>
<thead>
<tr>
<th></th>
<th>#clusters</th>
<th>maxClusterSize</th>
<th>edge compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>FI</td>
<td>31,215</td>
<td>400</td>
<td>15.6</td>
</tr>
<tr>
<td>Metis</td>
<td>30,000</td>
<td>335</td>
<td>9.616</td>
</tr>
</tbody>
</table>

Edge compression on wiki

<table>
<thead>
<tr>
<th></th>
<th>#clusters</th>
<th>maxClusterSize</th>
<th>edge compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>FI</td>
<td>11,305</td>
<td>1600</td>
<td>17.3</td>
</tr>
<tr>
<td>Metis</td>
<td>3,000</td>
<td>1,096</td>
<td>15.7</td>
</tr>
<tr>
<td>Metis</td>
<td>4,000</td>
<td>16,353</td>
<td>9.13</td>
</tr>
</tbody>
</table>

• Modified Nibble is able to achieve better edge compression than Metis.
Metis: performance on connection queries

- Metis performs really well on some keyword queries, while FI outperforms Metis on some others.
- Difference in performance can also be caused by the queries under consideration.
Metis: performance on near queries

In all cases, number of supernodes with near keywords match, for FI is lesser than Metis.

- Clusters produced by FI, also groups the paper titles in dblp3.
Metis: performance on near queries

- FI has significantly lesser cache misses than Metis.
FI outperforms Metis on almost all queries considered.
Time and space required for clustering

Modified Nibble Clustering algorithm:

<table>
<thead>
<tr>
<th>dataset</th>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp3 (132 MB)</td>
<td>~ 1.5 hrs</td>
<td>190 MB</td>
</tr>
<tr>
<td>wiki (1.9 GB)</td>
<td>~ 1.5 days</td>
<td>2 GB</td>
</tr>
</tbody>
</table>

- Space requirements of FI - very close to the size of the graph.
- It was found that difference in time and space required, for different maxClusterSize is negligible.
k vs memory for dblp3

- Space required grows almost linearly with k.
- Constants are very high (e.g. for $k = 40,000$ on dblp3, memory required is 12.8 GB).
- Time taken: dblp3 - 5 mins, wiki - 1.5 hrs.
- But, since clustering is done offline, time may not be an issue, but space may be.
Conclusions

- We proposed an algorithm called Modified Nibble Clustering algorithm, for clustering data represented as graphs, using the technique of random walks. It improved upon the earlier Nibble algorithm.
-Outlined several heuristics that improved its performance.
-Compared our algorithm with EBFS and Metis, where the metrics used were edge compression, keyword search performance, time & space requirements for clustering, on sample graphs.
-Results showed that Modified Nibble clustering outperformed EBFS uniformly, and Metis, for some metrics.
Future Work

- Formulating a clustering objective for getting good connection query performance, on external memory search systems.
- Test the effect of combinations of heuristics.
- Test the performance of Modified Nibble clustering algorithm on larger graphs, that fit in memory.
- Modifying the algorithm to run in a distributed environment, so that massive graphs can be handled.
- Improve the speed of clustering process, by nibbling out multiple clusters in parallel.
References


Extra Slides
Detailed pseudocode

Overall clustering algorithm

**input:** Graph $G$

1. Set $G' = G$. If co-citation heuristic $H_9$ is used, set $G'$ to the remainder graph, after removing hub nodes.
2. Choose start node $n_s$ according to $H_1$.
3. Obtain cluster $C_s = \text{ModifiedNibble}(n_s, G')$
4. Set $G' = G' - C_s$, and save $C_s$.
5. Repeat from step (2), until $G'$ is null.
6. Compact the clusters obtained, using $H_8$ procedure.
**ModifiedNibble I**

**input:** start node $n_s$, Graph $G'$

1. **Initialization:**
   - Set nodeProbability of $n_s$ to 1 and add it to the activeNodes set.
   - Set maxSteps according to H5.
   - If number of active nodes are bounded, calculate maxActiveNodeBound using H6.
   - Set totalSteps to 0.

2. **Batch i:**
   - **Initialization:**
     - Get term $t_i$ from the series chosen using H4.
     - Set batchSteps to $(t_i - \text{totalSteps})$.
     - But, if $t_i$ exceeds maxSteps, set batchSteps to $(\text{maxSteps} - \text{totalSteps})$. 
do the following for `batchSteps` number of times:

1. spread from all nodes in `activeNodes` or a single node, according to H2.
2. the amount of spreading is determined by `spreadProbability` as chosen in H3.
3. update `nodeProbability` of all nodes, with the probabilities accumulated from their neighbors.
4. update `activeNodes` set to contain all nodes with positive values for their `nodeProbabilities`.
5. if number of active nodes are bounded, check if `maxActiveNodeBound` has been reached. If yes, then, according to the choice of H7, do as below:
   - H7(a): stop this batch, and proceed directly to step 3.
   - H7(b): continue this batch, but spreading is done to only those nodes, which are already in `activeNodes`. 
3. obtain cluster $C_i = \text{ModifiedFindBestCluster}(\text{activeNodes, } G')$.
4. find conductance of $C_i$ w.r.t the current graph $G'$, $\Phi_{G'}(C_i)$.
   - if $\Phi_{G'}(C_i) \geq \Phi_{G'}(C_{i-1})$, set $C_{\text{best}}$ to $C_{i-1}$, and go to step 6.
   - else, set $C_{\text{best}}$ to $C_i$
5. do the following and repeat from step 2 onwards (Batch i+1).
   - if $t_i$ exceeds $\text{maxSteps}$, go to step 6.
   - else, set $\text{totalSteps}$ to $t_i$.
6. if graph heuristic H10 is being used, and is set to H10(a),
   set $C_{\text{best}}$ to $C_{\text{best}} \cup \{n_c \mid n_c \text{ is abandoned by } C_{\text{best}}\}$
7. return $C_{\text{best}}$ as the best cluster of $n_s$. 
ModifiedFindBestCluster

**input**: set `activeNodes`, graph `G'`

1. normalize the `nodeProbability` of all nodes in `activeNodes`
2. sort the nodes in `activeNodes` set, in the decreasing order of their degree-normalized `nodeProbabilities`.
3. candidate clusters `C_j` - set of nodes from 1 to `j`, in sorted order, where `j = min(maxClusterSize, |activeNodes|)`.
4. if the graph heuristic H10 is used, and is set to H10(b), then do the following:
   - set each `C_j` to `C_j ∪ {n_c | n_c is abandoned by C_j}`
   - if for any `j`, `|C_j|` exceeds `maxClusterSize`, discard `C_j`.
5. for all remaining candidate clusters, compute `Φ_{G'}`.
6. return that candidate, which has the smallest conductance.
Clustering using Nibble Algorithm

**Objective:** find the cluster to which seed node belongs

**Nibble Algorithm:**

input: Start Vertex $v$, Graph $G$, Conductance $\theta_0$, a positive integer $b$

1. Compute $t_0 \propto \ln(m)/\theta_0^2$, $\gamma \propto \theta_0/\ln(m)$, $\epsilon_b \propto \theta_0/\ln(m)t_02^b$
2. Start a lazy random walk from $v$
3. At each step: (until $t_0$)
   - Do the Truncation Operation with threshold $= \epsilon_b$
   - Sort the nodes in the decreasing order of their probabilities
   - Check if a $\tilde{j}$ exists such that:
     - $\Phi(\{1, ..., \tilde{j}\}) \leq \theta_0$
     - $Pr(\tilde{j}) \geq \gamma/Vol(\{1, ..., \tilde{j}\})$
     - $Vol(\{1, ..., \tilde{j}\}) \leq \frac{5}{6} Vol(V)$, then, output $C = \{1, ..., \tilde{j}\}$
4. Do the next step of random walk and repeat from Step (3)
Random Nibble Algorithm:
input: $G, \theta_0$

1. Set $v$ to be the largest degree vertex of $G$
2. Choose $b$ in $1, \ldots, \lceil \log(m) \rceil$ according to
   $$Pr[b = i] \propto 2^{-i}$$
3. Call Nibble($G, v, \theta_0, b$)

Partition Algorithm:
input: $G, \theta_0, p \in (0, 1)$

1. Compute number of iterations $j$ ($\propto m \lceil \text{lg}(1/p) \rceil$)
2. Start with the entire graph, i.e., set $W$ to $V$
3. Call RandomNibble($G(W), \theta_0$)
4. Add the cluster nodes returned by RandomNibble to the answer
5. Now, remove these nodes from $W$
6. If $Vol(W) \leq \frac{5}{6} Vol(V)$, then stop
7. Else, repeat from Step (3)
Multiway Partition Algorithm:
input: $G$, $\theta$, $p$

1. Set $\theta_0$ to $(5/36)\theta$
2. Compute number of iterations $t$ ($\propto (\lg m)^2$)
3. Start with the entire vertex set, i.e, set $C_1$ to $V$
4. In each step: For each component $C \in C_t$,
   Call Partition($G(C), \theta_0, p/m$)
5. Add the two partitions returned to $C_{t+1}$ and repeat from Step 4
6. Final clustering is given by $C_{t+1}$

Running Time:
Nibble: $O(2^b \ln^4(m)/\theta_0^5)$
Multiway Partition: $O(m (\lg(1/p) \lg^{O(1)}(m))/\theta^5)$
Clustering using Seed Sets [AL06]

Objective: find the enclosing community of a “seed set” of nodes

Algorithm:
1. Assign equal probabilities to all nodes in the seed set, and start spreading probabilities.
2. Sort the vertices in descending order of their degree-normalized probabilities.
3. Truncate the walk for nodes with probabilities lesser than a predefined threshold.
4. Find a $j$ such that the set of first $j$ nodes, $C$, satisfy the test for a good community: the probability outside $C$ is lesser than a predetermined fraction of $\Phi(C) \times \#numSteps$
5. If a $j$ is found, stop and return that set as the community.
6. Else, continue the random walk from step (2) onwards.

Shortcoming: The seed set is chosen manually.
Compression obtained maxDegree start is always higher than that of minDegree.

<table>
<thead>
<tr>
<th></th>
<th>edge compression</th>
<th>maxClusterSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>start node</td>
<td>200   400 800</td>
<td></td>
</tr>
<tr>
<td>min degree</td>
<td>11.81 14.39 16.95</td>
<td></td>
</tr>
<tr>
<td>max degree</td>
<td>12.78 15.53 18.55</td>
<td></td>
</tr>
</tbody>
</table>

Table: Edge compression on dblp3
<table>
<thead>
<tr>
<th>spreadProbability</th>
<th># clusters</th>
<th>edge compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>79,065</td>
<td>16.052</td>
</tr>
<tr>
<td>50</td>
<td>78,435</td>
<td>16.163</td>
</tr>
<tr>
<td>75</td>
<td>74,356</td>
<td>17.495</td>
</tr>
<tr>
<td>85</td>
<td>71,364</td>
<td>18.371</td>
</tr>
<tr>
<td>95</td>
<td>65,616</td>
<td>19.367</td>
</tr>
</tbody>
</table>

Edge compression on dblp3. (*settings: H2(b), mcs = 1500, no compaction.*)

- Edge compression increases with `spreadProbability`.
- Number of clusters reduces by about 13,500 - clusters found are of larger size.
- With higher `spreadProbability`, larger fraction of total probability can escape the cluster boundary.
- Larger clusters could be merging together multiple smaller ones.
- To avoid such effects, we use 0.5 for all the experiments.
H8 - compaction techniques

Following compaction techniques tried:

CP1  Blind and greedy compaction of all clusters
CP2  Edge aware compaction of all clusters
CP3  Naïve compaction of tiny clusters

- CP1 and CP2 improves edge compression, since they combine clusters which may have edges across them.
- But, applying CP1 and CP2, made the supernode graph, denser.
- Searching in a dense supernode graph, quickly spreads to a very large fraction of it, and can incur more cache misses.
- CP3 doesn’t affect edge compression and does make the supernode graph denser.

We choose CP3, since we want to strike a balance between the following:

- number of supernodes in the graph
- denseness of the supernode graph
| Q1 | sudarshan soumen                        |
| Q2 | vapnik support vector                  |
| Q3 | divesh jignesh jagadish timber querying XML |
| Q4 | sudarshan widom                        |
| Q5 | giora fernandez                        |
| Q6 | david fernandez parametric             |
| Q7 | chaudhuri agrawal                      |
| Q8 | widom database                         |
| Q9 | raghu deductive databases              |
| Q10| “prabhakar raghavan” “raghu ramakrishnan” |
| Q11| rozenberg “petri nets”                 |
| Q12| rozenberg janssens “graph grammars”    |
| Q13| silberschatz “disk arrays”             |
| Q14| ramamritham “real time”                |
| Q15| “howard siegel” SIMD                   |
| Q16| frieze “random graphs”                 |
| Q17| romanski ada                           |
| Q18| banerjee “distributed memory” multicomputers |
| Q19| didier “possibilistic logic”           |
| Q20| tamassia “graph drawing”               |

connection queries for db1p3 dataset
near queries for dblp3 dataset