PART B. GEAR SESSIONS
SESSION 3: BIG GRAPH ANALYSIS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs

• CS535 Online

• Please read announcements on Canvas
  • Follow the directions carefully
  • If you have any questions, please post on Piazza

• GEAR Sessions
  • Reviewers: Please submit your reviews
  • Presenters: Record your presentations and submit them as well as your reviews
  • Presentation/Discussion for the GEAR Session III will be available on April 6
  • No discussion session on April 1

• Stay connected!
Topics of Today's Class

• Large graph analysis paradigm
  • Part 1: Introduction and Graph Partitioning
  • Part 2: Computing model of Pregel
  • Part 3: Bulk Synchronization model and examples
This material is built based on

- Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, Names C. Dehnert, I Ian Horn, Naty Leiser, Grzegorz Czajkowski, "Pregel: a system for large-scale graph processing", Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 135-146

Graph?

- Graph algorithms are becoming increasingly important for solving many problems
- Graphs provide a flexible abstraction for describing relationship between discrete objects
- A Graph consists of a finite set of vertices and a set of edges
  - Adjacency Matrix
  - Adjacency List
Graph analysis application in Biology

- Protein-protein interaction networks
  - E.g. LCMA (Local Clique Merging Algorithm) iteratively identifies local cliques and merges them if they overlap significantly

- Gene regulatory networks
  - E.g. Deriving a gene regulatory network from observed gene expression levels

- Metabolic networks
  - E.g. Identifying common patterns in graph-based representations of metabolic pathways

- Tissue modeling
  - E.g. Classifying tissue samples by segmenting tissue images to identify cells

Graph analysis application in Network Security

- Network vulnerability analysis
  - E.g. Specifying the pre- and post- conditions of attacks using attack graphs
  - E.g. Defining a suite of metrics to qualify the vulnerability of a network based on its attack graph

- Malware detection
  - E.g. Analyzing download graphs to identify droppers, malicious programs that download other programs to a host machine
  - E.g. Estimating the probability of a file being malicious using bipartite graph of machine and files

- Botnet detection
  - E.g. Identifying a P2P botnet in a communication graph

- Anomaly detection
- Video Surveillance
Graph analysis applications in Software Engineering

- Identifying and locating software bugs
  - E.g. Mining software behavior graphs

- Detect prediction
  - E.g. Mining software dependency graph

- Software plagiarism detection
  - E.g. GPLAG: mining the software dependency graphs

Graph analysis applications in Social Science

- Identifying important users
  - E.g. Modeling Twitter users based on the Authority scores using PageRank

- Identifying interesting posts
  - E.g. Using the HITS algorithm

- Science, Finance, Linguistics, etc.
Discussions

- What will be computational challenges in graph computation?
  - Please write down at least 2 unique challenges on your note.

Challenges in the Large-scale graph processing

- Voluminous graph will be partitioned across the cluster

- Ingression time
  - Time that the system to load and partition the graph before starting the actual execution

- Communication cost
  - The amount of network transfer required during the executing the algorithm

- Load balancing
  - Quality of load distribution
  - Balanced graph partitioning is known as an **NP-complete problem**
GEAR Session 3. Big Graph Analysis
Lecture 1. Distributed Large Graph Analysis
Graph Partitioning 101

Graph partitioning: Edge-cut

- Vertices are equally distributed among partitions
  - Then, edges are distributed across partitions

- Edges along with the corresponding vertices are replicated and passed according to the requirement between partitions
  - Communication cost associated with edge-cut algorithms is directly proportional to the number of edges cut

- Both edge data and vertex data are passed to between partitions
Graph partitioning: Edge-cut: Example

- Both edge data and vertex data are passed to between partitions
- Example
  - Vertices are partitioned with a hash function:
    - \((\text{vertex id}) \mod n\)
    - Where, \(n\) is the total number of partitions
  - Suppose that \(n\) is 2
    - Vertices 1, 3, and 5 will be grouped in the same partition
    - 2 and 4 will be grouped separately
  - If both source and destination of edge are in the same partition, that edge will be grouped in the same partition
  - Otherwise, it will follow the destination

Discussions

- If your program updates “authority” value of this graph, how many network communication should be initiated? Assume that current “hub” and “authority” values are stored in each vertex.
Discussions

• If your program updates “authority” value of this graph, how many network communications should be initiated? Assume that current “hub” and “authority” values are stored in each vertex.

```
<table>
<thead>
<tr>
<th>Vertex</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Answers:
In the partition of Vertex 1, 3, and 5
For V1: none
For V5: with 2 and 4
For V3: with 4
In the partition of Vertex 2 and 4
For V2: with 1 and 5
For V4: with 5

Graph partitioning: Vertex-cut

• Edges are equally distributed

• Then, vertices are cut and replicated across the partitions
  • Vertex data is passed between the partitions
  • Communication cost is directly proportional to the number of the vertex replicas
  • Load balancing factor is determined by the number of edges assigned to each of the partitions

• Passes just vertex data
Graph partitioning: Vertex-cut: Example

- **Passes just vertex data**
- **Example**
  - Distribute edges equally over 2 partitions with a hash function, \((V_s + V_d) \mod n\), where \(V_s\) is the source vertex id, and \(V_d\) is the destination vertex id
  - \((1,2), (5,2), (2,5), (5,4), (4,5), (4,3)\) are placed into the partition 1
    - Rest of them are placed into the partition 2
  - Finally, it will create incomplete subgraphs

Discussions

- If your program updates “authority” value of this graph, how many network communication should be initiated? Assume that current “hub” and “authority” values are stored in each vertex.
Discussions

- If your program updates “authority” value of this graph, how many network communications should be initiated? Assume that current “hub” and “authority” values are stored in each vertex.

Partition 1
1
2
5
3
4

Partition 2
1
2
5
3
4

Answer:
V1: none
V2: none
V3: After update local it should synchronize the value with V3 in other partition
V4: After update local it should synchronize the value with V4 in other partition
V5: After update local it should synchronize the value with V5 in other partition

Types of Graph Process Models

- **Vertex-centric (Edge-cut)**
  - Synchronous: Giraph, GraphLab, Blogel/Vertex
  - Asynchronous: GraphLab
  - These systems cannot handle the Power Law like uneven distribution

- **Edge-centric (Vertex-cut)**
  - PowerGraph
  - GraphX

- **Block-centric: Blogel-Block (Blogel-B)**
  - MapReduce and its extensions: Hadoop
  - Relational: Vertica
  - Streaming Graph: Flink Gelly
GEAR Session 3. Big Graph Analysis
Lecture 1. Distributed Large Graph Analysis
Pregel: “Think Like A Vertex!”

Graph analysis at Google?

• Cluster computing tasks
  • Google’s 80% of data analysis
• Large-scale web search indexing
  • Clustering problems for Google News
  • Produce reports for popular queries (e.g. Google Trend)
  • Processing of satellite imagery data
  • Language model processing for statistical machine translations • Large-scale machine learning problems
• Back-up/restore
• The other 20%?
Graph analysis at Google?

- Large graph analysis
- **Graph algorithms**
  - PageRank
  - Shortest path
  - Connected components
  - Clustering techniques
- **Graph data**
  - Web graph
  - Transportation routes
  - Citation relationships
  - Social networks

What is so unique about “processing large graphs”?

- **Poor locality of memory access** for graph algorithms
  - Very little work per vertex
  - A changing degree of parallelism over the course of execution

- MapReduce is NOT a solution for graph processing
  - Many iterations are needed for parallel graph processing
  - Materializations of intermediate results at every MapReduce iteration causes performance bottleneck
GEAR Session 3. Big Graph Analysis
Lecture 1. Distributed Large Graph Analysis
Pregel: “Think Like A Vertex!”

Understanding distributed graph process with the SSSP example

Single Source Shortest Path (SSSP)

- Find shortest path from a source node to all target nodes
- If you have a single processor machine?
- Dijkstra’s algorithm
GEAR Session 3. Big Graph Analysis
Lecture 1. Distributed Large Graph Analysis
Pregel: “Think Like A Vertex!”

Understanding distributed graph process
with the SSSP example in a Single Machine

Dijkstra’s Algorithm (single node)

<table>
<thead>
<tr>
<th>A(Source)</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
Dijkstra's Algorithm (single node)

![Graph for Dijkstra's Algorithm](image)

<table>
<thead>
<tr>
<th>A(Source)</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>5/A</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

Dijkstra's Algorithm (single node)

![Graph for Dijkstra's Algorithm](image)

<table>
<thead>
<tr>
<th>A(Source)</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>5/A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>5/A</td>
<td>14</td>
<td>7/C</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm (single node)

<table>
<thead>
<tr>
<th>A(Source)</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>5/A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>5/A</td>
<td>14</td>
<td>7/C</td>
</tr>
<tr>
<td>0</td>
<td>8/C</td>
<td>5/A</td>
<td>13</td>
<td>7/C</td>
</tr>
</tbody>
</table>

GEAR Session 3. Big Graph Analysis
Lecture 1. Distributed Large Graph Analysis
Pregel: “Think Like A Vertex!”
Understanding distributed graph process with the SSSP example with MapReduce
Dijkstra’s Algorithm (with MR)

**Adjacency Matrix**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Adjacency List**

- A: (B, 10), (C, 5)
- B: (C, 2), (D, 1)
- C: (B, 3), (D, 9)
- D: (E, 4)
- E: (A, 7), (D, 6)
Dijkstra's Algorithm (with MR)

- Representing this graph in your mapper
  - `<source, <total_dist_from_the_node_A, <an_adjacency_list_of_the_source_node>>`
  - E.g. node A has two outgoing edges to the node B and C
    - `<A, 0, <(B, 10), (C, 5)>>`

- Mapper will have input:
  - `<A, 0, <(B,10), (C, 5)>>`
  - `<B, inf, <(C,2), (D,1)>>`
  - `<C, inf, <(B,3), (D,9), (E,2)>>`
  - `<D, inf, <(E,4)>>`
  - `<E, inf, <(A,7), (D,6)>>`

---

Dijkstra's Algorithm (with MR)

- Mapper calculates `<destination_node_ID, dist>`
  - Input: `<A, 0, <(B,10), (C, 5)>>`
  - Output: `<A, 0, 0, <(B,10), (C, 5)>>`
  - `<B, inf, <(C,2), (D,1)>>`
  - `<C, inf, <(B,3), (D,9), (E,2)>>`
  - `<D, inf, <(E,4)>>`
  - `<E, inf, <(A,7), (D,6)>>`

- Key
  - Distance to the source + distance from the source

http://www.cs.colostate.edu/~cs535 Spring 2020 Colorado State University, page 19
Dijkstra’s Algorithm (with MR)

• A reducer will take inputs:
  • [ <A, <0, <(B,10), (C, 5)>>>, <A, 0>, <A, inf>]  

• Similarly, others will have:

  • [<B, <inf, <(C,2), (D,1)>>>, <B, inf>, <B, inf>, <B, 10>]
  • [<C, <inf, <(B,3), (D,9), (E,2)>>>, <C, 5>, <C, inf>, <C, inf>]
  • [<D, <inf, <(E,4)>>>, <D, inf>, <D, inf>, <D, inf>, <D, inf>]
  • [<E, <inf, <(A,7), (D,6)>>>, <E, inf>, <E, inf>, <E, inf>]

Dijkstra’s Algorithm (with MR)

• A reducer will select the minimum distance available, and update inputs

  • [ <A, <0, <(B,10), (C, 5)>>>, <A, 0>, <A, inf>]  
  • [<B, <inf, <(C,2), (D,1)>>>, <B, inf>, <B, inf>, <B, 10>]
  • [<C, <inf, <(B,3), (D,9), (E,2)>>>, <C, 5>, <C, inf>, <C, inf>]
  • [<D, <inf, <(E,4)>>>, <D, inf>, <D, inf>, <D, inf>, <D, inf>]
  • [<E, <inf, <(A,7), (D,6)>>>, <E, inf>, <E, inf>, <E, inf>]

http://www.cs.colostate.edu/~cs535
Dijkstra’s Algorithm (with MR)

- Mapper calculates <dist node ID, dist>
  - <A, <0, (B, 10), (C, 5)>, <B, 10>, <C, 5>
  - <B, <10, (C, 2), (D, 1)>, <C, 2+10>, <D, 1+10>
  - <C, <5, (B, 3), (D, 9), (E, 2)>, <C, 5, B, 3+5>, <D, 9+5>, <E, 2+5>
  - <D, <inf, (E, 4)>>, <D, <inf, (E, 4)>, <E, inf>
  - <E, <inf, (A, 7), (D, 6)>, <E, <inf, (A, 7), (D, 6)>, <A, inf>, <D, inf>

- A reducer will select the minimum distance available
  - [ <A, <0, (B, 10), (C, 5)>, <A, 0>, <A, inf> ]
  - [ <B, <10, (C, 2), (D, 1)>, <B, inf>, <B, 8>, <B, 10> ]
  - [ <C, <5, (B, 3), (D, 9), (E, 2)>, <C, 5, C, 12>, <C, inf> ]
  - [ <D, <inf, (E, 4)>, <D, 11>, <D, 12>, <D, inf>, <D, inf> ]
  - [ <E, <inf, (A, 7), (D, 6)>, <E, 7>, <E, inf>, <E, inf> ]
Dijkstra’s Algorithm (with MR)

- Mapper calculates <dist node ID, dist>
  - <A, <0, (<B,10>, (C, 5))>, <B, 10>, <C, 5>
  - <B, <8, (<C,2), (D,1)>>>, <B, <10, (<C,2), (D,1)>>>, <C, 2+8>, <D, 1+8>
  - <C, <5, (<B,3), (D,9), (E,2)>>>, <C, 5, <B, 3+5>, <D, 9+5>, <E, 2+5>
  - <D, <11, (<E,4)>>>, <D, <11, (<E,4)>>>, <E, 4+11>
  - <E, <7, (<A,7), (D,6)>>>, <E, <7, (<A,7), (D,6)>>>, <A, 7+7>, <D, 6+7>
  - Repeat these process

Discussions

- When should your computation stop?
  
  a. When there is no change in the vertex values
  b. When there is no “inf” value in tuples
  c. When the sum of vertex values reaches the maximum
  d. When the sum of vertex values reaches the minimum
Discussions

• When should your computation stop?

  a. When there is no change in the vertex values
  b. When there is no "inf" value in tuples
  c. When the sum of vertex values reaches the maximum
  d. When the sum of vertex values reaches the minimum

Discussions

• Can we convert this to Apache Spark programming?
Discussions

• Can we convert this to Apache Spark programming?
  • Create a RDD and use reduceByKey()
  • Repeat same functionality in reducers

GEAR Session 3. Big Graph Analysis
Lecture 1. Distributed Large Graph Analysis
Pregel: “Think Like A Vertex!”
Bulk Synchronized Parallel Model
Bulk Synchronized Parallel Model

Inspired by Valiant's Bulk Synchronous Paradigm (1991)

A Superstep

Local computation

Computation Barrier Synchronization

End of the first iteration

End of the second iteration
Computation Model

- Data partitioning: Edge-cut
  - Vertices are distributed evenly

- Inputs to the Pregel computation
  - A directed graph
  - Vertex
    - String vertex ID
    - Associated user defined value
  - Edge
    - Associated with their source vertices
    - User defined value and a target vertex ID

Computation Model

- Computation in the vertex
  - Performed in parallel
  - Executes the same user-defined function
  - Modifies the state
    - Sometimes changes the outgoing edges
  - Receive/send message
  - Mutate topology
    - There is no computation associated with the edges
Computation Model: Vertex State Machine

- Computation in the vertex
  - Performed in parallel
  - Executes the same user-defined function
  - Modifies the state
    - Sometimes changes the outgoing edges
  - Receive/send message
  - Mutate topology
    - There is no computation associated with the edges

Computation Model: Vertex State Machine

- In the Superstep 0
  - Every vertex is in the “ACTIVE” state
  - All active vertices participate in the computation of any given superstep
  - A vertex deactivates itself by voting to halt
    - This is a user-defined function
      - The vertex has no further work to do unless triggered externally
      - Pregel will not process that vertex in subsequent supersteps
  - Unless there is a messages passed from the previous superstep
  - Once a vertex is re-activated, it must explicitly deactivate itself again
Output of Pregel program

- Set of values explicitly output by the vertices
- Often a directed graph isomorphic to the input
  - E.g. clustering algorithm
    - Creates small set of disconnected vertices selected from a large graph
  - E.g. graph mining algorithm
    - Generates aggregated statistics mined from the graph

Message Passing

- Vertices communicate directly with one another by sending messages
  - Message value
  - Name of the destination vertex

- A vertex can send any number of messages in a superstep

- There is no guaranteed order of messages in the iterator
  - However, message is delivered reliably
  - There will be no duplicate
**Message Passing**

- **Common usage pattern**
  - A vertex $V$ to iterate over its outgoing edges and sending a message to the destination vertex of each edge

- **Destination vertex need not be a neighbor of $V$**
  - E.g. A vertex can learn the identifier of a non-neighbor from a message received earlier
  - E.g implicitly vertex info is distributed

- If destination does not exist, user-defined handler will be executed
  - Create the missing vertex or remove the dangling edge

---

**SSSP using Parallel BFS in Pregel**

Q: What is the state of each node?
SSSP using Parallel BFS in Pregel

Q: What is the state of each node? --Active

SSSP using Parallel BFS in Pregel

A sends "10" to B
B sends "∞" to C
B sends "∞" to D
C sends "∞" to B
C sends "∞" to D
C sends "∞" to E
D sends "∞" to E
E sends "∞" to A
E sends "∞" to D
E sends "∞" to A

Vote to halt
Message received
SSSP using Parallel BFS in Pregel

After the first iteration, the state of each node is updated:

After the first iteration, the state of each node is updated:

A does not send anything
B sends "12" to C
B sends "11" to D

C sends "8" to B
C sends "9" to D
C sends "7" to E

E does not send anything
D does not send anything
SSSP using Parallel BFS in Pregel

After the second iteration, the state of each node is updated:

Active: B, D, and E
Inactive: A and C

Discussion: What is the state of each node?

B sends “10” to C
B sends “9” to D
D sends “15” to E
A does not send anything
C does not send anything
E sends “14” to A
E sends “13” to D
SSSP using Parallel BFS in Pregel

After the third iteration, the state of each node is updated:

After the second iteration, the state of each node is updated:
SSSP using Parallel BFS in Pregel

Implementation with Pregel

class ShortestPathVertex
: public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(), mindist + iter.GetValue());
      VoteToHalt();
    }
  }
};
GEAR Session 3. Big Graph Analysis
Lecture 1. Distributed Large Graph Analysis
Pregel: “Think Like A Vertex!”

Bulk Synchronized Parallel Model: Applications

PageRank example with Pregel

```cpp
class PageRankVertex : public Vertex<double, void, double> {
public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() = 0.15 / NumVertices() + 0.85 * sum;
    }
    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else { VoteToHalt();
  }
};
```
Maximum Bipartite Matching

- Bipartite Matching
  - Finding a set of edges chosen in such a way that no two edges share an end point

- Maximum Bipartite Matching
  - Matching of maximum size

- Inputs
  - Two distinct sets of vertices
  - Edges only between the sets

- Outputs
  - Subset of edges with no common endpoints
Randomized Maximal Matching algorithm

- **Vertex value**
  - A tuple of two values
  - A flag indicating which set of vertex is in (L or R)
  - The name of its matched vertex once known

- **Edge value**
  - Type void (edges carry no information)
  - Messages are Boolean

- **Algorithm proceeds in cycles of four phases**
  - Phase index = the superstep index % 4
  - Uses a three-way handshake

---

Randomized Maximal Matching algorithm

- **Phase 0 of a cycle**
  - Each left vertex not yet matched sends a message to each of its neighbors to request a match
  - Unconditionally votes to halt

- **Phase 1 of a cycle**
  - Each right vertex not yet matched randomly chooses one of the messages it receives
  - Sends a message granting that request
  - Sends a denying message to others
  - Unconditionally votes to halt
Randomized Maximal Matching algorithm

• Phase 2 of a cycle
  • Each left vertex not yet matched chooses one of the grants in receives
  • Sends an acceptance message

• Phase 3 of a cycle
  • An unmatched right vertex receives at most one acceptance message
  • Notes the matched node
  • Unconditionally votes to halt

• Repeats Phase 0 ~ 3

Semi-Clustering

• Social graph mining
• Vertices
  • People
• Edges
  • Connections between people
  • Explicit actions (e.g., adding a friend in a social networking site)
  • Behavior (e.g., email conversations or co-publication)
  • May have weights, to represent the interactions’ frequency or strength
• A semi-cluster in a social graph is a group of people who interact frequently with each other and less frequently with others
  • A vertex may belong to more than one semi-cluster
Semi-Clustering: Parallel greedy semi-clustering

- Input is a weighted, undirected graph
- Output is at most $C_{\text{max}}$ semi-clusters
  - Each containing at most $V_{\text{max}}$ vertices, where $C_{\text{max}}$ and $V_{\text{max}}$ are user-specified parameters
- A semi-cluster $C$ is assigned a score
  - $S_C = \frac{I_C - f_B B_C}{V_C (V_C - 1) / 2}$
    - Where $I_C$ is the sum of the weights of all internal edges
    - $B_C$ is the sum of the weights of all boundary edges
    - $V_C$ is the number of vertices in the semi-cluster
    - $f_B$ is the boundary edge score factor (user specified parameter, usually between 0 and 1)
- The score is normalized
  - Divided by the number of edges in a clique of size $V_C$

Each vertex $V$ maintains a list of containing at most $C_{\text{max}}$ semi-clusters, sorted by score

In Superstep 0
- $V$ enters itself to the list as a semi-cluster of size 1 and score 1
- Publishes itself to all of its neighbors
- Vertex $V$ iterates over the semi-clusters $c_1, c_2, \ldots, c_k$ sent to it on the previous superstep
- If $c$ does not contain any of $V$, and $V_C < M_{\text{max}}$, then $V$ is added to $c$ to form $c'$
- The semi-clusters $c_1, c_2, \ldots, c_k, c', c_2', \ldots, c_k'$ are sorted by their scores and the best ones are sent to $V$'s neighbors
- Vertex $V$ updates its list of semi-clusters with the semi-clusters from $c_1, c_2, \ldots, c_k, c', c_2', \ldots, c_k'$ that contain $V$. This algorithm terminates either when the semi-clusters stop changing or when the number of supersteps reaches a user-specified limit
Questions?