Enabling Scalable Social Group Analytics via Hypergraph Analysis Systems

Benjamin Heintz, Abhishek Chandra
University of Minnesota
Big Social Data

• Rapid growth in *social* data
  – Likes
  – Tweets
  – Publications

• Transform into *knowledge*
  – Importance / centrality / influence
  – Community detection
  – Information flow
State of the Art: *Graph* Systems

- *Pairwise* interactions
  - Writing a paper
  - Attending an event
  - Appearing on a TV show

- Wave of systems work
  - Pregel
  - GraphLab
  - GraphX

Vertex- or edge-centric programming models
What about *Groups*?

- Interactions: more than just *pairs*
  - Papers
  - Photos
  - Events

- How to model groups?

*Groups are the basis for many social interactions.*

*Graphs poorly model groups.*
Hypergraphs Model Groups

\[ H = (V, E) \]
Hypergraphs Model Groups

$H = (V, E)$

vertex

hyperedge
Hypergraphs Model Groups

\[ H = (V, E) \]
Hypergraphs Model Groups

To better model group interactions, we need hypergraph analysis systems.

$H = (V, E)$
# Alternative Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affiliation network</td>
<td>Models groups explicitly</td>
<td>Tedious to implement; wrong level of abstraction</td>
</tr>
<tr>
<td>1-mode graph projection</td>
<td>No need for hypergraph systems</td>
<td>Tedious; large size; not always applicable</td>
</tr>
<tr>
<td><em>Hypergraphs</em></td>
<td>Level of abstraction matches the domain</td>
<td>Requires hypergraph systems!</td>
</tr>
</tbody>
</table>
Roadmap

- Motivation
- **Interface**
- Implementation
- Evaluation
- Discussion
Example: PageRank

- Rank vertices by importance
Example: PageRank

• Rank vertices by importance
Example: PageRank

- Rank vertices by importance
- Extension: rank **hyperedges** too
Example: PageRank

• Rank vertices by importance
• Extension: rank *hyperedges* too
Example: PageRank

• Rank vertices by importance
• Extension: rank *hyperedges* too
• Extension: arbitrary hyperedge behavior
Example: PageRank

- Rank vertices by importance
- Extension: rank hyperedges too
- Extension: arbitrary hyperedge behavior

In a hypergraph system, hyperedges are first-class objects.
Iterative Computation

Graph

Hypergraph

v1
v2
v3

v1
v2
v3

v1
v2
v3

v1
v2
v3

v1
v2
v3

v1
v2
v3

v1
v2
v3

Heintz & Chandra | HotCloud '15
Iterative Computation

Graph

Hypergraph

Heintz & Chandra | HotCloud '15
A Hypergraph API

```scala
trait HyperGraph[HVD, HED] {
  def compute[ToE, ToV](
    maxIters: Int,
    initialMsg: ToV,
    hvProgram: Program[HVD, ToV, ToE],
    heProgram: Program[HED, ToE, ToV]) :
    HyperGraph[HVD, HED]
}

object HyperGraph {
  trait Program[A, InMsg, OutMsg] {
    def messageCombiner: MessageCombiner[OutMsg]
    def procedure: Procedure[A, InMsg, OutMsg]
  }

  type MessageCombiner[Msg] = (Msg, Msg) => Msg

  type Procedure[A, InMsg, OutMsg] =
    (Int, NodeId, A, InMsg, Context[A, OutMsg]) => Unit

  trait Context[A, OutMsg] {
    def become(attr: A): Unit
    def send(msgF: NodeId => OutMsg,
              to: Recipients): Unit
  }
}
```

Iterative computation

Vertex and hyperedge programs

Messages to neighbors
Roadmap

• Motivation
• Interface
• Implementation
• Evaluation
• Discussion
Implementation

• Challenges
  – Representation
  – Partitioning

• Initial approach
  – Build upon existing graph systems
## Hypergraph Representations

<table>
<thead>
<tr>
<th>Representation</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bipartite Graph</strong></td>
<td>Portable to any graph system</td>
<td>Obscures hyperedge / vertex differences</td>
</tr>
<tr>
<td><img src="image" alt="Bipartite Graph Diagram" /></td>
<td><img src="image" alt="Bipartite Graph Diagram" /></td>
<td><img src="image" alt="Bipartite Graph Diagram" /></td>
</tr>
<tr>
<td><strong>Multigraph</strong></td>
<td>Exploits hyperedge / vertex differences</td>
<td>Size overhead</td>
</tr>
<tr>
<td><img src="image" alt="Multigraph Diagram" /></td>
<td><img src="image" alt="Multigraph Diagram" /></td>
<td><img src="image" alt="Multigraph Diagram" /></td>
</tr>
</tbody>
</table>
Hypergraph Partitioning

• Partition the underlying graph
  – Cut vertices, cut edges, both

• Or use hypergraph-aware partitioning
  – Differentiate hyperedges and vertices
Roadmap

• Motivation
• Interface
• Implementation

• Evaluation
• Discussion
### Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Vertices</th>
<th>Hyperedges</th>
<th>Bipartite Edges</th>
<th>1-mode Projection Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBLP</strong></td>
<td>952,115 authors</td>
<td>916,947 collaborations</td>
<td>2,768,930</td>
<td>21,592,883</td>
</tr>
<tr>
<td><strong>Friendster</strong></td>
<td>7,944,949 users</td>
<td>1,620,991 communities</td>
<td>23,479,217</td>
<td>&gt;15.1B</td>
</tr>
</tbody>
</table>
Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Vertices</th>
<th>Hyperedges</th>
<th>Bipartite Edges</th>
<th>1-mode Projection Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP</td>
<td>952,115</td>
<td>916,947</td>
<td>2,768,930</td>
<td>21,592,883</td>
</tr>
<tr>
<td>Friendster</td>
<td>7,944,949 users</td>
<td>1,620,991 communities</td>
<td>23,479,217</td>
<td>&gt; 15.1 B</td>
</tr>
</tbody>
</table>

Real-world hypergraphs exhibit diverse characteristics.
Prototype

• *Proof-of-concept* prototype
• Implemented on Apache Spark GraphX 1.2.1
• Run on shared 6-node cluster (2x6-core, 24GB RAM each)
• Using bipartite graph representation
Experimental Results (1/2)

**DBLP**
- **PR**
- **PR-Entropy**

**Friendster**
- **PR**
- **PR-Entropy**
Experimental Results (1/2)

Scalability is a critical challenge.

Graphs showing the execution time (s) vs. the number of bipartite edges (millions) for DBLP and Friendster datasets. The graphs compare the performance of PR and PR-Entropy algorithms.
Experimental Results (2/2)

Partitioning: DBLP

- Cut Vertices
- Cut Hyperedges
- Cut Both

Execution Time (s)

<table>
<thead>
<tr>
<th></th>
<th>PR</th>
<th>PR-Entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut Vertices</td>
<td>400</td>
<td>500</td>
</tr>
<tr>
<td>Cut Hyperedges</td>
<td>400</td>
<td>500</td>
</tr>
<tr>
<td>Cut Both</td>
<td>500</td>
<td>600</td>
</tr>
</tbody>
</table>
Experimental Results (2/2)

Performance significantly affected by dataset + algorithm + partitioning
Conclusion

• Social data $\rightarrow$ knowledge
• State of the art: graphs
• Need to model groups
• Hypergraphs are the right abstraction
• System challenges
Roadmap

• Motivation
• Interface
• Implementation
• Evaluation
• Discussion
Discussion

• *Programming Model & API*
  – Synchronous vs. asynchronous
  – Directed, temporal, ...

• *Implementation*
  – Build on graph systems or from scratch?
  – Representations, partitioning techniques
Thank you!
dcsg.cs.umn.edu