Signal Processing on Databases

Jeremy Kepner

Lecture 0: Introduction

This work is sponsored by the Department of the Air Force under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.
Acknowledgements

• Nicholas Arcolano
• Michelle Beard
• Nadya Bliss
• Josh Haines
• Matthew Schmidt
• Ben Miller
• Benjamin O’Gwynn
• Tamara Yu
• Bill Arcand
• Bill Bergeron

• David Bestor
• Chansup Byun
• Matt Hubbell
• Pete Michaleas
• Julie Mullen
• Andy Prout
• Albert Reuther
• Tony Rosa
• Charles Yee
• Dylan Hutchinson
Outline

• Introduction
• Course Outline
• Example Implementation
• Summary
Example Applications of Graph Analytics

**ISR**
- Graphs represent entities and relationships detected through multi-INT sources
- 1,000s – 1,000,000s tracks and locations
- GOAL: Identify anomalous patterns of life

**Cyber**
- Graphs represent communication patterns of computers on a network
- 1,000,000s – 1,000,000,000s network events
- GOAL: Detect cyber attacks or malicious software

**Social**
- Graphs represent relationships between individuals or documents
- 10,000s – 10,000,000s individual and interactions
- GOAL: Identify hidden social networks

**Cross-Mission Challenge:** Detection of subtle patterns in massive multi-source noisy datasets
Example Applications of Graph Analytics

**ISR**
- Graphs represent entities and relationships detected through multi-INT sources
- 1,000s – 1,000,000s tracks and locations
- GOAL: Identify anomalous patterns of life

**Social**
- Graphs represent relationships between individuals or documents
- 10,000s – 10,000,000s individual and interactions
- GOAL: Identify hidden social networks

**Cyber**
- Graphs represent communication patterns of computers on a network
- 1,000,000s – 1,000,000,000s network events
- GOAL: Detect cyber attacks or malicious software

**Cross-Mission Challenge**: Detection of subtle patterns in massive multi-source noisy datasets
Example: Web Traffic Graph

Graph Statistics
- 90 minutes worth of traffic
- 1 frame = 1 minute of traffic
- Number of source computers: 4,063
- Number of web servers: 16,397
- Number of logs: 4,344,148

Malicious Activity Statistics
- Number of infected IPs: 1
- Number of event logs: 16,000
- % infected traffic: 0.37%
- Existing tools did not detect event
- Detection took 10 days and required manual log inspection

Challenge: Activity signature is typically a weak signal
Big Data Challenge: Data Representation

- Raw data sources are rarely stored in a graph format
- Data is often derived from multiple collection points

- Many different graphs can be built from a single data source
- Constructing a single graph may require many sources
- Building multi-graphs requires that entities be normalized

Challenge: Raw data source representations do not enable the efficient construction of graphs of interest
Technology Stack

Graph Analytics

High Level Languages

Distributed Storage and Indexing

High Performance Processing

Applicability
- Cyber, COIN, ISR, Bioinformatics

Resiliency
- Uncertainty in data and observation

Scalability
- Parallel language support

Programmability
- Automated performance optimization

Portability
- Bindings to multiple databases

Elasticity
- Virtual machine development

Performance
- Novel instruction set architectures

Efficiency
- Specialized circuitry and communication
Outline

• Introduction
• Course Outline
• Example Implementation
• Summary
The MIT Formula

**Theory**
- Academics
- Departments
  - EECS, Math, Physics, …
- Mathematics
- Algorithms
- Software

**Experiment**
- Research
- Laboratories
  - Lincoln, CSAIL, Media, …
- Measurement
- Data
- Bytes

---

**Discovery**
Software and Bytes Live on Parallel Computers

Parallel Architecture

- CPU
- RAM
- Disk

Network Switch

Memory Hierarchy

- Registers
- Instruction Operands

- Cache
- Blocks

- Local Memory
- Messages

- Remote Memory
- Pages

- Disk

Implications

- Bandwidth: High
- Latency: High
- Programmability: High
- Capacity: High

Nearly all modern computers are Von Neumann architectures with multi-level memory hierarchies. The architecture selects the algorithms and data that run well on it.
Software Performance vs. Parallel Programmer Effort

- Goal: Software that does a lot with the least effort
Data Use Cases

- Data volume and data request size determine the best approach.
- Always want to start with the simplest and move to the most complex.

<table>
<thead>
<tr>
<th>Total Data Volume</th>
<th>Serial Program</th>
<th>Parallel Program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Serial or Parallel Program + Database</td>
<td>Parallel Program + Parallel Database</td>
</tr>
<tr>
<td></td>
<td>Serial Program + Files</td>
<td>Parallel Program + Parallel Files</td>
</tr>
</tbody>
</table>

- Serial memory
- Parallel memory / Serial storage
- Parallel storage
The class teaches the highest performance and lowest effort software techniques that are currently known.
Key Course Concepts

• Bigger definition of a graph
  – How to move beyond random, undirected, unweighted graphs to power-law, directed, multi-hyper graphs

• Bigger definition of linear algebra
  – How to move beyond real numbers to doing math with words and strings

• Bigger definition of processing
  – How to move beyond map/reduce to distributed arrays programming

• These abstract concepts are the foundation for high performance signal processing on large unstructured data sets
Course Outline

• Introduction
  – Review course goals and structure

• Using Associative Arrays
  – Schemas, incidence matrices, and directed multi-hyper graphs

• Group Theory
  – Extending linear algebra to words using fuzzy algebra

• Entity Analysis in Unstructured Data
  – Reading and parsing unstructured data

• Analysis of Structured Data
  – Graph traversal queries

• Power Law Data
  – Models and fitting

• Cross Correlation
  – Sequence data, computing degree distributions, and finding matches

• Parallel Processing
  – Kronecker graphs, parallel data generation and computation

• Databases
  – Relational, triple store, and exploded schemas
References

• Book: “Graph Algorithms in the Language of Linear Algebra”
• Editors: Kepner (MIT-LL) and Gilbert (UCSB)
• Contributors:
  – Bader (Ga Tech)
  – Bliss (MIT-LL)
  – Bond (MIT-LL)
  – Dunlavy (Sandia)
  – Faloutsos (CMU)
  – Fineman (CMU)
  – Gilbert (USCB)
  – Heitsch (Ga Tech)
  – Hendrickson (Sandia)
  – Kegelmeyer (Sandia)
  – Kepner (MIT-LL)
  – Kolda (Sandia)
  – Leskovec (CMU)
  – Madduri (Ga Tech)
  – Mohindra (MIT-LL)
  – Nguyen (MIT)
  – Radar (MIT-LL)
  – Reinhardt (Microsoft)
  – Robinson (MIT-LL)
  – Shah (USCB)
Outline

• Introduction
• Course Outline
• Example Implementation
• Summary
Constructing Graph Representations of Raw Data Source

- Raw data sources can contain information about multiple types of relations between entities.
- The process of constructing a graph representation is specific to both the data source and the relationships represented by the graph.

- The development time of parsing and graph construction algorithms can overwhelm the runtime of the algorithm.

1. Parse edge and vertex information from raw data.
   - Developed Once Per Data Source Per Graph.

2. Convert edge lists into adjacency matrices.
   - Developed Once.

Diagram:
- Raw Data → Vertex and Edge Lists
- Information from raw data
  - Developed Once Per Data Source Per Graph
- (2) Convert edge lists into adjacency matrices
  - Developed Once
Graph Construction Using D4M

- D4M provides needed flexibility in the construction of large-scale, dynamic graphs at different resolutions and scopes.
Graph Construction Using D4M: Parsing Raw Data Into Dense Tables

Proxy Logs

```
128.0.0.1 208.29.69.138 "-" [10/May/2011:09:52:53] "GET http://www.thedailybeast.com/ HTTP/1.1" 200
1024 8192 "http://www.theatlantic.com/" "Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.13)
Gecko/20101209 CentOS/3.6-2.el5.centos Firefox/3.6.13" "bl" - "text/html" "MITLAB" 0.523 "-" Neutral TCP_MISS
10296 "-" "Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.13) Gecko/20101209 CentOS/3.6-
2.el5.centos Firefox/3.6.13" "bu" - "text/html" "MITLAB" 0.784 "-" Neutral TCP_MISS
...
```

<table>
<thead>
<tr>
<th>log_id</th>
<th>src_ip</th>
<th>server_ip</th>
<th>time_stamp</th>
<th>req_line</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>128.0.0.1</td>
<td>208.29.69.138</td>
<td>10/May/2011:09:52:53</td>
<td>GET <a href="http://www.thedailybeast.com/">http://www.thedailybeast.com/</a> HTTP/1.1</td>
</tr>
<tr>
<td>003</td>
<td>128.0.0.1</td>
<td>74.125.224.72</td>
<td>13/May/2011:11:05:12</td>
<td>GET <a href="http://www.google.com/">http://www.google.com/</a> HTTP/1.1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Graph Construction Using D4M: Explode Schema

- Raw Data
- CSV Files
- Distributed Database
- Assoc. Arrays

Use as row indices

Create columns for each unique type/value pair

<table>
<thead>
<tr>
<th>log_id</th>
<th>src_ip</th>
<th>server_ip</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>128.0.0.1</td>
<td>208.29.69.138</td>
</tr>
<tr>
<td>002</td>
<td>192.168.1.2</td>
<td>157.166.255.18</td>
</tr>
<tr>
<td>003</td>
<td>128.0.0.1</td>
<td>74.125.224.72</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>log_id</th>
<th>src_ip 128.0.0.1</th>
<th>src_ip 192.168.1.2</th>
<th>server_ip 157.166.255.18</th>
<th>server_ip 208.29.69.138</th>
<th>server_ip 74.125.224.72</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_id</td>
<td>001</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>log_id</td>
<td>002</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>log_id</td>
<td>003</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
**Graph Construction Using D4M: Storing Exploded Data as Triples**

D4M stores the triple data representing both the exploded table and its transpose.

### Table Triples

<table>
<thead>
<tr>
<th>Row</th>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_id</td>
<td>001</td>
<td>src_ip</td>
</tr>
<tr>
<td>log_id</td>
<td>001</td>
<td>server_ip</td>
</tr>
<tr>
<td>log_id</td>
<td>002</td>
<td>src_ip</td>
</tr>
<tr>
<td>log_id</td>
<td>002</td>
<td>server_ip</td>
</tr>
<tr>
<td>log_id</td>
<td>003</td>
<td>src_ip</td>
</tr>
<tr>
<td>log_id</td>
<td>003</td>
<td>server_ip</td>
</tr>
</tbody>
</table>

### Table Transpose Triples

<table>
<thead>
<tr>
<th>Row</th>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_ip</td>
<td>157.166.255.18</td>
<td>log_id</td>
</tr>
<tr>
<td>server_ip</td>
<td>208.29.69.138</td>
<td>log_id</td>
</tr>
<tr>
<td>server_ip</td>
<td>74.125.224.72</td>
<td>log_id</td>
</tr>
<tr>
<td>src_ip</td>
<td>128.0.0.1</td>
<td>log_id</td>
</tr>
<tr>
<td>src_ip</td>
<td>128.0.0.1</td>
<td>log_id</td>
</tr>
<tr>
<td>src_ip</td>
<td>192.168.1.2</td>
<td>log_id</td>
</tr>
</tbody>
</table>
Graph Construction Using D4M: Construct Associative Arrays

**D4M Query #1**

```plaintext
keys = T(:, 'time_stamp|10/May/2011:00:00:00', :, ...
    'time_stamp|13/May/2011:23:59:59',);

('log_id|001', 'time_stamp|11/May/2011:09:52:53', 1)
('log_id|003', 'time_stamp|13/May/2011:11:05:12', 1)
...
Graph Construction Using D4M: Construct Associative Arrays

D4M Query #1
\[ \text{keys} = T(:, 'time\_stamp|10/\text{May}/2011:00:00:00', :, \ldots 'time\_stamp|13/\text{May}/2011:23:59:59',); \]

D4M Query #2
\[ \text{data} = T(Row(\text{keys}), :); \]

\[
\begin{align*}
\text{('log\_id|001','server\_ip|208.29.69.138',1)} \\
\text{('log\_id|001','src\_ip|128.0.0.1',1)} \\
\text{('log\_id|001','time\_stamp|11/\text{May}/2011:09:52:53',1)} \\
\text{...}
\text{('log\_id|002','server\_ip|157.166.255.18',1)}
\text{('log\_id|002','src\_ip|192.168.1.2',1)}
\text{('log\_id|002','time\_stamp|12/\text{May}/2011:13:24:11',1)}
\text{...}
\text{('log\_id|003','server\_ip|74.125.224.72',1)}
\text{('log\_id|003','src\_ip|128.0.0.1',1)}
\text{('log\_id|003','time\_stamp|13/\text{May}/2011:11:05:12',1)}
\text{...}
\end{align*}
\]
Graph Construction Using D4M: Construct Associative Arrays

Raw Data → CSV Files → Distributed Database → Assoc. Arrays

\[
\text{D4M Query #1:} \quad \text{keys} = T(:,'time_stamp|10/May/2011:00:00:00',:,
| 'time_stamp|13/May/2011:23:59:59',);
\]

\[
\text{D4M Query #2:} \quad \text{data} = T(Row(\text{keys}), :);
\]

\[
\text{Associative Array} \quad \mathcal{A} = ( \text{('src\_ip|128.0.0.1','server\_ip|208.29.69.138',1)} \\
(\text{('src\_ip|128.0.0.1','server\_ip|74.125.224.72',1)} \\
(\text{('src\_ip|192.168.1.2','server\_ip|157.166.255.18',1)} \\
\ldots
\]

\[
\mathcal{G} \quad \text{data}(::'\text{src\_ip|*')} \cdot \text{data}(::'\text{server\_ip|*'});
\]

(\text{('src\_ip|128.0.0.1','server\_ip|208.29.69.138',1)} \\
(\text{('src\_ip|128.0.0.1','server\_ip|74.125.224.72',1)} \\
(\text{('src\_ip|192.168.1.2','server\_ip|157.166.255.18',1)} \\
\ldots
Graph Construction Using D4M: Construct Associative Arrays

Graphs can be constructed with minimal effort using D4M queries and associative array algebra.

D4M Query #1

keys $T(:, 'time_stamp|10/May/2011:00:00:00', :, \ldots 'time_stamp|13/May/2011:23:59:59',);$

D4M Query #2

data $T(\text{Row}(\text{keys}), :);$ 

Associative Array Algebra

$G \text{ data}(:, 'src_ip|*').' * \text{ data}(:, 'server_ip|*');$

Adj(G);
Constructing Graph Representation of One Week’s Worth of Proxy Data

- Ingested ~130 million proxy log records resulting in ~4.5 billion triples
- Constructed 604,800 secondwise source IP to server IP graphs
- Constructing graphs with different vertex types could be done without re-parsing or re-ingesting data

- Utilizing D4M could allow analysis to be run in nearly real-time (dependent on raw data availability)
Summary

• Big data is found across a wide range of areas
  – Document analysis
  – Computer network analysis
  – DNA Sequencing

• Currently there is a gap in big data analysis tools for algorithm developers

• D4M fills this gap by providing algorithm developers composable associative arrays that admit linear algebraic manipulation
Example Code and Assignment

• Example code
  - D4Muser_share/Examples/1Intro/1AssocIntro

• Assignment
  - Test your LLGrid account and D4M
  - Copy the D4Muser_share/Examples to your LL Grid home directory
  - Verify that you can run the above examples
    • Start Matlab
    • CD to your copy of the example
    • Run the Examples