Detection of Anomalous Events in Large-Scale Graphs

Matthew C. Schmidt

GraphEx 2012

April 18th, 2012

This work is supported by the Intelligence Advanced Research Projects Activity (IARPA) via Air Force Contract FA8721-05-C-0002. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA or the U.S. Government.
## Example Applications of Graph Analytics

<table>
<thead>
<tr>
<th>ISR</th>
<th>Social</th>
<th>Cyber</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="ISR Image" /></td>
<td><img src="image2.png" alt="Social Image" /></td>
<td><img src="image3.png" alt="Cyber Image" /></td>
</tr>
</tbody>
</table>

- **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 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
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
Big Data Challenge: Activity Signatures

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

Data Sources

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

Graph Construction

- 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
Challenge: Many graph techniques are intractable on modern graphs
Big Data Challenge: Latency Requirements

1977: Zachary’s Karate Club

- 3 Years Observation Period
- 34 Vertices
- 78 Edges
- None Latency Requirements

2011: Web Proxy Logs

- 90 Minutes Observation Period
- 20,000 Vertices
- 400,000 Edges
- Minutes to Seconds Latency Requirements

Challenge: To keep up with data collection rates the latency of results must decrease
Big Data Challenges

Activity Signatures

Data Representations

Problem Scales

Latency Requirements
Data and Information Processing Architecture

### Big Data Challenges

- Activity Signatures
- Data Representations
- Problem Scales
- Latency Requirements

### Architecture Elements

- Graph Analytics
- High Level Languages
- Distributed Storage and Indexing
- High Performance Processing
Data and Information Processing Architecture

Big Data Challenges

- Activity Signatures
- Data Representations
- Problem Scales
- Latency Requirements

Architecture Elements

- Graph Analytics
- High Level Languages
- Distributed Storage and Indexing
- High Performance Processing
Data and Information Processing Architecture

Big Data Challenges

- Activity Signatures
- Data Representations
- Problem Scales
- Latency Requirements

Architecture Elements

- Graph Analytics
- High Level Languages
- Distributed Storage and Indexing
- High Performance Processing
Data and Information Processing Architecture

Big Data Challenges

- Activity Signatures
- Data Representations
- Problem Scales
- Latency Requirements

Architecture Elements

- Graph Analytics
- High Level Languages
- Distributed Storage and Indexing
- High Performance Processing
Focus of Talk: Graph Analytics and High Level Languages

Big Data Challenges

- Activity Signatures
- Data Representations
- Problem Scales
- Latency Requirements

Architecture Elements

- Graph Analytics: SPG: Signal Processing for Graphs Framework
- High Level Languages: D4M: Dynamic Distributed Dimensional Data Model
- Distributed Storage and Indexing
- High Performance Processing
Outline

• Introduction

• Graph Analytics
  – SPG Framework
  – Applications

• High Level Languages
  – Graph Construction Using D4M
  – Application

• Summary
Statistical Detection Framework for Graphs

Graph Theory

Detection Theory

Develop fundamental graph signal processing concepts

Demonstrate in simulation

Apply to real data
Residuals Example: Expected Topology

Graph Model

\( E[G] \)

- Graph model defines the “expected value” of a graph’s topology
- Model can be predefined or computed from observed data

The SPG framework uses graph models of the expected topology to detect anomalous subgraphs
Residual graph represents the difference between the observed and expected.
Residuals Example: Anomalous Subgraph

- Observed Graph $G_1$
- Graph Model $E[G]$
- Residual Graph $R[G_1]$

$H_1 - H = R[G_1]$

- Residual graph represents the difference between the observed and expected
- *Coordinated* residuals will produce much stronger signal than uncoordinated residuals

Detection framework is designed to detect coordinated deviations from the expected topology
Processing Chain

Statistical detection framework for graph data
Processing Chain

1. **Input**
   - Graph
   - No cue

2. **Graph Model Construction**

3. **Residual Decomposition**

4. **Component Selection**

5. **Anomaly Detection**

6. **Identification**

7. **Temporal Integration**

8. **Dimensionality Reduction**
Processing Chain

Input:
- Graph
- No cue

Output:
- Statistically anomalous subgraph(s)
Processing Chain

GraphEx 2012 - 24
MCS 04/18/12
Temporal Integration

TEMPORAL INTEGRATION → GRAPH MODEL CONSTRUCTION → RESIDUAL DECOMPOSITION → COMPONENT SELECTION → ANOMALY DETECTION → IDENTIFICATION

Time Series = A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9

Filter = \( w_1 \ w_2 \ w_3 \ w_4 \ w_5 \ w_6 \ w_7 \ w_8 \ w_9 \)
Temporal Integration

- Temporal Integration
- Graph Model Construction
- Residual Decomposition
- Component Selection
- Anomaly Detection
- Identification

GraphEx 2012 - 26
MCS 04/18/12
Temporal Integration

- Temporal Integration
- Graph Model Construction
- Residual Decomposition
- Component Selection
- Anomaly Detection
- Identification
Temporal Integration

TEMPORAL INTEGRATION \rightarrow GRAPH MODEL CONSTRUCTION \rightarrow RESIDUAL DECOMPOSITION \rightarrow COMPONENT SELECTION \rightarrow ANOMALY DETECTION \rightarrow IDENTIFICATION
Temporal Integration

TEMPORAL INTEGRATION → GRAPH MODEL CONSTRUCTION → RESIDUAL DECOMPOSITION → COMPONENT SELECTION → ANOMALY DETECTION → IDENTIFICATION

\[ A = \sum (w_i \cdot A_i) \]
Graph Model Construction

- TEMPORAL INTEGRATION
- GRAPH MODEL CONSTRUCTION
- RESIDUAL DECOMPOSITION
- COMPONENT SELECTION
- ANOMALY DETECTION
- IDENTIFICATION

A

E(A)

R(A)

Observed

Expected

Residuals
Dimensionality Reduction

Select vectors pointing towards the strongest residuals

\[ \lambda_1, \lambda_2, \ldots, \lambda_N \]
Dimensionality Reduction

TEMPORAL INTEGRATION → GRAPH MODEL CONSTRUCTION → RESIDUAL DECOMPOSITION → COMPONENT SELECTION → ANOMALY DETECTION → IDENTIFICATION

**GraphEx 2012 - 32**

**MCS 04/18/12**

Vertex 1
Vertex 2
...
Vertex N

**Dimension 1**
**Dimension 2**

Project the data into a 2-dimensional space and compute the test statistic
Anomaly Detection

TEMPORAL INTEGRATION → GRAPH MODEL CONSTRUCTION → RESIDUAL DECOMPOSITION → COMPONENT SELECTION → ANOMALY DETECTION → IDENTIFICATION

GraphEx 2012 - 33
MCS 04/18/12

LINCOLN LABORATORY
Massachusetts Institute of Technology
Anomaly Detection

Use false alarm rate based on requirements and capabilities
Anomaly Detection

TEMPORAL INTEGRATION → GRAPH MODEL CONSTRUCTION → RESIDUAL DECOMPOSITION → COMPONENT SELECTION → ANOMALY DETECTION → IDENTIFICATION

GraphEx 2012 - 35
MCS 04/18/12

Equal Error Rate: 16.7%
Anomaly Detection

TEMPORAL INTEGRATION → GRAPH MODEL CONSTRUCTION → RESIDUAL DECOMPOSITION → COMPONENT SELECTION → ANOMALY DETECTION → IDENTIFICATION

Threshold

Receiver Operating Characteristic

1% False Alarm Rate
Anomaly Detection

GraphEx Model Construction
Residual Decomposition
Component Selection
Anomaly Detection
Identification

Threshold

Beyond 0.01% false alarm rate
Identification

Identify clusters to identify anomalous subgraph vertices and edges
Separate clusters to identify foreground and background vertices
Identification

Detection framework both detected anomaly and identified the anomalous subgraph.
Outline

• Introduction

• Graph Analytics
  – SPG Framework
  – Applications

• High Level Languages
  – Graph Construction Using D4M
  – Application

• Summary
Kill Chain of Example Malware

Possible Malware Kill Chain

(1) System infected while browsing the Internet

(2) Infected system signals command and control server that it is infected

(3) C&C server sends list of website ads to visit

(4) Infected system visits the listed website ads

(5) Ads pay C&C owner for the click-throughs

Malware Detection

- Majority of malware analysis is done at step (1)
- Most of the functional behavior of malware is independent of step (1)
- This allows malware developers large degrees of freedom to avoid detection strategies at step (1)
- Graph analysis can detect malware at steps (2) through (4)

Graph analysis detects malicious behavior at more functionally constrained steps in the malware kill chain

GraphEx 2012 - 42
MCS 04/18/12
The SPG detection framework is applied to detect malware behavior in graphs built from proxy logs.
Automated Event Detection

- **Normal Activity**
  - Source Machines
  - Web Servers

- **Event Activity**
  - Infected system
  - Web servers

- **Automatic detection within 15 minutes of infection (vs 10 days)**
- **Infected system clearly separated in 2D space and low (negligible) false alarm rate**
Another Application: Eigenanalysis of Citation Graph

- 4,668,824 documents (documents in the database and those cited)
- 549,726 unique authors
- Modularity matrices (directed for citation, undirected for coauthor) integrated over a 5-year window with a linear ramp filter
  - Top 30 eigenvalues shown
  - Eigendecomposition of 4.6M-vertex graph takes less than 1 hour on a single commercial processor
  - Gradual upward trend with several large deviations in both cases
  - For each graph, one year has a large deviation in many eigenvalues
Emerging Clusters: Citation Graph

- Under the clutter, two subsets of vertices with significant internal connectivity
  - Most documents are in biochemistry and microbiology
  - Largely focus on metabolic properties of acids and proteins
- Temporal integration increases the strength of these subsets in the residuals space
  - Eigenvectors of similar rank correspond to star subgraphs with eigenvalues more than twice as large

Emerging interconnectedness in a major research area emphasized by linear ramp filter
Emerging Clusters: Coauthor Graph

- Again, under the clutter, two tightly-connected subgraphs emerge
  - Similar periodicals in the (newly-founded) American J. Med comprise the other cluster
  - In both cases, teams of medical researchers form gradually over time
- Ramp filter emphasizes the densifying behavior
  - Increases strength to the same order as clutter subgraphs with spectral norms 50%–100% larger

Temporal integration boosts signal power and enables detection of slowly-growing collaboration network while still (relatively) weak
Outline

• Introduction

• Graph Analytics
  – SPG Framework
  – Applications

• High Level Languages
  – Graph Construction Using D4M
  – Application

• Summary
High Level Language: D4M

D4M binds associative arrays to databases, enabling rapid prototyping of data-intensive cloud analytics and visualization.

Distributed Database

D4M
Dynamic
Distributed
Dimensional
Data
Model

Query:
Alice
Bob
Cathy
David
Earl

A D4M query returns a sparse matrix or a graph...

...for statistical signal processing or graph analysis in MATLAB

Associate Arrays
Numerical Computing Environment
Constructing Graph Representations of Raw Data Sources

(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

- 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 algorithms
Graph Construction Using D4M

1. Parse fields from raw data
   - Developed Once Per Data Source

2. Explode schema and store in database
   - D4M

3. Construct associative arrays using D4M queries
   - Developed Once Per Graph

4. Convert associative arrays into adjacency matrices
   - 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 192.168.1.1 157.166.255.18 "-" [12/May/2011:13:24:11] "GET http://www.cnn.com/ HTTP/1.1" 335 256 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 ...

Dense Table

<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>
</tbody>
</table>
Graph Construction Using D4M: Explode Schema

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

**Dense Table**

<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>

**Use as row indices**

<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>

**Create columns for each unique type/value pair**

<table>
<thead>
<tr>
<th>src_ip</th>
<th>128.0.0.1</th>
<th>src_ip</th>
<th>192.168.1.2</th>
<th>server_ip</th>
<th>157.166.255.18</th>
<th>server_ip</th>
<th>208.29.69.138</th>
<th>server_ip</th>
<th>74.125.224.72</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_id</td>
<td>001</td>
<td>002</td>
<td>003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Exploded Table**

GraphEx 2012 - 53
MCS 04/18/12
Graph Construction Using D4M: Storing Exploded Data as Triples

Exploded Table

<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>src_ip</td>
</tr>
<tr>
<td>log_id</td>
<td>002</td>
<td>server_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>server_ip</td>
</tr>
</tbody>
</table>

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
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**

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

**D4M Query #2**

```matlab
data = T(Row(keys), :);
```

```
('log_id|001','server_ip|208.29.69.138',1)
('log_id|001','src_ip|128.0.0.1',1)
('log_id|001','time_stamp|11/May/2011:09:52:53',1)
...
('log_id|002','server_ip|157.166.255.18',1)
('log_id|002','src_ip|192.168.1.2',1)
...
('log_id|003','server_ip|74.125.224.72',1)
('log_id|003','src_ip|128.0.0.1',1)
('log_id|003','time_stamp|13/May/2011:11:05:12',1)
...```
Graph Construction Using D4M: Construct Associative Arrays

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

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

Associative Array Algebra
\[
G = \text{data}(:, 'src\_ip|*') .* \text{data}(:, 'server\_ip|*');
\]

\[
\begin{align*}
('src\_ip|128.0.0.1', 'server\_ip|208.29.69.138',1) \\
('src\_ip|128.0.0.1', 'server\_ip|74.125.224.72',1) \\
('src\_ip|192.168.1.2', 'server\_ip|157.166.255.18',1) \\
\ldots
\end{align*}
\]
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',);
```

D4M Query #2

```plaintext
data = T(Row(keys), :);
```

Associative Array Algebra

```plaintext
G = data(:, 'src_ip|*'). * data(:, 'server_ip|*');
```

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

• Introduction

• Graph Analytics
  – SPG Framework
  – Applications

• High Level Languages
  – Graph Construction Using D4M
    – Application

• Summary
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)
Outline

• Introduction

• Graph Analytics
  – SPG Framework
  – Applications

• High Level Languages
  – Graph Construction Using D4M
  – Application

• Summary
Ongoing Efforts and Next Steps

Graph Analytics

- Applicability
  - Cyber, COIN, ISR, Bioinformatics

- Resiliency
  - Uncertainty in data and observation

High Level Languages

- Scalability
  - Parallel language support

- Programmability
  - Automated performance optimization

Distributed Storage and Indexing

- Portability
  - Bindings to multiple databases

- Elasticity
  - Virtual machine development

High Performance Processing

- Performance
  - Novel instruction set architectures

- Efficiency
  - Specialized circuitry and communication
• Processing data sets of emerging scales requires new computing and analytics techniques

• SPG framework enables development of scalable (novel) graph algorithms

• Approach can identify subtle anomalies in large graph datasets

• D4M framework can keep up with data rates and number of modalities
Acknowledgements

• Nicholas Arcolano
• Michelle Beard
• Nadya Bliss
• Josh Haines
• Jeremy Kepner
• Ben Miller
• Benjamin O’Gwynn
• Patrick Wolfe
• Tamara Yu
Questions
### Big Data Challenges

<table>
<thead>
<tr>
<th>Activity Signature</th>
<th>Data Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Detection</strong></td>
<td><strong>Data Source</strong></td>
</tr>
<tr>
<td>Cued</td>
<td>Single</td>
</tr>
<tr>
<td>Uncued</td>
<td>Many</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal-to-Noise</th>
<th>Dimensionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Increasing Difficulty</td>
</tr>
<tr>
<td>Low</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem Scale</th>
<th>Latency Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entities</strong></td>
<td><strong>Collection Time</strong></td>
</tr>
<tr>
<td>Few</td>
<td>Short</td>
</tr>
<tr>
<td>Many</td>
<td>Long</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relationships</th>
<th>Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>Increasing Difficulty</td>
</tr>
<tr>
<td>Global</td>
<td></td>
</tr>
</tbody>
</table>

**The Big Data Problem**