E6893 Big Data Analytics Lecture 9:

Linked Big Data — Graphical Models (II)

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science

November 2nd, 2016
Outline

• Overview and Background
• Bayesian Network and Inference
• Node level parallelism and computation kernels
• Architecture-aware structural parallelism and scheduling
• Conclusion

Thanks to Dr. Yinglong Xia’s contributions!!
Big Data Analytics and High Performance Computing

- **Big Data Analytics**
  - Widespread impact
  - Rapidly increasing volume of data
  - Real time requirement
  - Growing demanding for High Performance Computing algorithms

- **High Performance Computing**
  - Parallel computing capability at various scales
  - Multicore is the de facto standard
  - From multi-/many-core to clusters
  - Accelerate Big Data analytics

- Twitter has 340 million tweets and 1.6 billion queries a day from 140 million users
- BlueGene/Q Sequoia has 1.6 million cores offering 20 PFlops, ranked #1 in TOP500
Example Graph Sizes and Graph Capacity

Scale is the number of nodes in a graph, assuming the node degree is 25, the average degree in Facebook.

Throughput is the tera-number of computations taking place in a second.

- **Scale**: 1G, 10G, 100G, 1T, 1P
- **Throughput**: 1E, 100P, 10P, 1E
Challenges – Graph Analytic Optimization based on Hardware Platforms

- Development productivity v.s. System performance
  - High level developers → limited knowledge on system architectures
  - Not consider platform architectures → difficult to achieve high performance
- Performance optimization requires tremendous knowledge on system architectures
- Optimization strategies vary significantly from one graph workload type to another

Multicore → Mutex locks for synchronization, possible false sharing in cache data, collaboration between threads

Manycore → high computation capability, but also possibly high coordination overhead

Heterogeneous capacity of cores of systems where tasks affiliation is different

High latency in communication among cluster compute nodes
Understanding computation and comm/synchronisation overheads help improve system performance

- The ratio is HIGHLY dependent on input graph topology, graph property
  - Different algorithms lead to different relationship
  - Even for the same algorithm the input can impact the ratio

Parallel BFS in BSP model

Comp/Sync. in thread

Different input leads to different ratios

BFS with spinlock barriers

4-Grid 8-Grid 16-Grid

Good_1 Good_2 Rand
Graph Workload

- Traditional (non-graph) computations
  - Example → scientific computations
  - Characteristics
- Graph analytic computations
  - Case Study: Probabilistic inference in graphical models is a representative
    - (a) Vertices + edges → Graph structure
    - (b) Parameters (CPTs) on each node → Graph property
    - (c) Changes of graph structure → Graph dynamics

Graphical Model $\mathcal{B}(\mathcal{G}(\mathcal{V}, \mathcal{E}), \mathcal{P})$
- Graph + Parameters (Properties)
- Node → Random variables
- Edge → Conditional dependency
- Parameter → Conditional distribution

Example: Matrix $A \times B$
- Regular memory access
- Relatively good data locality
- Computational intensive

Inference for anomalous behaviors in a social network

Conditional distribution table (CPT)
Graph Workload Types

- **Type 1: Computations on graph structures / topologies**
  - Example → converting Bayesian network into junction tree, graph traversal (BFS/DFS), etc.
  - Characteristics → Poor locality, irregular memory access, limited numeric operations

- **Type 2: Computations on graphs with rich properties**
  - Example → Belief propagation: diffuse information through a graph using statistical models
  - Characteristics
    - Locality and memory access pattern depend on vertex models
    - Typically a lot of numeric operations
    - Hybrid workload

- **Type 3: Computations on dynamic graphs**
  - Example → streaming graph clustering, incremental k-core, etc.
  - Characteristics
    - Poor locality, irregular memory access
    - Operations to update a model (e.g., cluster, sub-graph)
    - Hybrid workload
Scheduling on Clusters with Distributed Memory

- Necessity for utilizing clusters with distributed memory
  - Increasing capacity of aggregated resources
  - Accelerate computation even though a graph can fit into a shared memory
- Generic distributed solution remains a challenge
  - Optimized partitioning is NP-hard for large graphs, especially for dynamic graphs
- Graph RDMA enables a virtual shared memory platform
  - Merits → remote pointers, one-side operation, near wire-speed remote access

Graph RDMA uses key-value queues with remote pointers

The overhead due to remote commutation among 3 m/c is very low due to RDMA
Explore BlueGene and HMC for Superior Performance

- BlueGene offers super performance for large scale graph-based applications
  - 4-way 18-core PowerPC x 1024 compute nodes → 20 PFLOPS
  - Winner of GRAPH500 (Sequoia, LLNL)
  - Exploit RDMA, FIFO and PAMI to achieve high performance

Inference in BSP model

- Hybrid Memory Cube for Deep Computing
  - Parallelism at 3 levels
  - Data-centric computation
  - Innovative computation model
Outline

- Overview and Background
- Bayesian Network and Inference
- Node level parallelism and computation kernels
- Architecture-aware structural parallelism and scheduling
- Conclusion
Many critical Big Data problems are based on inter-related entities.

Typical analysis for Big Data involves reasoning and prediction under uncertainties.

Graph-based representation of conditional independence.

Key computations include structure learning and probabilistic inference.

\[ B(G(V, E), P) \]
Graphical Models for Big Data Analytics

Challenges
- Problem $\Rightarrow$ graph
- Compact representation
- Parallelism
- Target platforms

Applications
- Atmospheric troubleshooting (PNNL)
- Disease diagnosis (QMR-DT)
- Anomaly Detection (DARPA)
- Customer Behavior Prediction ……

Traditional ML or AI graphical models do not scale to thousands of inference nodes

Even for this 15-node BN of 5 states for each node, the conditional probability distribution table (PDT) for the degree of interest consists of 9,765,625 entries
Systematic Solution for Parallel Inference

Big Data Problems

C1 & C2 – Modeling with reduced computation pressure (PDCS’09, UAI’12)
C2 - Parallel conversion of Bayesian Network to Junction tree (ParCo’07)

Inference

Evidence

Junction tree

Evidence Propagation

C3 - Identify primitives in probabilistic inference (PDCS’09 Best Paper…)

Output

Posterior probability of query variables

Challenges

C1. Problem ⇨ graph
C2. Effective representation
C3. Parallelism
C4. Target platforms ……

Analytics

Graphical Model

HPC

Parallel Computing Platforms

C3&C4 - Efficiently map algorithms to HPC architectures (IPDPS’08,PDCS’10 Best Paper,……)
Target Platforms

- Homogeneous multicore processors
  - Intel Xeon E5335 (Clovertown)
  - AMD Opteron 2347 (Barcelona)
  - Netezza (FPGA, multicore)
- Homogeneous manycore processors
  - Sun UltraSPARC T2 (Niagara 2), GPGPU
- Heterogeneous multicore processors
  - Cell Broadband Engine
- Clusters
  - HPCC, DataStar, BlueGene, etc.
Target Platforms (Cont') - What can we learn from the diagrams?

- What are the differences between these architectures?
- What are the potential impacts to user programming models?
Bayesian Network

- Directed acyclic graph (DAG) + Parameters
  - Random variables, dependence and conditional probability tables (CPTs)
  - Compactly model a joint distribution

\[
\begin{array}{c|cc}
C & p(S=0|C) & p(S=1|C) \\
0 & 0.5 & 0.5 \\
1 & 0.9 & 0.1 \\
\end{array}
\]

\[
\begin{array}{c|cc}
C & p(R=0|C) & p(R=1|C) \\
0 & 0.8 & 0.2 \\
1 & 0.2 & 0.8 \\
\end{array}
\]

\[
\begin{array}{c|cc}
S & R & p(W=0|S,R) & p(W=1|S,R) \\
0 & 0 & 1.0 & 0.0 \\
1 & 0 & 0.1 & 0.9 \\
0 & 1 & 0.1 & 0.9 \\
1 & 1 & 0.01 & 0.99 \\
\end{array}
\]

- The joint probability of all the nodes:
  \[
p(C, S, R, W) = p(C) \times p(S|C) \times p(R|C,S) \times p(W|C,S,R)
\]
  \[
p(C, S, R, W) = p(C) \times p(S|C) \times p(R|C) \times p(W|S,R)
\]

Compact representation!
Question: How to Update the Bayesian Network Structure?

- How to update Bayesian network structure if the sprinkler has a rain sensor? Note that the rain sensor can control the sprinkler according to the weather.

\[
p(C, S, R, W) = p(C) \times p(S \mid C) \times p(R \mid C) \times p(W \mid S, R)
\]
Updated Bayesian Network Structure

- Sprinkler is now independent of Cloudy, but becomes directly dependent on Rain
- The structure update results in the new compact representation of the joint distribution

\[
P(C, S, R, W) = P(C)P(R|C)P(S|R, C)P(W|R, C, S) = P(C)P(R|C)P(S|R)P(W|R, S)
\]
Inference in a Simple Bayesian Network

- What is the probability that the grass got wet (W=1), if we know someone used the sprinkler (S=1) and the it did not rain (R=0)?
  - Compute or table-lookup?
- What is the probability that the grass got wet, if there is no rain?
  - Is there a single answer? $P(W=1|R=0)$
  - Either 0.0 or 0.9?
- What is the probability that the grass got wet? $P(W=1)$
  - Think about the meaning of the net!
- What is the probability that the sprinkler is used, given the observation of wet grass? $P(S=1|W=1)$
  - Think about the model name!

\[
\begin{array}{c|cc|c|cc|c|cc|c}
C & p(S=0|C) & p(S=1|C) & \hline
0 & 0.5 & 0.5 & \hline
1 & 0.9 & 0.1 & \hline
\end{array}
\]

\[
\begin{array}{c|cc|c|cc|c|cc|c}
C & p(R=0|C) & p(R=1|C) & \hline
0 & 0.8 & 0.2 & \hline
1 & 0.2 & 0.8 & \hline
\end{array}
\]

\[
\begin{array}{c|cc|c|cc|c|cc|c}
S & R & p(W=0|S, R) & p(W=1|S, R) & \hline
0 & 0 & 1.0 & 0.0 & \hline
1 & 0 & 0.1 & 0.9 & \hline
0 & 1 & 0.1 & 0.9 & \hline
1 & 1 & 0.01 & 0.99 & \hline
\end{array}
\]
What is the probability that the sprinkler is used, given the observation of wet grass?

\[ p(S=1|W=1) \]

The joint distribution:

\[ p(C, S, R, W) = p(C) \times p(S|C) \times p(R|C) \times p(W|S,R) \]

How much time does it take for the computation?

What if the variables are not binary?
Inference on Large Scale Graphs Example - Atmospheric Radiation Measurement (PNNL)

• Atmosphere combines measurements of different atmospheric parameters from multiple instruments that can identify unphysical or problematic data
• ARM instruments run unattended 24 hours, 7 days a week
• Each site has > 20 instruments and is measuring hundreds of data variables

Random variables represent climate sensor measurements

States for each variable are defined by examining range of and patterns in historical measurement data

Links identify highly-correlated measurements as detected by computing the Pearson Correlation across pairs of variables

4,550,812 CPT entries
2,846 cliques
3.9 averages states per node
Use Case: Smarter another Planet

**Goal:** Atmospheric Radiation Measurement (ARM) climate research facility provides 24x7 *continuous field observations* of cloud, aerosol and radiative processes. **Graphical models** can automate the validation with improvement efficiency and performance.

**Approach:** BN is built to represent the dependence among sensors and replicated across timesteps. BN parameters are learned from over 15 years of ARM climate data to support distributed climate sensor validation. Inference validates sensors in the connected instruments.

![Dynamic Bayesian Network - Execution Time](image)

**Bayesian Network**
- 3 timesteps
- 63 variables
- 3.9 avg states
- 4.0 avg indegree
- 16,858 CPT entries

**Junction Tree**
- 67 cliques
- 873,064 PT entries in cliques
Story – Espionage Example

- **Personal stress:**
  - Gender identity confusion
  - Family change (termination of a stable relationship)

- **Job stress:**
  - Dissatisfaction with work
  - Job roles and location (sent to Iraq)
  - Long work hours (14/7)

- **Unstable Mental Status:**
  - Fight with colleagues, write complaining emails to colleagues
  - Emotional collapse in workspace (crying, violence against objects)
  - Large number of unhappy Facebook posts (work-related and emotional)

- **Planning:**
  - Online chat with a hacker confiding his first attempt of leaking the information

- **Attack:**
  - Brought music CD to work and downloaded/copied documents onto it with his own account
Behavior Analysis – Large Scale Graphical Models

User behaviors can be observed and analyzed at several levels

Single user BN for detection

Cross-user BN for detection and validation

Demo
Example of Graphical Analytics and Provenance

IBM Anomaly Detection at Multiple Scales (ADAMS) Analytics

Markov Network
Latent Network
Bayesian Network

Email (Anger)

Distribution of Anger for all people for all days (not showing the 32109 zero values)

Top Person/Days -- Anger, group Self

<table>
<thead>
<tr>
<th>UserId</th>
<th>Date</th>
<th>Anger</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A4D96980D1E5E62F0C7F1919110D0</td>
<td>2012-07-18</td>
<td>81</td>
</tr>
<tr>
<td>7B131181E7A0C09F3A8E605E53227B8</td>
<td>2012-07-17</td>
<td>77</td>
</tr>
<tr>
<td>41C50A44B3E2C2C34EDC6FD129DC3F</td>
<td>2012-07-10</td>
<td>50</td>
</tr>
<tr>
<td>369730CC2965FD97E435A43565AF6FF6</td>
<td>2012-07-17</td>
<td>53</td>
</tr>
<tr>
<td>2FF547A39E2D3D05D6ACB12654411</td>
<td>2012-07-27</td>
<td>62</td>
</tr>
<tr>
<td>6826DFED7FEE7436C4FF7C3809E79D</td>
<td>2012-07-10</td>
<td>62</td>
</tr>
<tr>
<td>CBD7C165A41C749983731039E2817</td>
<td>2012-07-24</td>
<td>62</td>
</tr>
<tr>
<td>C00A90D836F136DCC56712936CD4E</td>
<td>2012-07-13</td>
<td>62</td>
</tr>
<tr>
<td>9C2568C8AF5099AA64C53EB8257AB00</td>
<td>2012-07-10</td>
<td>62</td>
</tr>
<tr>
<td>215291F41174E0D5462A43606615A8</td>
<td>2012-07-24</td>
<td>81</td>
</tr>
<tr>
<td>BCE365D13A8362900BF4CCB1D6E36</td>
<td>2012-07-25</td>
<td>50</td>
</tr>
</tbody>
</table>
Independency & Conditional Independence in Bayesian Inference

- “Multiple, competing explanation” scenario

\[ S \perp R \mid W \times \]

- An “explain-away” dependency between \( S \) and \( R \).
- \( R \) is explained away (get lower probability) by knowing \( S \) given \( W \).

Three canonical graphs:
Naïve Inference – Conditional Independence Determination

- A intuitive “reachability” algorithm for naïve inference - CI query
- Bayes Ball Algorithm to test CI query

\[ X_A \perp X_B \mid X_C \]

- Shade all nodes of \( X_C \)
- Place a ball at each of the nodes \( X_A \). Let the balls bounce on the graph according to certain rules that depend on if nodes are shaded, and on arrow directions
- Ask whether any of the balls reach one of the nodes \( X_B \):
  - If none of the balls reach \( X_B \), then we assert \( X_A \perp X_B \mid X_C \).
  - If a ball reaches \( X_B \), then we assert \( X_A \not\perp X_B \mid X_C \times \).
• Conditional dependence among random variables allows information propagated from a node to another \( \Rightarrow \) foundation of probabilistic inference

\[
\text{Given evidence (observations) } E, \text{ output the posterior probabilities of query } P(Q|E)
\]

Bayes’ theorem can not be applied directly to non-singly connected networks, as it would yield erroneous results

Therefore, junction trees are used to implement exact inference
Probabilistic Inference in Junction Tree

- Inference in junction trees relies on the probabilistic representation of the distributions of random variables.
- Probabilistic representation: Tabular v.s. Parametric models
  - Use Conditional Probability Tables (CPTs) to form Potential Tables (POTs).
  - Multivariable nodes per clique.
  - Property of running intersection (shared variables).

\[ \Psi(8) = P(8) \]

\[ \Psi(3,2,1) = P(3|1,2)P(1)P(2) \]

<table>
<thead>
<tr>
<th>3,1,2</th>
<th>TT</th>
<th>TF</th>
<th>FT</th>
<th>FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0.075</td>
<td>0.105</td>
<td>0.090</td>
<td>0.245</td>
</tr>
<tr>
<td>F</td>
<td>0.075</td>
<td>0.245</td>
<td>0.060</td>
<td>0.105</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3,1,2</th>
<th>TT</th>
<th>TF</th>
<th>FT</th>
<th>FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0.5</td>
<td>0.3</td>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>F</td>
<td>0.5</td>
<td>0.7</td>
<td>0.4</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Convert CPTs in Bayesian network to POTs in Junction tree.
Conversion of Bayesian Network into Junction Tree

- **Moralization** connects all parents of each node
- **Parallel Moralization** connects all parents of each node
- **Parallel Triangulation** chordalizes cycles with more than 3 edges
- **Clique identification** finds cliques using node elimination
  - Node elimination is a step look-ahead algorithms that brings challenges in processing large scale graphs
- **Parallel Junction tree construction** builds a hypergraph of the Bayesian network based on running intersection property
- Define weight of a node
  - The weight of a node $V$ is the number of values of $V$
  - The weight of a cluster is the product of the weights of its constituent nodes.
- Criterion for selecting nodes to add edges (based on Node Elimination)
  - Choose the node that causes the least number of edges to be added
  - Break ties by choosing the node that induces the cluster with the smallest weight

Node Elimination Algorithm
• We can identify the cliques from a triangulated graph as it is being constructed. Our procedure relies on the following two observations:
  ● Each clique in the triangulated graph is an induced cluster
  ● An induced cluster can not be a subset of another induced cluster

Fairly easy? Think about if this process can be parallelized?
Conversion of Bayesian Network into Junction Tree – JT construction

- For each distinct pair of cliques X and Y
  - Create a candidate sepset $X \cap Y$, referd as $S_{XY}$ and insert into set A
- Repeat until n-1 sepsets have been inserted into the JT forest
  - Select a sepset $S_{XY}$ from A, according to the criterion: a) choose the candidate sepset with the largest mass; b) break ties by choosing the sepset with the smallest cost
    - Mass: the number of variables it contains
    - Cost: weight of X + weight of Y
  - Insert the sepset $S_{XY}$ between the cliques X and Y only if X and Y are on different trees
Outline

• Overview and Background
• Bayesian Network and Inference
• Node level parallelism and computation kernels
• Architecture-aware structural parallelism and scheduling
• Conclusion
Scalable Node Level Computation Kernels

- Evidence propagation in junction tree is a set of computations between the POTs of adjacent cliques and separators.
  
- For each clique, evidence propagation is to use the POT of the updated separator to update the POT of the clique and the separators between the clique and its children.

- Primitives for inference:
  - Marginalize
  - Divide
  - Extend
  - Multiply
  - and Marginalize again

Contributions:
* Parallel node level primitives
* Computation kernels
* POT organization
Update Potential Tables (POTs)

- **Efficient** POT representation stores the probabilities only
  - Saves memory
  - Simplifies exact inference
- State strings can be converted from the POT indices

\[
s_i = \left\lceil \frac{t - 1}{\prod_{j=1}^{w} r_j} \right\rceil \mod r_i
\]

\[
t = 1 + \sum_{i=1}^{w} s_i \prod_{j=1}^{r_i} \]

- \( r \): number of states of variables
- \( w \): clique width

Dynamically convert POT indices into state strings

Each of the following arrays has \( r^w \) entries

<table>
<thead>
<tr>
<th>POT Index</th>
<th>State string</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \cdots \cdots )</td>
<td>( (a=0, b=0, c=0, d=0, e=1, f=1) )</td>
</tr>
<tr>
<td>3 = ( (000011)_2 )</td>
<td>( (a=0, b=0, c=0, d=1, e=0, f=0) )</td>
</tr>
<tr>
<td>( \cdots \cdots )</td>
<td>( \cdots \cdots )</td>
</tr>
</tbody>
</table>

\( \Psi_c \)

Only this array is stored with clique \( C \)
Update POTs (Marginalization)

- **Example**: Marginalization
  - Marginalize a clique POT $\Psi(C)$ to a separator POT $\Psi(C')$
  - $C' = \{b, c\}$ and $C = \{a, b, c, d, e\}$
Evaluation of Data Parallelism in Exact Inference

- Exact inference with parallel node level primitives
- A set of junction trees of cliques with various parameters
  - \( w \): Clique width, \( r \): Number of states of random variables
- Suitable for junction trees with large POTs

Inference on DataStar

![Graph showing performance comparison for different values of \( w \) and \( r \)]

Inference on HPCC

![Graph showing performance comparison for different values of \( w \) and \( r \)]

Xia et. al. IEEE Trans. Comp ’10
Outline

• Overview and Background
• Bayesian Network and Inference
• Node level parallelism and computation kernels
• Architecture-aware structural parallelism and scheduling
• Application on anomaly detection
• Conclusion
Structural Parallelism in Inference

- Evidence collection and distribution
- Computation at cliques: node level primitives
From Inference into Task Scheduling

- Computation of a node level primitive
- Evidence collection and distribution \(\Rightarrow\) task dependency graph (DAG)
- Construct DAG based on junction tree and the precedence of primitives
Task Graph Representation

- Global task list (GL)
  - Array of tasks
  - Store tasks from DAG
  - Keep dependency information
- Task data
  - Weight (estimated task execution time)
  - Dependency degree
  - Link to successors
  - Application specific metadata

![A portion of DAG](image)

![Task elements](image)

GL

For task scheduling

For task execution

- Dependence Degree
- Successors
- Task ID
- Application Specific Parameters
Handle Scheduling Overhead – Self-scheduling on Multicore

- Collaborative scheduling - task sharing based approach
  - Objective: Load balance, Minimized overhead

![Diagram showing task scheduling and dependencies](image)

- Local ready lists (LLs) are shared to support load balance
- Coordinator overhead

Xia et. al. J.Supercomputing ‘11
Inference on Manycore – An Adaptive Scheduling Approach

- Centralized or distributed method for exact inference?
  - Centralized method
    - A single manager can be too busy
    - Workers may starve
  - Distributed method
    - Employing many managers limits resources
    - High synchronization overhead

Supermanager adjusts the size of thread group dynamically to adapt to the input task graph.
Probabilistic Inference on CPU-GPGPU Platform

(Xia, et. al. ICPP’10, …)

- CPU-GPGPU heterogeneous platform
  - A global task list in Host Mem
  - Two ready task lists
  - Work sharing between CPU cores and GPU threads
  - Work sharing among CPU cores, and among GPU threads
  - Work stealing is not good for slow read from \( Q_g \)

![Diagram showing CPU-GPGPU architecture with task lists and workers]

- workers
- manager
- ready task list for worker cores
- ready task list for GPU

(Xia, et. al. ICPP’10, …)
Performance Comparison with Multicore CPUs

- Exact inference in QMR-DT network, a real-life application used for medical and clinical research
- Comparison between SYNC and ASYNC communication
- Comparison with other platforms

Y. Xia et al, Parallel Exact Inference on a CPU-GPGPU Heterogeneous System, International Conference on Parallel Processing (ICPP), 2010.
Outline

• Overview and Background

• Bayesian Network and Inference

• Node level parallelism and computation kernels

• Architecture-aware structural parallelism and scheduling

• Conclusion
Conclusions

- Parallelization of large scale networks should be solved at various levels.
- Heterogeneity in parallelism exists between different parallelism levels.
- Fully consider architectural constraints in graphical model algorithm design.
- Algorithm must be re-designed to achieve high performance.
- Some parameters need to be tuned to improve performance.

More opportunities for us to explore:
- A scheduler working at cluster level for coordination.
- Scheduling policies on heterogeneous platforms, such as CPU+GPGPU.
- Machine learning techniques to find affinity between a segment of graphical model code and a type of architectures (compute nodes).
- From exact inference to approximate inference on large scale networks.
- (Can you add more bullets here?)
IBM System G Bayesian Network Tool — Short Intro
BN Framework

• Graph Configuration
• Rule Configuration
• Feature Configuration
• BN Tool Command
Graph Configuration (gShell interface)

- Run "./gShell interactive < g_shell.txt"
- Within "g_shell.txt" script:
  - create --graph Esp1 --type directed
  - add_vertex --graph Esp1 --id 0 --label Attack --prop obs:0
  - add_vertex --graph Esp1 --id 1 --label Communication --prop obs:0
  - add_vertex --graph Esp1 --id 2 --label GatherInfo --prop obs:0
  - add_vertex --graph Esp1 --id 3 --label Eml --prop obs:1 col:x1 thresh:[0.85]
  - add_vertex --graph Esp1 --id 4 --label LOF --prop obs:1 col:x2 thresh:[0.005]
  - add_vertex --graph Esp1 --id 5 --label HTTP --prop obs:1 col:x3 thresh:[0.3]
  - add_vertex --graph Esp1 --id 6 --label queryTerm --prop obs:1 col:x4 thresh:[0.3]
Graph Configuration (gShell interface) Cont.

• Within “g_shell.txt” script:
  • add_edge --graph Esp1 --src 1 --targ 0
  • add_edge --graph Esp1 --src 2 --targ 0
  • add_edge --graph Esp1 --src 5 --targ 2
  • add_edge --graph Esp1 --src 6 --targ 2
  • add_edge --graph Esp1 --src 3 --targ 1
  • add_edge --graph Esp1 --src 4 --targ 1
Graph Configuration (gShell interface)

Cont.

- Run `print_all --graph Esp1 --redirect ~/BN_pipeline/testdir/input.json`

- Within output json format file:
  ```json
  {"edges":[{"source":"1","target":"0","eid":"0","label":"\_"},
  {"source":"2","target":"0","eid":"1","label":"\_"},
  {"source":"3","target":"1","eid":"4","label":"\_"},
  {"source":"4","target":"1","eid":"5","label":"\_"},
  {"source":"5","target":"2","eid":"2","label":"\_"},
  {"source":"6","target":"2","eid":"3","label":"\_"}],
  "nodes":[{"id":"0","label":"Attack","obs":"0"},
  {"id":"1","label":"Communication","obs":"0"},
  {"id":"2","label":"GatherInfo","obs":"0"},
  {"id":"3","label":"Eml","obs":"1","col":"x1","thresh":"[0.85]"},
  {"id":"4","label":"LOF","obs":"1","col":"x2","thresh":"[0.005]"},
  {"id":"5","label":"HTTP","obs":"1","col":"x3","thresh":"[0.3]"},
  {"id":"6","label":"queryTerm","obs":"1","col":"x4","thresh":"[0.3]"}],
  "properties":[{"type":"node","name":"thresh","value":"string"},
  {"type":"node","name":"col","value":"string"},
  {"type":"node","name":"obs","value":"string"}],
  "summary":[{"number of nodes":7,"number of edges":6}],
  "time":[{"TIME":"9.39369e-05"}]
  ```
Rule Configuration (runRules())

- `<BNRules>`
  - `<category name="spatial" rulenum="2">
    - `<ruleItem index="1" relation="and" target="1">
      - `<rule val="1">3</rule>
      - `<rule val="0">4</rule>
    </ruleItem>
    - `<ruleItem index="2" relation="and" target="2">
      - `<rule val="0">5</rule>
      - `<rule val="0">6</rule>
    </ruleItem>
  </category>
  - `<category name="temporal" rulenum="1">
    - `<ruleItem index="1" target="5">
      - `<rule>4</rule>
    </ruleItem>
  </category>
- `<BNRules>`

Rules are written in “~/BN_pipeline/testdir/BNRules.xml”

The codes make sure all the unobservable node has according rules while all the rules are applied within junction tree cliques. Currently, temporal rules are not considered. Function “runRules” for parsing.
Feature Configuration
(runFeature())

This is a feature configuration sample “~/BN_pipeline/testdir/BN_feature.json”, when Evidence feature appeared at 07-03-2015, and history features which is used to generate for POT were collected at 07-02-2015 and 07-02-2015. The feature dimension is one. Unobserved one equals “-1”. QueryClq and QueryIdx is the query node we are interested in.
BN Tool Command

• In order to run the code
  • Run “chmod 777 ~/BN_pipeline/bin/moralize”
  • Run “chmod 777 ~/BN_pipeline/bin/inferenceEngine”
  • Under folder “~/BN_pipeline/testdir/”, run “..~/bin/pipeline.sh”, you could check whether the output results are the same as the files under folder “~/BN_pipeline/testdir/expected_results/”
  • The inference results will be in files “output_inf_std.txt” and “output_inf_visual.txt”.
Demo by Eric Johnson