ramBLE: A Parallel Framework for Constraint-Based Bayesian Network Learning via Markov Blanket Discovery

Ankit Srivastava
Motivation

- Machine Learning (ML) models are being used for decision-making in a diverse set of fields – spam detection, recommender systems, etc.
  - “Black box” models are typically used for the purpose

Image Source: XKCD – Machine Learning
https://xkcd.com/1838/
Motivation

• Machine Learning (ML) models are being used for decision-making in a diverse set of fields – spam detection, recommender systems, etc.
  • “Black box” models are typically used for the purpose – NOT interpretable

Image Source: Interpretable Machine Learning — Fairness, Accountability, and Transparency in ML systems
Motivation

• Machine Learning (ML) models are being used for decision-making in a diverse set of fields – spam detection, recommender systems, etc.
  • “Black box” models are typically used for the purpose – NOT interpretable

• Increasingly, ML is being used in high human-impact areas, e.g., criminal justice, healthcare, law enforcement, etc.
  • Apprehensions regarding use of black box models in these areas is growing
Motivation

• Machine Learning (ML) models are being used for decision-making in a diverse set of fields – spam detection, recommender systems, etc.
  • “Black box” models are typically used for the purpose – NOT interpretable

• Increasingly, ML is being used in high human-impact areas, e.g., criminal justice, healthcare, law enforcement, etc.
  • Interpretable ML models are the need of the hour
Motivation

- **Bayesian networks (BNs)** enable probabilistic reasoning about links between the variables of interest – interpretable decisions
  - Used for medical diagnosis, legal reasoning, epidemiology, etc.
- Learning structure of BNs is compute-intensive – needs parallelism

- Existing libraries for learning BNs support limited or no parallelism
  - e.g., bnlearn, pcalg, Tetrad
- Parallelization strategies have been proposed for various BN learning algorithms – difficult to integrate disparate strategies
  - Parallel library with support for multiple algorithms is desirable
Background – Bayesian Networks

• BN structure represents dependence graph of a set of variables
  • Case study – Stock prices of companies related to cloud computing

• Parents and Children (PC) set of a variable consists of the variables that are dependent on it, given any conditioning set
  • e.g., $PC(\text{GOOGL}) = \{\text{NVDA, SPOT, NFLX}\}$

• Markov blanket (MB) of a variable consists of the variables that render the variable independent of other variables
  • Assuming faithfulness
    $MB(X) = PC(X) \cup (Parents(Y) \forall Y \in Children(X))$
    $\Rightarrow MB(\text{GOOGL}) = \{\text{NVDA, SPOT, NFLX, AMZN}\}$
Blanket Learning Algorithms

• **Constraint-based** algorithms learn BN by conducting repeated CI tests using given data set of $m$ observations for the $n$ variables
  • Statistical tests, e.g., $G^2$ test for discrete data

• **Blanket learning** algorithms are *constraint-based* algorithms that first learn MB sets of all the variables separately to get the BN structure
  • *Grow-Shrink (GS)* (Margaritis and Thrun, 2000)
  • *Incremental Association MB (IAMB)* (Tsamardinos et al., 2003)
  • *Interleaved IAMB (Inter-IAMB)* (Tsamardinos et al., 2003)
Blanket Learning Algorithms

- Use variations of the Grow-Shrink scheme for learning MB sets
  - *Grow* phase: Add variables to candidate MB sets
  - *Shrink* phase: Remove false positive variables from candidate MB sets
- Differ in the specifics of how the scheme is iterated
  - Choosing variables to be added in *Grow* phase
    - *IAMB* and *Inter-IAMB* pick the “most dependent” variable given the current candidate MB set
    - *GS* picks the first dependent variable
  - Order of *Grow* and *Shrink* phases
    - *GS* and *IAMB* execute multiple iterations of *Grow* phase followed by one *Shrink* phase
    - *Inter-IAMB* interleaves the execution of *Grow* and *Shrink* phases in every iteration
- Perform *symmetry correction* for MB sets ($X \in MB(T) \Leftrightarrow T \in MB(X))$
- Learn *PC from MB* sets (PC $\subseteq$ MB)
Related Works

• Nikolova et al. (2011) parallelized two similar *constraint-based* algorithms: *MMPC* (Tsamardinos et al., 2006) and *GetPC* (Peña et al., 2007)
  • Scales well up to 512 cores for learning neighborhoods of 1,000 variables
  • Scaling tapers off as the number of cores or variables are increased

• *bnlearn* contains implementations of the three algorithms
  • Scutari et al. (2017) added support for parallelizing the implementations using a master-worker paradigm for small-scale parallelism

• Both these approaches distribute learning of variable neighborhoods
Parallel Framework – Key Design Ideas

• Distribute learning across processors – how?
  • Previous approaches have distributed learning neighborhoods of variables

\[
p_1 = O\left(\frac{n}{p}\right)
\]

\[
p_2 = O\left(\frac{n}{p}\right)
\]

Variables

Candidates = \(O(n)\)

Work per processor \(\propto O\left(\frac{n^2}{p}\right)\)
Parallel Framework – Key Design Ideas

• Distribute learning across processors – how?
  • Previous approaches have distributed learning neighborhoods of variables

\[ \begin{array}{c}
  p_1 \\
  \begin{array}{c}
    \text{Selected} \\
    \text{Candidate Variables}
  \end{array}
\end{array} \quad \begin{array}{c}
  \text{Iteration 1}
  \begin{array}{c}
    \text{Iteration 1}
  \end{array}
\end{array} \quad \begin{array}{c}
  p_2 \\
  \begin{array}{c}
    \text{Selected} \\
    \text{Candidate Variables}
  \end{array}
\end{array} \]
Parallel Framework – Key Design Ideas

• Distribute learning across processors – how?
  • Previous approaches have distributed learning neighborhoods of variables

\[ p_1 \]
\[ p_2 \]

Iteration 1  Iteration 2
Parallel Framework – Key Design Ideas

• Distribute learning across processors – how?
  • Previous approaches have distributed learning neighborhoods of variables

Load Imbalance
Unclear how to fix
Parallel Framework – Key Design Ideas

• Distribute learning across processors – how?
  • Previous approaches have distributed learning neighborhoods of variables

Iteration 1  Iteration 2  Iteration 3  Iteration 4

Idle Processor
Coarse-grained work distribution
Parallel Framework – Key Design Ideas

• Distribute learning across processors – how?
  • Previous approaches have distributed learning neighborhoods of variables

• **Observation:** Variables have different neighborhood sizes – distributing variables to processors is suboptimal

• **Idea:** Distribute all the target and candidate variable pairs in parallel
Parallel Framework – Key Design Ideas

• **Idea:** Distribute all the target and candidate variable pairs in parallel

\[
\text{Work per processor } \propto \text{ Pairs per processor}
= O\left(\frac{n^2}{p}\right)
\]
Parallel Framework – Key Design Ideas

- **Idea:** Distribute all the target and candidate variable pairs in parallel

*Iteration 1*

Selected Candidate Pairs
Parallel Framework – Key Design Ideas

• *Idea:* Distribute all the target and candidate variable pairs in parallel

\[ p_1 \]

\[ p_2 \]

*Iteration 1*  *Iteration 2*
Parallel Framework – Key Design Ideas

- **Idea**: Distribute all the target and candidate variable pairs in parallel

 Load Imbalance Can be alleviated
Parallel Framework – Key Design Ideas

• **Idea:** Distribute all the target and candidate variable pairs in parallel

```

Iteration 1  Iteration 2  Iteration 3  Iteration 4
p1
p2

Idle Processor
Can be avoided
```
Parallel Framework – Primary Data Structure

• $c$–scores is a list of tuples $< X, Y, \theta_{XY} >$ s.t. $X \in \mathcal{X}, Y \in \mathcal{X} \setminus \{X\}$
  • $\theta_{XY}$ is the score of $Y$ for addition to the MB set of $X$
  • Tuples with the same $X$ are contiguously arranged in the list

• Work distribution in parallel by distributing the tuples
  • $c$–scores is block-distributed across processors – $c$–scores$_j$ on processor $j$
Parallel Framework – Components

• Parallel Grow phase on processor $j$
  • Update $\theta_{XY}$ for all the tuples $\in c\text{-}scores_j$
  • Computation of $\theta_{XY}$ is dependent on the algorithm
  • Add $Y$ to the MB of $X$ corresponding to the best $\theta_{XY}$
  • Can be identified using two segmented parallel prefix operations for all the variables

• Parallel Shrink phase on processor $j$
  • Candidate MBs are available for local target variables – no communication

• Parallel Symmetry Correction using algorithm by Nikolova et al. (2011)

• Parallel PC from MB for local target variables on processor $j$
Parallel Skeleton – Blanket Learning

```plaintext
1 function PARALLEL-SKELETON-INTERIAMB():
   \[ \text{Input: } \mathcal{X}, D, \text{APPLY-HYBRIDISTIC}, \text{REDUCE-HYBRIDISTIC} \]
   \[ \text{Output: } \mathcal{PC}(T) \text{ sets for all } T \in \mathcal{X} \]
2   parallel \( j = \text{processor's rank} \) do
3       Initialize \( c\text{-scores}_j, \text{variables}_j, M_B(\cdot) \) as described in subsection 3.2.1
4       Initialize \text{neighbors} as empty list of tuples
5       repeat
6           \text{GROW-PHASE}(D, c\text{-scores}, \text{variables}, M_B, \text{APPLY-HYBRIDISTIC},
7                \text{REDUCE-HYBRIDISTIC})
8           \text{+ SHRINK-PHASE}(D, \text{variables}, M_B)
9       until no \( M_B(X) \) changes on any of the processors
10      \text{- SHRINK-PHASE}(D, \text{variables}, M_B)
11      \text{SYMMETRY-CORRECTION}(\text{variables, } M_B)
12      Synchronize \( M_B(\cdot) \) across all the processors
13   \text{CONSTRUCT-PC}(D, \text{variables}, M_B, \text{neighbors})
```
Implementation

• Implemented using C++ and MPI (conforms to C++14 and MPI 3.1)
  Available at https://github.com/asrivast28/ramBLE

• Optimizations for fast execution in practice
  • Algorithm specific optimizations – GS work reduction
  • Experimented with different statistic computation strategies for CI tests
  • Dynamic load balancing scheme
Experiments and Results

• Experimental setup
  • 64 nodes of the Hive cluster, 16 MPI processes per node – **1024 processes**
  • RHEL 7.6, gcc v9.2.0, MVAPICH2 v2.3.3

• Used real gene-expression data sets to learn gene networks

<table>
<thead>
<tr>
<th>Name</th>
<th>Organism</th>
<th>Genes (n)</th>
<th>Observations (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>S. cerevisiae</td>
<td>5,716</td>
<td>2,577</td>
</tr>
<tr>
<td>D2</td>
<td>A. thaliana</td>
<td>18,373</td>
<td>5,102</td>
</tr>
<tr>
<td>D3</td>
<td>A. thaliana</td>
<td>18,380</td>
<td>16,838</td>
</tr>
</tbody>
</table>

• Used three simulated data sets (S1, S2, and S3) to show scalability
  • $n = 30,000; m = 10,000$; edge addition probabilities: $5e^{-5}, 1e^{-4},$ and $5e^{-4}$
Experiments and Results

- Sequential comparison with prior state-of-the-art – *bnlearn*
  - Popular library for learning BNs; C implementation interfaces with R

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Data set</th>
<th>Run-time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><em>bnlearn</em></td>
<td><em>Ours</em></td>
</tr>
<tr>
<td>GS</td>
<td>$D_1$</td>
<td>8720.0</td>
<td>240.1</td>
</tr>
<tr>
<td></td>
<td>$D_2$</td>
<td>$\times$</td>
<td>6760.3</td>
</tr>
<tr>
<td></td>
<td>$D_3$</td>
<td>$\times$</td>
<td>18695.0</td>
</tr>
<tr>
<td>IAMB</td>
<td>$D_1$</td>
<td>975.9</td>
<td>624.6</td>
</tr>
<tr>
<td></td>
<td>$D_2$</td>
<td>40605.7</td>
<td>14529.8</td>
</tr>
<tr>
<td></td>
<td>$D_3$</td>
<td>84403.1</td>
<td>46603.2</td>
</tr>
<tr>
<td>Inter-IAMB</td>
<td>$D_1$</td>
<td>992.0</td>
<td>624.1</td>
</tr>
<tr>
<td></td>
<td>$D_2$</td>
<td>40819.0</td>
<td>14559.0</td>
</tr>
<tr>
<td></td>
<td>$D_3$</td>
<td>89839.7</td>
<td>48442.4</td>
</tr>
</tbody>
</table>
Experiments and Results

• Sequential comparison with prior state-of-the-art – *bnlearn*
  • Popular library for learning BNs; C implementation interfaces with R

• BNs learned by our implementations are similar to those by *bnlearn*
  • Recalled 99.84% edges with a precision of 99.92% for $D_1$ data set
  • Changes in the ordering of the variables caused the differences

• Parallelism in *bnlearn* yields diminishing returns beyond a single node
  • e.g., *IAMB* shows a self-speedup of 3.4X on 16 cores for $D_3$ data set
    while the self-speedup using 64 cores on four nodes is 3.9X
Experiments and Results

• Parallel performance of our framework – notions of scalability

• Strong Scaling
  • Fixed total work; how does the run-time scale with increasing parallelism? 
    \((n \text{ is kept constant as } p \text{ increases})\)

• Weak Scaling
  • Fixed work per processor; how does the run-time scale with increasing parallelism? 
    \((n \text{ is increased as } p \text{ increases})\)

• Speedup and efficiency are measured
  • Perfect parallel algorithm shows **linear speedup** and **100% efficiency**
Experiments and Results

- Strong scaling of our framework – IAMB

![Graph showing strong scaling speedup and efficiency for different datasets (D1, D2, D3) across varying numbers of cores (p), with linear speedup indicated.]
Experiments and Results

- Strong scaling of our framework – *Inter-IAMB*

![Graph showing strong scaling speedup and efficiency](image)

**Strong Scaling Speedup**

- **D1**
- **D2**
- **D3**
- **Linear Speedup**

**Strong Scaling Efficiency (%)**

- **D1**
- **D2**
- **D3**

*Number of cores (p)*

*Number of cores (p)*
Experiments and Results

- Strong scaling of our framework – GS
Experiments and Results

- Investigating the scaling performance of GS
  - High communication overhead due to lower total work?

![Graph showing the fraction of total run-time spent in communication for different number of cores. The graph compares GS, IAMB, and Inter-IAMB methods.](image-url)
Experiments and Results

• Scaling performance of GS – real data versus simulated data
Experiments and Results

• Weak scaling of our framework

• Fixed work per processor – how do we vary \( n \) with increasing \( p \)?
  • Choose all the variables when using the largest \( p \), a subset of variables for smaller \( p \)

• Estimated work per processor = \( \frac{n^2}{p} \)
  • Chosen number of variables scale as \( \sqrt{p} \), i.e., \( n_p = n \sqrt{\frac{p}{p_{max}}} \)
  • We chose the first \( n_p \) variables in the data sets for our experiments
Experiments and Results

- Weak scaling of our framework – \textit{D2}

![Weak Scaling Efficiency Graph](image)
Experiments and Results

• Our parallel algorithms learn genome-scale BNs in < 1 minute on 1024 cores, down from more than 13 hours sequentially
  • Maximum speedup of 844.8X and 82.5% scaling efficiency on 1024 cores
  • IAMB and Inter-IAMB show a sustained efficiency of > 75% for $D_2$ and $D_3$

• Learning BNs from simulated data sets takes < 2 minutes on 1024 cores, as compared to more than a day sequentially
  • Maximum speedup of 845X and 82.5% scaling efficiency on 1024 cores
  • GS shows an improved efficiency of > 60% for all the data sets
Thanks! Questions?