USENIX
ATC '24

Fast Inference for Probabilistic Graphical Models

Jiantong Jiang [1], Zeyi Wen [2,3], Atif Mansoor [1], Ajmal Mian [1]

[1] *The University of Western Australia*

[2] *Hong Kong University of Science and Technology (Guangzhou)*

[3] *Hong Kong University of Science and Technology*

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

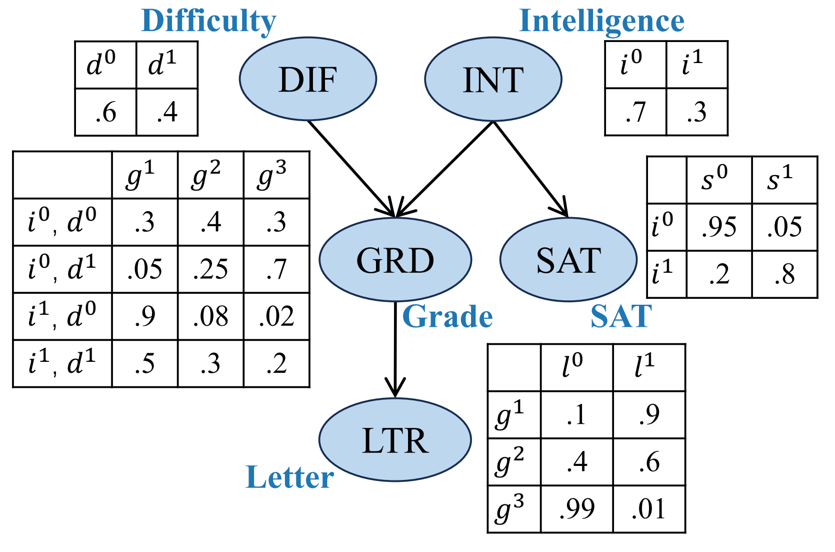
THE UNIVERSITY OF WESTERN AUSTRALIA

Outline

- **Background and Motivation**
- Objective I: Good Generality and Flexibility
- Objective II: High Efficiency
- Experimental Evaluation
- Conclusion

Background of PGMs

- Probabilistic graphical models (PGMs):
  - A **graph**: illustrates random variables and their relationships.
  - **Parameters**: quantify the strength of the relationships: a set of tables for discrete PGMs, e.g. *conditional probability tables (CPTs)* for BNs.
    - Joint distribution can be factorized into local CPTs of each node.



d^0	d^1
.6	.4

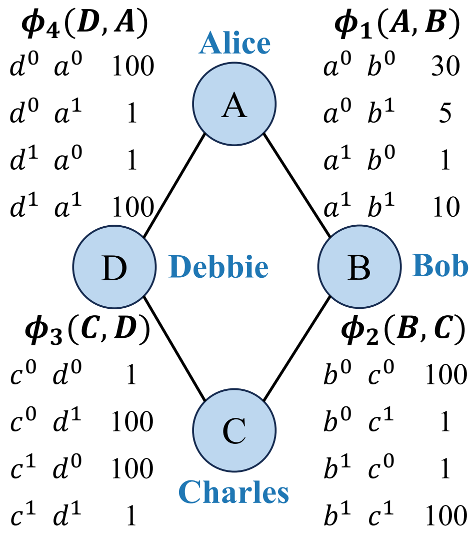
i^0	i^1
.7	.3

	g^1	g^2	g^3
i^0, d^0	.3	.4	.3
i^0, d^1	.05	.25	.7
i^1, d^0	.9	.08	.02
i^1, d^1	.5	.3	.2

	s^0	s^1
i^0	.95	.05
i^1	.2	.8

	l^0	l^1
g^1	.1	.9
g^2	.4	.6
g^3	.99	.01

Bayesian network (BN)



$\boldsymbol{\phi_4(D, A)}$

d^0	a^0	100
d^0	a^1	1
d^1	a^0	1
d^1	a^1	100

$\boldsymbol{\phi_1(A, B)}$

a^0	b^0	30
a^0	b^1	5
a^1	b^0	1
a^1	b^1	10

$\boldsymbol{\phi_3(C, D)}$

c^0	d^0	1
c^0	d^1	100
c^1	d^0	100
c^1	d^1	1

$\boldsymbol{\phi_2(B, C)}$

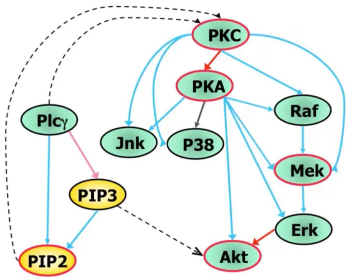
b^0	c^0	100
b^0	c^1	1
b^1	c^0	1
b^1	c^1	100

Markov random field (MRF)

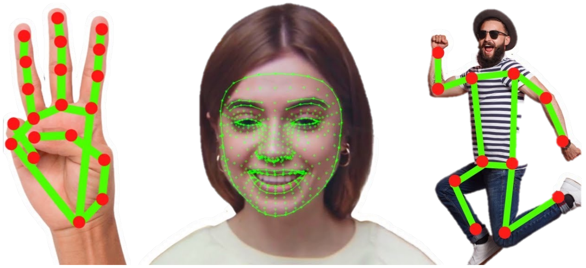
Background of PGMs

- Key advantages:
  - Transparent and intuitive graphical representation.
  - Solid theoretical foundations (probability theory).
- Applications:

Medical diagnosis

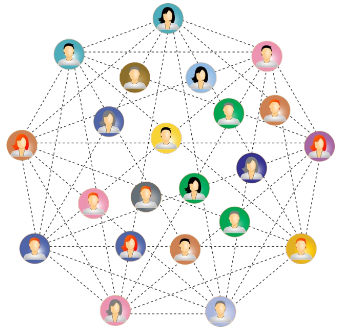


Biological informatics



Computer vision

Financial analysis

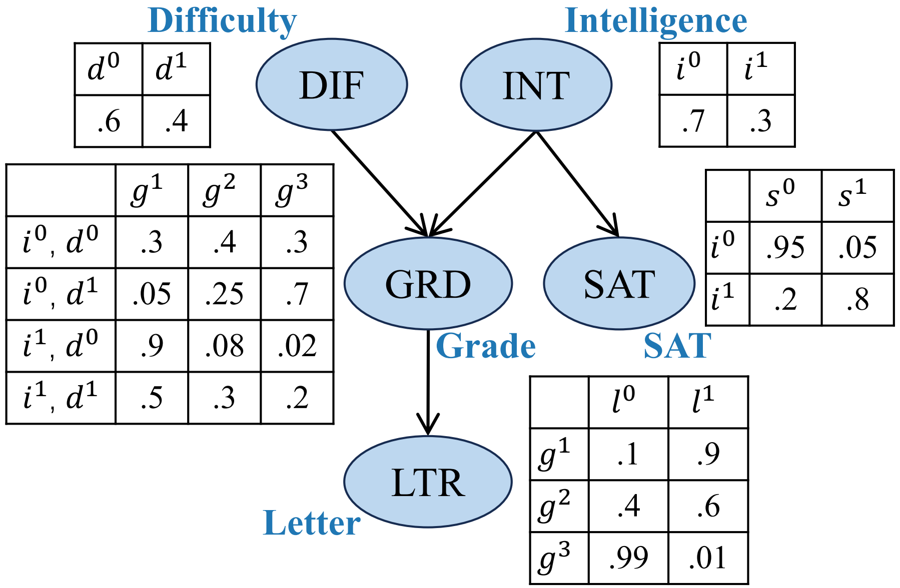


Social network analysis

Background of PGM Inference

- **Inference**: calculate the posterior probability distribution of *query* variables, given values of other *evidence* variables.
  - e.g. evidence: DIF = d^1, INT = i^1; query variable: LTR.

    -> P(LTR | DIF = d^1, INT = i^1).



d^0	d^1
.6	.4

i^0	i^1
.7	.3

	g^1	g^2	g^3
i^0, d^0	.3	.4	.3
i^0, d^1	.05	.25	.7
i^1, d^0	.9	.08	.02
i^1, d^1	.5	.3	.2

	s^0	s^1
i^0	.95	.05
i^1	.2	.8

	l^0	l^1
g^1	.1	.9
g^2	.4	.6
g^3	.99	.01

Background of PGM Inference

- **Approximate inference**: less time but lower accuracy.
  - **Importance sampling-based methods**: use *importance function* to generate samples and estimate probability from the samples.
    - ★ Importance function in PGMs: a probability distribution, can be decomposed into multiple *importance conditional probability tables (ICPTs)* of variables.
  - Widely used sampling-based algorithms:
    - Probabilistic logic sampling (PLS).
    - Likelihood weighting (LW).
    - Self-importance sampling (SIS).
    - AIS-BN.
    - EPIS-BN.

Motivation

- Major challenges:
  - NP-hardness of approximate inference problem.
  - Irregular nature of the graphical structure.
  - Stochastic nature of the sampling process.
  - Difficulty in abstracting and integrating various algorithm.
- We provide ***Fast-PGM***, a system for importance sampling-based PGM inference. Main objectives:
  - **Good generality and flexibility.**
  - **High efficiency.**

Outline

- Background and Motivation
- **Objective I: Good Generality and Flexibility**
- Objective II: High Efficiency
- Experimental Evaluation
- Conclusion

System Abstraction

- All the importance sampling-based algorithms can be abstracted into a general framework with four crucial modules.

Algorithm 3: Fast-PGM

---

input : prior probability $P(\mathcal{V})$, weight v^0, # of samples required q, updating interval l

output : estimated posterior marginal probability for all non-evidence variables

$k \leftarrow 0$, $\mathbf{scrArr} \leftarrow \mathbf{0}$, $\mathbf{w}_{curScr} \leftarrow 0$, $\mathbf{w}_{allScr} \leftarrow 0$.

/* importance function initialization module */

$f^0(\mathcal{V}) \leftarrow \text{initImpFunc}(P(\mathcal{V}))$

for $i \leftarrow 1$ **to** q **do**

    if $i \% l == 0$ **then**

        $k \leftarrow k + 1$

        /* importance function update module */

        $f^k(\mathcal{V}), v^k \leftarrow \text{updImpFunc}(\mathbf{w}_{curScr}, \mathbf{w}_{allScr})$

    /* sample generation module */

    $s_i, w_{iScr} \leftarrow \text{genSamp}(f^k(\mathcal{V}), P(\mathcal{V}))$

    /* importance score accumulation module */

    $\mathbf{w}_{curScr}, \mathbf{w}_{allScr}, \mathbf{scrArr} \leftarrow \text{accScr}(s_i, w_{iScr}, v^k)$

normalize $\mathbf{scrArr}$ for each variable

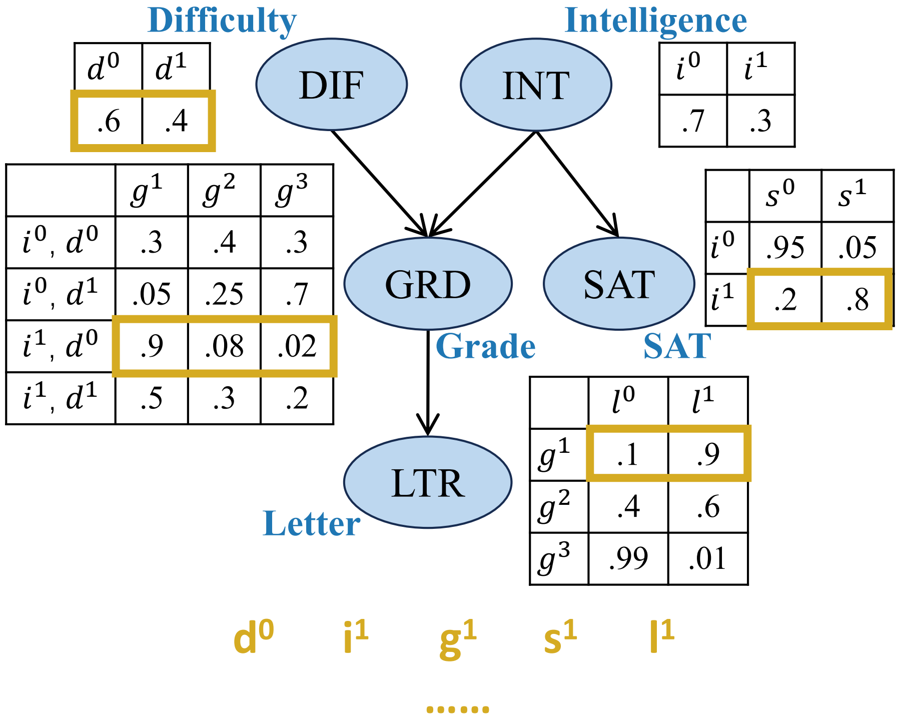
System Abstraction

- Sample generation: generate a sample by sampling each variable (via topological order), and calculate its *importance score*.
  - Distinguish variables
  - Does not distinguish

Steps to handle each variable V_j:

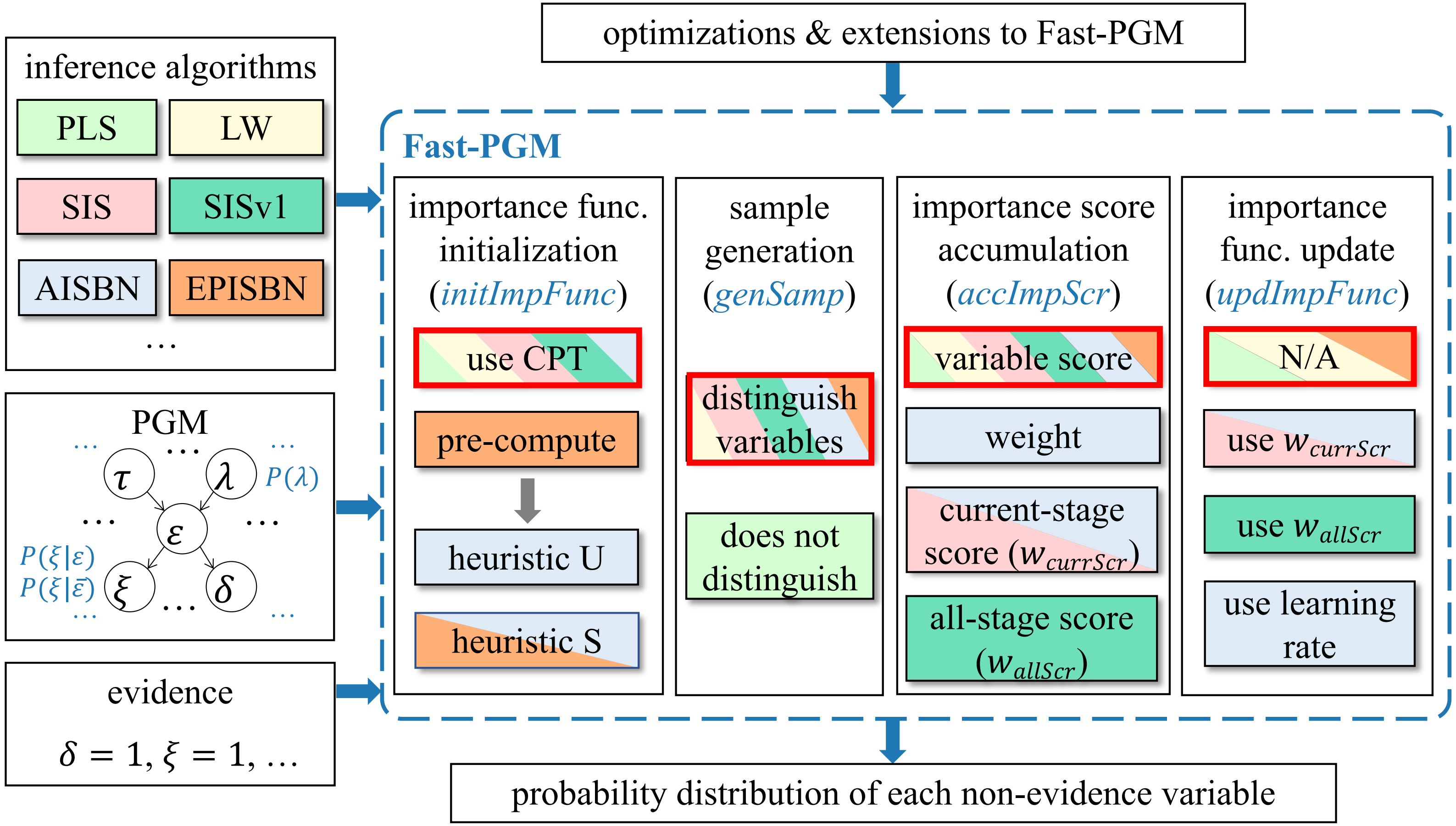
1. Get the instantiation of V_j's parents.
2. Get the weight vector of V_j based on V_j's ICPT and the instantiation of its parents.
3. Randomly pick a value of V_j based on the weight vector.
4. Compute the score and multiply it into the importance score.

Example: INT = i^1.



Fast-PGM Overview

- Fast-PGM enables good generality and flexibility.



Key features:

Versatile modules and functionalities support.

Quick new algorithm implementation.

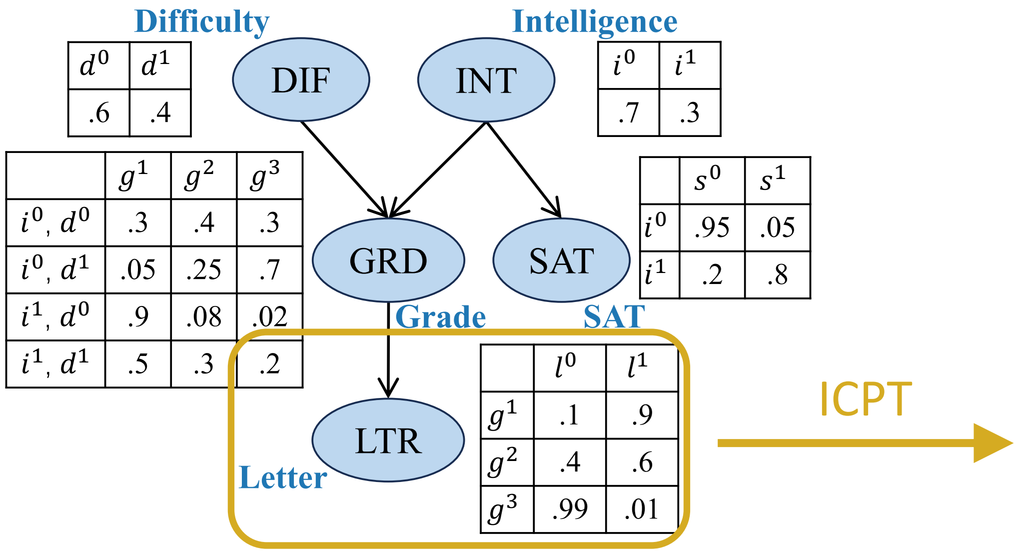
Ease of optimization and extension.

Outline

- Background and Motivation
- Objective I: Good Generality and Flexibility
- **Objective II: High Efficiency**
  - Memory management
  - Computation simplification
  - Parallelization
- Experimental Evaluation
- Conclusion

Memory : Basic Data Structure

- Challenge 1: Large memory requirement for storing ICPTs.
  - The computations are done by maintaining an ICPT of each variable.



d^0	d^1
.6	.4

i^0	i^1
.7	.3

	g^1	g^2	g^3
i^0, d^0	.3	.4	.3
i^0, d^1	.05	.25	.7
i^1, d^0	.9	.08	.02
i^1, d^1	.5	.3	.2

	s^0	s^1
i^0	.95	.05
i^1	.2	.8

	l^0	l^1
g^1	.1	.9
g^2	.4	.6
g^3	.99	.01

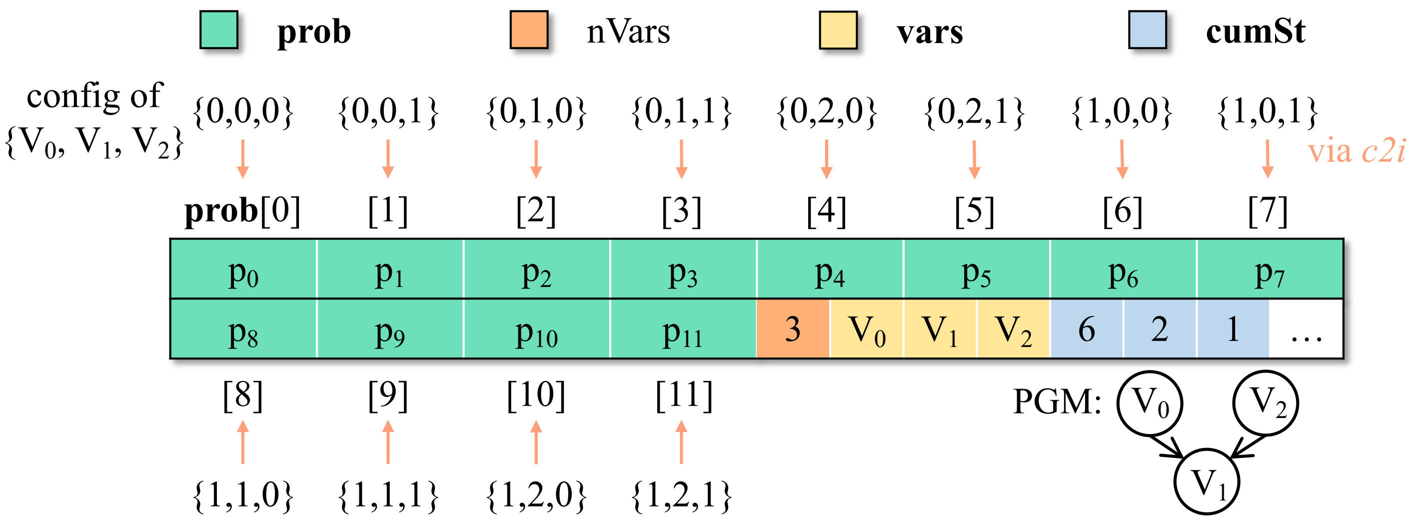
(l^0, g^1)	p_0
(l^0, g^2)	p_1
(l^0, g^3)	p_2
(l^1, g^1)	p_3
(l^1, g^2)	p_4
(l^1, g^3)	p_5

Related variables: *LTR*, *GRD*.
Keys: state configurations.
Values: probabilities.

Number of entries = 2 * 3 = 6.

Memory: Basic Data Structure

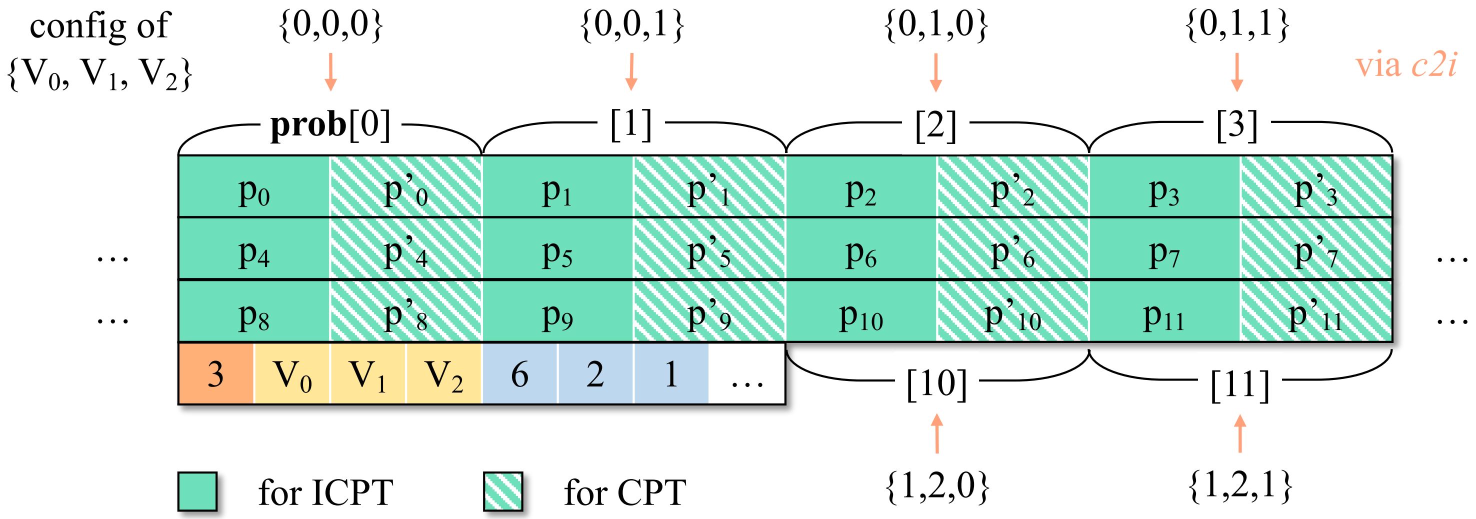
- Solution 1: Avoid storing state configurations for each entry.




```
int ProbabilityTable::c2i(int *config) {
  int idx=0;
  for(int i=0;i<this->nVars;++i)
    idx+=config[i]*this->cumSt[i]; // use cumSt
  return idx;
}
```

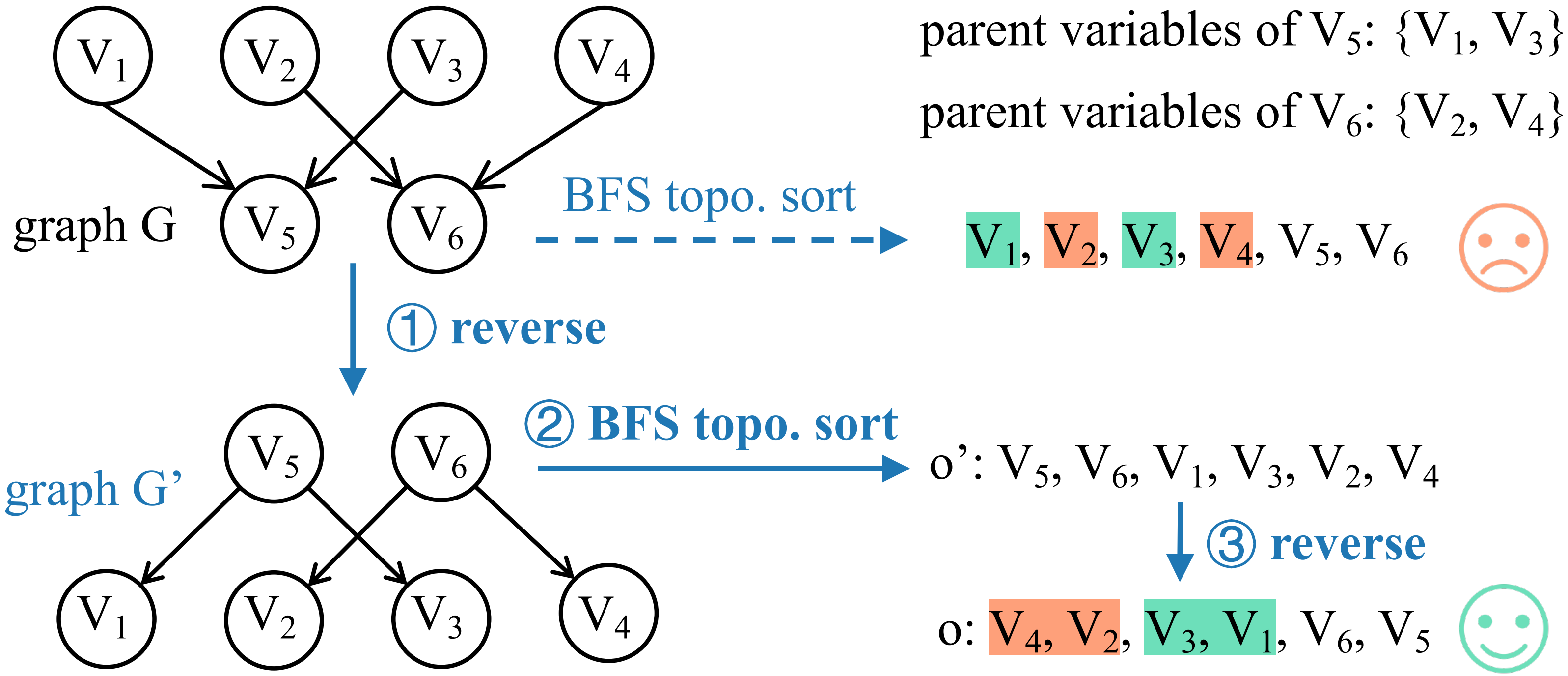
Memory: Data Fusion

- Challenge 2: Irregular memory accesses to ICPTs and CPTs caused by the stochastic nature of the sampling process.
- Solution 2: Fusing ICPT and CPT of each variable to enhance data locality.



Memory: Data Reordering

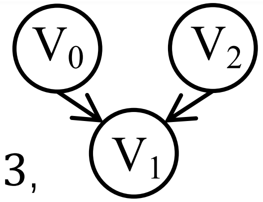
- Challenge 3: Irregular memory accesses caused by the graphical structure.
- Solution 3: Data reordering to maximize proximity of each node's parents.



Computation: Table Reorganization

- Most expensive operation: Get the weight vector of variables.
  - By reducing the ICPT of each variable based on the instantiation of its parents.

PGM:



$r_{V_1} = 3$,
$r_{V_0} = r_{V_2} = 2$.

Parent instantiation:
$V_0 = 1, V_2 = 0$.

$(V_0 = 0, V_1 = 0, V_2 = 0)$	p_0
$(V_0 = 0, V_1 = 0, V_2 = 1)$	p_1
$(V_0 = 0, V_1 = 1, V_2 = 0)$	p_2
$(V_0 = 0, V_1 = 1, V_2 = 1)$	p_3
$(V_0 = 0, V_1 = 2, V_2 = 0)$	p_4
$(V_0 = 0, V_1 = 2, V_2 = 1)$	p_5
$(V_0 = 1, V_1 = 0, V_2 = 0)$	p_6
$(V_0 = 1, V_1 = 0, V_2 = 1)$	p_7
$(V_0 = 1, V_1 = 1, V_2 = 0)$	p_8
$(V_0 = 1, V_1 = 1, V_2 = 1)$	p_9
$(V_0 = 1, V_1 = 2, V_2 = 0)$	p_{10}
$(V_0 = 1, V_1 = 2, V_2 = 1)$	p_{11}

Size of ICPT: $r_{V_j} \times \prod_{V_i \in Par(V_j)} r_{V_i}$
(exponential in $|Par(V_j)| + 1$)!

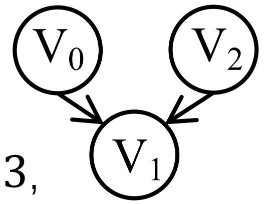
Overall Complexity:
$O(q \times \sum_{V_j \notin E}(|Par(V_j)| \times r_{V_j} \times \prod_{V_i \in Par(V_j)} r_{V_i}))$.

Weight vector:

p_6	p_8	p_{10}

Computation: Table Reorganization

PGM:



$r_{V_1} = 3,$
$r_{V_0} = r_{V_2} = 2.$

- Table reorganization optimization to simplify computations.

prob in data structure

idx	prob		config of $\{V_0,V_1,V_2\}$
0	p_0	p'_0	{0, 0, 0}
1	p_1	p'_1	{0, 0, 1}
2	p_2	p'_2	{0, 1, 0}
3	p_3	p'_3	{0, 1, 1}
4	p_4	p'_4	{0, 2, 0}
5	p_5	p'_5	{0, 2, 1}
6	p_6	p'_6	{1, 0, 0}
7	p_7	p'_7	{1, 0, 1}
8	p_8	p'_8	{1, 1, 0}
9	p_9	p'_9	{1, 1, 1}
10	p_{10}	p'_{10}	{1, 2, 0}
11	p_{11}	p'_{11}	{1, 2, 1}

operations

conConfig: $\{V_0=1, V_2=0\}$

↓ ①

config:

{1, 0, 0}
{1, 1, 0}
{1, 2, 0}

↓ ②

idx:

6	8	10

↓ ③

p_6	p'_6
p_8	p'_8
p_{10}	p'_{10}

$$O\Big(q\times \sum_{V_j\notin E}\big(r_{V_j}\times(|Par(V_j)|+1)\big)\Big)$$

Variable itself is in the rightmost.

prob in data structure

idx	prob		config of $\{V_0,V_2,V_1\}$
0	p_0	p'_0	{0, 0, 0}
1	p_2	p'_2	{0, 0, 1}
2	p_4	p'_4	{0, 0, 2}
3	p_1	p'_1	{0, 1, 0}
4	p_3	p'_3	{0, 1, 1}
5	p_5	p'_5	{0, 1, 2}
6	p_6	p'_6	{1, 0, 0}
7	p_8	p'_8	{1, 0, 1}
8	p_{10}	p'_{10}	{1, 0, 2}
9	p_7	p'_7	{1, 1, 0}
10	p_9	p'_9	{1, 1, 1}
11	p_{11}	p'_{11}	{1, 1, 2}

operations

conConfig: $\{V_0=1, V_2=0\}$

↓ ①

config1:

{1, 0, 0}

↓ ②

idx1:

6

↓ ③

p_6	p'_6
p_8	p'_8
p_{10}	p'_{10}

$$O\Big(q\times \sum_{V_j\notin E}\big(|Par(V_j)|+1\big)\Big)$$

Parallelization

- Case-level parallelism (coarse-grained parallelism):
  - Load unbalancing due to different evidence variables.
- Variable-level parallelism (fine-grained parallelism):
  - Requirement of graph partitioning that relies on PGM structures.
  - Small workloads but high parallel overhead.

- **Sample-level parallelism**:
  - Parallelize the generation of samples within each importance updating stage.

Outline

- Background and Motivation
- Objective I: Good Generality and Flexibility
- Objective II: High Efficiency
- **Experimental Evaluation**
- Conclusion

USENIX ATC '24

Experimental Evaluation

THE UNIVERSITY OF WESTERN AUSTRALIA

- Overall comparison with existing work:

Table 2: Execution time comparison of Fast-PGM ("Ours") with SMILE ("S") and BNJ ("B") on 1000 test cases. Speedups of Fast-PGM over SMILE and BNJ are also reported. "N/A" means that the library does not support the corresponding algorithm.

PGM	PLS					LW					SIS				
	Time (sec)			Speedup		Time (sec)			Speedup		Time (sec)			Speedup	
	S	B	Ours	S	B	S	B	Ours	S	BNJ	S	B	Ours	S	B
Alarm	0.39	1.59	0.13	3.0	12.0	0.42	1.50	0.11	3.7	13.2	N/A	3.0	0.14	N/A	20.7
Hailfinder	0.76	5.7	0.26	3.0	22.1	0.66	2.7	0.21	3.1	13.1	N/A	15.8	0.26	N/A	60.0
Pathfinder	5.0	76.0	1.1	4.4	66.8	6.1	88.0	1.0	5.8	84.2	N/A	1.1k	1.3	N/A	850
Pigs	21.4	1.2k	1.4	15.0	841	14.6	1.2k	1.5	10.1	854	N/A	4.9k	1.8	N/A	2.8k
Munin2	68.4	6.2k	4.5	15.1	1.4k	48.9	6.0k	4.5	10.8	1.3k	N/A	27k	5.9	N/A	4.5k
Munin4	67.8	6.1k	5.1	13.4	1.2k	47.7	7.6k	5.2	9.2	1.5k	N/A	32k	6.9	N/A	4.6k

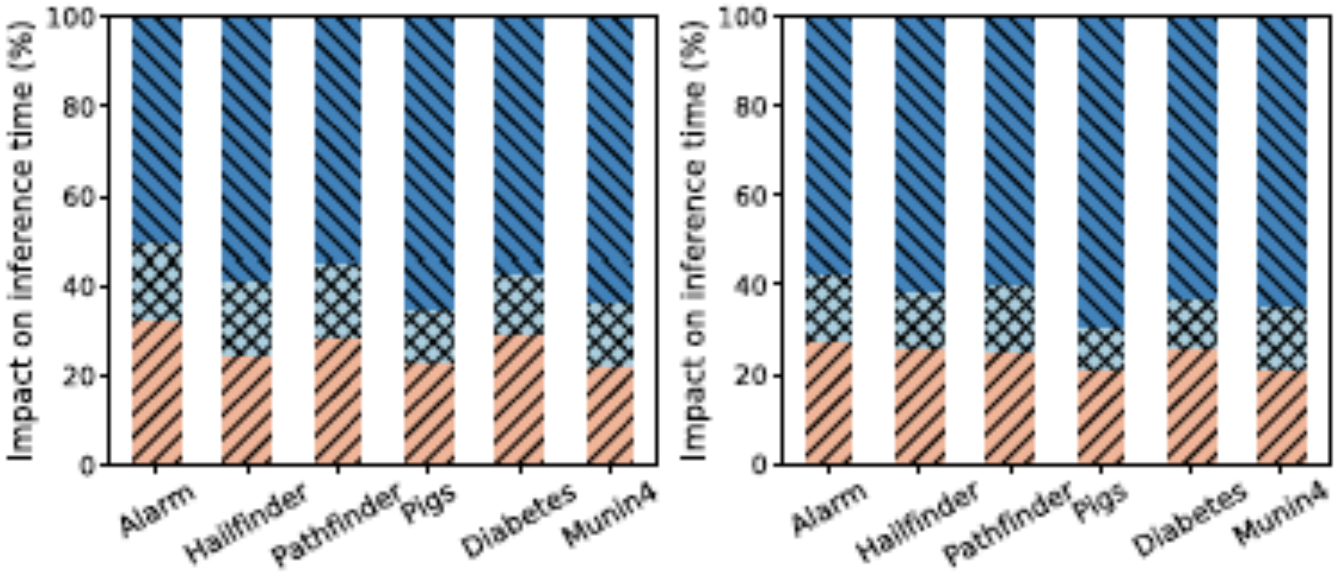
PGM	SISv1					AIS-BN					EPIS-BN				
	Time (sec)			Speedup		Time (sec)			Speedup		Time (sec)			Speedup	
	S	B	Ours	S	B	S	B	Ours	S	B	S	B	Ours	S	B
Alarm	0.46	N/A	0.15	3.2	N/A	0.77	10.2	0.15	5.2	69.3	0.65	N/A	0.14	4.5	N/A
Hailfinder	0.83	N/A	0.27	3.1	N/A	1.3	27.7	0.27	4.9	104	0.83	N/A	0.26	3.2	N/A
Pathfinder	13.0	N/A	1.3	10.0	N/A	11.5	2.7k	1.3	8.8	2.1k	6.0	N/A	1.3	4.5	N/A
Pigs	30.0	N/A	1.8	16.8	N/A	32.0	6.9k	1.8	17.9	3.9k	17.8	N/A	1.8	10.2	N/A
Munin2	94.2	N/A	6.1	15.4	N/A	121	47k	6.2	19.7	7.4k	56.3	N/A	5.8	9.7	N/A
Munin4	63.9	N/A	6.8	9.3	N/A	119	55k	6.9	17.2	7.9k	62.9	N/A	6.6	9.6	N/A

Our solution support more algorithms.

3 – 20 x faster than SMILE; two – four orders of magnitude speedup than BNJ.

Experimental Evaluation

- Impact of individual optimizations:



(e) AIS-BN

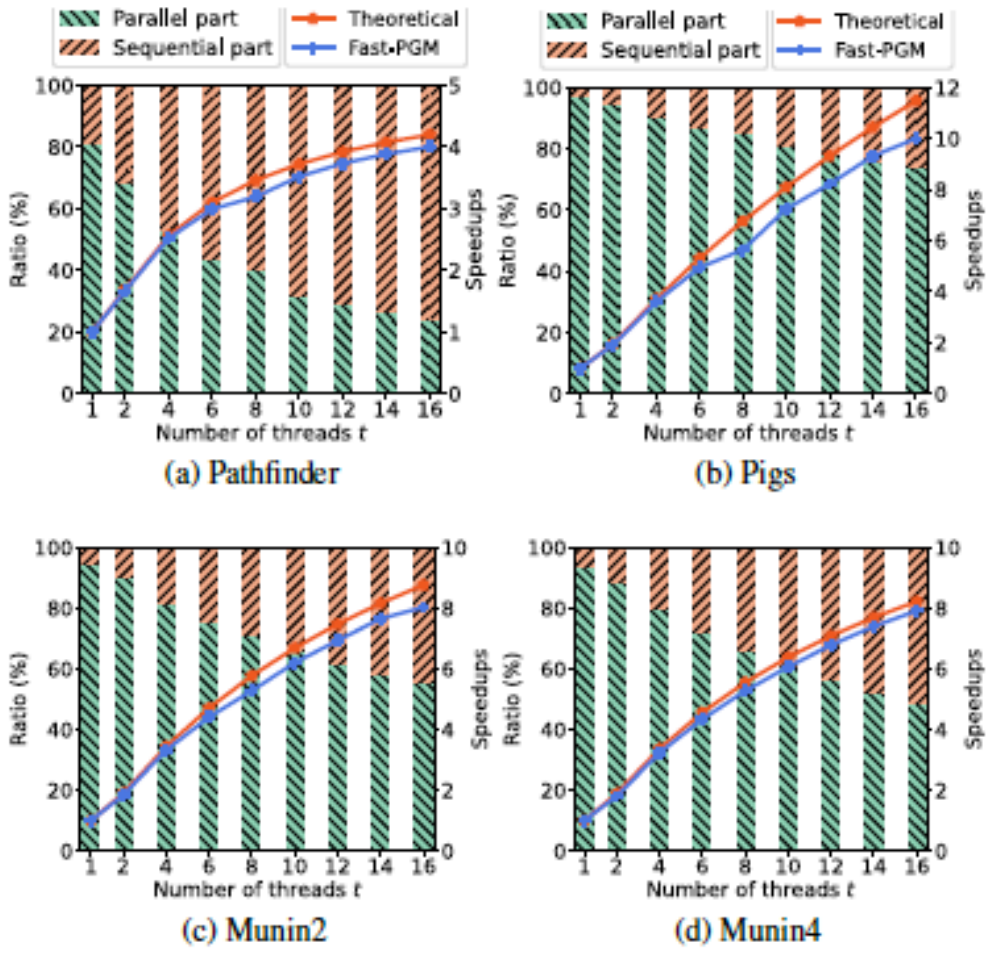
(f) EPIS-BN

Overall: 25% from memory management, 14% from computation simplification, 61% from parallelization.

Learning-based algorithms benefit more from memory management and computation simplification compared to non-learning-based algorithms.

Experimental Evaluation

- Comparison with theoretical speedup:



Theoretical speedup is computed by Amdahl's law:

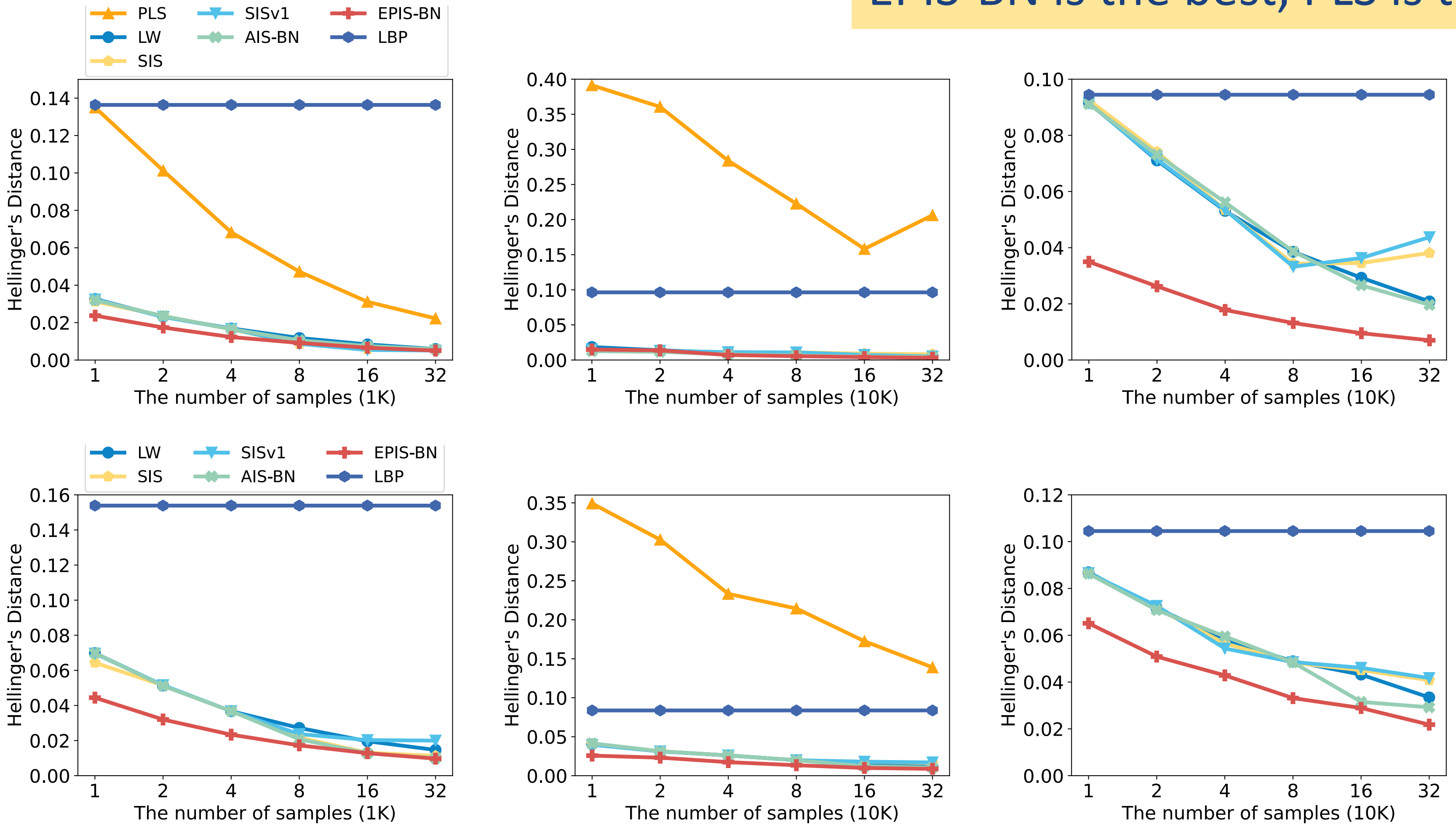
$$Speedup(t) = \frac{1}{(1 - rp) + \frac{rp}{t}}$$

Our practical speedups approach the theoretical speedups.

Experimental Evaluation

- Accuracy of approximation:

EPIS-BN is the best, PLS is the worst.



Outline

- Background and Motivation
- Objective I: Good Generality and Flexibility
- Objective II: High Efficiency
- Experimental Evaluation
- **Conclusion**

Conclusion

- We propose Fast-PGM, a system for PGM inference. Through systematic abstraction, Fast-PGM provides a general framework with rich interfaces, enabling fast and easy optimization, extension, and customization.
- We incorporate memory management, computation simplification, and parallelization techniques, which are applicable to other PGM topics and can inspire acceleration for a broader class of graph-based algorithms.
- We conduct experiments to study the effectiveness of Fast-PGM and the impact of our optimizations.
- Future work could extend Fast-PGM to distributed environment and incorporate additional inference algorithms.

USENIX
ATC '24

Thank you for listening!
Q & A

Fast Inference for Probabilistic Graphical Models

Jiantong Jiang, Zeyi Wen, Atif Mansoor, Ajmal Mian

Contact information:

Email: jjiantong@gmail.com

Code: github.com/jjiantong/FastPGM

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

THE UNIVERSITY OF WESTERN AUSTRALIA