Fast Parallel Bayesian Network Structure Learning

Jiantong Jiang, Zeyi Wen, Ajmal Mian

The University of Western Australia

THE UNIVERSITY OF WESTERN AUSTRALIA

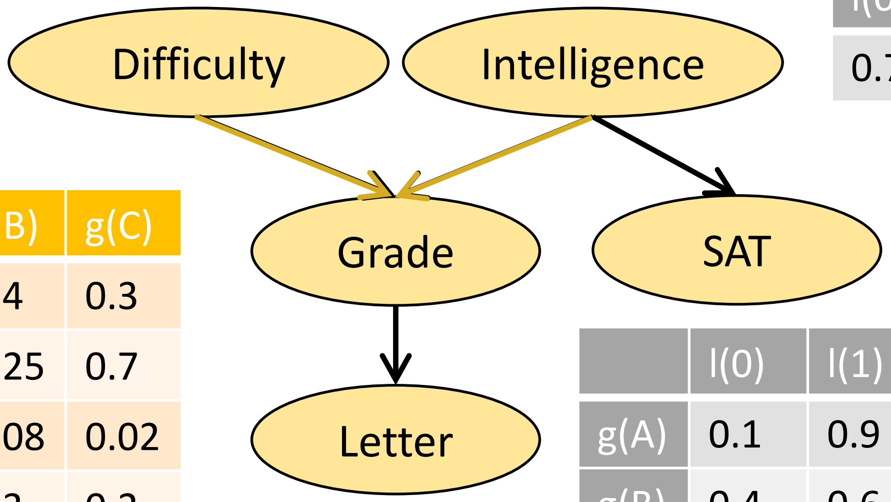
Outline

- **Background**
- Our proposed Fast-BNS
- Experimental results
- Conclusion

Bayesian Networks

- Bayesian Networks (BNs) are probabilistic graphical models.
- A BN is defined by:
  - a network structure (a DAG G)
  - a joint probability distribution
    - ➢ can be factorized into local Conditional Probability Distributions (CPDs) of each node in *G*.

d(0)	d(1)
0.6	0.4



i(0)	i(1)
0.7	0.3

	g(A)	g(B)	g(C)
i(0),d(0)	0.3	0.4	0.3
i(0),d(1)	0.05	0.25	0.7
i(1),d(0)	0.9	0.08	0.02
i(1),d(1)	0.5	0.3	0.2

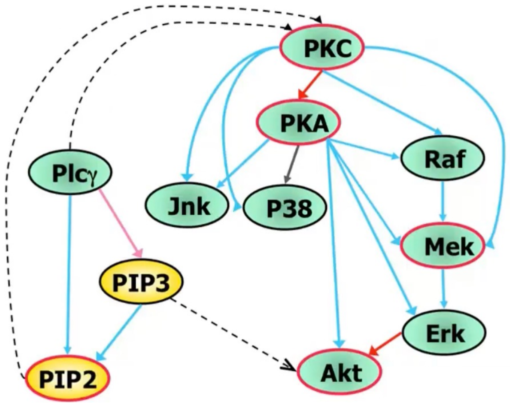
	s(0)	s(1)
i(0)	0.95	0.05
i(1)	0.2	0.8

	l(0)	l(1)
g(A)	0.1	0.9
g(B)	0.4	0.6
g(C)	0.99	0.01

Bayesian Network Applications

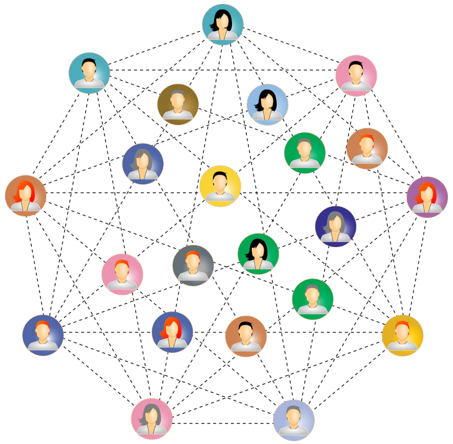
- BNs are suitable for representing knowledge with uncertainty.
- BNs have been applied in a wide range of applications.

Medical diagnosis



Biological network reconstruction

Forecasting



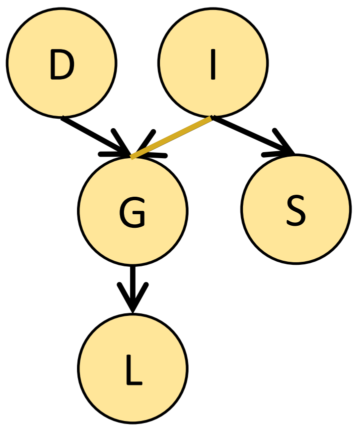
Social network models

BN Structure Learning

- **Structure learning**: learn DAGs that are well matched the observed data
- Constraint-based approaches:
  - Test independencies, build based on independencies
    - ➢ By conditional independence (CI) tests

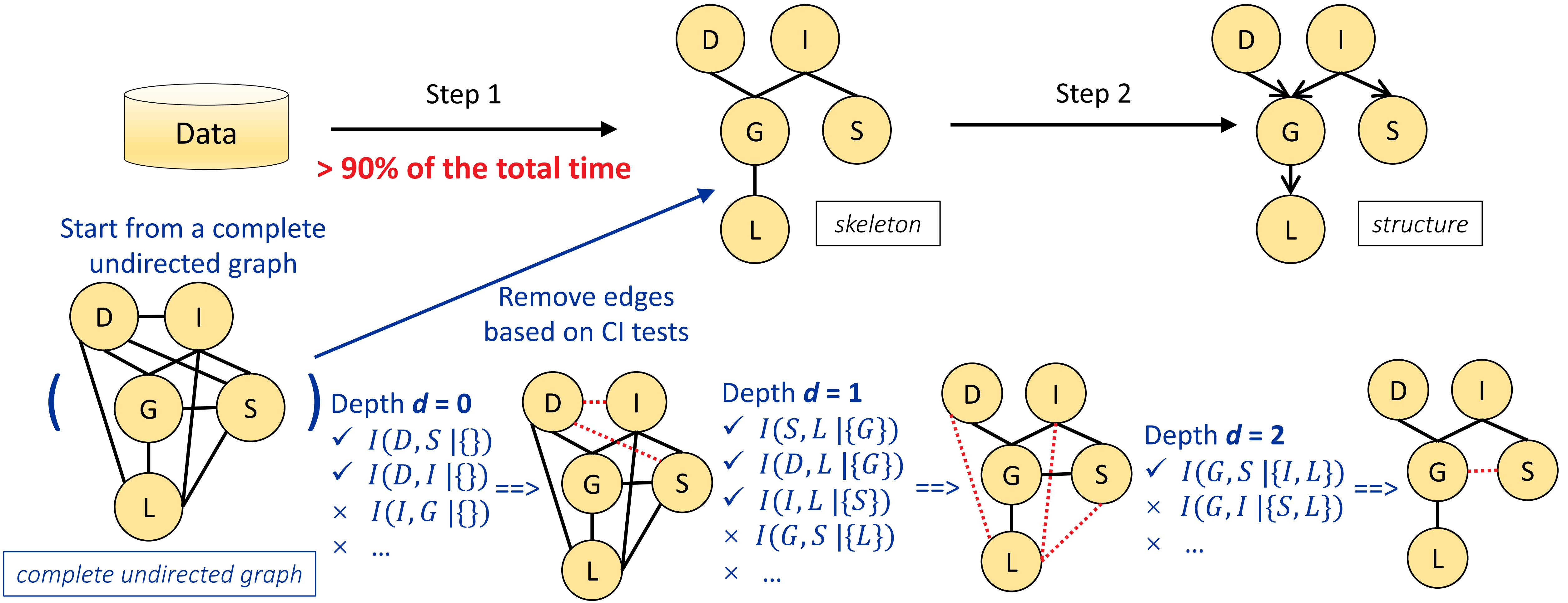
      e.g. $I(I, G|\{D\})$

      Basic theory: no $\boldsymbol{S}$ s.t. $I(I, G\ |\boldsymbol{S})$ => $I - G$
  - Most are based on the PC-stable algorithm



× $I(I, G\ |\{\})$
× $I(I, G\ |\{D\})$
× $I(I, G\ |\{S\})$
× $I(I, G\ |\{L\})$
× $I(I, G\ |\{D, S\})$
× $I(I, G\ |\{D, L\})$
× $I(I, G\ |\{S, L\})$
× $I(I, G\ |\{D, S, L\})$

Key Steps of PC-stable



PC-stable Libraries

- **Key barrier**: a large number of CI tests
- Sequential implementations:
  - bnlearn [Scutari, 2009]
  - pcalg [Kalisch et al, 2012]
  - tetrad [Ramsey et al, 2018]
- Parallel implementations:
  - bnlearn [Scutari, 2014]
  - Parallel-PC [Le et al, 2016]
  - BIB-based method [Madsen et al, 2017]

IPDPS 2022 · VIRTUAL ·

Limitations of Parallel Implementations

THE UNIVERSITY OF WESTERN AUSTRALIA

- Coarse-grained scheme: **edge-level parallelism**
  - Parallelize the processing of edges inside each depth
  - Limitation: load unbalancing
    - Different number of adjacent nodes
    - Undetermined number of CI tests
- Fine-grained scheme: **sample-level parallelism**
  - Parallelize among samples inside each CI test
    - i.e. parallelize traversing of the whole data set
  - Limitations:
    - Expensive atomic operations
    - High parallel overhead

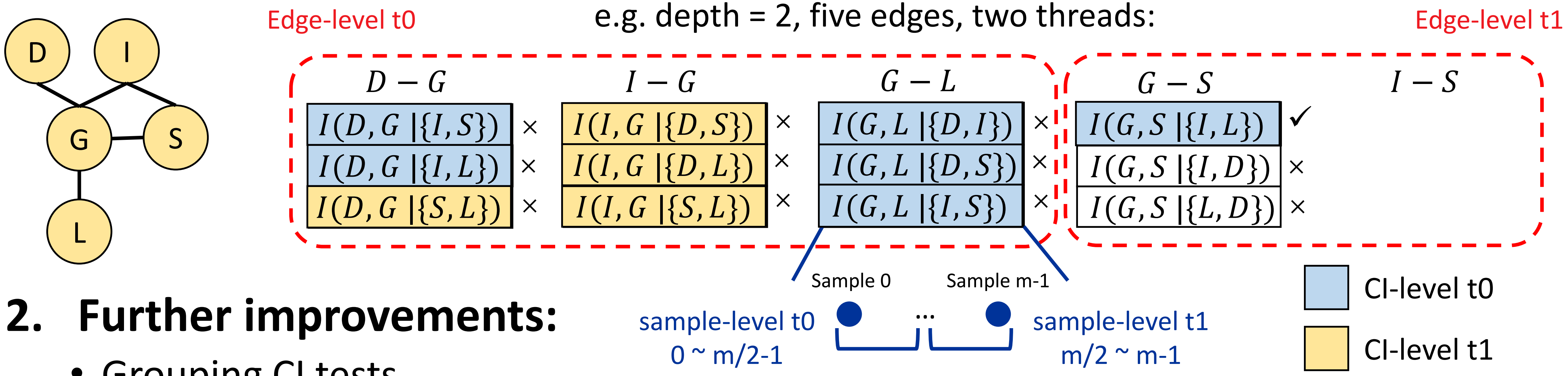
Outline

- Background
- **Our proposed Fast-BNS**
- Experimental results
- Conclusion

Overview of Fast-BNS

1. CI-level parallelism:

- between edge-level and sample-level
- Parallelize CI tests of different edges, implemented using a dynamic work pool

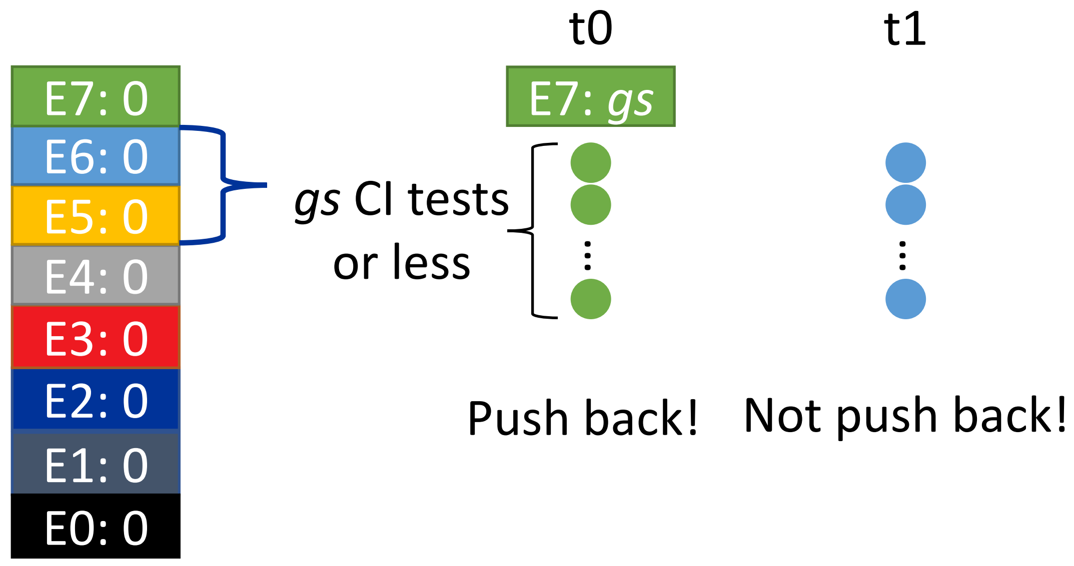


2. Further improvements:

- Grouping CI tests
- Using a cache-friendly data storage
- Generating conditioning sets on-the-fly

CI-Level Parallelism

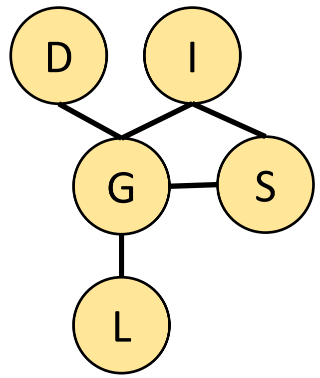
- Key idea: a dynamic work pool, contains:
  1. The edges required to be processed
  2. The edges' processing progresses with respect to the CI tests



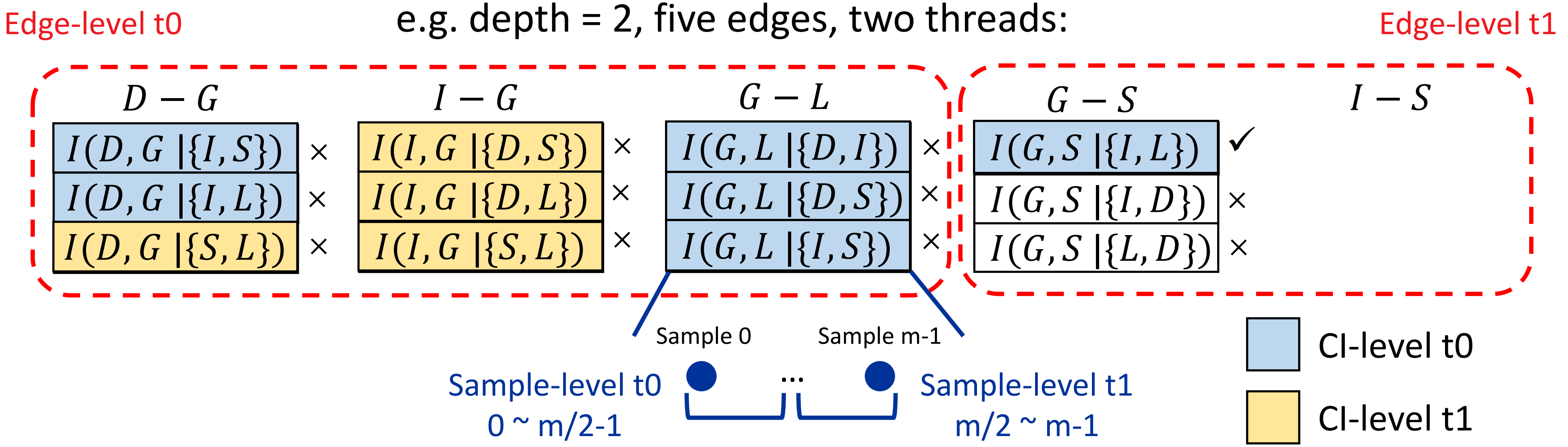
Work pool

Intuition:
Multiple threads processing multiple CI tests on different edges in parallel, but **a thread is never bounded to a fixed edge**.

Comparison of Parallelism



e.g. depth = 2, five edges, two threads:



Summary:

- CI-level parallelism relieves the efficiency issues in edge-level and sample-level parallelism.

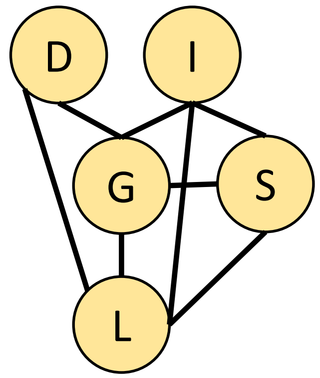
TABLE I: Comparison between edge-level parallelism, sample-level parallelism and the proposed CI-level parallelism.

Granularity of parallelism	Load balance	Avoid atomic operations	Reasonable workloads
Edge-level parallelism	✗	✓	✓
Sample-level parallelism	✓	✗	✗
CI-level parallelism	✓	✓	✓

Further Improvements

✓**Grouping CI tests of the edges with the same endpoints**

- e.g. view edges $D - L$ and $L - D$ as the same edge
- To reduce unnecessary CI tests



✓**Using a cache-friendly data storage**

- To reduce cache misses

✓**Generating conditioning sets on-the-fly**

- Generate set given any *d* and processing progress
- To reduce memory consumption

Outline

- Background
- Our proposed Fast-BNS
- **Experimental results**
- Conclusion

Experimental Setup

- Two 26-core 2GHz Intel Xeon Platinum 8167M CPUs and 768GB main memory
- Implemented using OpenMP in C++

- Baselines:
  - Sequential: bnlearn-seq [Scutari, 2009] , pcalg [Kalisch et al, 2012] , tetrad [Ramsey et al, 2018]
  - Parallel: bnlearn-par [Scutari, 2014] , parallel-PC [Le et al, 2016]
- Datasets:
  - *Alarm*, *Insurance*, *Hepar2*, *Munin1*, *Diabetes*, *Link*, *Munin2*, *Munin3*
  - # nodes from 37 to 1041; # edges from 46 to 1306

Overall Comparison

- Overall comparison of execution time and speedup

TABLE II: Execution time and speedup.

Data set	Sequential implementation				Parallel implementation		
	Time (sec)	Speedup			Time (sec)	Speedup	
	Fast-BNS	bnlearn	tetrad	pcalg	Fast-BNS	bnlearn	parallel-PC
Alarm	0.12	3.5	45.1	450	0.017	24.5	890
Insurance	0.24	1.4	55	302	0.037	9.2	687
Hepar2	1.57	2.8	24	133	0.19	15.2	852
Munin1	15.5	7.2	49.8	140	1.78	9.3	91.3
Diabetes	23.3k	4.9	> 7.4		1203	6.4	44.9
Link	62.9k	> 2.7			4349	11.4	> 39.7
Munin2	3496	8.0	> 49.4		293	9.3	> 590
Munin3	8081	4.8	> 21.4		751	4.8	> 230

Sequential: 1.4 - 8 times faster than bnlearn-seq
Parallel: 4.8 – 24.5 times faster than bnlearn-par

Overall Comparison

- Detailed measurement
  - Use perf Linux profiler

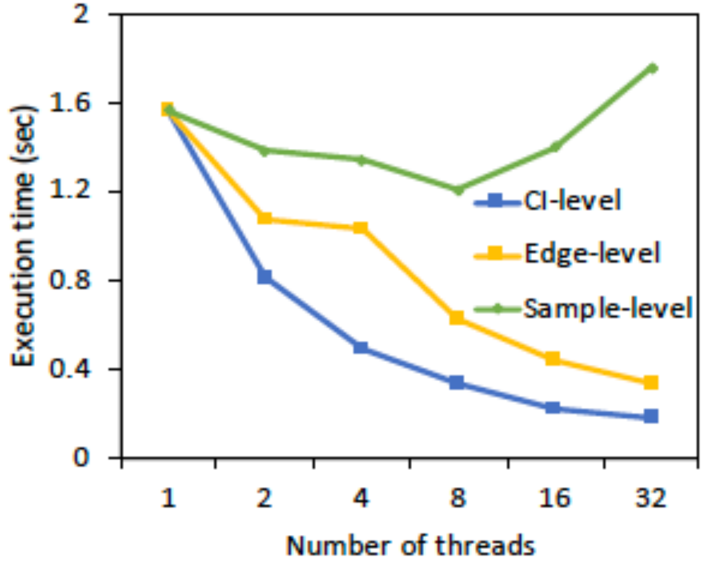
TABLE IV: Detailed comparison.

Hepar2	L1-cache accesses	L1-cache misses (rate)	LL-cache accesses	LL-cache misses (rate)	FLOPS	CPU utilization
Fast-BNS-par	4.5×10^9	7.9×10^7 (1.78%)	1.6×10^6	8.1×10^4 (5.1%)	1.4×10^9	12.7
Fast-BNS-seq	4.1×10^9	7.2×10^7 (1.73%)	2.5×10^5	1.5×10^4 (6.0%)	2.3×10^8	1
bnlearn-par	1.5×10^{10}	4.7×10^8 (3.17%)	4.2×10^7	1.7×10^7 (39.9%)	7.0×10^7	3.7

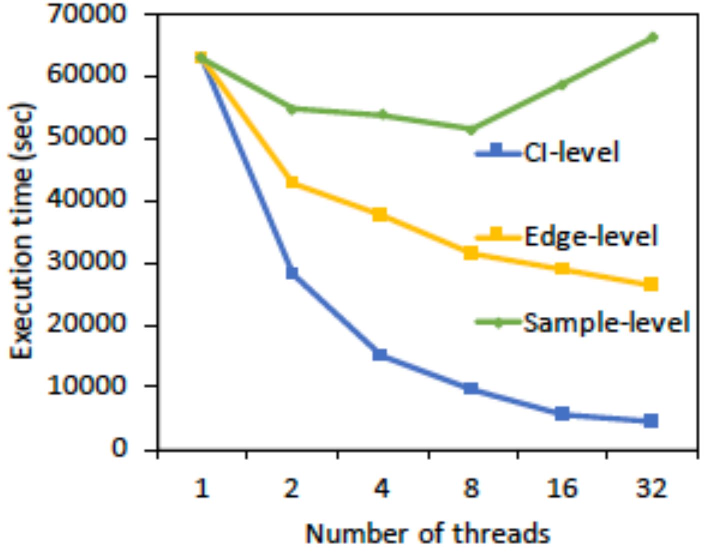
Advantage: increase CPU utilization and FLOPs, decrease L1 cache, LL cache accesses and rate of cache misses.

Different Granularities

- Comparison of different granularities:



(c) Hepar2

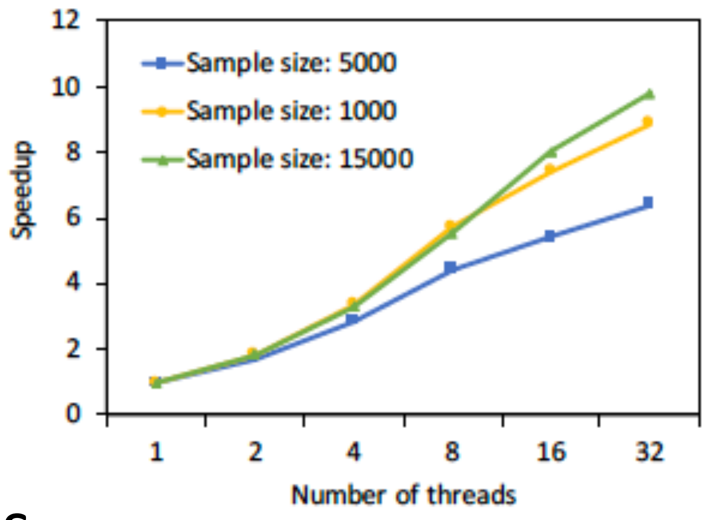


(f) Link

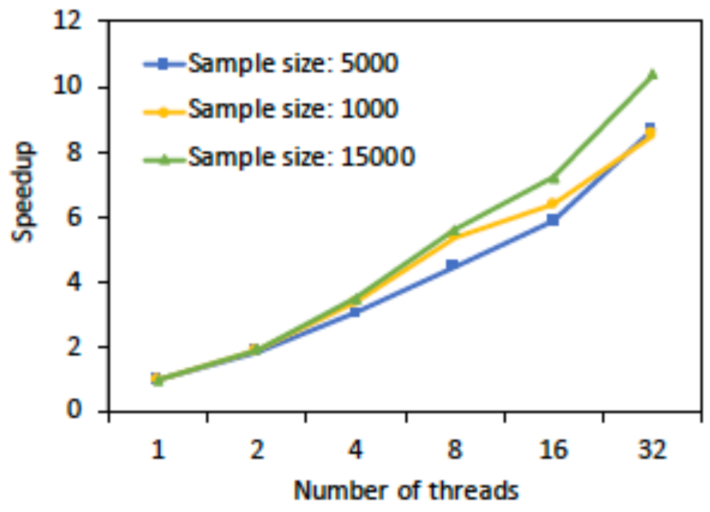
CI level parallelism always leads to the shortest execution time under different number of threads.

Sensitivity Studies

- Varying sample size

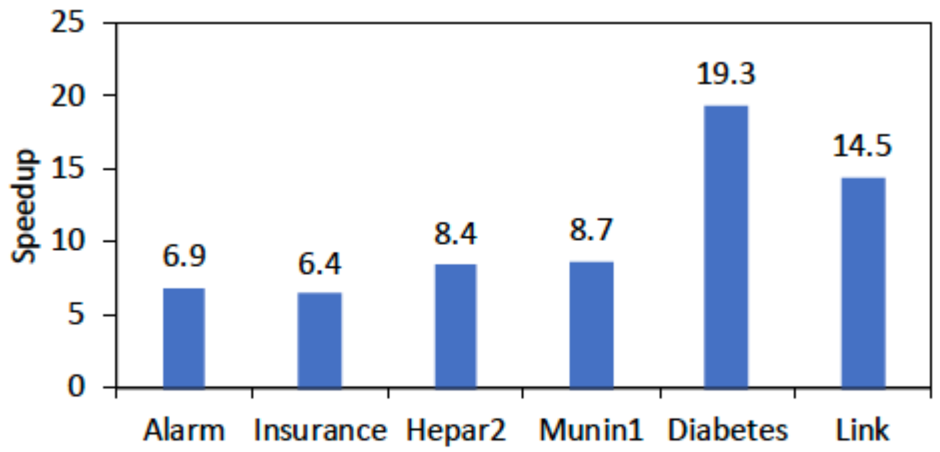


(b) Insurance



(d) Munin1

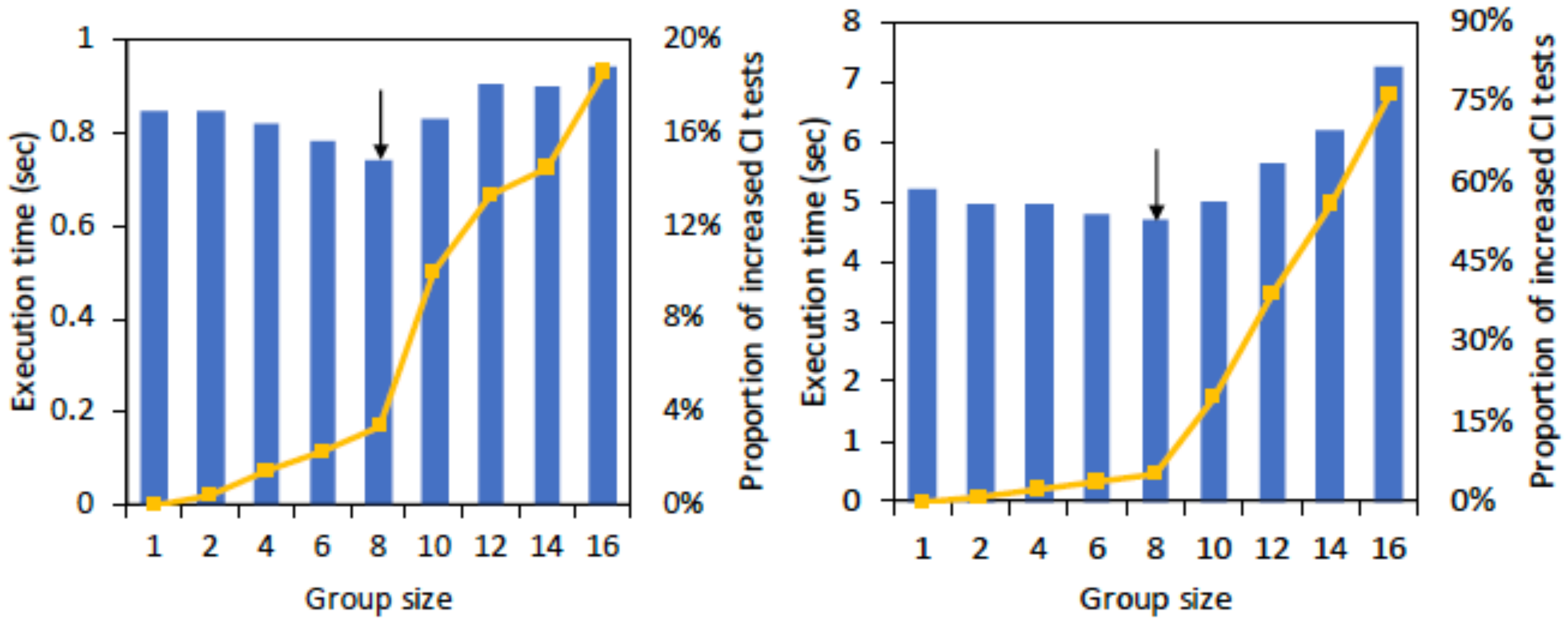
- Different network sizes



Good scalability of Fast-BNS to sample size and network size

Sensitivity Studies

- Varying group size
  - Group size (*gs*): trade-off between the number of CI tests and memory accesses



(c) Hepar2 (d) Munin1

6 or 8 is good choice in our experiments.

IPDPS 2022 • VIRTUAL •

Conclusion

THE UNIVERSITY OF WESTERN AUSTRALIA

- We proposed Fast-BNS for efficient BN structure learning which exploits the **CI-level parallelism** and employs a series of novel techniques.

- Fast-BNS tackles the challenges of addressing **load unbalancing** issues, reducing **atomic operations** and amortizing **parallel overhead**.

- Experimental results show that Fast-BNS-seq is 1.4 - 8 times faster and Fast-BNS-par is 4.8 - 24.5 times faster. Moreover, it has good scalability to the network size and sample size.

Thank you for listening!