Inference Complexity As Learning Bias

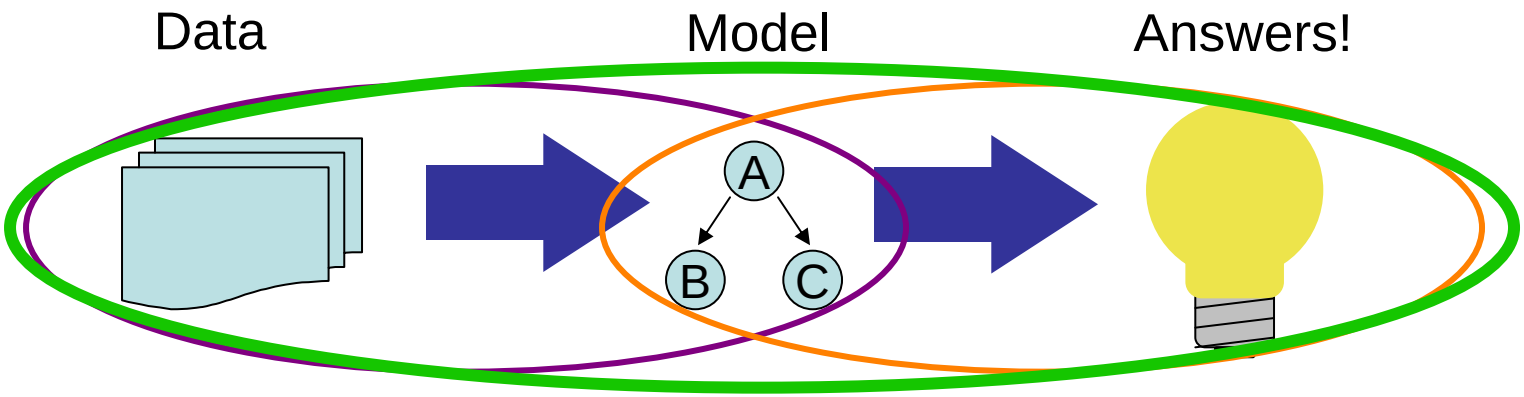
Pedro Domingos
Dept. of Computer Science & Eng.
University of Washington

Joint work with Daniel Lowd

Don’t use model complexity as your learning bias …

Use inference complexity.

The Goal



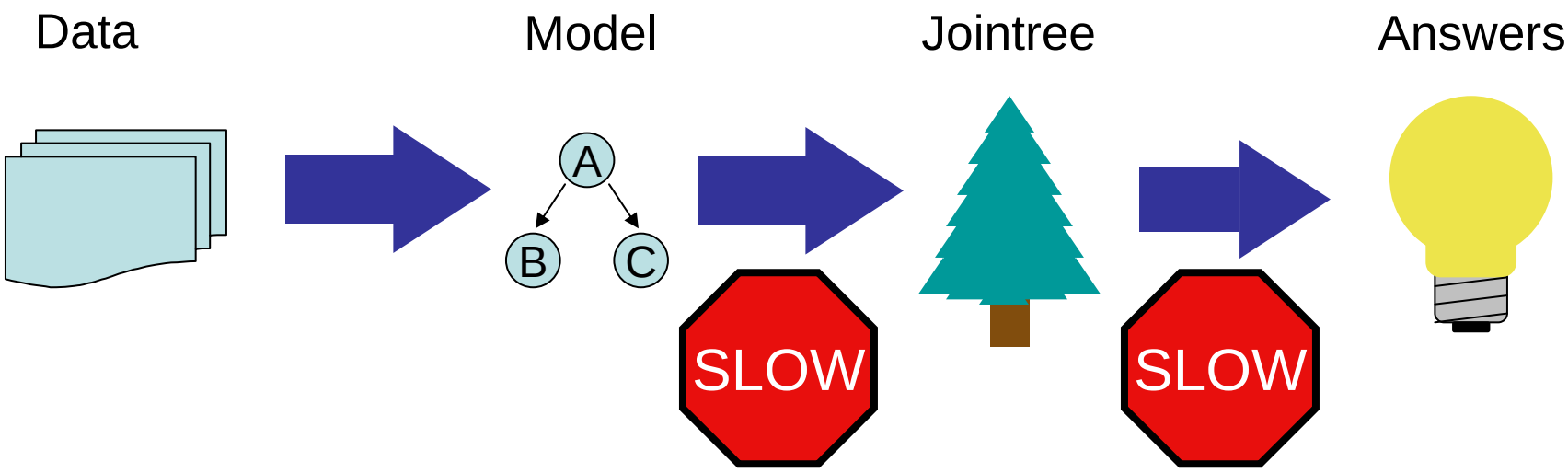
This talk:

- How to learn accurate and efficient models by tightly integrating learning and inference
- Experiments: exact inference in < 100ms in models with treewidth > 100

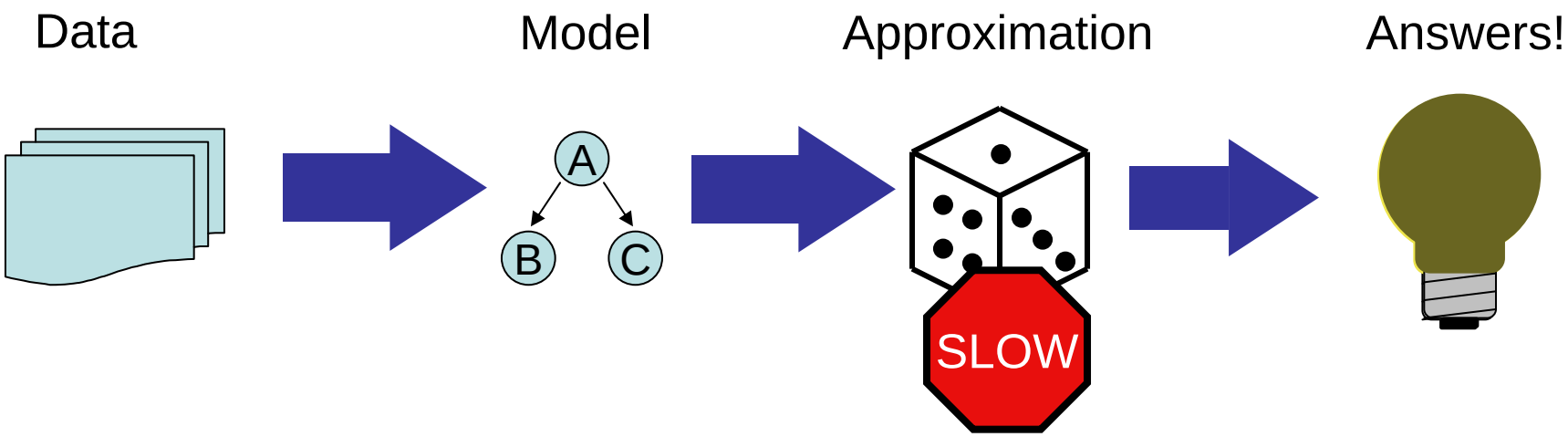
Outline

- Standard solutions (and why they fail)
- Background
  - Learning with Bayesian networks
  - Inference with arithmetic circuits
- Learning arithmetic circuits
  - Scoring
  - Search
  - Efficiency
- Experiments
- Conclusion

Solution 1: Exact Inference



Solution 2: Approximate Inference

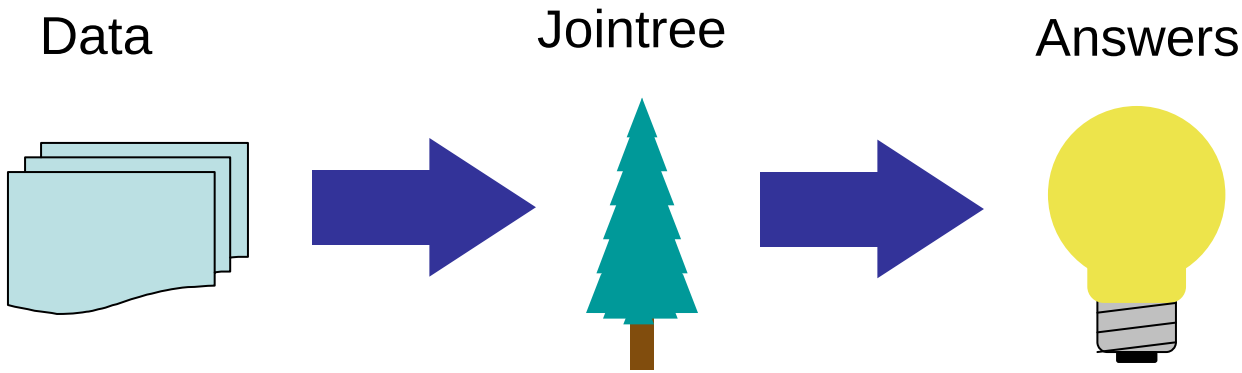


Approximations are often too inaccurate.

More accurate algorithms tend to be slower.

Solution 3: Learn a tractable model

Related work: Thin junction trees



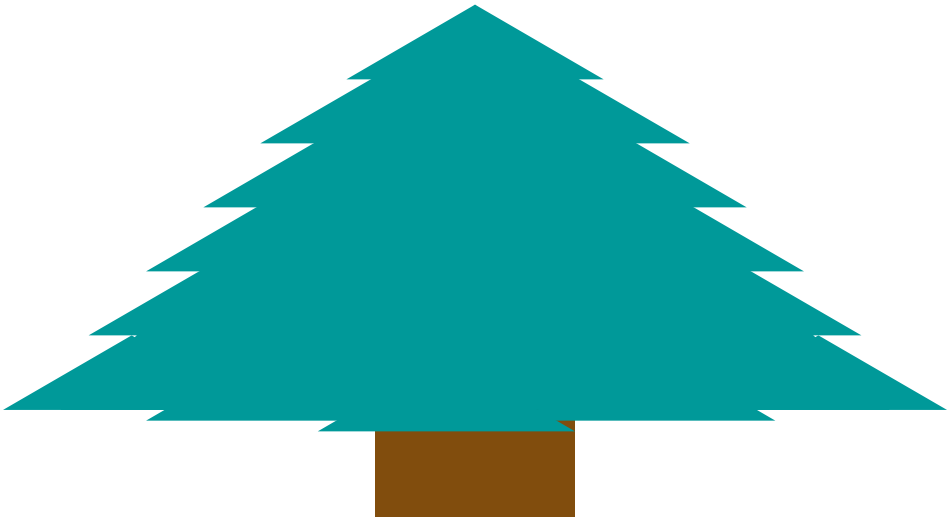
Polynomial in data, but still exponential in treewidth

[E.g.: Chechetka & Guestrin, 2007]

Thin junction trees are thin

Their junction trees

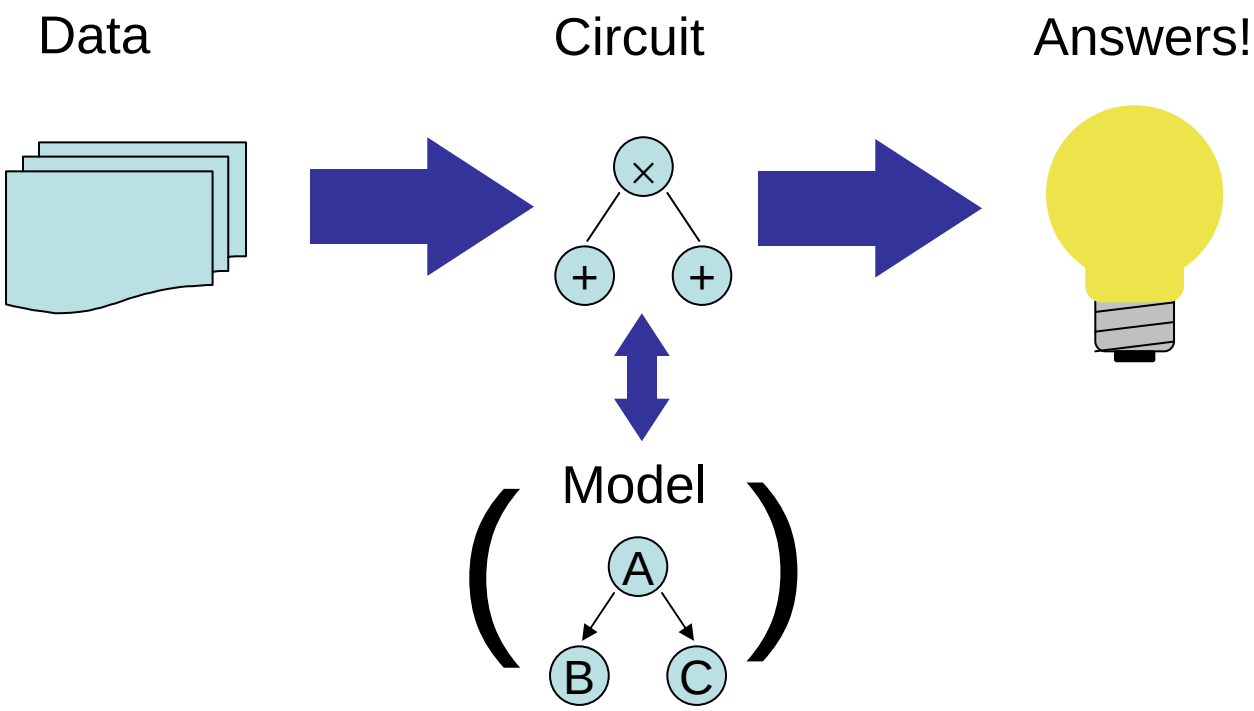
Our junction trees



- Maximum effective treewidth is 2-5
- We have learned models with treewidth >100

Solution 3: Learn a tractable model

Our work: Arithmetic circuits with penalty on circuit size



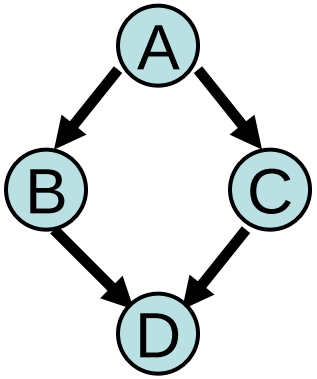
Outline

- Standard solutions (and why they fail)
- Background
  - Learning with Bayesian networks
  - Inference with arithmetic circuits
- Learning arithmetic circuits
  - Overview
  - Optimizations
- Experiments
- Conclusion

Bayesian networks

Problem: Compactly represent probability distribution over many variables

Solution: Conditional independence



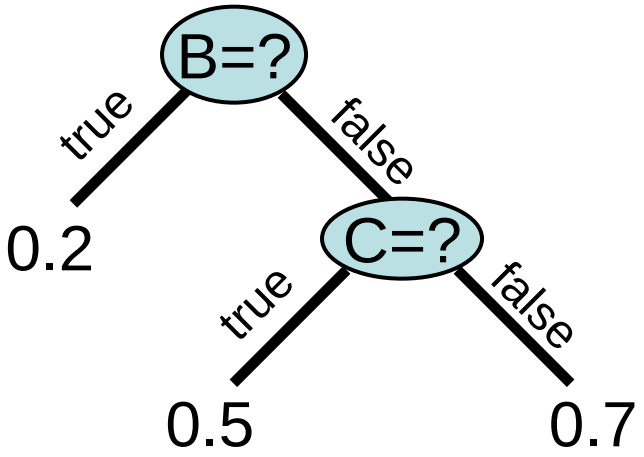
$$P(A,B,C,D) = P(A)\,P(B|A)\,P(C|A)\,P(D|B,C)$$

… With decision-tree CPDs

Problem: Number of parameters is exponential in the maximum number of parents

Solution: Context-specific independence

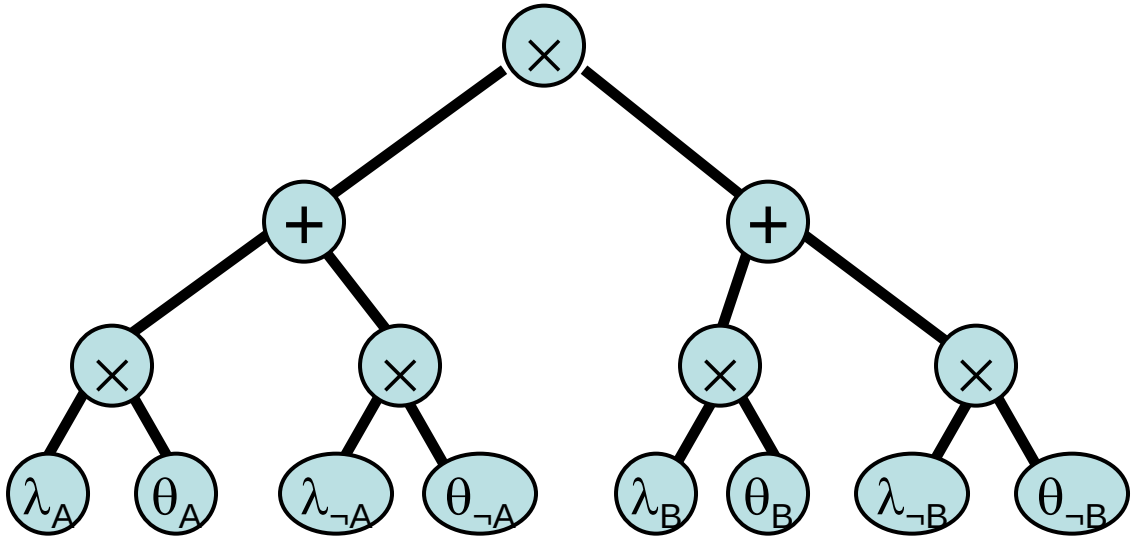
P(D|B,C) =



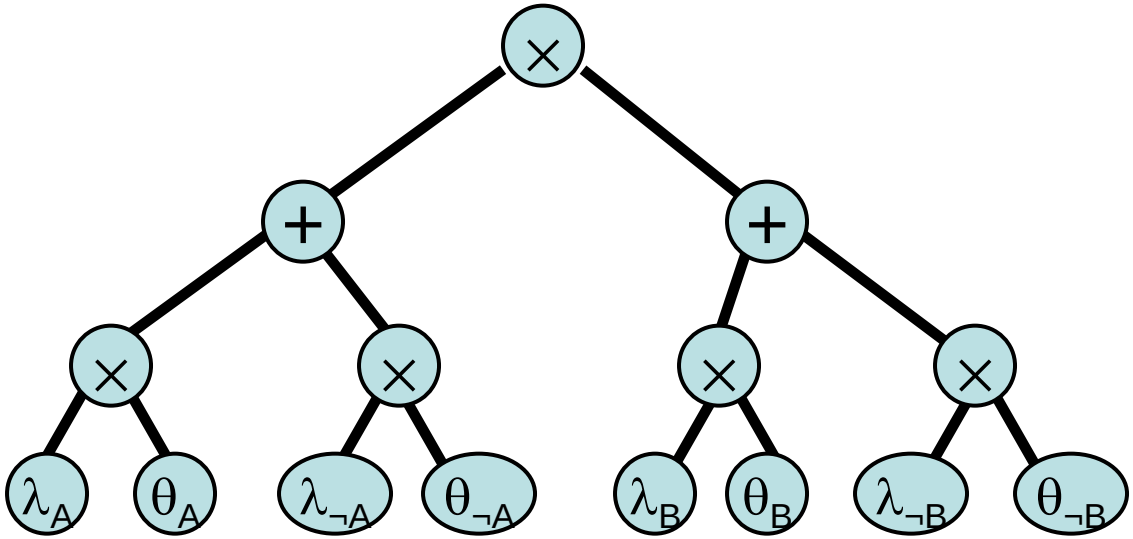
… Compiled to circuits

Problem: Inference is exponential in treewidth

Solution: Compile to arithmetic circuits



Arithmetic circuits



- Directed, acyclic graph with single root
  - Leaf nodes are inputs
  - Interior nodes are addition or multiplication
  - Can represent any distribution
- Inference is linear in model size!
  - Never larger than junction tree
  - Can exploit local structure to save time/space

ACs for Inference

- Bayesian network:
  $P(A,B,C) = P(A)\ P(B)\ P(C|A,B)$
- Network polynomial:
  $\lambda_A \lambda_B \lambda_C\ \theta_A \theta_B \theta_{C|AB} + \lambda_{\neg A} \lambda_B \lambda_C\ \theta_{\neg A} \theta_B \theta_{C|\neg AB} + \ldots$
- Can compute arbitrary marginal queries by evaluating network polynomial.
- Arithmetic circuits (ACs) offer efficient, factored representations of this polynomial.
- Can take advantage of local structure such as context-specific independence.

BN Structure Learning

[Chickering et al., 1996]

- Start with an empty network
- Greedily add splits to decision trees one at a time, enforcing acyclicity

$$\text{score}(C,T) = \log P(T|C) - k_p\, n_p(C)$$

(accuracy – # parameters)

Key Idea

For an arithmetic circuit C on an i.i.d. training sample T:
Typical cost function:

$$\text{score}(C,T) = \log P(T|C) - k_p\, n_p(C)$$
(accuracy – # parameters)

Our cost function:

$$\text{score}(C,T) = \log P(T|C) - k_p\, n_p(C) - k_e\, n_e(C)$$
(accuracy – # parameters – circuit size)

Basic algorithm

Following Chickering et al. (1996), we induce our statistical models by greedily selecting splits for the decision-tree CPDs. Our approach has two key differences:

1. We optimize a different objective function

2. We return a Bayesian network that has already been compiled into a circuit

Efficiency

Compiling each candidate AC from scratch at each step is too expensive.

Instead: Incrementally modify AC as we add splits.

Algorithm

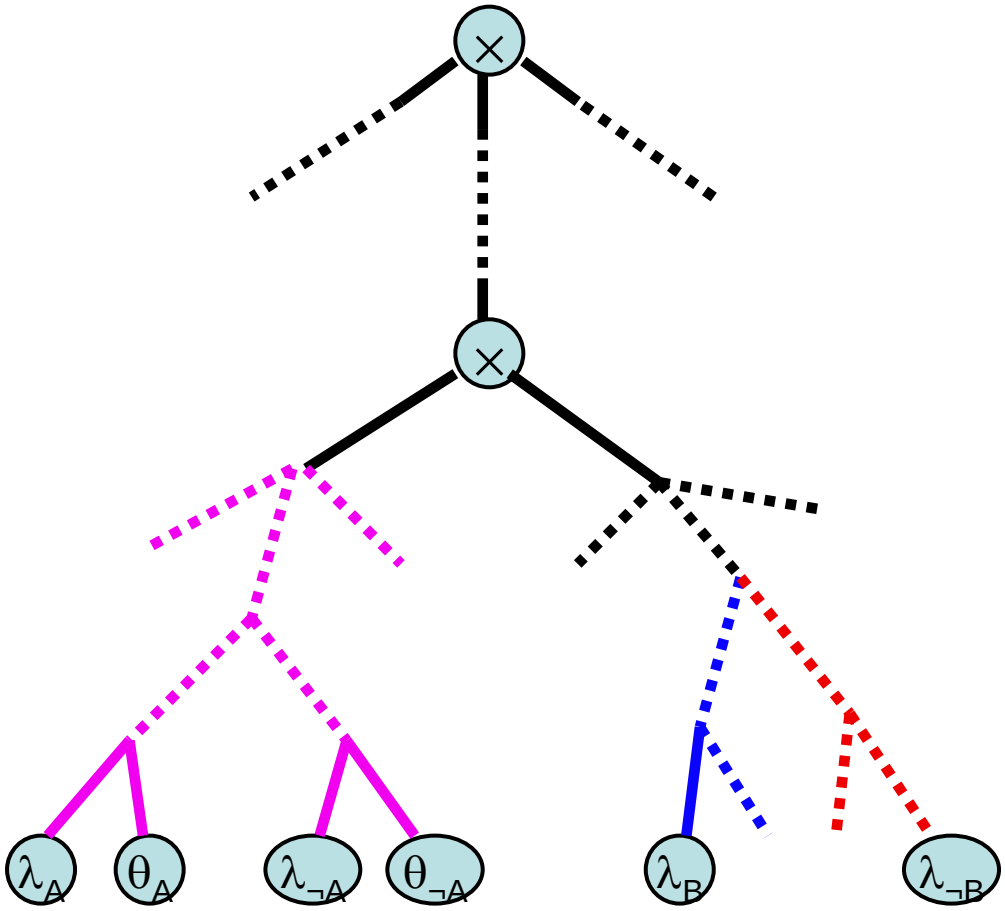
Create initial product of marginals circuit

Create initial split list

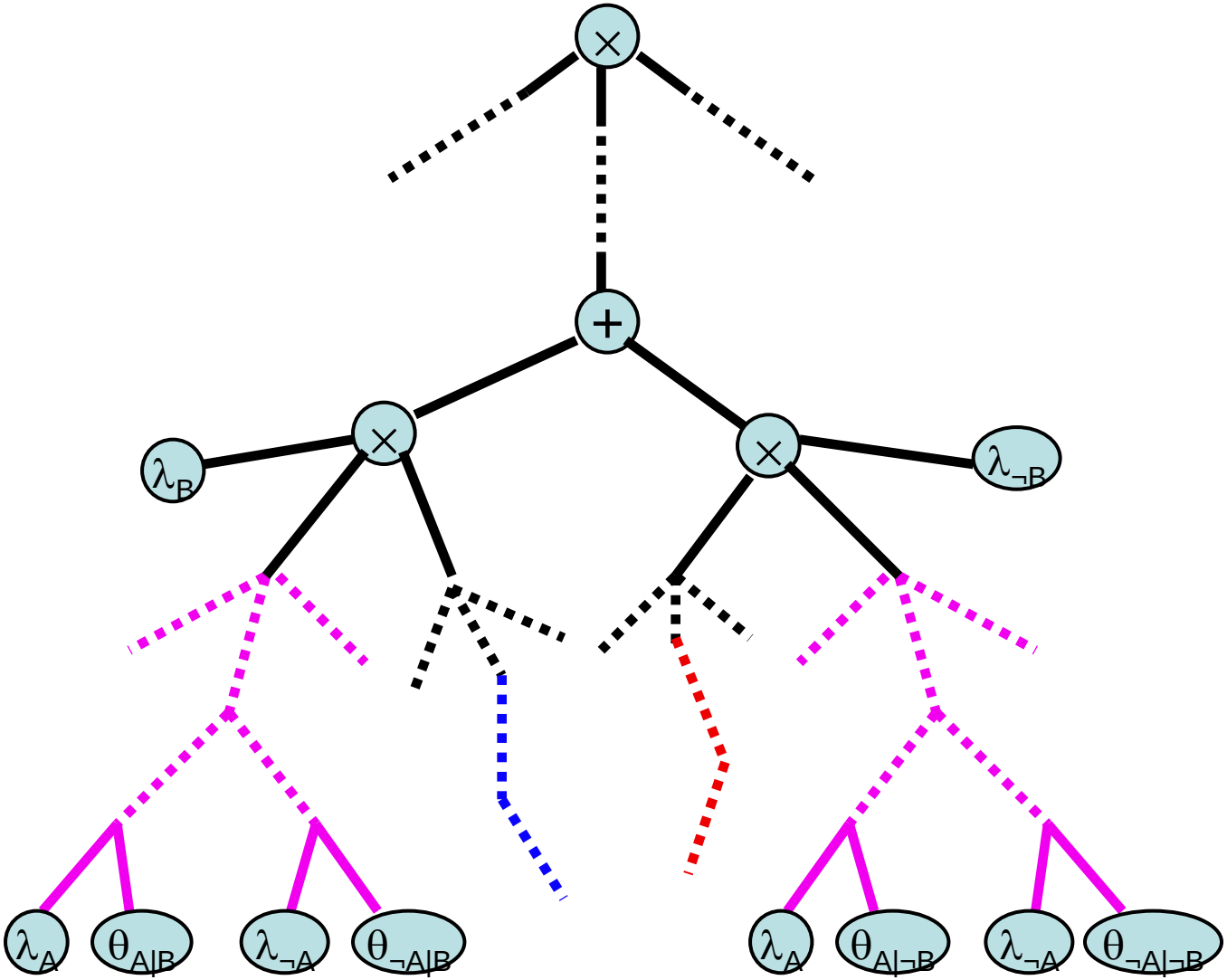
Until convergence:

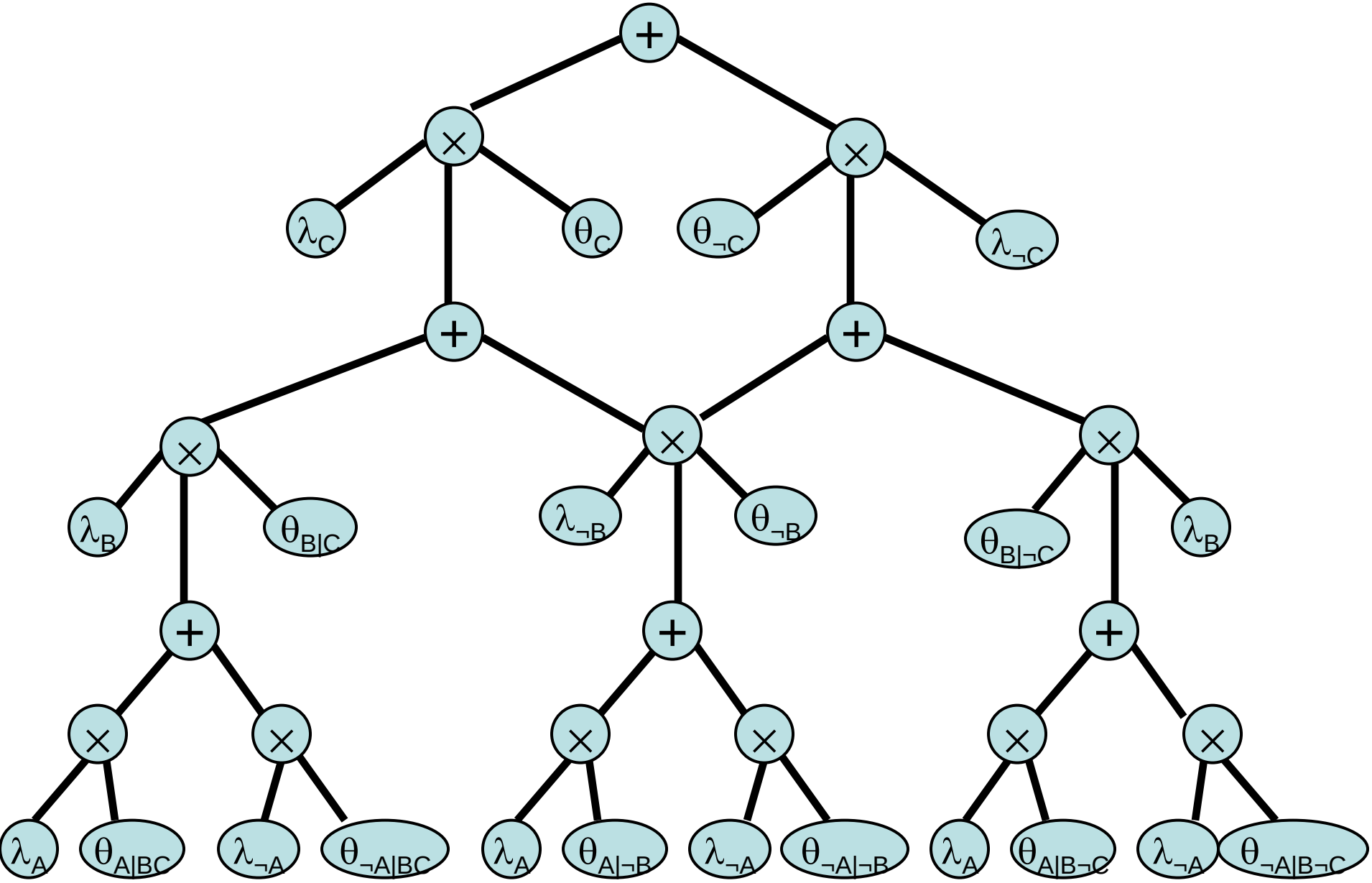
- For each split in list
  - Apply split to circuit
  - Score result
  - Undo split
- Apply highest-scoring split to circuit
- Add new child splits to list
- Remove inconsistent splits from list

Before split



After split


×
+
×
×
λ_B
$\lambda_{\neg B}$
λ_A
$\theta_{A|B}$
$\lambda_{\neg A}$
$\theta_{\neg A|B}$
λ_A
$\theta_{A|\neg B}$
$\lambda_{\neg A}$
$\theta_{\neg A|\neg B}$


+
×
×
λC
θC
θ¬C
λ¬C
+
+
×
×
×
λB
θB|C
λ¬B
θ¬B
θB|¬C
λB
+
+
+
×
×
×
×
×
×
λA
θA|BC
λ¬A
θ¬A|BC
λA
θA|¬B
λ¬A
θ¬A|¬B
λA
θA|B¬C
λ¬A
θ¬A|B¬C

How to split a circuit

D: Parameter nodes to be split

V: Indicators for the splitting variable

M: First mutual ancestors of D and V

For each indicator λ in V,
    Copy all nodes between M and D or V, conditioned on λ.

For each m in M,
    Replace children of m that are ancestors of D or V with a sum over copies of the ancestors times the λ each copy was conditioned on.

Optimizations

We avoid rescoring splits every iteration by:

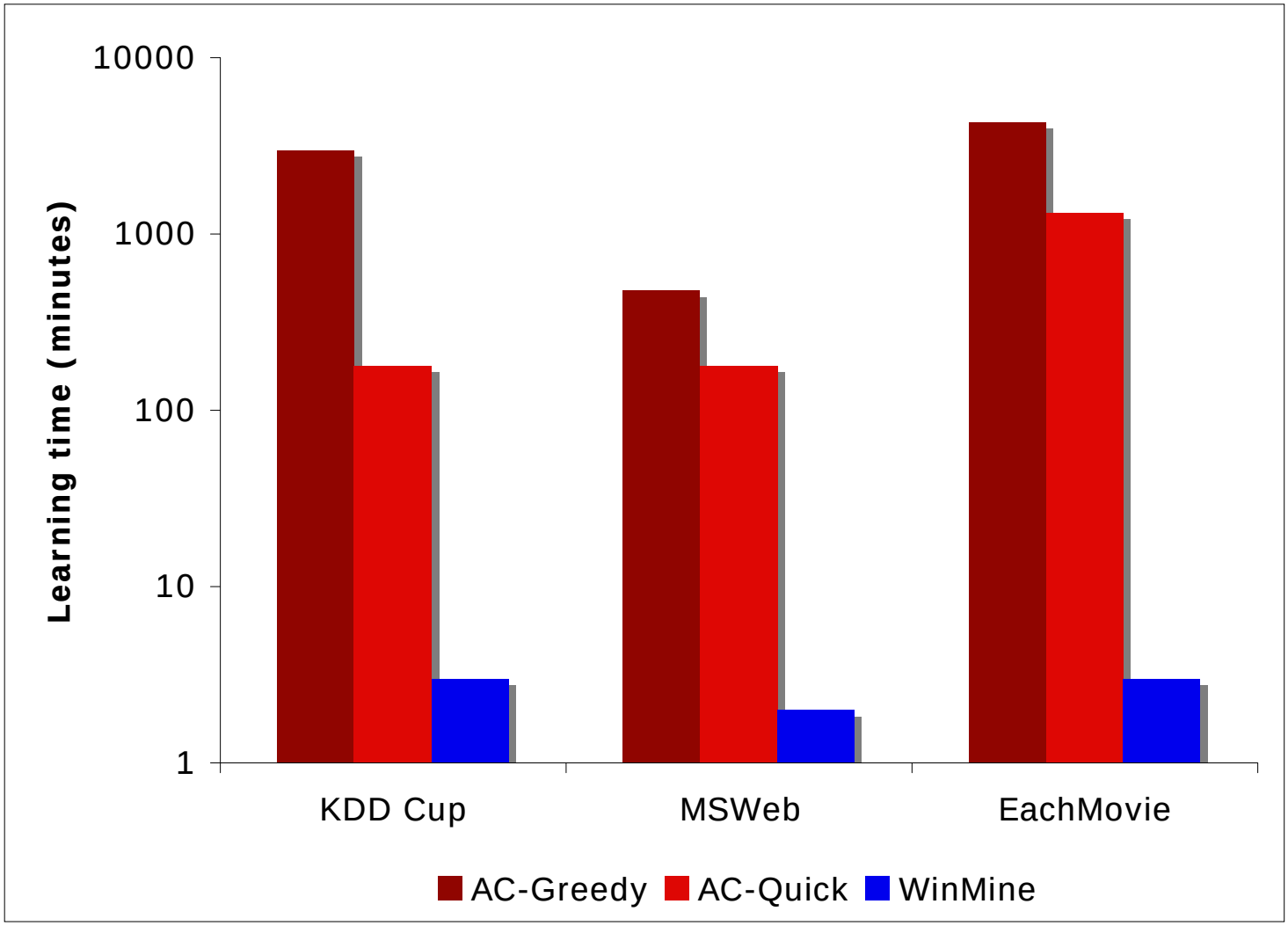
1. Noting that likelihood gain never changes, only number of edges added
2. Evaluating splits with higher likelihood gain first, since likelihood gain is an upper bound on score.
3. Re-evaluate number of edges added only when another split may have affected it (AC-Greedy).
4. Assume the number of edges added by a split only increases as the algorithm progresses (AC-Quick).

Experiments

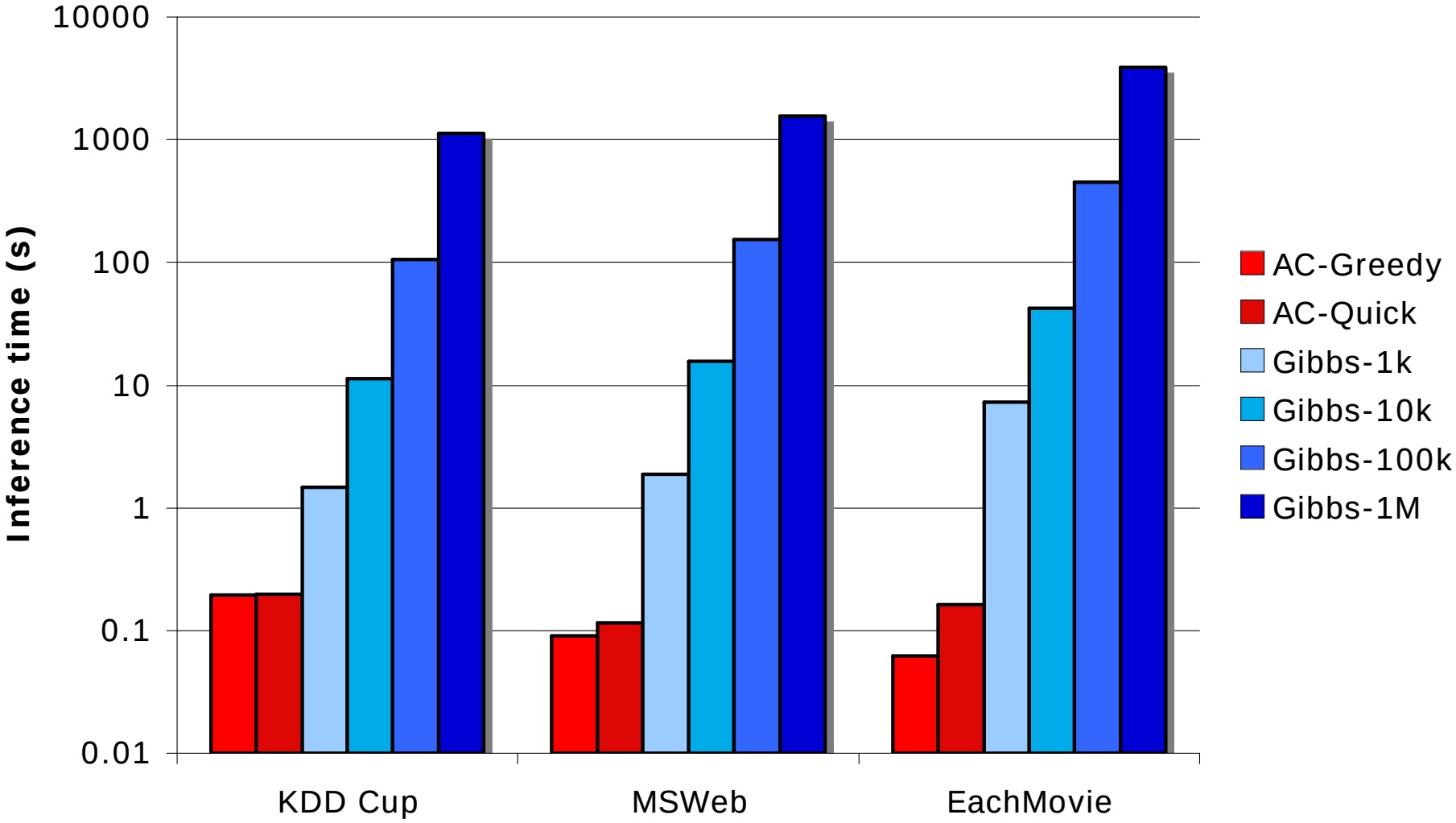
We applied our algorithms (AC-Greedy, AC-Quick) to three real-world datasets, using the WinMine Toolkit as the baseline. WinMine's algorithm is very similar to that of Chickering et al. (1996).

For inference, we generated queries from the test data with varying numbers of evidence and query variables. We used Gibbs sampling on the WinMine models since exact inference was not feasible.

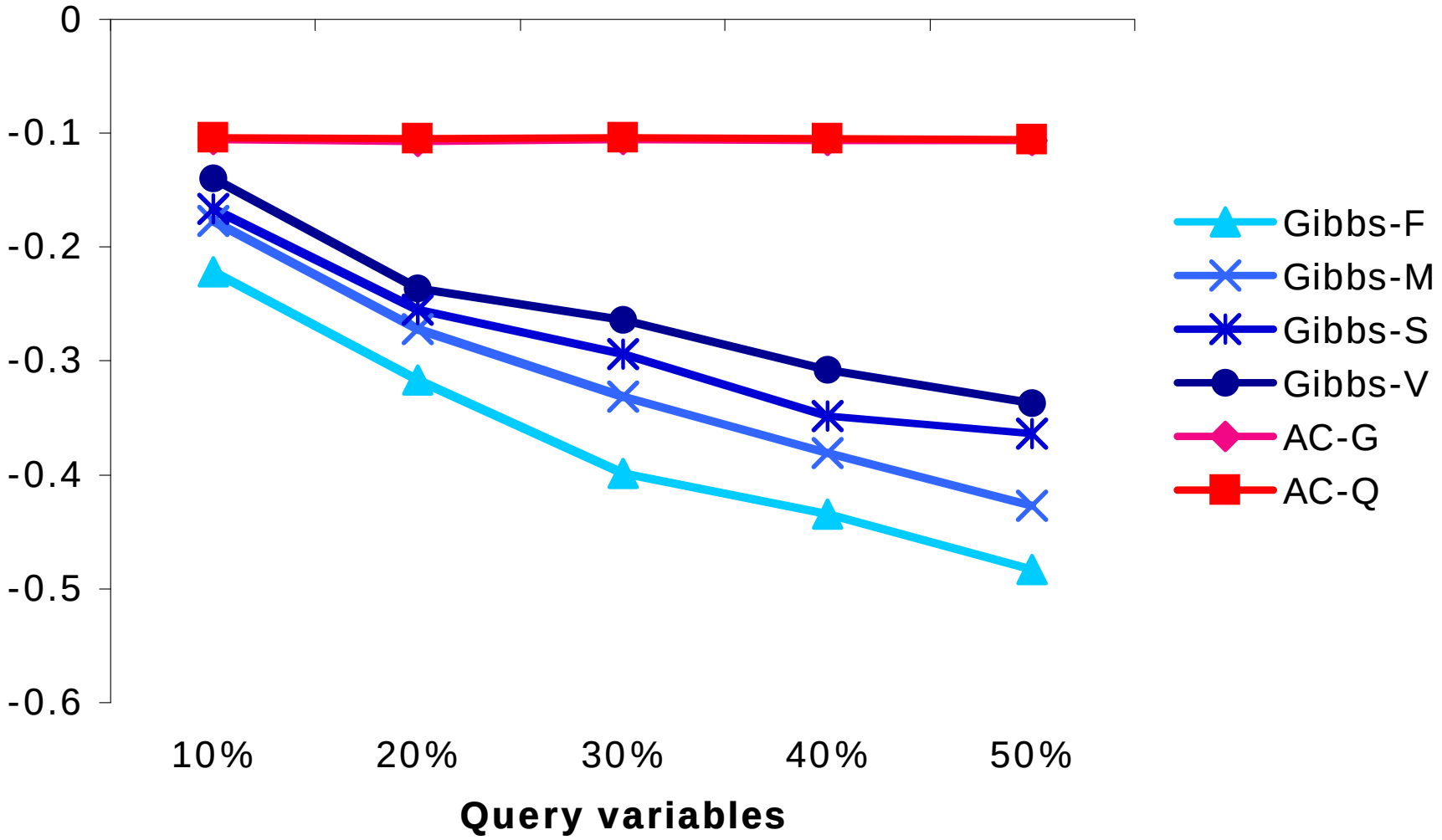
Learning time



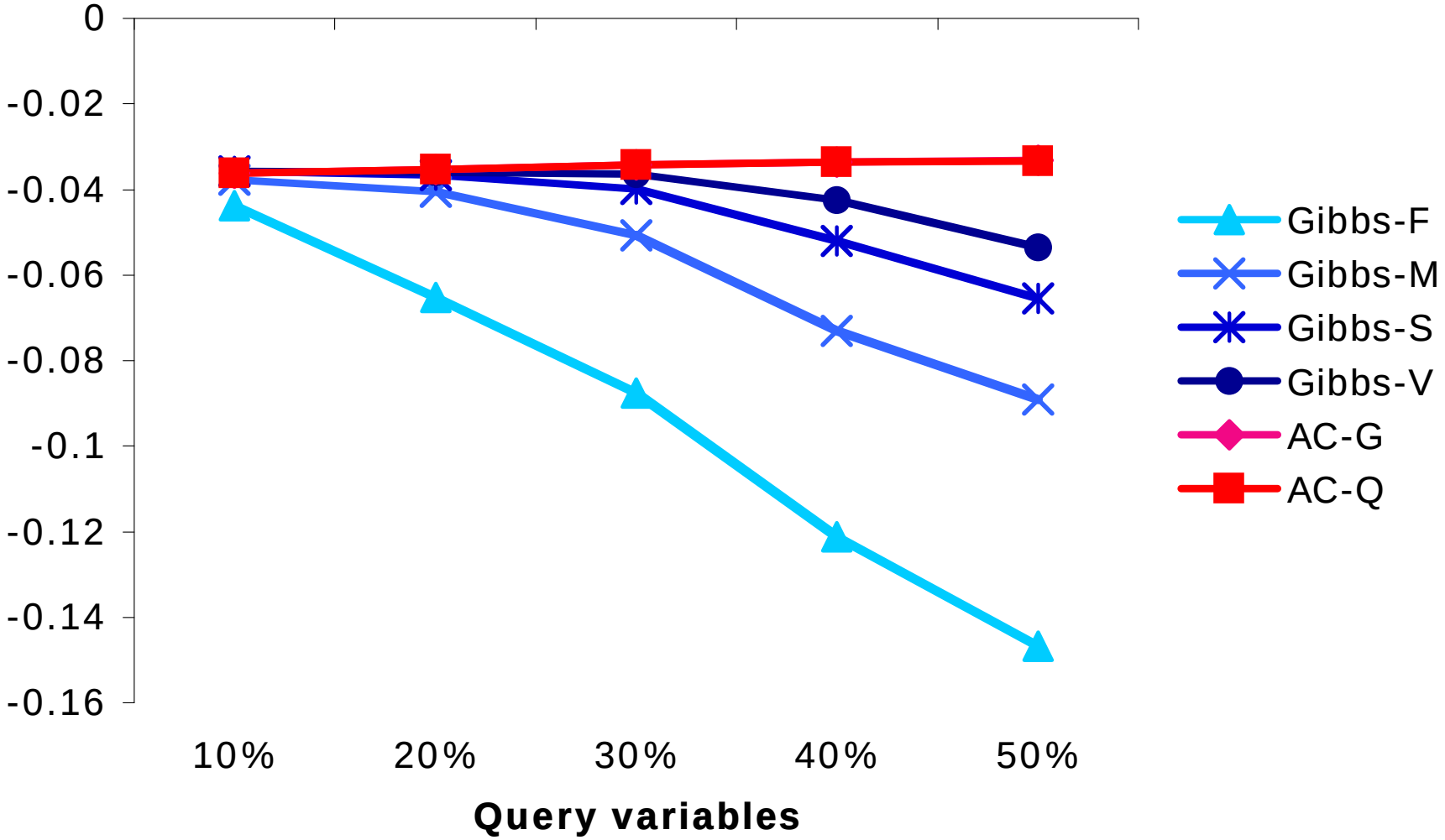
Inference time



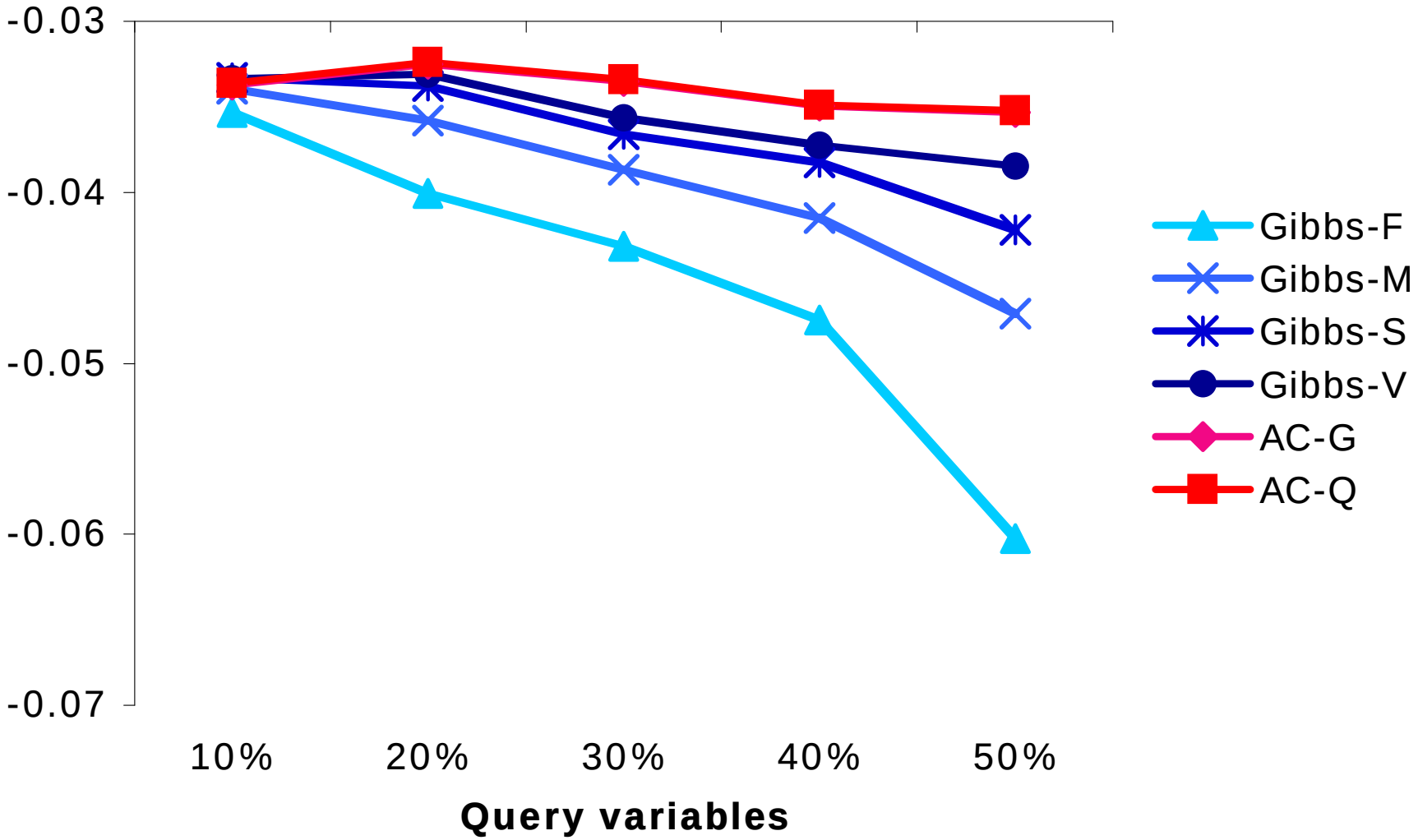
Accuracy: EachMovie



Accuracy: MSWeb



Accuracy: KDD Cup



Conclusion

- **Problem:** Learning accurate intractable models = Learning inaccurate models
- **Solution:** Use inference complexity as learning bias
- **Algorithm:** Learn arithmetic circuits with penalty on circuit size
- **Result:** Much faster and more accurate inference than standard Bayes net learning

Algorithm	EachMovie	KDD Cup	MSWeb
AC-Greedy	62ms	194ms	91ms
AC-Quick	162ms	198ms	115ms
Gibbs: 1k	7.22s	1.46s	1.89s
Gibbs: 10k	42.5s	11.3s	15.6s
Gibbs: 100k	452s	106s	154s
Gibbs: 1M	3912s	1124s	1556s

EachMovie	AC-Greedy	AC-Quick	WinMine
Log-likelih.	–55.7	–54.9	–53.7
Edges	155k	372k	
Leaves	4070	6521	4830
Treewidth	35	54	281
Time	>72h	22h	3m

KDD Cup	AC-Greedy	AC-Quick	WinMine
Log-likelih.	–2.16	–2.16	–2.16
Edges	382k	365k	
Leaves	4574	4463	2267
Treewidth	38	38	53
Time	50h	3h	3m

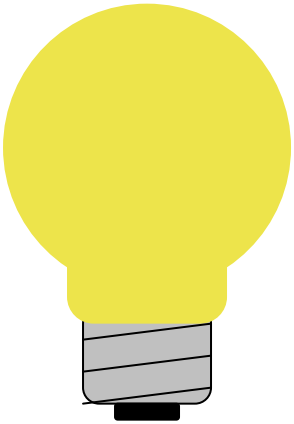
MSWeb	AC-Greedy	AC-Quick	WinMine
Log-likelih.	–9.85	–9.85	–9.69
Edges	204k	256k	
Leaves	1353	1870	1710
Treewidth	114	127	118
Time	8h	3h	2m

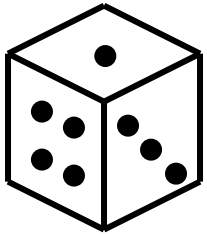
Compiling Bayes nets

- Exact inference: Jointree algorithm
- AC may be more efficient because it can exploit local structure, including determinism and context-specific independencies

Context-specific independence

- Can represent many distributions more efficiently with local structure
- We will focus on Bayesian networks where the conditional probability distributions (CPDs) are represented by decision trees
- This allows compact distributions even for nodes with many parents
- ACs can exploit local structure






SLOW