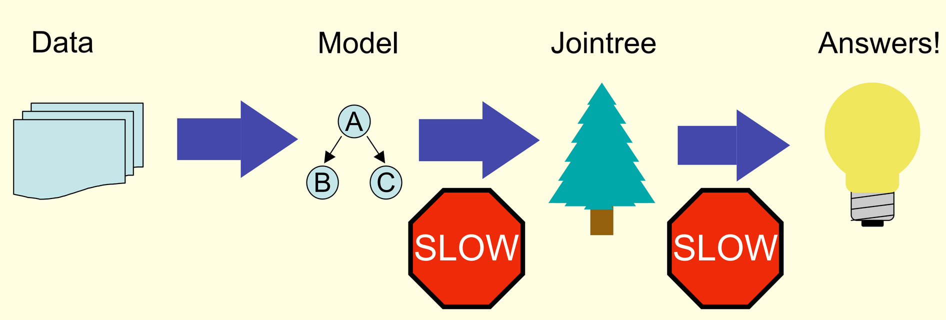
Daniel Lowd
University of Washington
<lowd@cs.washington.edu>

Learning Arithmetic Circuits

Pedro Domingos
University of Washington
<pedrod@cs.washington.edu>

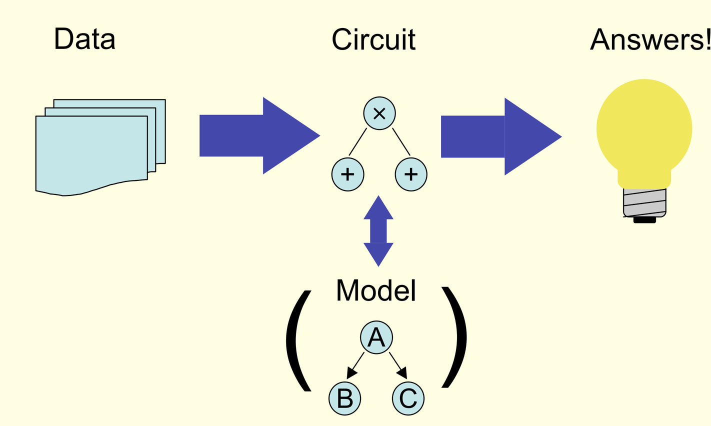
KEY IDEA: Prefer models that allow for more efficient inference

Traditional: Bayesian network structure learning often selects models for which inference is intractable.



Our new approach:

- Apply standard structure learning algorithm but penalize models with high inference cost.
- Represent the distribution more compactly using arithmetic circuits and context-specific independence.

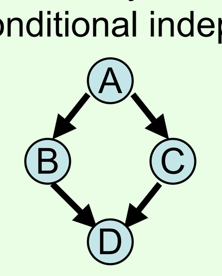


Now we can learn complex models that allow exact inference in milliseconds!

BACKGROUND: From Bayesian networks to arithmetic circuits

Bayesian networks…

Problem: Compactly represent probability distribution over many variables
Solution: Conditional independence

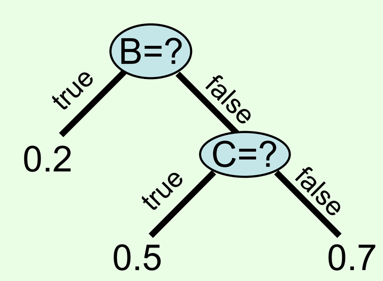


P(A,B,C,D) = P(A) P(B|A) P(C|A) P(D|B,C)

with decision-tree CPDs…

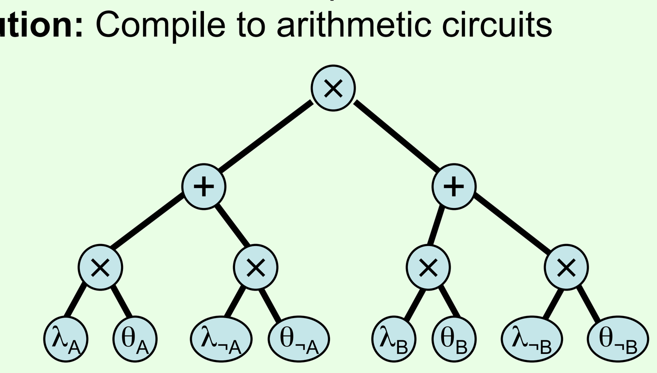
Problem: Number of parameters is exponential in the maximum number of parents
Solution: Context-specific independence

P(D|B,C) =



compiled to circuits.

Problem: Inference is exponential in tree-width
Solution: Compile to arithmetic circuits



Details: 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.
- ACs can take advantage of local structure such as context-specific independence.

ALGORITHM: Struct. learning + Circuit size penalty + Incremental compilation

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

Objective function

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

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

Inference time is linear in circuit size, so this penalizes models with slow inference. Each split effects a constant change in model accuracy and number of parameters. The change in circuit size depends on circuit structure and may increase or decrease as other splits are applied.

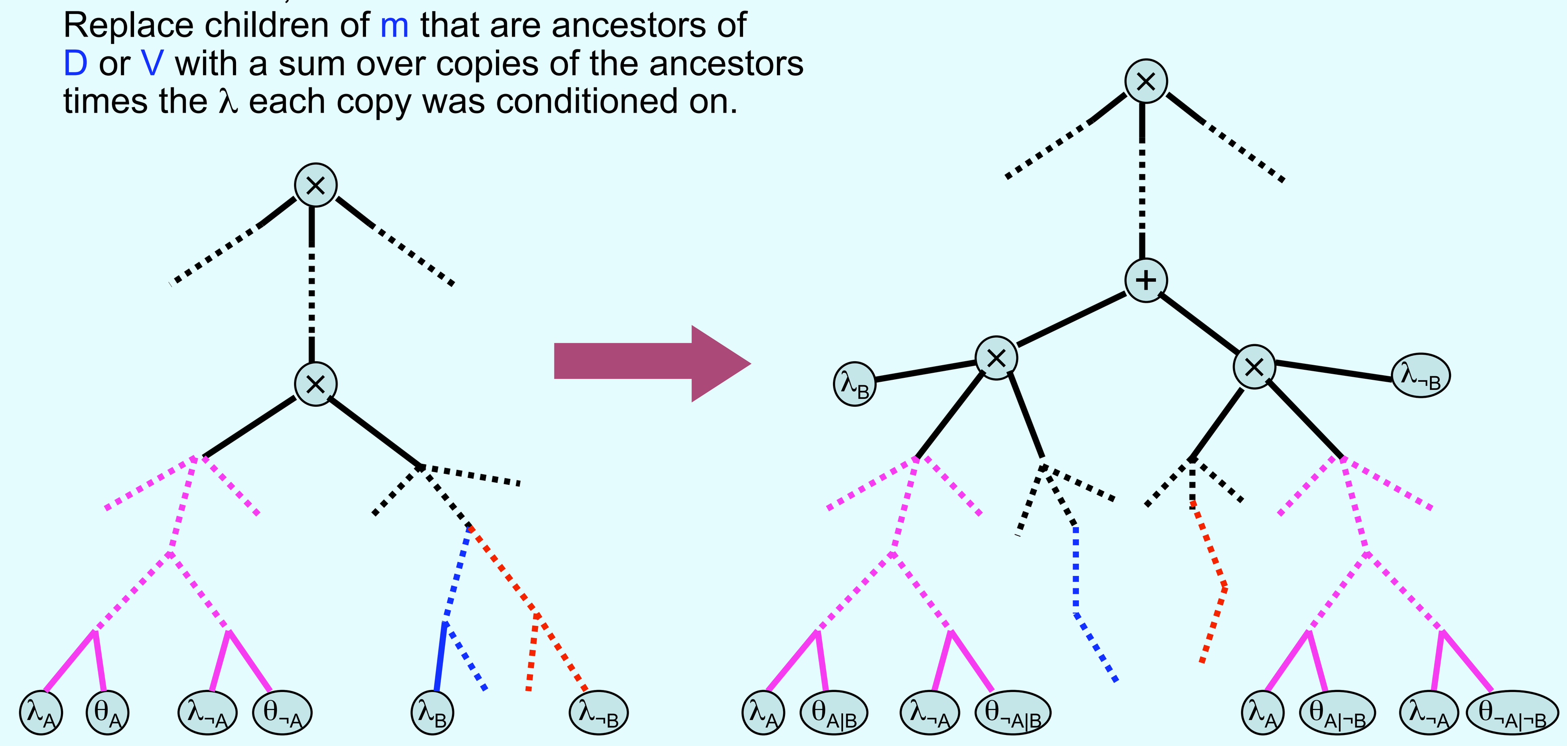
Efficiency

Compiling each candidate AC from scratch at each step is too expensive. Instead, we incrementally modify the circuit as we add splits.

How to split a circuit

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.

D: parameter nodes to be split
V: indicators for the splitting variable
M: first mutual ancestors of D and V



Pseudocode

```
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
```

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. Reevaluate 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 progress. (AC-Quick)

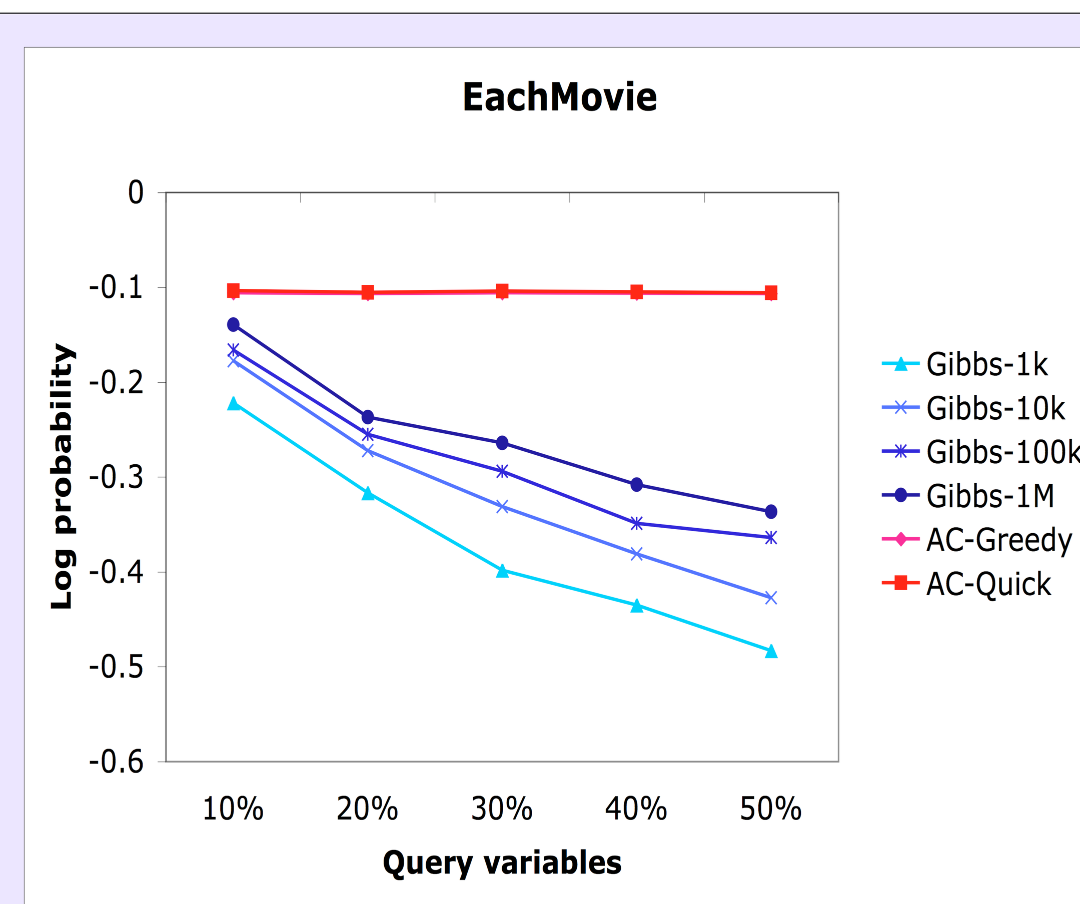
EXPERIMENTS: Better accuracy, >10,000 times faster inference

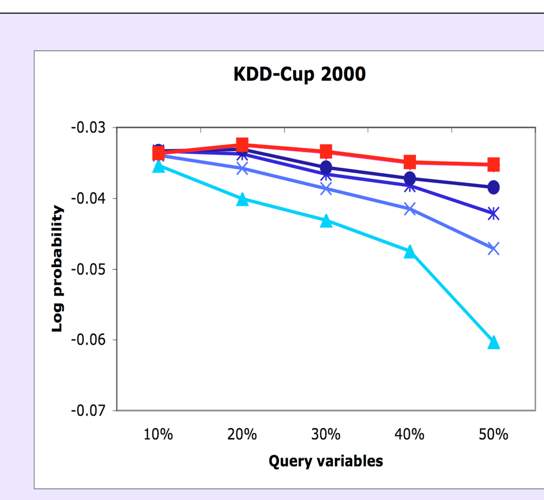
We applied our algorithms (AC-Greedy, AC-Quick) to three real-world datasets, using the WinMine Toolkit as the baseline. WinMine's algorithm is very similar to that of Chickering et al. (1996).

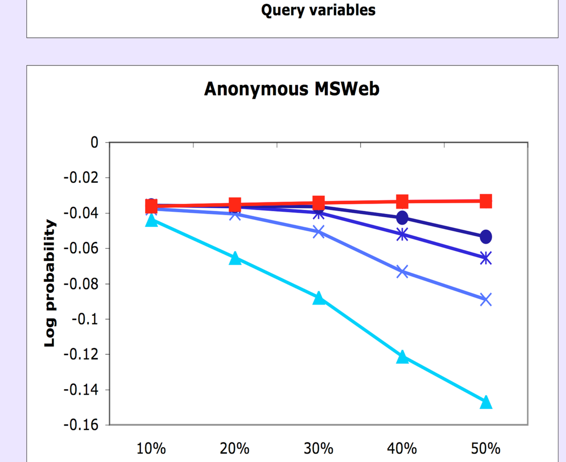
For inference, we generated queries from the test data with varying numbers of evidence and query variables. We used Gibbs sampling on the WinMine models since exact inference was not feasible.

Results: Inference time

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







Results: Learned Models

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