Approximate Inference via Compilation to Arithmetic Circuits

Work by Daniel Lowd and Pedro Domingos

Overview

- Arithmetic circuits (ACs) allow for exact inference in networks with high treewidth by exploiting context-specific independence and determinism
- This work introduces *approximate* inference methods using ACs

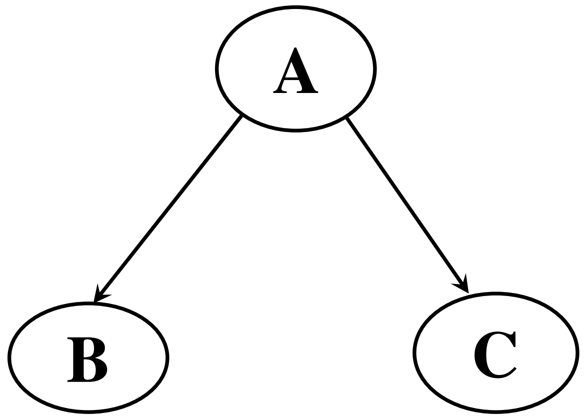
Network Polynomial

- Defines a Bayesian network as a multi-linear function
- λ_x - Evidence indicators: used to select the a value
- $\theta_{x|\mathbf{u}}$ - Network parameters: parameters from the CPTs.

$$f = \sum_{\mathbf{x}} \prod_{x\mathbf{u} \sim \mathbf{x}} \lambda_x \theta_{x|\mathbf{u}}$$

Arithmetic Circuit

- Rooted, directed acyclic graph
- Leaves - numeric constants/variables
- Interior nodes - Addition and multiplication operations

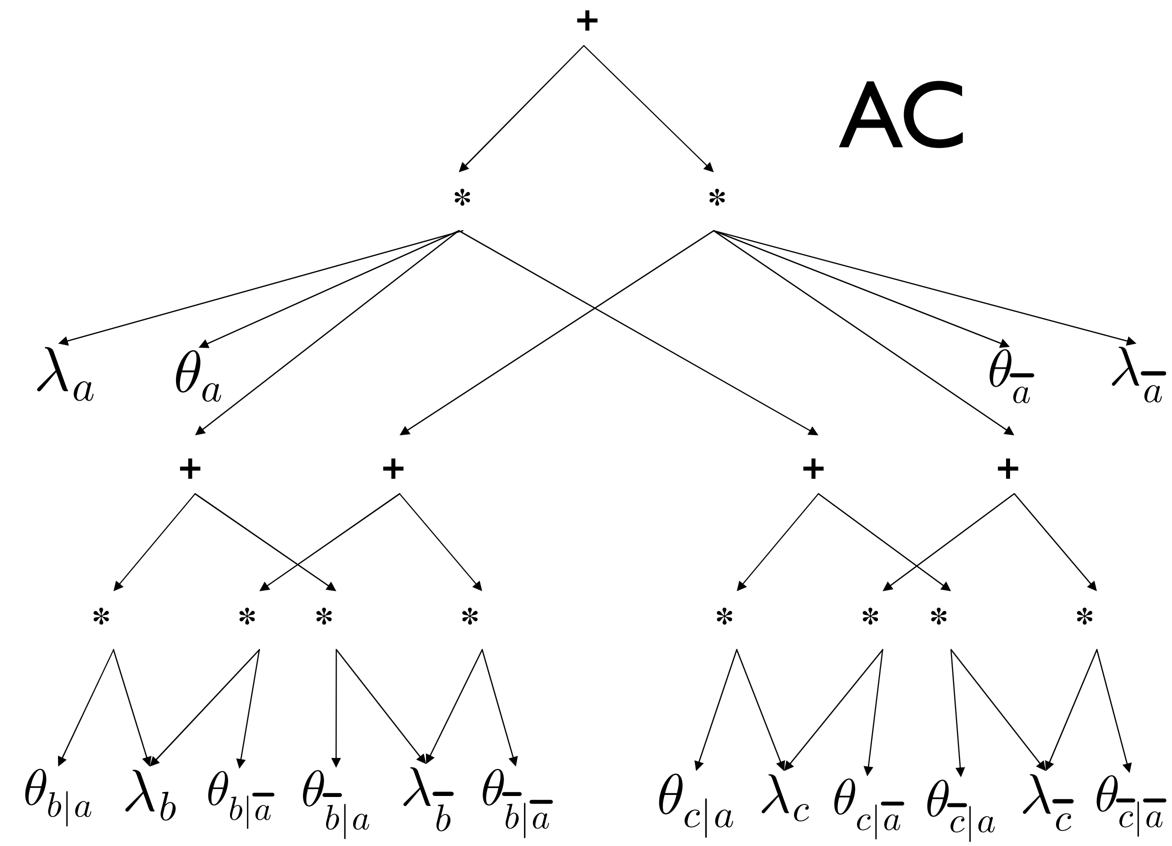


A	θ_A
true	.5
false	.5

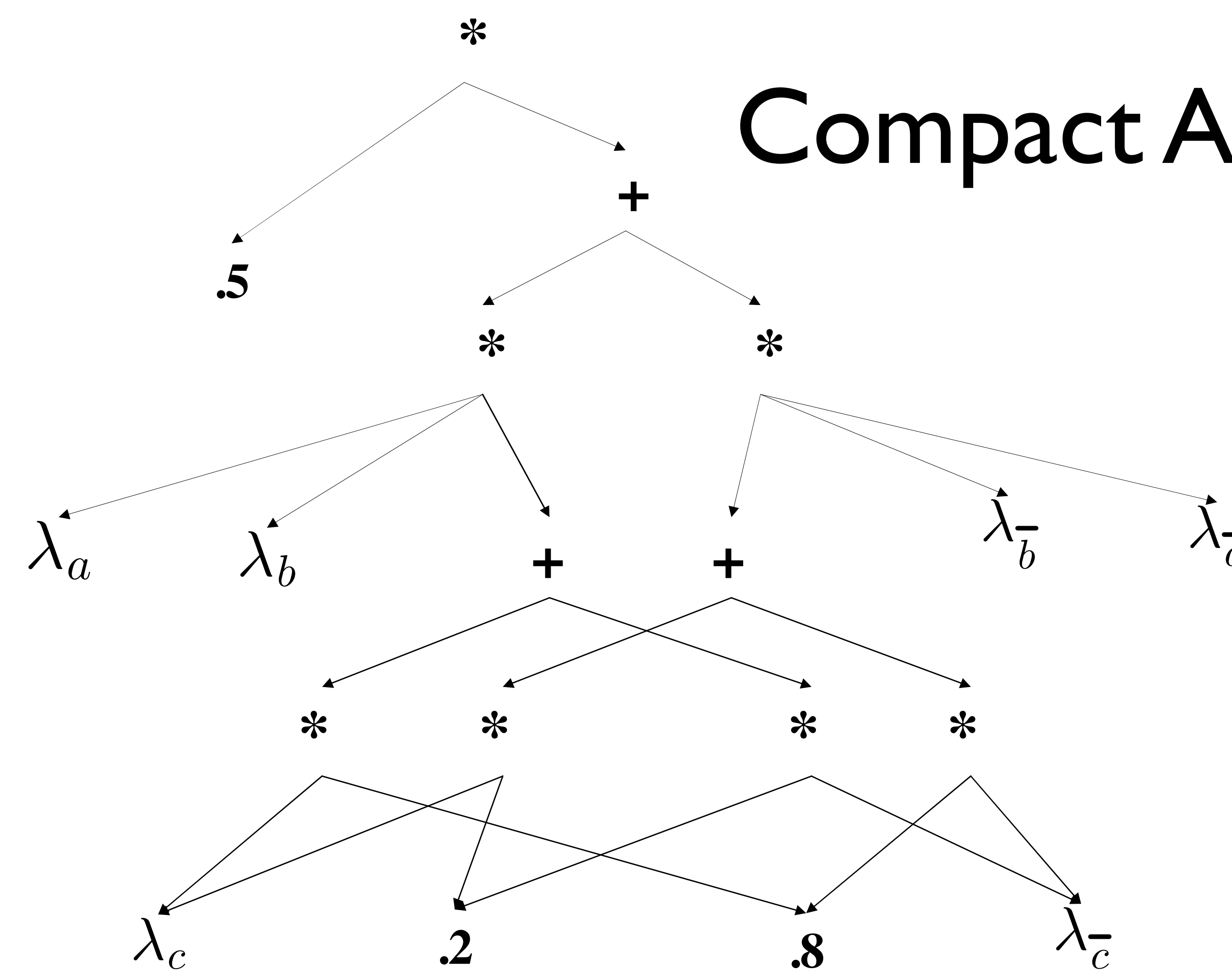
A	B	$\theta_{B\|A}$
true	true	1
true	false	0
false	true	0
false	false	1

A	C	$\theta_{C\|A}$
true	true	.8
true	false	.2
false	true	.2
false	false	.8

AC



Compact AC



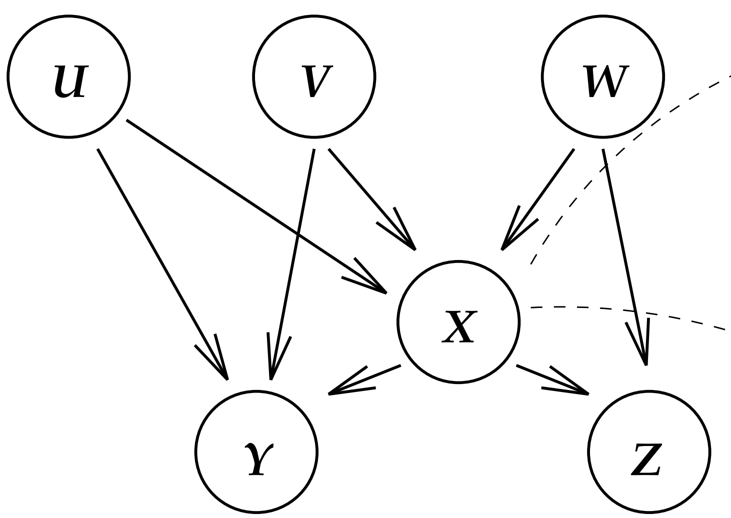
Network polynomial

$$
\begin{aligned}
f \;=\; & \lambda_a \lambda_b \lambda_c \theta_a \theta_{b|a} \theta_{c|a} + \\
& \lambda_a \lambda_b \lambda_{\bar{c}} \theta_a \theta_{b|a} \theta_{\bar{c}|a} + \\
& \vdots \\
& \lambda_{\bar{a}} \lambda_{\bar{b}} \lambda_{\bar{c}} \theta_{\bar{a}} \theta_{\bar{b}|\bar{a}} \theta_{\bar{c}|\bar{a}}.
\end{aligned}
$$

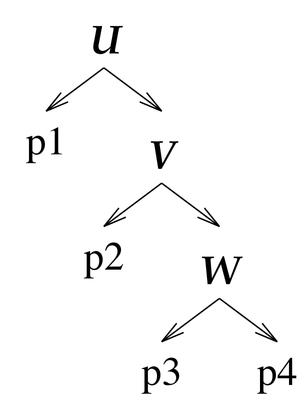
(Figures from Darwiche 2003)

Context-Specific Independence

- Take advantage of *context-specific independencies*
  - Variables may be independent given a certain instantiation
- Use a *decision-tree CPD* representation
- Arithmetic circuits can also capture these independencies



U	V	W	$P(x)$
t	t	t	p1
t	t	f	p1
t	f	t	p1
t	f	f	p1
f	t	t	p2
f	t	f	p2
f	f	t	p3
f	f	f	p4



(Figure from Boutilier et al. 1996)

Log-Linear Model Representation

- f_i - feature: represents an assignment i to a configuration
- w_i - weight: represents the value corresponding to assignment i

$$\log P(X = x) = -\log Z + \sum_i w_i f_i(x)$$

KL Divergence

- Average log difference between two distributions
- Q is typically the approximate distribution, while P is the true distribution
- In variational methods, the goal is to find a tractable distribution Q that minimizes this

$$\mathrm{KL}(Q \,\|\, P) = \sum_x Q(x) \log \frac{Q(x)}{P(x)} = -H_Q(x) - \sum_i w_i E_Q[f_i] + \log Z_P$$

Approximate Compilation

- Main idea: Compile an AC that approximates the true BN and perform using that AC
- Two stages
  - Structure search (Performed once)
  - Parameter optimization (Fine tunes the circuit to specific evidence)

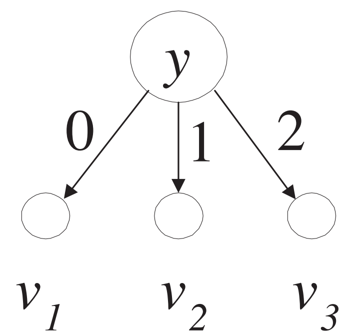
Structure Search

- Two methods covered in this work
  - Prune and compile
  - Learn from samples

Decision Tree CPD Structure Learning

- Split operators
  - Complete - C(v, π) add leaf nodes as children to v, where each leaf corresponds to an assignment of π
  - Binary - B(v, π, k) adds 2 leaf nodes as children to v, where one leaf corresponds to assignment k of π and the other for all other assignments
  - Merge - M(v_1,v_2) merges 2 leaf nodes; resulting node inherits all parents from both.

Decision Tree CPD Structure Learning

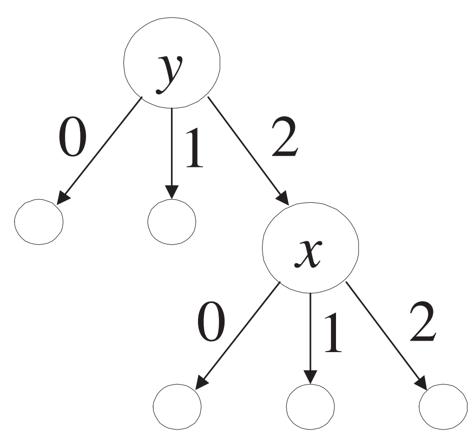


(a)

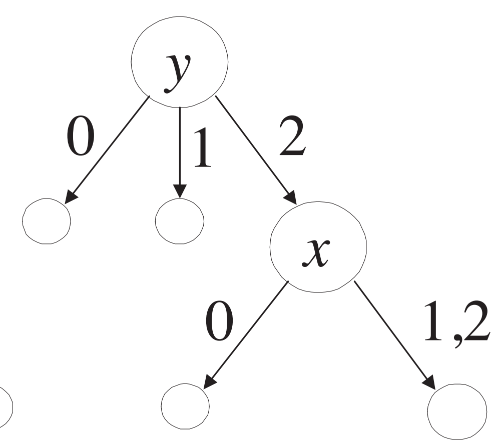
$v_1 = P(z \mid y=0)$

$v_2 = P(z \mid y=1)$

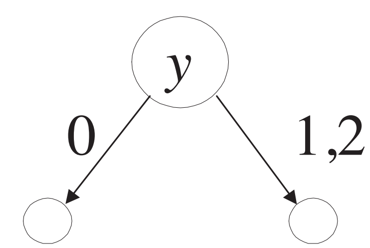
$v_3 = P(z \mid y=2)$



(b)



(c)



(d)

$C(v_3, x)$ $B(v_3, x, 0)$ $M(v_2, v_3)$

Prune and Compile

- Goal: Find a simplified network Q from pruning splits in the decision tree CPD
- Q's structure is a subset of P's - can decompose KL divergence
- Need to compute parent distributions $P(\pi_i)$
  - Intractable
  - Need to approximate
    - P-Samp: Estimate joint distributions with samples
    - P-MF: Mean field

$$\mathrm{KL}(P \,\|\, Q) = \sum_i \sum_{\pi_i} P(\pi_i) \sum_{x_i} P(x_i|\pi_i) \log \frac{P(x_i|\pi_i)}{Q(x_i|\pi_i)}$$

Prune and Compile

- Algorithm
  - Initialize with fully pruned network
  - Add in splits greedily that best decrease the KL divergence
  - Every 10 splits, compile the network to an AC check the number of edges
    - Stop if this value exceeds a pre-specified bound

Learning from Samples

- Generate samples with forward sampling
- Use samples as data for learning AC structure directly

Learning from Samples

- The LearnAC algorithm (Lowd and Domingos, UAI 2008)
  - Initialize circuit as a product of marginals (a BN with no edges)
  - Apply AC splits that add the fewest edges while increasing a score function
    - The score function computes the log-likelihood of the training data, with penalties on the number of edges and parameters
    - Modified in this work to stop when a size limit is reached

Parameter Optimization

- Three methods
  - Forward sampling
  - Variational optimization
  - Gibbs sampling

AC^2-F

- Generate a set of samples from the original BN
- Use maximum likelihood estimation to set AC parameters
- Can be viewed as approximately minimizing KL(P||Q)
- For conditional queries, we are more interested in the divergence of the conditional distributions, KL(P(. | x_{ev})||Q(. | x_{ev})
    - Expected to perform more poorly on rare evidence (see bound below)

$$KL(P(.|x_{ev}) \,\|\, Q(.|x_{ev})) \leq \frac{1}{P(x_{ev})} KL(P \,\|\, Q)$$

AC^2-V

- Try to address the problem when conditioning on poor evidence
- Minimize KL(Q||P) where P is the original BN conditioned on evidence

AC²-V

- Representing P and Q as log-linear models...

$$\log P(x) = -\log Z_P + \sum_i w_i f_i(x) \qquad \log Q(x) = -\log Z_Q + \sum_j v_j g_j(x)$$

- KL divergence and gradient

$$\mathrm{KL}(Q \,\|\, P) = \sum_j v_j E_Q(g_j) - \sum_i w_i E_Q(f_i) + \log \frac{Z_P}{Z_Q}$$

$$\frac{\partial}{\partial v_j} \mathrm{KL}(Q \,\|\, P) = \sum_k v_k (E_Q(g_k g_j) - Q(g_k) Q(g_j)) - \sum_i v_i (E_Q(f_i g_j) - Q(f_i) Q(g_j))$$

AC^2-G

- Minimize KL(P||Q), since expectations over Q may assign small or zero probabilities to important modes of P
- Approximate the expectations of P with Gibbs sampling

$$\mathrm{KL}(P \| Q) = \sum_i w_i E_P(f_i) - \sum_j v_j E_P(g_j) + \log Z_Q / Z_P$$

$$\frac{\partial}{\partial v_j}\mathrm{KL}(P \| Q) = E_Q(g_j) - E_P(g_j)$$

Experiments

- Structure Selection
  - Compute KL divergences (used random samples)
    - Prune and compile (P-Samp and P-MF)
    - Chow-Liu Trees
    - LearnAC
  - Parameters set using AC^2-F
  - Training times ranged from 17 minutes to 8 hours

$$D(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} = E_P[\log(P(x)/Q(x))] \approx \frac{1}{m} \sum_i \log(P(x^{(i)})/Q(x^{(i)}))$$

Experiments

	P-MF	P-Samp	C-L	LAC
KDD Cup	2.44	0.10	0.23	0.07
Plants	8.41	2.29	4.48	1.27
Audio	4.99	3.31	4.47	2.12
Jester	5.14	3.55	5.08	2.82
Netflix	3.83	3.06	4.14	2.24
MSWeb	1.78	0.52	0.70	0.38
Book	4.90	2.43	2.84	1.89
EachMovie	29.66	17.61	17.11	11.12

Experiments

- Generated 100 random samples
- Select random subset of variables to use as evidence
  - Generate additional samples (up to 100,000) and pick out ones consistent with the evidence in the 100 samples.
- Measure the log conditional probability of non-evidence variables of the samples
- Serves as an approximation of the KL divergence between the true and inferred conditional distributions

Experiments

- Mean Query Times

	AC^2-F	AC^2-V	AC^2-G	BP	MF	Gibbs
KDD Cup	0.022	3803	11.2	0.050	0.025	2.5
Plants	0.022	2741	11.2	0.081	0.073	2.8
Audio	0.023	4184	14.4	0.063	0.048	3.4
Jester	0.019	3448	13.8	0.054	0.057	3.3
Netflix	0.021	3050	12.3	0.057	0.053	3.3
MSWeb	0.022	2831	12.2	0.277	0.046	4.3
Book	0.020	5190	16.1	0.864	0.059	6.6
EachMovie	0.022	10204	28.6	1.441	0.342	11.0

Experiments

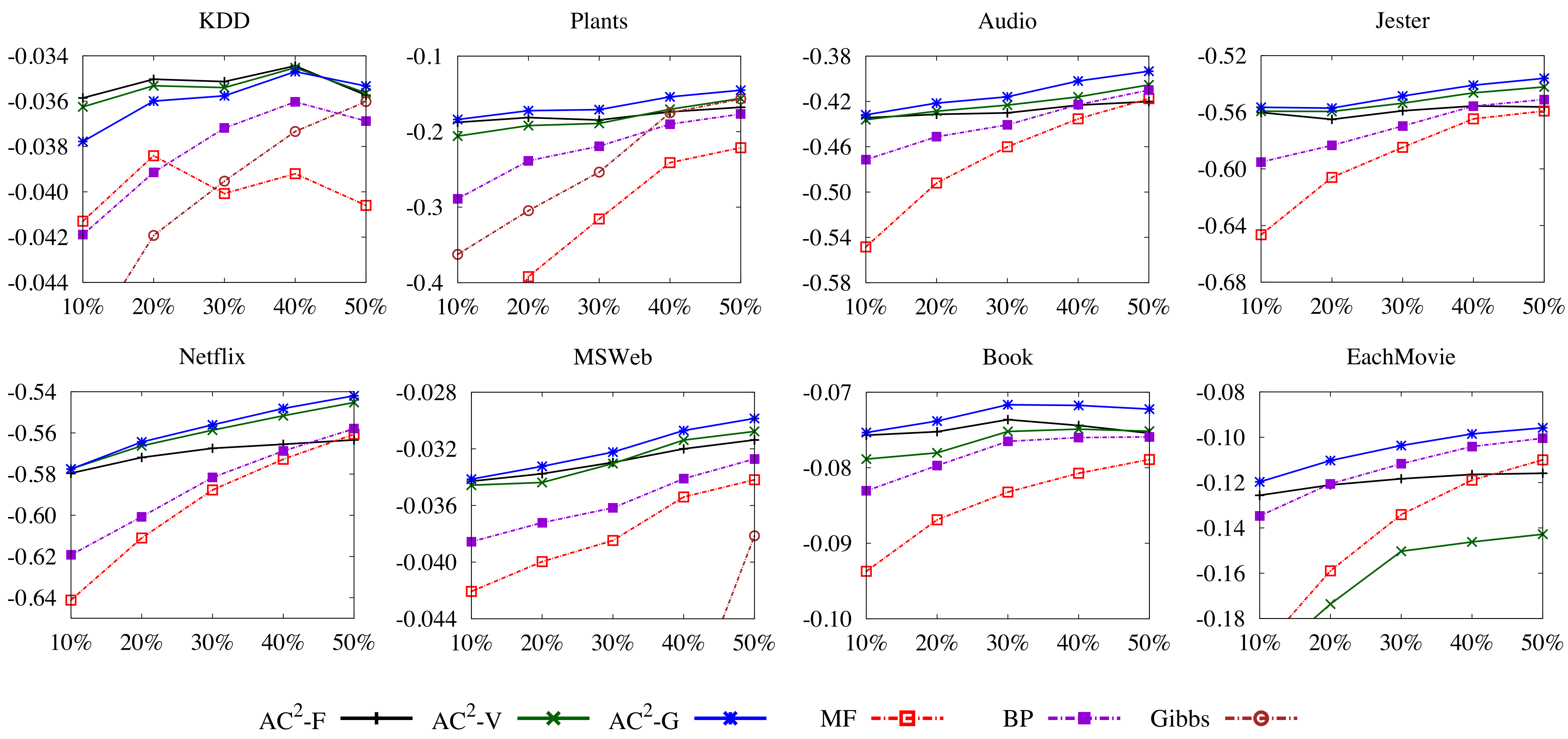


Figure 1: Average conditional log likelihood of the query variables (y axis), divided by the number of query variables (x axis). Higher is better. Gibbs often performs too badly to appear in the frame.

Experiments

- AC^2-F: Good for fast inference on with small to moderate amounts of evidence
- AC^2-V: Better when there is more evidence
  - Reducing KL(Q||P) may increase KL(P||Q)
  - Slower than the other algorithms
- AC^2-G: Most accurate
  - Dominates BP, MF, and Gibbs on the datasets
  - Takes longer than Gibbs due to parameter optimization step and computations of expectations

Conclusion

- ACs are alternatives to junction trees
  - Exploits determinism and context-specific independence
- Combining sampling and learning is a good strategy for accurate approximate inference
  - sampling - get a coarse approximation
  - learning - smooth the approximation