UCLA Computer Science

STAR AI RESEARCH LAB UCLA

Computational Abstractions of Probability Distributions

Guy Van den Broeck

PGM - Sep 24, 2020

Manfred Jaeger Tribute Band

1997-2004-2005

Relational Bayesian Networks

Manfred Jaeger*
Computer Science Department,
Stanford University, Stanford CA 94305
jaeger@robotics.stanford.edu

Abstract

A new method is developed to represent probabilistic relations on multiple random events. Where previously knowledge bases containing probabilistic rules were used for this purpose,

In case we ind… about several observed events, e. $burglary(\omega_2)$ = … ing a query $X($… evidence about … $P(X(\omega) = x$ |

In Int. J. of Uncertainty, Fuzziness

Probabilistic Decision Graphs – Combining Verification and AI Techniques for Probabilistic Inference

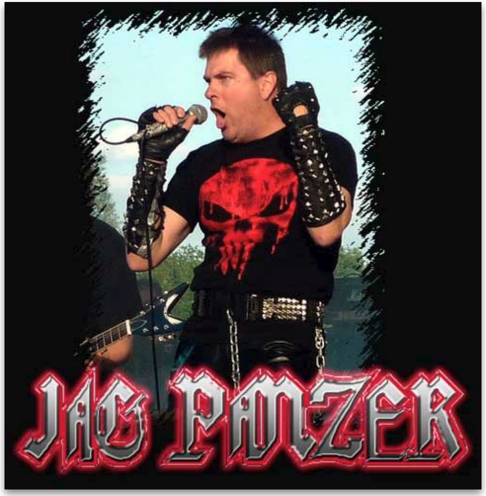
MANFRED JAEGER
Aalborg Universitet, Institut for Datalogi
Fredrik Bajers Vej 7, 9220 Aalborg Ø, Denmark
jaeger@cs.auc.dk

ELSEVIER
Available online at www.sciencedirect.com
SCIENCE DIRECT
International Journal of Approximate Reasoning
42 (2006) 4–20
INTERNATIONAL JOURNAL OF APPROXIMATE REASONING
www.elsevier.com/locate/ijar

Compiling relational Bayesian networks for exact inference

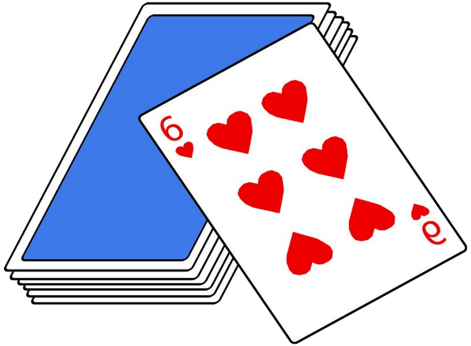
Mark Chavira [a,*], Adnan Darwiche [a], Manfred Jaeger [b]
[a] Computer Science Department, UCLA, Los Angeles, CA 90095, United States
[b] Institut for Datalogi, Aalborg Universitet, Fredrik Bajers Vej 7 E, DK-9220 Aalborg Ø, Denmark

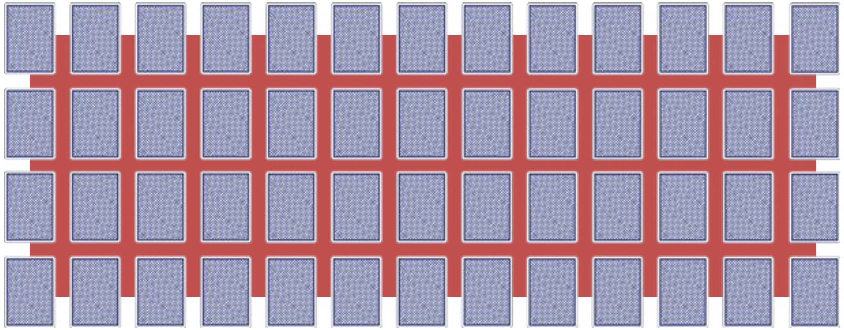
Available online 15 November 2005



Let me be provocative

Graphical models of variable-level (in)dependence are a broken abstraction.



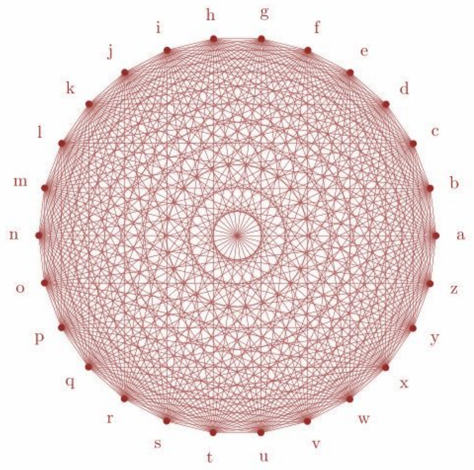


[VdB KRR15]

Let me be provocative

Graphical models of variable-level (in)dependence are a broken abstraction.

3.14 Smokes(x) $\wedge$ Friends(x,y) $\Rightarrow$ Smokes(y)



[VdB KRR15]

Let me be provocative

Graphical models of variable-level (in)dependence are a broken abstraction.

Bean Machine

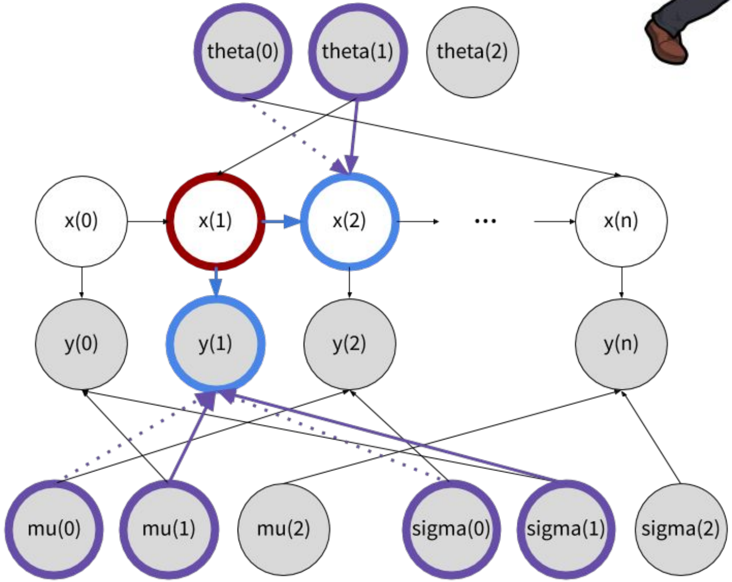
$$\mu_k \sim \text{Normal}(\alpha, \beta)$$

$$\sigma_k \sim \text{Gamma}(\nu, \rho)$$

$$\theta_k \sim \text{Dirichlet}(\kappa)$$

$$x_i \sim \begin{cases} \text{Categorical}(init) & \text{if } i = 0 \\ \text{Categorical}(\theta_{x_{i-1}}) & \text{if } i > 0 \end{cases}$$

$$y_i \sim \text{Normal}(\mu_{x_i}, \sigma_{x_i})$$



[Tehrani et al. PGM20]

Let me be **even more** provocative

Graphical models of variable-level (in)dependence are a broken abstraction.

We may have gotten stuck in a local optimum?

- Exact probabilistic inference still independence-based
  - Huge effort to extract more local structure from individual tables
- *What do you mean, compute probabilities exactly?*
  - Statistician: inference = Hamiltonian Monte Carlo
  - Machine learner: inference = variational
- Variable-level causality

Let me be provocative

Graphical models of variable-level (in)dependence are a broken abstraction.

The choice of representing a distribution primarily by its variable-level (in)dependencies is a little arbitrary…

What if we made some different choices?

Computational Abstractions

Let us think of distributions as objects that are computed.

Abstraction = Structure of Computation

‘closer to the metal’

Two examples:

- Probabilistic Circuits
- Probabilistic Programs

Probabilistic Circuits

Fully factorized

NaiveBayes AndOrGraphs PDGs

Trees PSDDs CNets LTMs SPNs NADEs

Thin Junction Trees ACs MADEs MAFs VAEs

DPPs FVSBNs TACs IAFs NAFs RAEs

Mixtures BNs NICE FGs GANs

RealNVP MNs

The Alphabet Soup of probabilistic models

Fully factorized

NaiveBayes AndOrGraphs PDGs

Trees PSDDs CNets LTMs SPNs NADEs

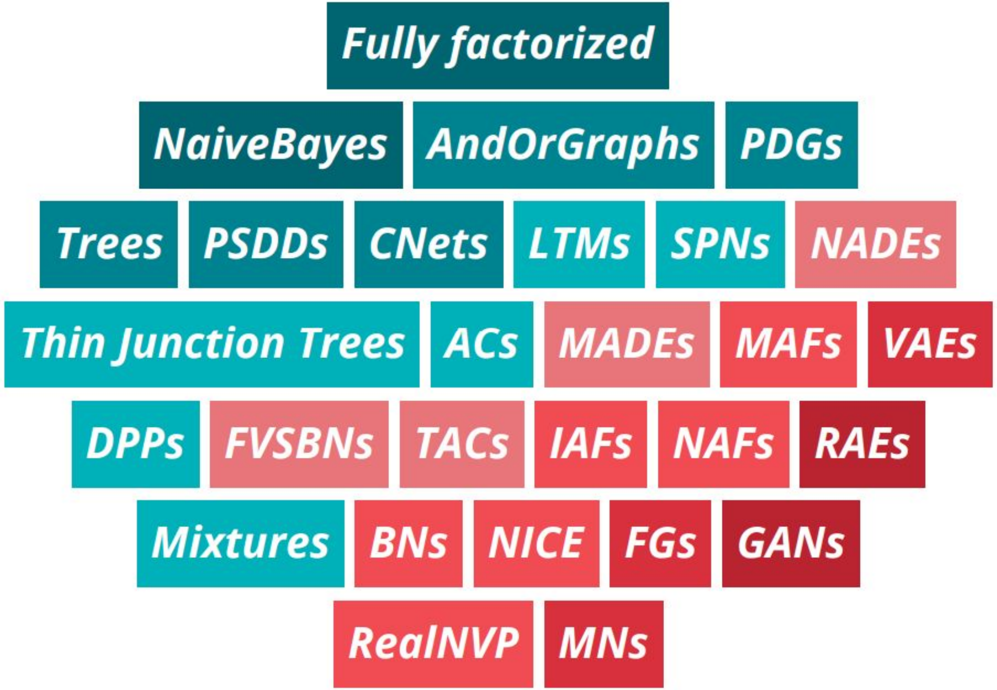
Thin Junction Trees ACs MADEs MAFs VAEs

DPPs FVSBNs TACs IAFs NAFs RAEs

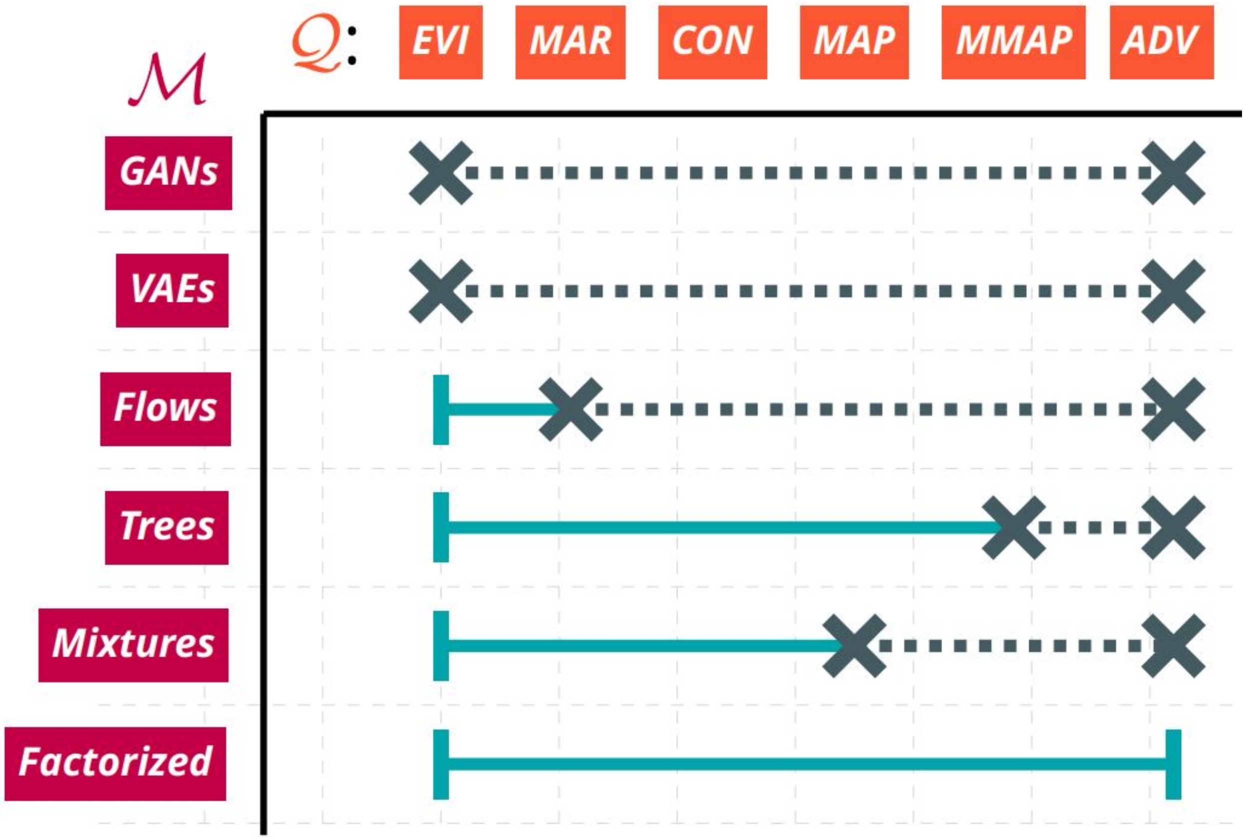
Mixtures BNs NICE FGs GANs

RealNVP MNs

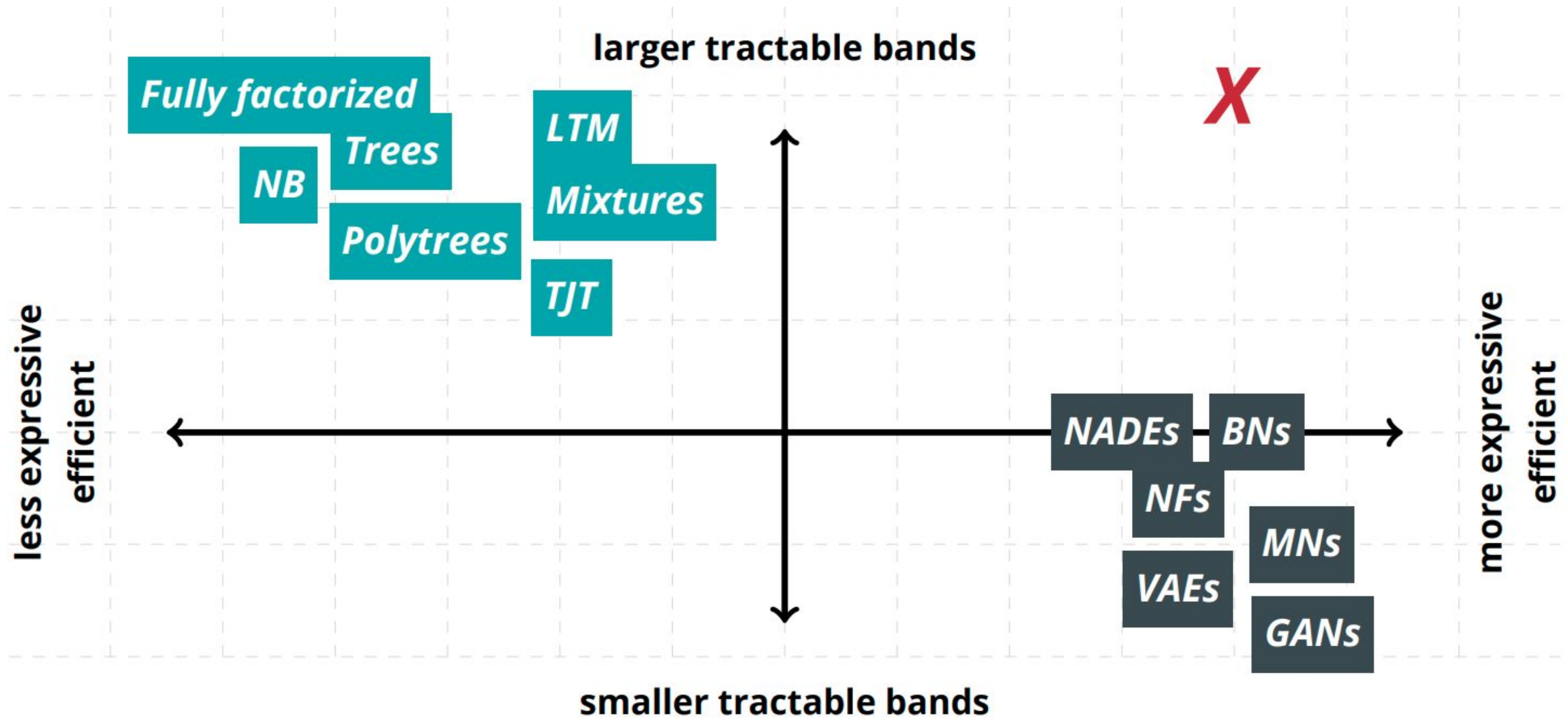
Intractable and *tractable* models

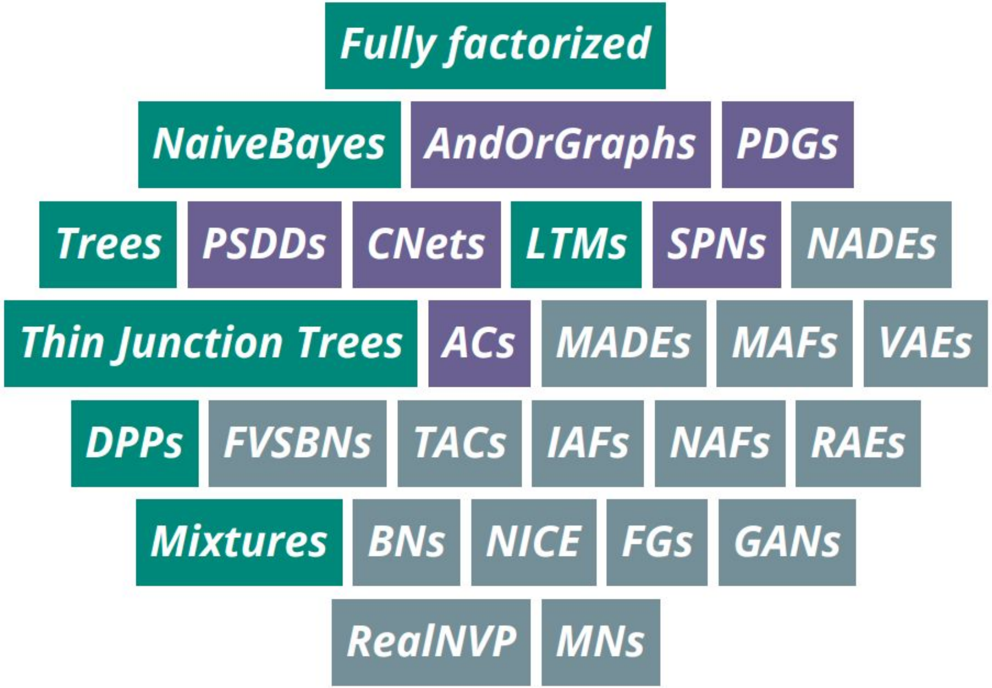


tractability is a spectrum

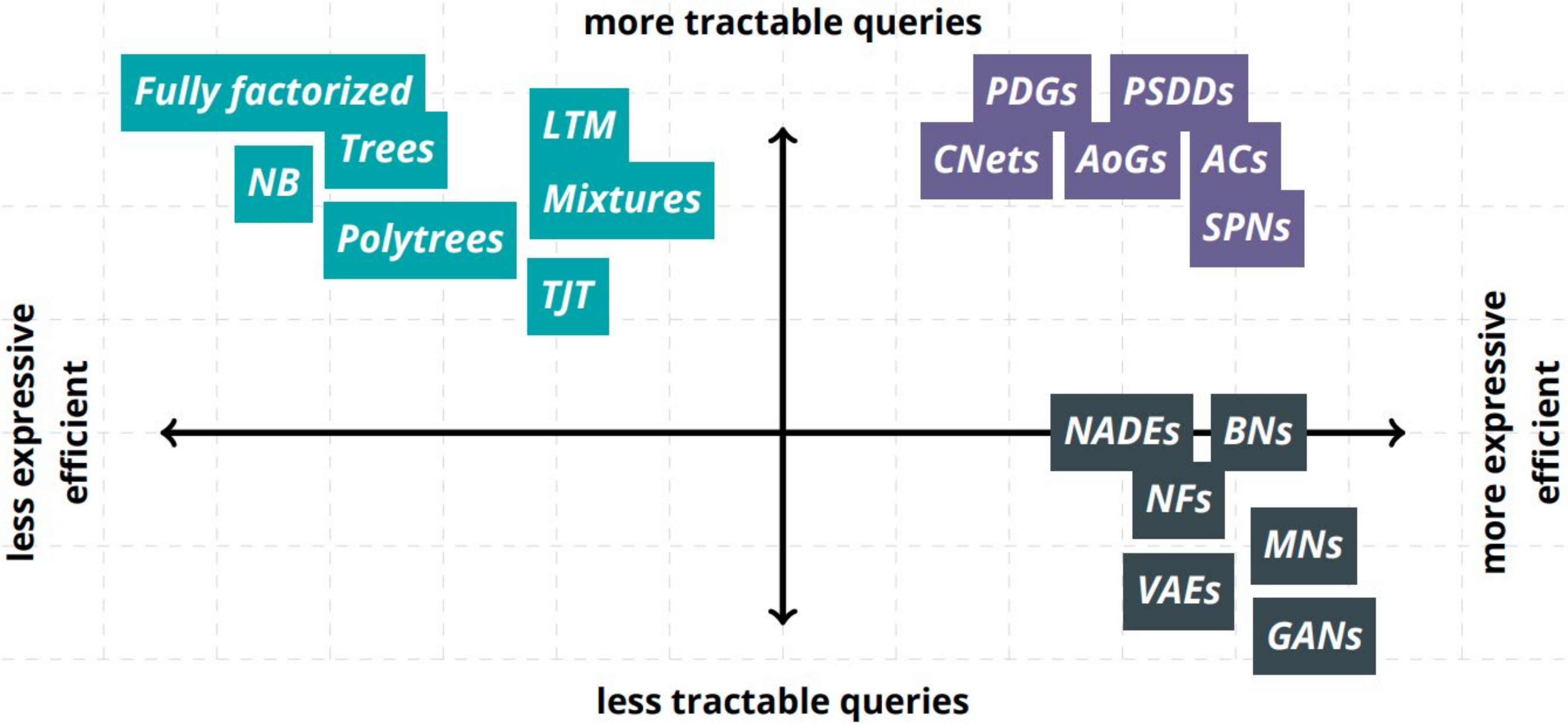


$\mathcal{Q}$:
EVI
MAR
CON
MAP
MMAP
ADV
$\mathcal{M}$
GANs
VAEs
Flows
Trees
Mixtures
Factorized

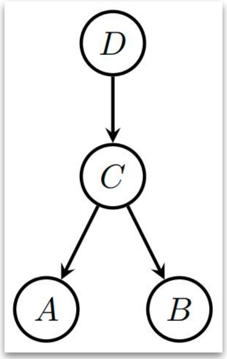
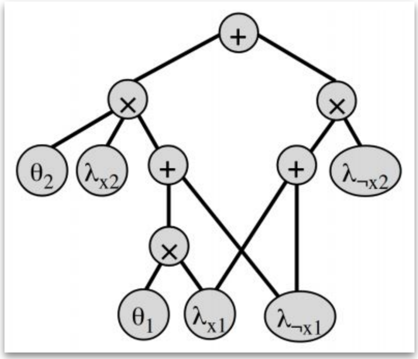
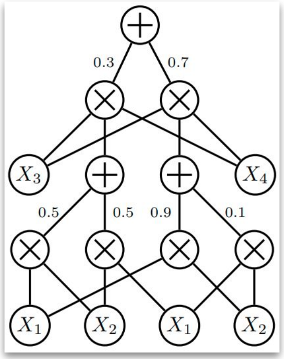
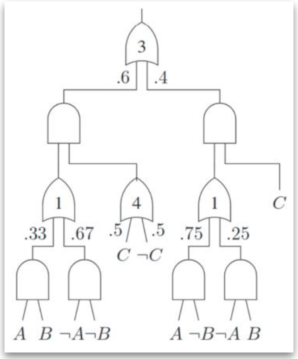
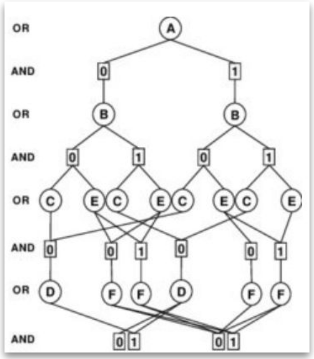
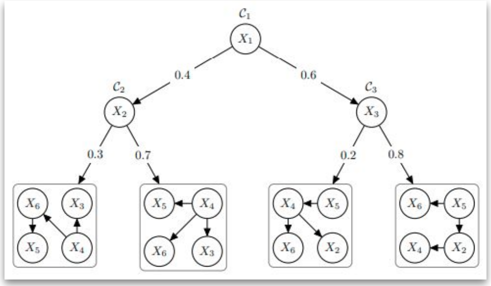
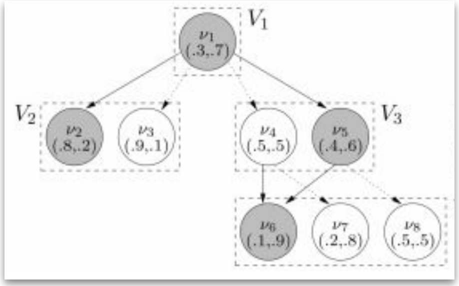
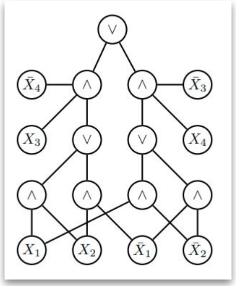
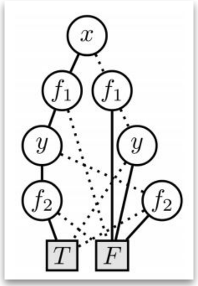
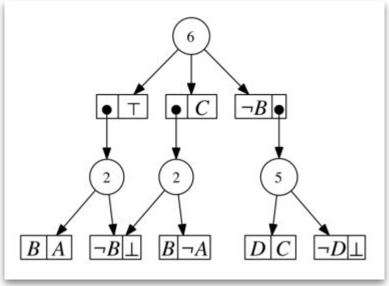

larger tractable bands
X
Fully factorized
Trees
NB
Polytrees
LTM
Mixtures
TJT
less expressive
efficient
more expressive
efficient
NADEs
BNs
NFs
MNs
VAEs
GANs
smaller tractable bands



Expressive models without *compromises*


more tractable queries
Fully factorized
Trees
NB
Polytrees
LTM
Mixtures
TJT
PDGs
PSDDs
CNets
AoGs
ACs
SPNs
less expressive
efficient
more expressive
efficient
NADEs
BNs
NFs
MNs
VAEs
GANs
less tractable queries

Tractable Probabilistic Models

"*Every keynote needs a joke and a literature overview slide, not necessarily distinct*"

- after Ron Graham

Fully factorized

NaiveBayes AndOrGraphs PDGs

Trees PSDDs CNets LTMs SPNs NADEs

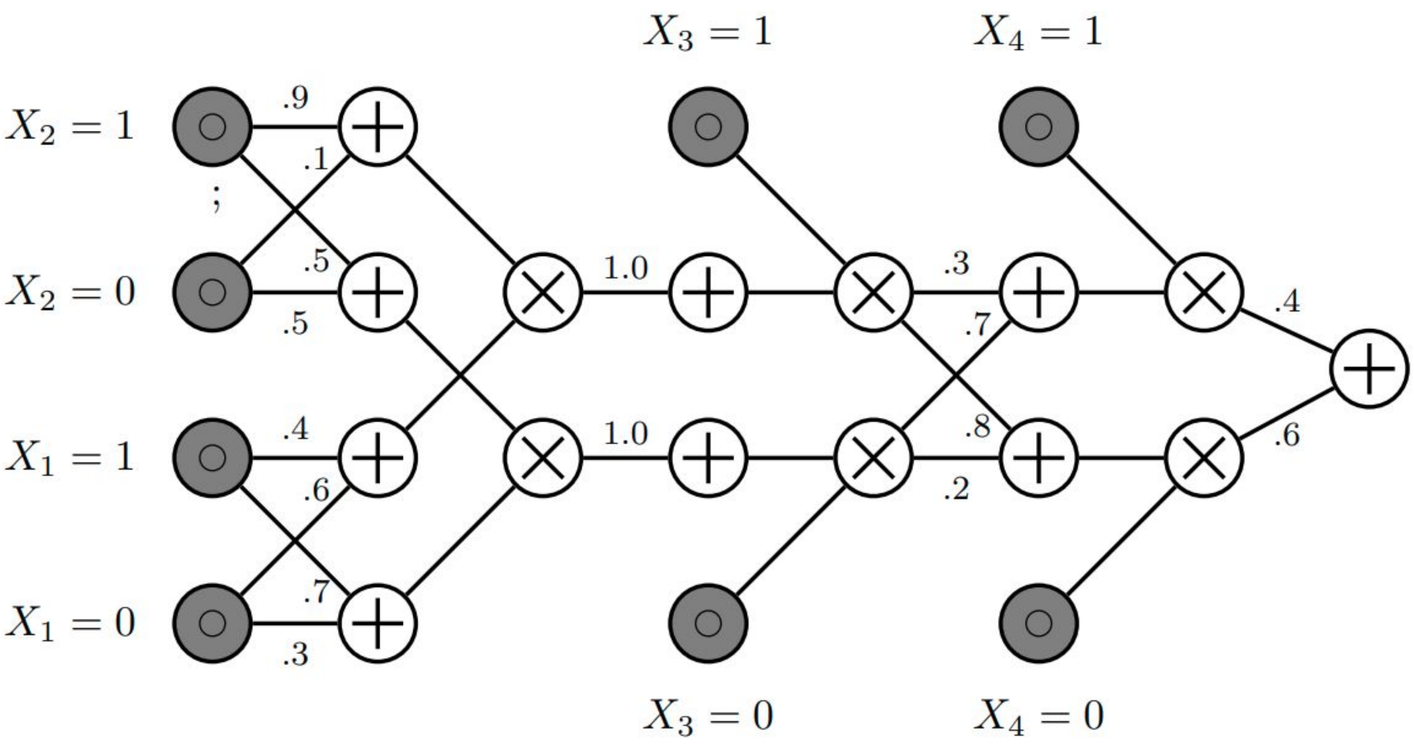
Thin Junction Trees ACs MADEs MAFs VAEs

DPPs FVSBNs TACs IAFs NAFs RAEs

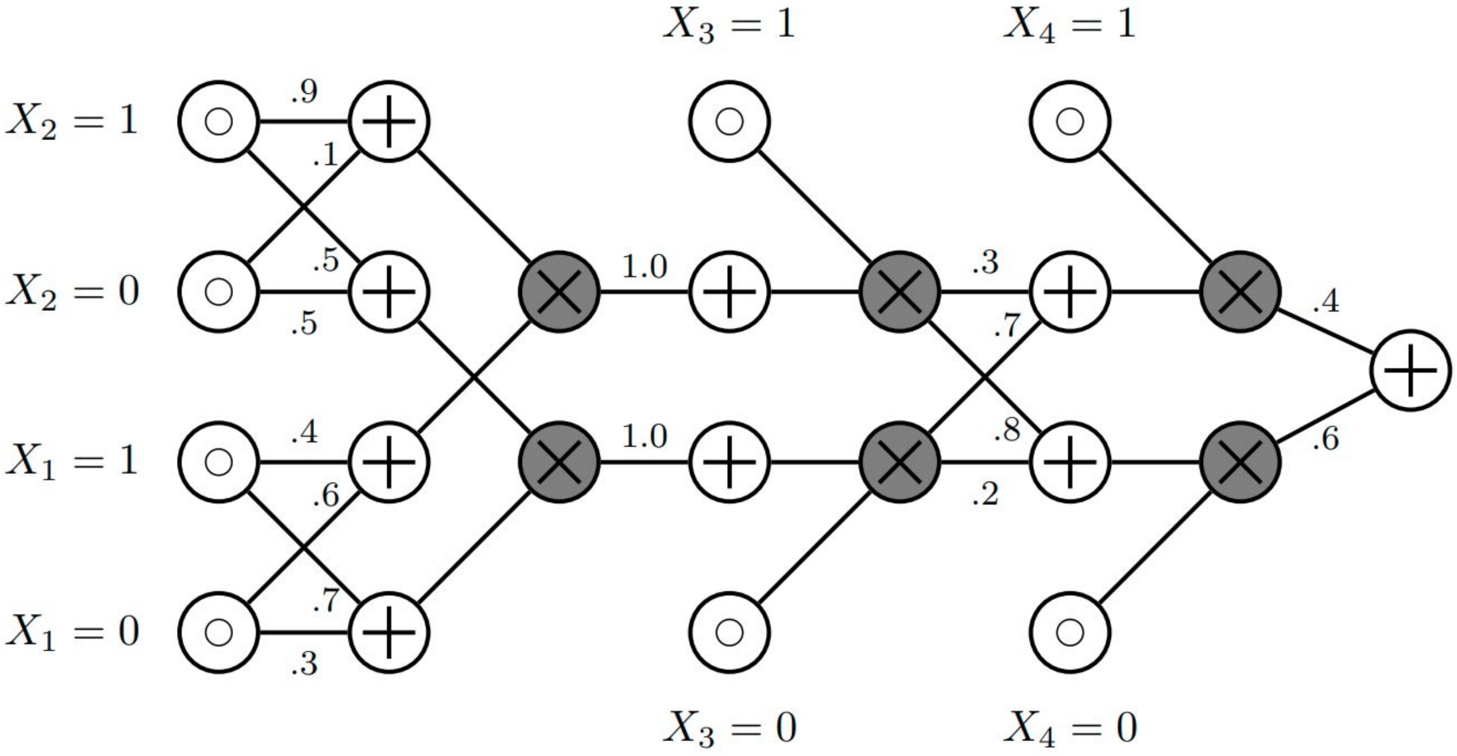
Mixtures BNs NICE FGs GANs

RealNVP MNs

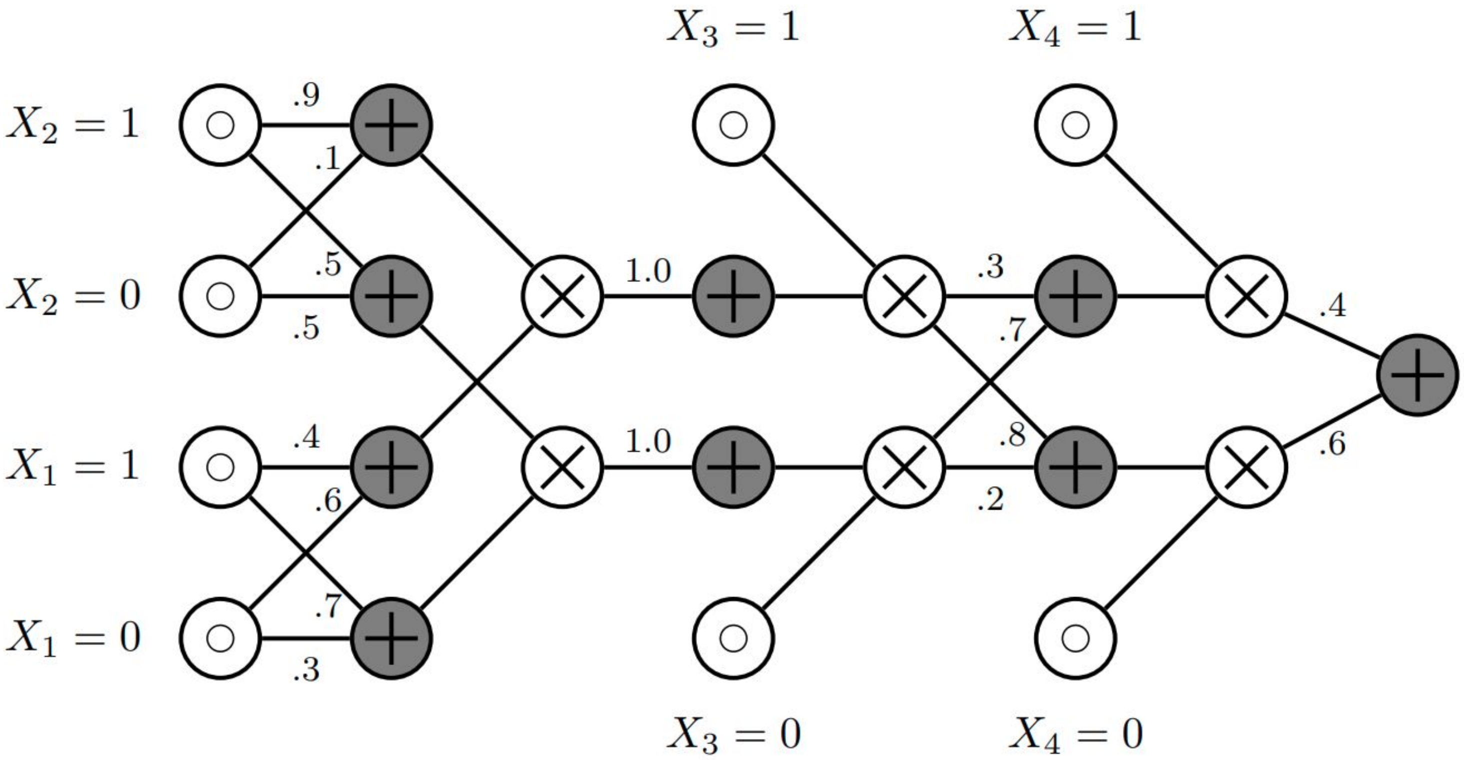
a *unifying framework* for tractable models



Input nodes are tractable (simple) distributions, e.g., indicator functions $p_n(X=1) = [X=1]$



Product nodes are factorizations $\prod_{c \in \mathsf{in}(n)} \mathrm{p}_c(\mathbf{x})$



Sum nodes are mixture models $\sum_{c \in \mathsf{in}(n)} \theta_{n,c} \, \mathrm{p}_c(\mathbf{x})$

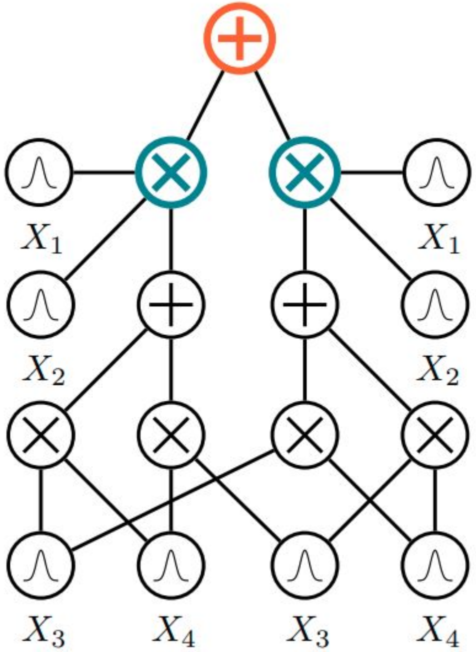
Smoothness + decomposability = tractable MAR

If $\boldsymbol{p}(\mathbf{x}) = \sum_i w_i \boldsymbol{p}_i(\mathbf{x})$, (***smoothness***):

$$\int \boldsymbol{p}(\mathbf{x}) d\mathbf{x} = \int \sum_i w_i \boldsymbol{p}_i(\mathbf{x}) d\mathbf{x} =$$

$$= \sum_i w_i \int \boldsymbol{p}_i(\mathbf{x}) d\mathbf{x}$$

$\Rightarrow$ *integrals are "pushed down" to children*



[Darwiche & Marquis JAIR 2001, Poon & Domingos UAI11]

Smoothness + decomposability = tractable MAR

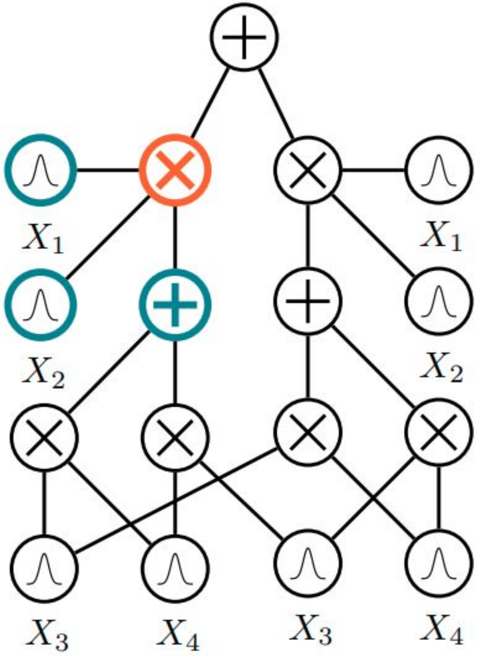
If $p(\mathbf{x}, \mathbf{y}, \mathbf{z}) = p(\mathbf{x})p(\mathbf{y})p(\mathbf{z})$, (***decomposability***):

$$\int\int\int p(\mathbf{x}, \mathbf{y}, \mathbf{z}) d\mathbf{x} d\mathbf{y} d\mathbf{z} =$$

$$= \int\int\int p(\mathbf{x})p(\mathbf{y})p(\mathbf{z}) d\mathbf{x} d\mathbf{y} d\mathbf{z} =$$

$$= \int p(\mathbf{x}) d\mathbf{x} \int p(\mathbf{y}) d\mathbf{y} \int p(\mathbf{z}) d\mathbf{z}$$

$\Rightarrow$ *integrals decompose into easier ones*



Smoothness + decomposability = tractable MAR

Forward pass evaluation for MAR

$\Rightarrow$ *linear in circuit size!*

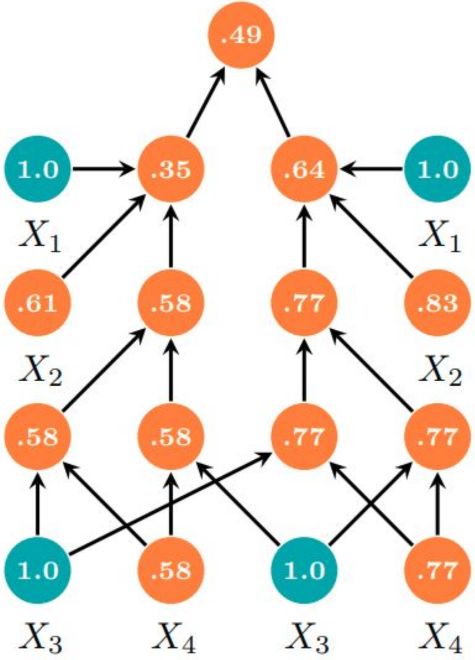
E.g. to compute $p(x_2, x_4)$:

- leafs over X_1 and X_3 output $\boldsymbol{Z}_i = \int p(x_i)dx_i$

  $\Rightarrow$ *for normalized leaf distributions:* 1.0

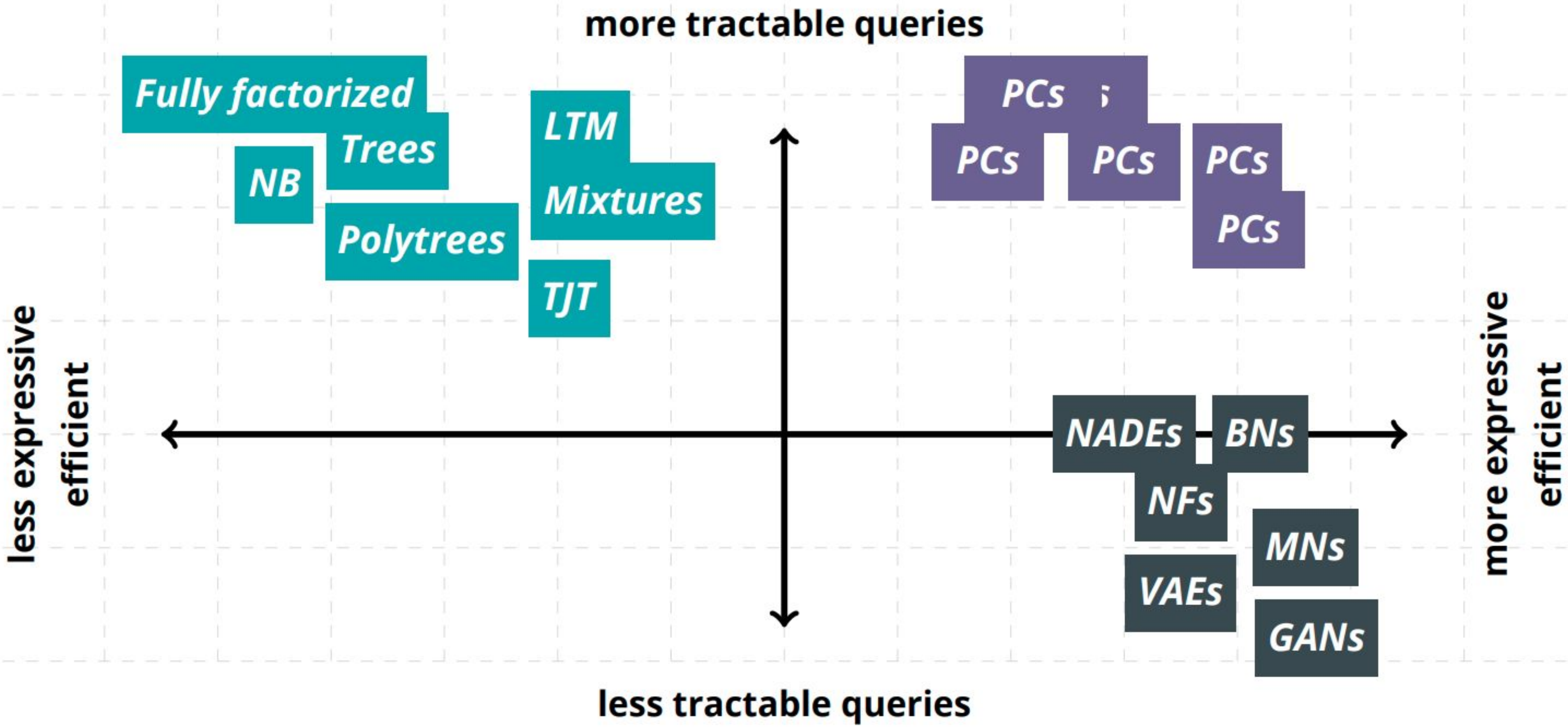
- leafs over X_2 and X_4 output ***EVI***

- feedforward evaluation (bottom-up)



$\mathcal{P}$ \ $\mathcal{Q}$:	MAR marginal queries	CON conditional queries	MOM moments (mean,...)	MAP maximum a posteriori	MMAP marginal MAP	ENT entropy	DIV divergences (KLD,...)	EXP expected predictions
smoothness SMO	✓	✓	✓	✗	✗	✓	✓	✓
decomposability DEC	✓	✓	✓	✗	✗	✗	✗	✗
consistency CON	✗	✗	✗	✓	✓	✗	✗	✗
determinism DET	✗	✗	✗	✓	✗	✗	✗	✗
marginal determinism MAR-DET	✗	✗	✗	✗	✓	✓	✓	✗
structured decomposability STR-DEC	✗	✗	✗	✗	✗	✓	✗	✗
paired str. decomposability P-STR-DEC	✗	✗	✗	✗	✗	✗	✓	✓

	smooth	*dec.*	**det.**	**str.dec.**
Arithmetic Circuits (ACs) *[Darwiche 2003]*	✔	✔	✔	✘
Sum-Product Networks (SPNs) *[Poon et al. 2011]*	✔	✔	✘	✘
Cutset Networks (CNets) *[Rahman et al. 2014]*	✔	✔	✔	✘
Probabilistic Decision Graphs *[Jaeger 2004]*	✔	✔	✔	✔
PSDDs *[Kisa et al. 2014a]*	✔	✔	✔	✔
AndOrGraphs *[Dechter et al. 2007]*	✔	✔	✔	✔



more tractable queries
Fully factorized
Trees
NB
Polytrees
LTM
Mixtures
TJT
PCs
PCs
PCs
PCs
PCs
less expressive
efficient
more expressive
efficient
NADEs
BNs
NFs
MNs
VAEs
GANs
less tractable queries

How expressive are probabilistic circuits?

density estimation benchmarks

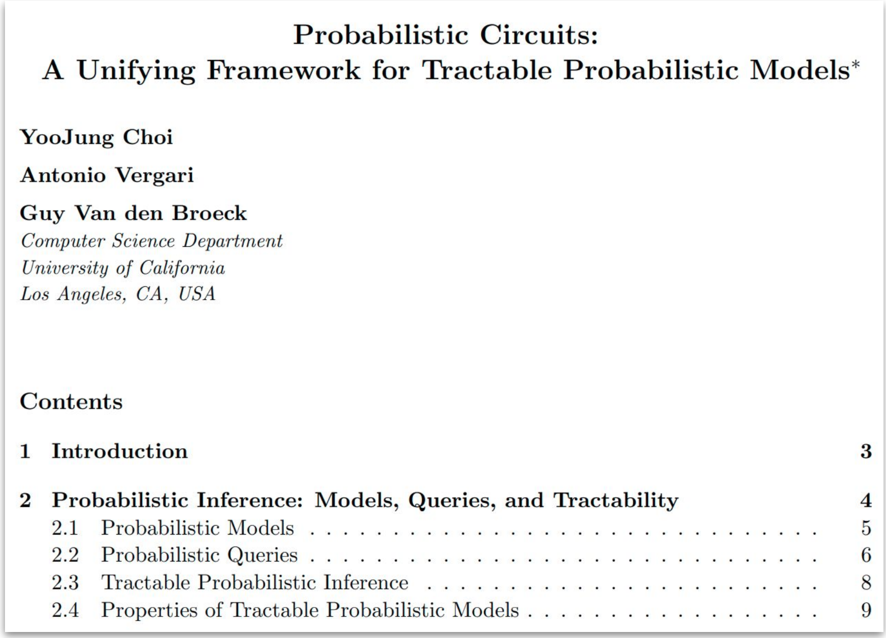
dataset	best circuit	BN	MADE	VAE	dataset	best circuit	BN	MADE	VAE
nltcs	**-5.99**	-6.02	-6.04	**-5.99**	***dna***	**-79.88**	-80.65	-82.77	-94.56
msnbc	**-6.04**	**-6.04**	-6.06	-6.09	***kosarek***	**-10.52**	-10.83	-	-10.64
kdd	-2.12	-2.19	**-2.07**	-2.12	***msweb***	-9.62	-9.70	**-9.59**	-9.73
plants	**-11.84**	-12.65	-12.32	-12.34	***book***	-33.82	-36.41	-33.95	**-33.19**
audio	-39.39	-40.50	-38.95	**-38.67**	***movie***	-50.34	-54.37	-48.7	**-47.43**
jester	-51.29	**-51.07**	-52.23	-51.54	***webkb***	-149.20	-157.43	-149.59	**-146.9**
netflix	-55.71	-57.02	-55.16	**-54.73**	***cr52***	-81.87	-87.56	-82.80	**-81.33**
accidents	-26.89	**-26.32**	-26.42	-29.11	***c20ng***	-151.02	-158.95	-153.18	**-146.9**
retail	**-10.72**	-10.87	-10.81	-10.83	***bbc***	**-229.21**	-257.86	-242.40	-240.94
pumbs*	-22.15	**-21.72**	-22.3	-25.16	***ad***	-14.00	-18.35	**-13.65**	-18.81

Want to learn more?

Tutorial (3h)

https://youtu.be/2RAG5-L9R70

Overview Paper (80p)



Probabilistic Circuits:
A Unifying Framework for Tractable Probabilistic Models*

YooJung Choi
Antonio Vergari
Guy Van den Broeck
Computer Science Department
University of California
Los Angeles, CA, USA

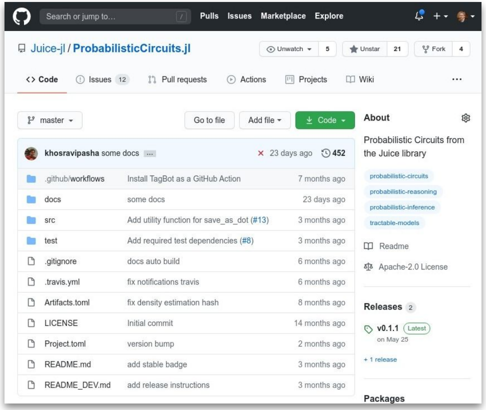
Contents

http://starai.cs.ucla.edu/papers/ProbCirc20.pdf

Training PCs in Julia with Juice.jl

Training maximum likelihood parameters of probabilistic circuits

```
julia> using ProbabilisticCircuits;
julia> data, structure = load(...);
julia> num_examples(data)
17412
julia> num_edges(structure)
270448
julia> @btime estimate_parameters(structure , data);
  63 ms
```



Custom SIMD and CUDA kernels to parallelize over layers and training examples.

https://github.com/Juice-jl/

Probabilistic circuits seem awfully general.

Are all tractable probabilistic models probabilistic circuits?

Determinantal Point Processes (DPPs)

DPPs are models where probabilities are specified by (sub)determinants

$$L = \begin{bmatrix} 1 & 0.9 & 0.8 & 0 \\ 0.9 & 0.97 & 0.96 & 0 \\ 0.8 & 0.96 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Pr_L(X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0) = \frac{1}{\det(L + I)} \det(L_{\{1,2\}})$$

Computing marginal probabilities is *tractable.*

[Zhang et al. UAI20]

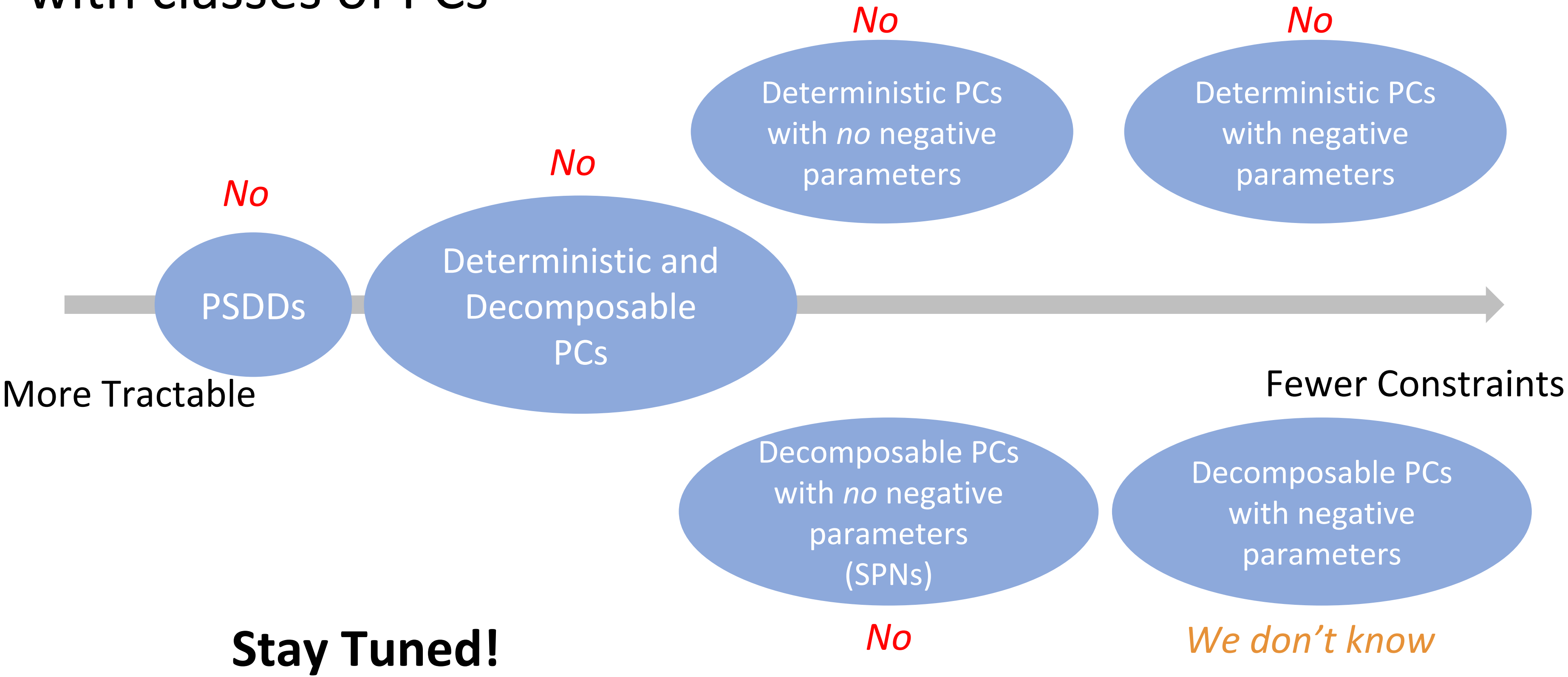
Representing the Determinant as a PC is *not easy*

Gaussian Elimination → Branching and Division

Laplace Expansion → Exponentially many subdeterminants

[Zhang et al. UAI20]

We cannot tractably represent DPPs with classes of PCs



[Zhang et al. UAI20; Martens & Medabalimi Arxiv15]

The AI Dilemma

Pure Logic **Pure Learning**

The AI Dilemma

Pure Logic **Pure Learning**

- Slow thinking: deliberative, cognitive, model-based, extrapolation
- Amazing achievements until this day
- "*Pure logic is brittle*"

  noise, uncertainty, incomplete knowledge, …

The AI Dilemma

Pure Logic **Pure Learning**

- Fast thinking: instinctive, perceptive, model-free, interpolation
- Amazing achievements recently
- "*Pure learning is brittle*"

  bias, algorithmic fairness, interpretability, explainability, adversarial attacks, unknown unknowns, calibration, verification, missing features, missing labels, data efficiency, shift in distribution, general robustness and safety

  fails to incorporate a sensible model of the world

Pure Logic   Probabilistic World Models   Pure Learning

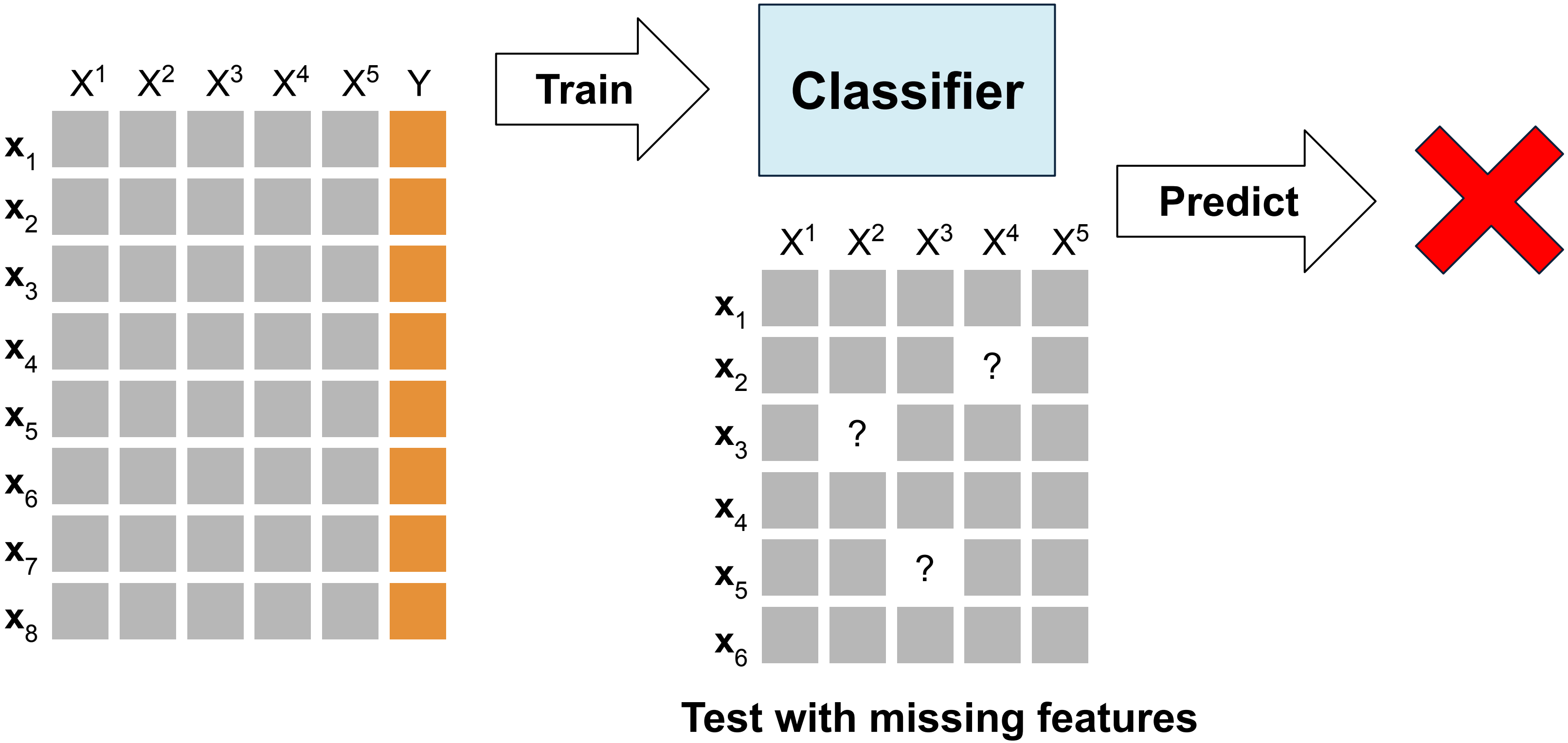
A New Synthesis of Learning and Reasoning

"Pure learning is brittle"

bias, **algorithmic fairness**, interpretability, **explainability**, adversarial attacks, unknown unknowns, calibration, verification, **missing features**, missing labels, data efficiency, shift in distribution, general robustness and safety

We need to incorporate a sensible probabilistic model of the world

Prediction with Missing Features



Expected Predictions

Consider **all possible complete inputs** and **reason** about the expected behavior of the classifier

$$\mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{x}^m | \mathbf{x}^o)} \left[f\left(\mathbf{x}^m \mathbf{x}^o\right) \right]$$

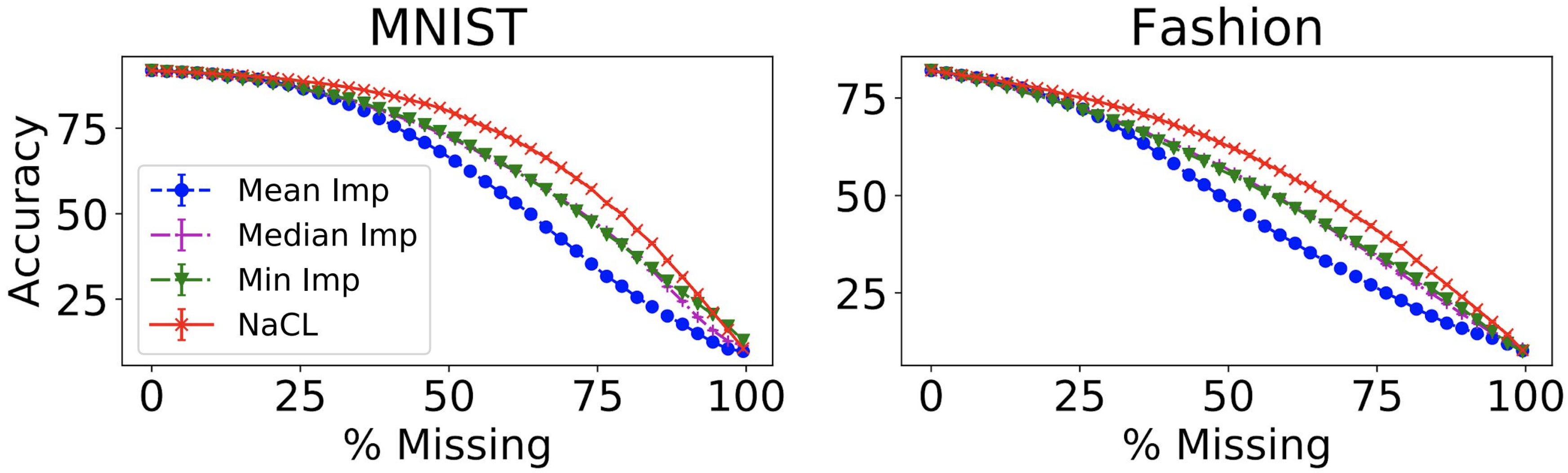
$\boldsymbol{x}^{\mathrm{o}}$ = observed features
$\boldsymbol{x}^{\mathrm{m}}$ = missing features

Generalizes what we've been doing all along...

$$\begin{aligned} P(C|\boldsymbol{y}) &= \sum_{\boldsymbol{m}} P(C, \boldsymbol{m}|\boldsymbol{y}) \\ &= \sum_{\boldsymbol{m}} P(C|\boldsymbol{m}, \boldsymbol{y})\, P(\boldsymbol{m}|\boldsymbol{y}) \\ &= \mathbb{E}_{\boldsymbol{m} \sim P(M|\boldsymbol{y})} P(C|\boldsymbol{m}, \boldsymbol{y}) \end{aligned}$$

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]

Experiments with simple distributions (Naive Bayes) to reason about missing data in logistic regression



“Conformant learning”

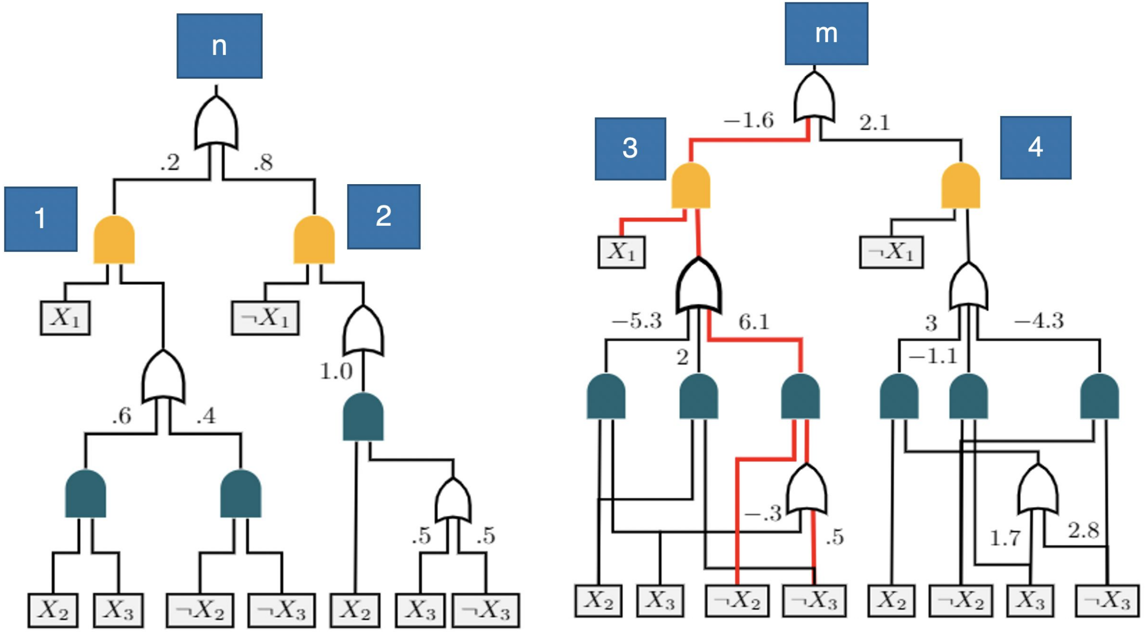
[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]

What about complex classifiers and distributions?

Tractable expected predictions if the classifier is a regression circuit, and the feature distribution is a compatible probabilistic circuits

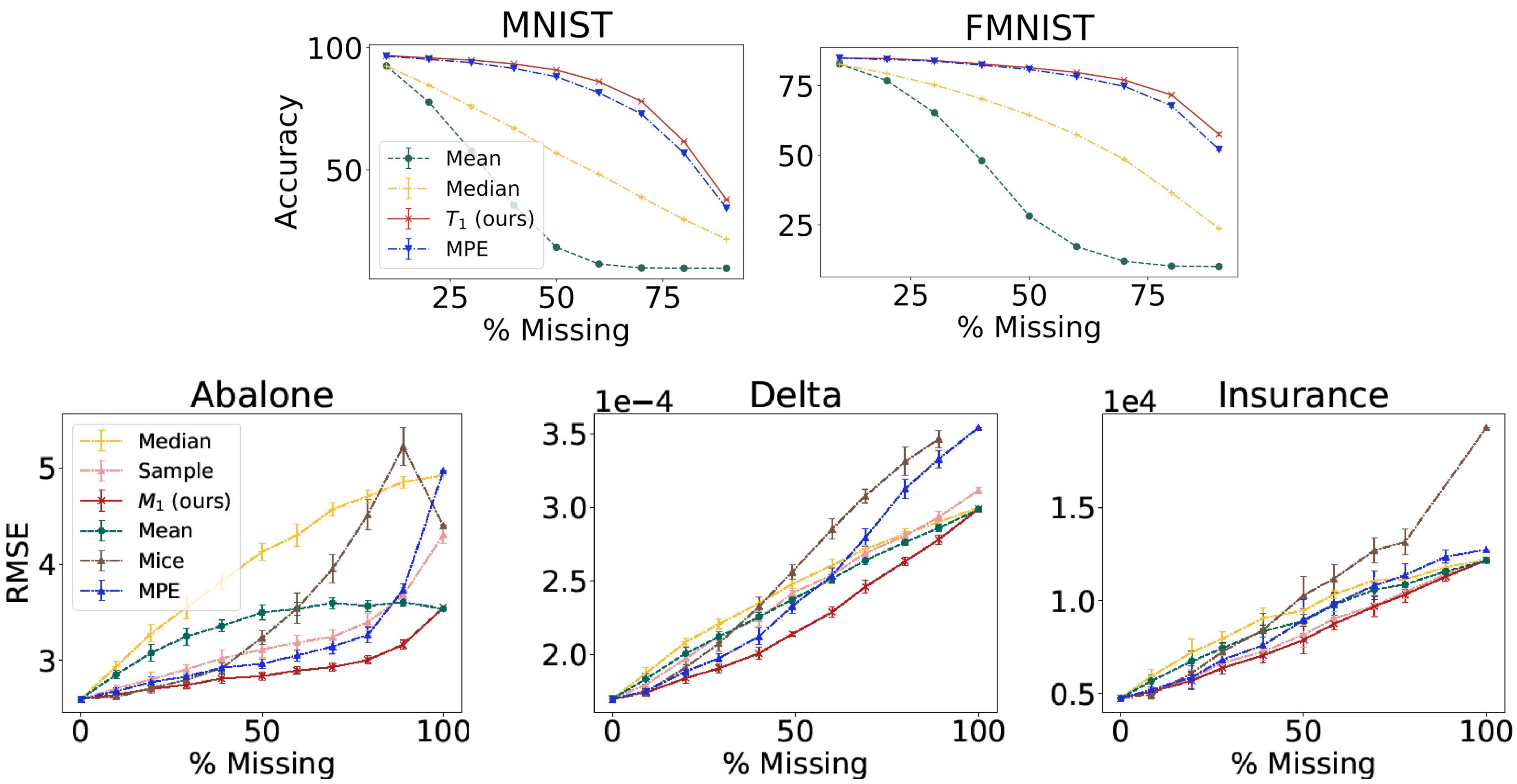
Recursion that "breaks down" the computation.

For + nodes (n,m), look at subproblems (1,3), (1,4), (2,3), (2,4)



[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]

Experiments with Probabilistic Circuits



[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]

ADV inference in Julia with Juice.jl

```julia
using ProbabilisticCircuits
pc = load_prob_circuit(zoo_psdd_file("insurance.psdd"));
rc = load_logistic_circuit(zoo_lc_file("insurance.circuit"), 1);
```

q_8: *How different is the insurance costs between smokers and non smokers?*

```julia
groups = make_observations([["!smoker"], ["smoker"]])
exps, _ = Expectation(pc, rc, groups);
println("Smoker     : \$ $(exps[2])");
println("Non-Smoker: \$ $(exps[1])");
println("Difference: \$ $(exps[2] - exps[1])");
Smoker     : $ 31355.32630488978
Non-Smoker: $ 8741.747258310648
Difference: $ 22613.57904657913
```

What If Training Also Has Missingness

This time we consider decision trees as the classifier

$$\mathcal{L}(\Theta; \mathsf{D}_{\mathsf{train}}) = \frac{1}{|\mathsf{D}_{\mathsf{train}}|} \sum_{\mathbf{x}^o, y \in \mathsf{D}_{\mathsf{train}}} \mathbb{E}_{p_\Phi(\mathbf{X}^m | \mathbf{x}^o)} \left[l(y, f_\Theta(\mathbf{x})) \right]$$

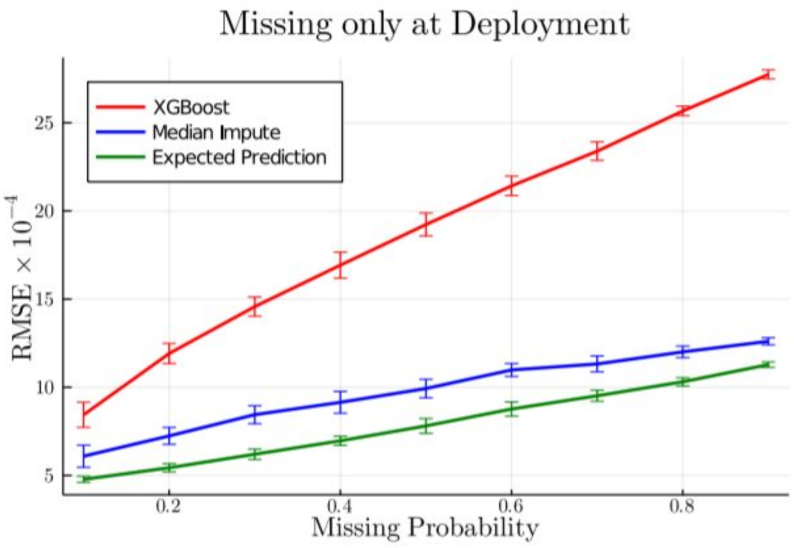
For one decision tree and using MSE loss, can be computed exactly

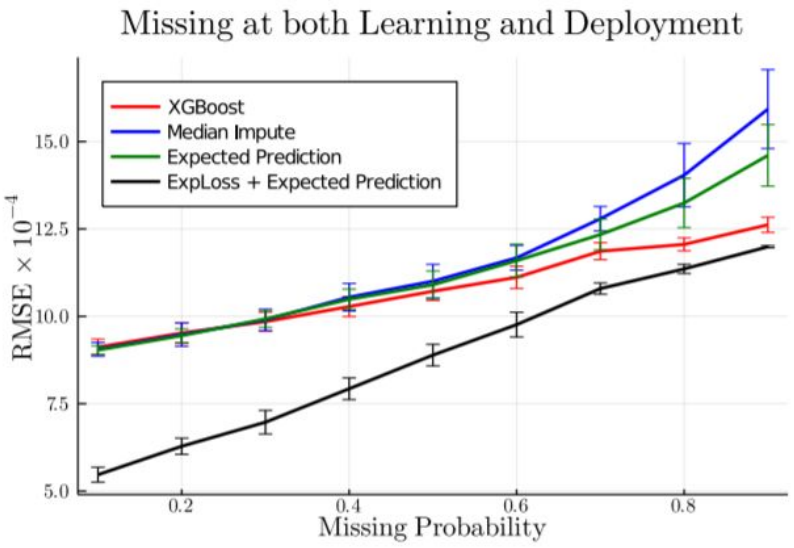
$$\theta_\ell^* = \frac{\sum_{\mathbf{x}^o, y \in \mathsf{D}_{\mathsf{train}}} y \cdot p_\ell(\mathbf{x}^o) / p(\mathbf{x}^o)}{\sum_{\mathbf{x}^o, y \in \mathsf{D}_{\mathsf{train}}} p_\ell(\mathbf{x}^o) / p(\mathbf{x}^o)}$$

More scenarios such as bagging/boosting in the paper.

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss 20]

Preliminary Experiments





[Khosravi et al. IJCAI19, NeurIPS20, Artemiss 20]

ADV inference in Julia with Juice.jl

```julia
using ProbabilisticCircuits
pc = load_prob_circuit(zoo_psdd_file("insurance.psdd"));
rc = load_logistic_circuit(zoo_lc_file("insurance.circuit"), 1);
```

q_9: *Is the predictive model biased by gender?*

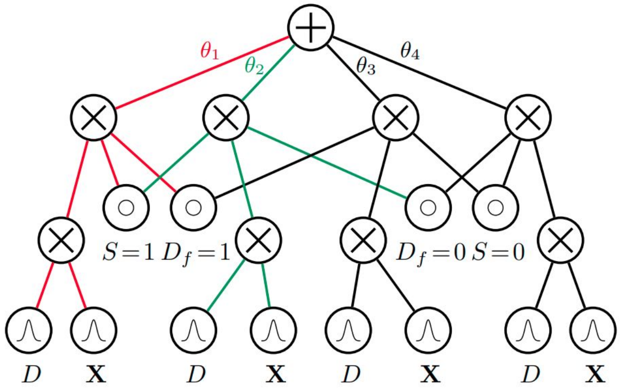
```julia
groups = make_observations([["male"], ["female"]])
exps, _ = Expectation(pc, rc, groups);
println("Female  : \$ $(exps[2])");
println("Male    : \$ $(exps[1])");
println("Diff    : \$ $(exps[2] - exps[1])");
```

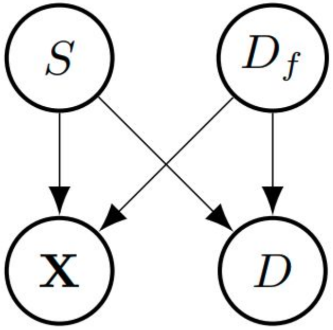
```
Female  : $ 14170.125469335406
Male    : $ 13196.548926381849
Diff    : $ 973.5765429535568
```

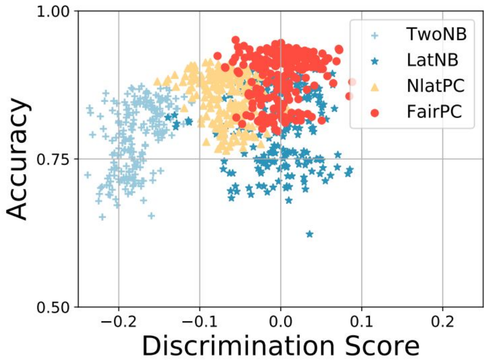
Model-Based Algorithmic Fairness: FairPC

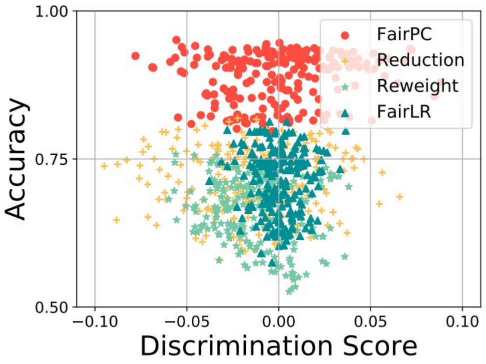
Learn classifier given
- features S and X
- training labels D

Fair decision *Df* should be independent of the sensitive attribute *S*









[Choi et al. Arxiv20]

Probabilistic Sufficient Explanations

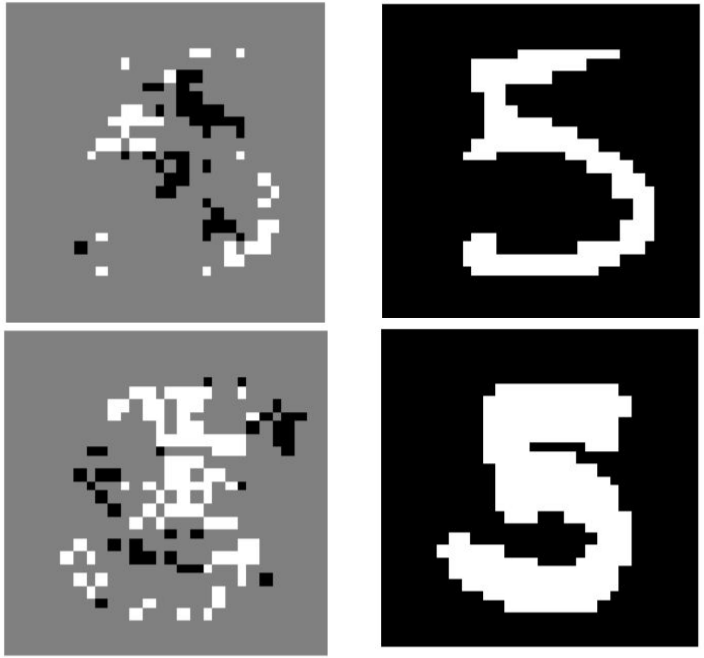
Goal: explain an instance of classification

Choose a **subset** of features s.t.

1. Given only the explanation it is "probabilistically sufficient"

   Under the feature distribution, it is likely to make the prediction to be explained

2. It is minimal and "simple"



[Khosravi et al. IJCAI19, Wang et al. XXAI20]

Pure Logic **Probabilistic World Models** **Pure Learning**

A New Synthesis of Learning and Reasoning

"Pure learning is brittle"

bias, **algorithmic fairness**, interpretability, **explainability**, adversarial attacks, unknown unknowns, calibration, verification, **missing features**, missing labels, data efficiency, shift in distribution, general robustness and safety

We need to incorporate a sensible probabilistic model of the world

Probabilistic Programs

What are probabilistic programs?

```
let x = flip 0.5 in
let y = flip 0.7 in
let z = x || y in
let w = if z then
      my_func(x,y)
else
      ...
in
observe(z);
```

means "flip a coin, and output true with probability ½"

Standard (functional) programming constructs: let, if, ...

means "reject this execution if z is not true"

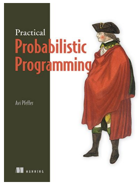
Why Probabilistic Programming?

PPLs are proliferating

Edward

HackPPL

Figaro



Pyro

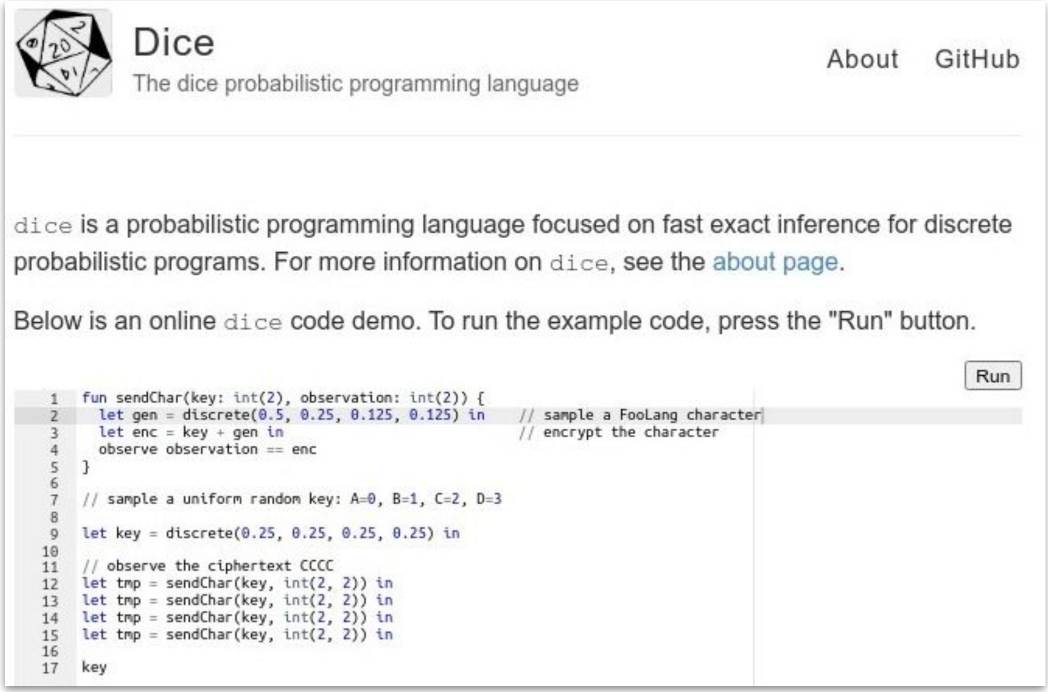
Stan

Venture, Church, IBAL, WebPPL, Infer.NET, Tensorflow Probability, ProbLog, PRISM, LPADs, CPLogic, CLP(BN), ICL, PHA, Primula, Storm, Gen, PRISM, PSI, Bean Machine, etc. … *and many many more*

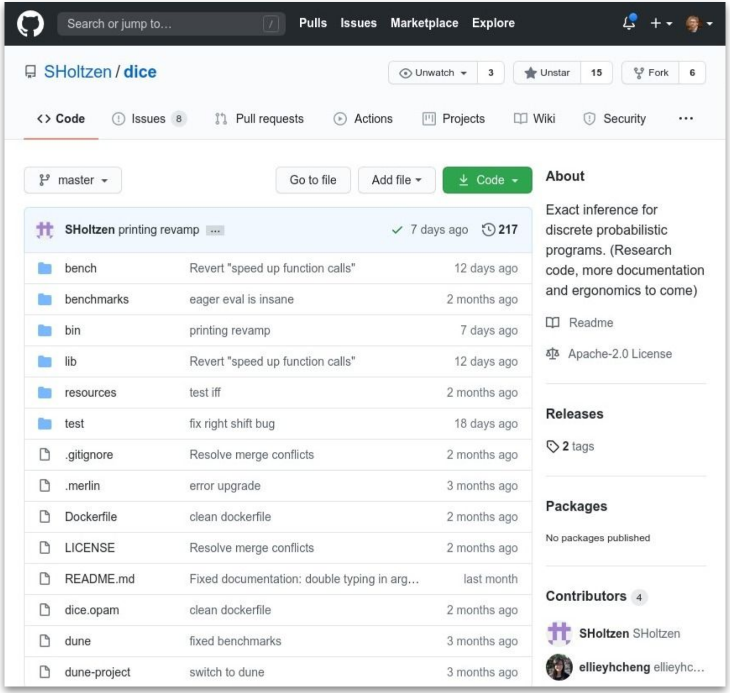
Programming languages are humanity's biggest knowledge representation achievement!

Dice probabilistic programming language

http://dicelang.cs.ucla.edu/



https://github.com/SHoltzen/dice



[Holtzen et al. OOPSLA20 (tentative)]

What is a possible world?

Program	Execution A	Execution B	Execution C	Execution D
let x = flip 0.4 in	x=1	x=1	x=0	x=0
let y = flip 0.7 in	x=1, y=1	x=1, y=0	x=0, y=1	x=0, y=0
let z = x || y in	x=1, y=1, z=1	x=1, y=0, z=1	x=0, y=1, z=1	x=0, y=0, z=0
let x = if z then				
x	x=1, y=1, z=1	x=1, y=0, z=1	x=0, y=1, z=1	
else				
1				x=1, y=0, z=0
in (x,y)	(1, 1)	(1,0)	(0,1)	(1,0)
	P = 0.4*0.7	**P = 0.4*0.3**	**P = 0.6*0.7**	**P = 0.6*0.3**

Why should I care? I like PGMs

- Better abstraction:
  - Beyond variable-level dependencies
  - modularity through functions
    reuse (cf. templative graphical models)
  - intuitive language for local structure; arithmetic
  - data structures
  - first-class observations

First-Class Observations, Functions

```
fun EncryptChar(key:int, obs:char):Bool {
  let randomChar = ChooseChar() in
  let ciphertext = (randomChar + key) % 26 in
  let _ = observe ciphertext = obs in
  true}
let k = UniformInt(0, 25) in
let _ = EncryptChar(k, 'H') in ···
let _ = EncryptChar(k, 'D') in k
```

Frequency Analyzer for a Caesar cipher in Dice

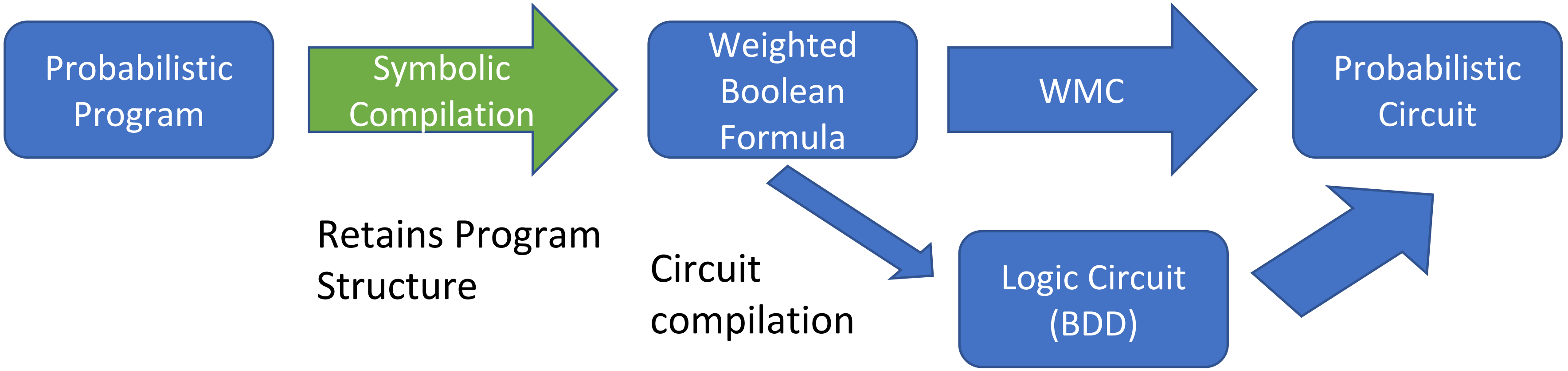
What do PGMs bring to the table?

1. Real programs have inherently discrete structure (e.g. if-statements)
2. Discrete structure is inherent in many domains (graphs, text/topic models, ranking, etc.)
3. Many existing PPLs assume smooth and differentiable densities and do not handle these programs correctly.

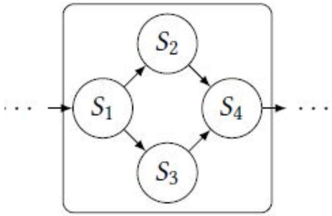
Discrete probabilistic programming is the important unsolved open problem!

PGM community knows how to solve this!

Symbolic Compilation to Probabilistic Circuits



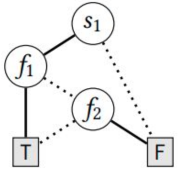
Inference in Dice



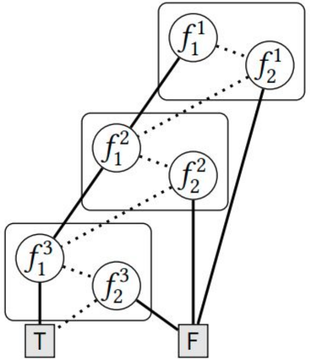
(a) Network diagram.

```
fun diamond(s1:Bool):Bool {
  let route = flip1 0.5 in
  let s2 = if route then s1 else F in
  let s3 = if route then F else s1 in
  let drop = flip2 0.0001 in
  s2 ∨ (s3 ∧ ¬drop)}
diamond(diamond(diamond(T)))
```

(b) Probabilistic program defining the network.



(c) diamond function.



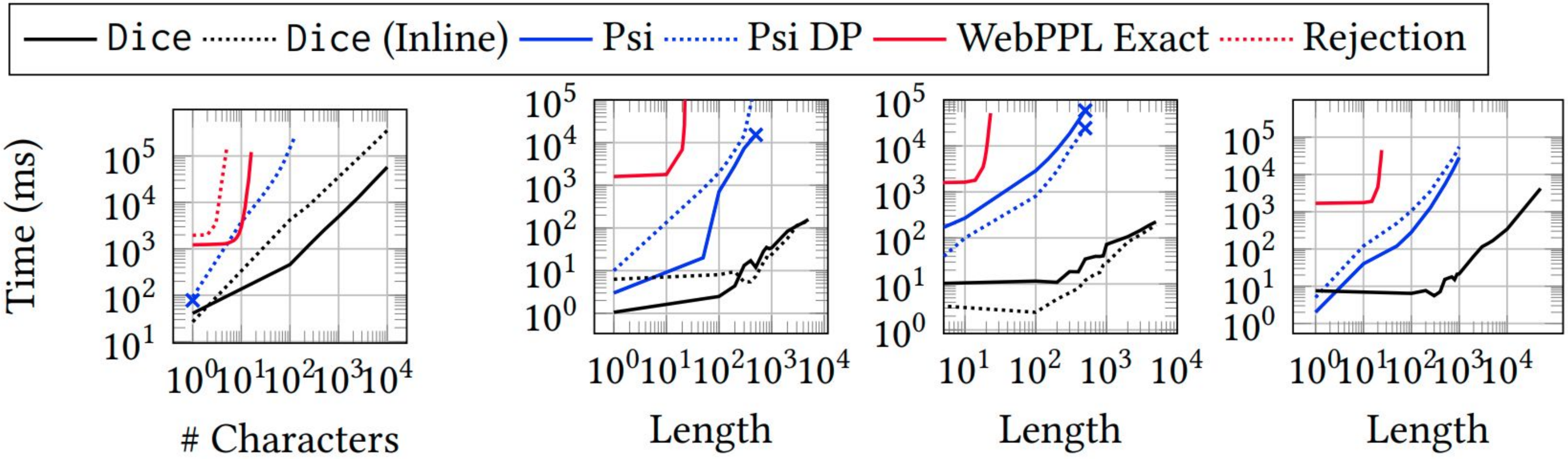
(d) Final BDD.

Network Verification

PPL benchmarks from PL community

Benchmark	Psi (ms)	DP (ms)	Dice (ms)	# Paths	BDD Size
Grass	167	57	**1.0**	9.5×10^1	15
Burglar Alarm	98	10	**1.1**	2.5×10^2	11
Coin Bias	94	23	**1.0**	4	13
Noisy Or	81	152	**1.0**	1.6×10^4	35
Evidence1	48	32	**1.0**	9	5
Evidence2	59	28	**1.0**	9	6
Murder Mystery	193	75	**1.0**	1.6×10^1	6

Scalable Inference

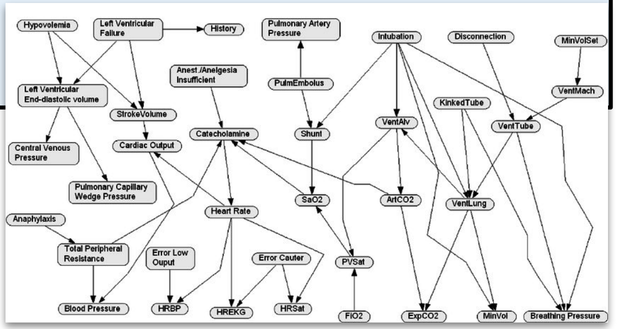


Scalable Inference

Benchmark	Psi (ms)	DP (ms)	Dice (ms)	# Parameters	# Paths	BDD Size
Cancer [48]	772	46	**1.0**	10	1.1×10^{3}	28
Survey [73]	2477	152	**2.0**	21	1.3×10^{4}	73
Alarm [5]	✗	✗	**9.0**	509	1.0×10^{36}	1.3×10^{3}
Insurance [7]	✗	✗	**75.0**	984	1.2×10^{40}	1.0×10^{5}
Hepar2 [63]	✗	✗	**54.0**	1453	2.9×10^{69}	1.3×10^{3}
Hailfinder [1]	✗	✗	**526.0**	2656	2.0×10^{76}	6.5×10^{4}
Pigs	✗	✗	**32.0**	5618	7.3×10^{492}	35
Water [43]	✗	✗	**2926.0**	1.0×10^{4}	3.2×10^{54}	5.1×10^{4}
Munin [3]	✗	✗	**1945.0**	8.1×10^{5}	2.1×10^{1622}	1.1×10^{4}

Alarm Bayesian Network

```
let HYPOVOLEMIA = flip 0.2 in
let LVFAILURE = flip 0.05 in
let STROKEVOLUME =
     if (HYPOVOLEMIA) then
          (if (LVFAILURE) then (discrete(0.98,0.01,0.01)) else (discrete(0.50,0.49,0.01)))
     else
          (if (LVFAILURE) then (discrete(0.95,0.04,0.01)) else (discrete(0.05,0.90,0.05)))
in
let LVEDVOLUME =
     if (HYPOVOLEMIA) then
          (if (LVFAILURE) then (discrete(0.95,0.04,0.01)) else (discrete(0.01,0.09,0.90)))
     else
          (if (LVFAILURE) then (discrete(0.98,0.01,0.01)) else (discrete(0.05,0.90,0.05)))
in
...
```



Why should I care? I like PGMs

- Better abstraction:
  - Beyond variable-level dependencies
  - modularity through functions
    reuse (cf. templative graphical models)
  - intuitive language for local structure; arithmetic
  - data structures
  - first-class observations
- Better inference? correctness? analysis?
  `import PL.*`

Denotational Semantics

- Goal: associate with every expression “e” a semantic object.
- Notation: semantic bracket: [[.]]
  - In Bayesian network: [[BN]] = $Pr_{BN}(.)$
  - In probabilistic programs: [[e]](.) for all expressions
  - Accepting and distributional semantics:

$$[\![\mathrm{e}]\!]_A \triangleq \sum_v [\![\mathrm{e}]\!](v), \qquad [\![\mathrm{e}]\!]_D(v) \triangleq \frac{1}{[\![\mathrm{e}]\!]_A} [\![\mathrm{e}]\!](v)$$

- Idea: don’t need to run ‘flip 0.4’ infinite times to know meaning

Denotational Semantics
+ Formal Inference Rules

$$[\![v_1]\!](v) \triangleq \big(\delta(v_1)\big)(v) \qquad [\![\texttt{fst}\ (v_1, v_2)]\!](v) \triangleq \big(\delta(v_1)\big)(v) \qquad [\![\texttt{snd}\ (v_1, v_2)]\!](v) \triangleq \big(\delta(v_2)\big)(v)$$

$$[\![\texttt{if}\ v_g\ \texttt{then}\ \mathsf{e}_1\ \texttt{else}\ \mathsf{e}_2]\!](v) \triangleq \begin{cases} [\![\mathsf{e}_1]\!](v) & \text{if } v_g = \mathsf{T} \\ [\![\mathsf{e}_2]\!](v) & \text{if } v_g = \mathsf{F} \\ 0 & \text{otherwise} \end{cases} \qquad [\![\texttt{flip}\ \theta]\!](v) \triangleq \begin{cases} \theta & \text{if } v = \mathsf{T} \\ 1-\theta & \text{if } v = \mathsf{F} \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\texttt{observe}\ v_1]\!](v) \triangleq \begin{cases} 1 & \text{if } v_1 = \mathsf{T} \text{ and } v = \mathsf{T}, \\ 0 & \text{otherwise} \end{cases} \qquad [\![f(v_1)]\!](v) \triangleq \Big(\big(T(f)\big)(v_1)\Big)(v)$$

$$[\![\texttt{let}\ x = \mathsf{e}_1\ \texttt{in}\ \mathsf{e}_2]\!](v) \triangleq \sum_{v'} [\![\mathsf{e}_1]\!](v') \times [\![\mathsf{e}_2[x \mapsto v']]\!](v)$$

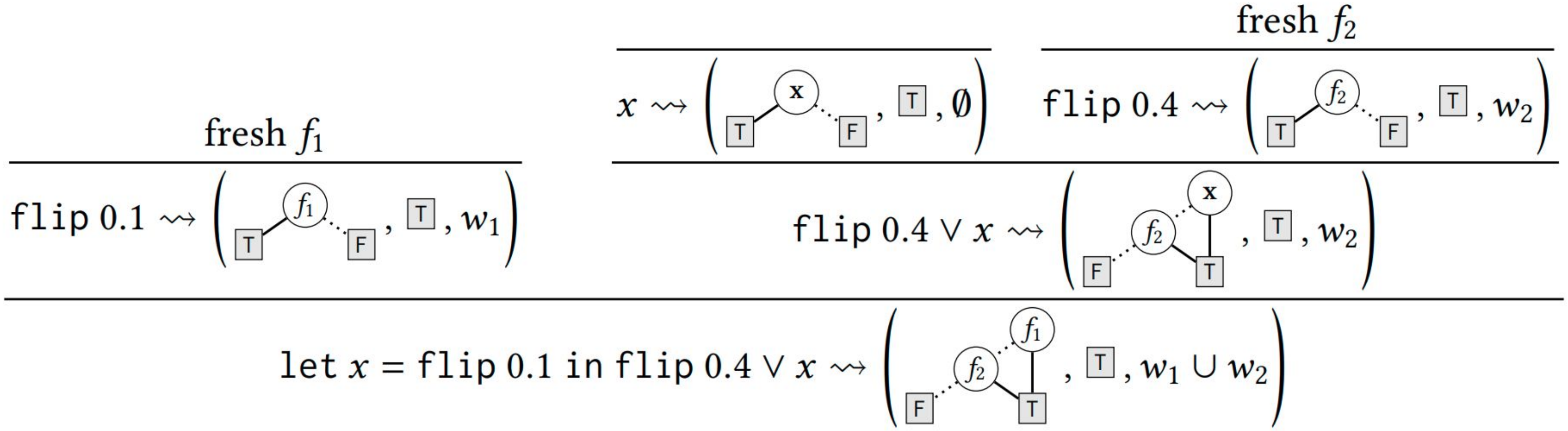
$$\frac{}{\mathsf{T} \rightsquigarrow (\mathsf{T}, \mathsf{T}, \emptyset)}\ (\text{C-True}) \qquad \frac{}{\mathsf{F} \rightsquigarrow (\mathsf{F}, \mathsf{T}, \emptyset)}\ (\text{C-False}) \qquad \frac{}{x \rightsquigarrow (\mathbf{x}, \mathsf{T}, \emptyset)}\ (\text{C-Ident})$$

$$\frac{\text{fresh } \mathbf{f}}{\texttt{flip}\ \theta \rightsquigarrow \big(\mathbf{f}, \mathsf{T}, (\mathbf{f} \mapsto \theta, \mathsf{T}, \overline{\mathbf{f}} \mapsto 1-\theta)\big)}\ (\text{C-Flip}) \qquad \frac{\mathsf{aexp} \rightsquigarrow (\varphi, \mathsf{T}, \emptyset)}{\texttt{observe}\ \mathsf{aexp} \rightsquigarrow (\mathsf{T}, \varphi, \emptyset)}\ (\text{C-Obs})$$

$$\frac{\mathsf{aexp} \rightsquigarrow (\varphi_g, \mathsf{T}, \emptyset) \qquad \mathsf{e}_T \rightsquigarrow (\varphi_T, \gamma_T, w_T) \qquad \mathsf{e}_E \rightsquigarrow (\varphi_E, \gamma_E, w_E)}{\texttt{if}\ \mathsf{aexp}\ \texttt{then}\ \mathsf{e}_\mathsf{T}\ \texttt{else}\ \mathsf{e}_\mathsf{E} \rightsquigarrow \Big(\big((\varphi_g \wedge \varphi_T) \vee ((\overline{\varphi}_g \wedge \varphi_E), ((\varphi_g \wedge \gamma_T) \vee ((\overline{\varphi}_g \wedge \gamma_E), w_T \cup w_E\Big)}\ (\text{C-Ite})$$

$$\frac{\mathsf{e}_1 \rightsquigarrow (\varphi_1, \gamma_1, w_1) \qquad \mathsf{e}_2 \rightsquigarrow (\varphi_2, \gamma_2, w_2)}{\texttt{let}\ x = \mathsf{e}_1\ \texttt{in}\ \mathsf{e}_2 \rightsquigarrow (\varphi_2[\mathbf{x} \mapsto \varphi_1], \gamma_1 \wedge \gamma_2[\mathbf{x} \mapsto \varphi_1], w_1 \cup w_2)}\ (\text{C-Let})$$

Provably Correct Inference!



Better Inference?

Exploit modularity

1. AI modularity:
   Discover contextual independencies and factorize
2. PL modularity:
   Compile procedure summaries and reuse at each call site

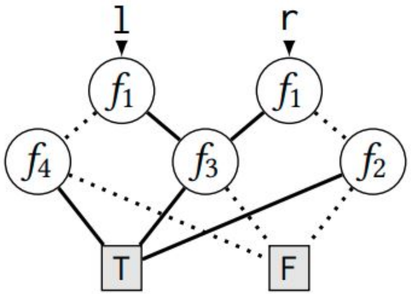
Reason about programs! Compiler optimizations.
Quick preview:

3. Flip hoisting optimization
4. Eager compilation

From programs to circuits directly:

```
let z = flip1 0.5 in
let x = if z then flip2 0.6 else flip3 0.7 in
let y = if z then flip4 0.7 else x in (x, y)
```

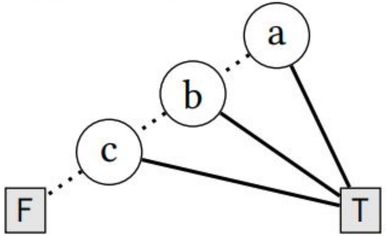
(a) Context-specific independence.



(b) Compiled BDD.

```
fun foo(a:Bool, b:Bool, c:Bool):Bool {
  a ∨ b ∨ c
}
```

(c) Structure without independence.



(d) Compiled BDD.

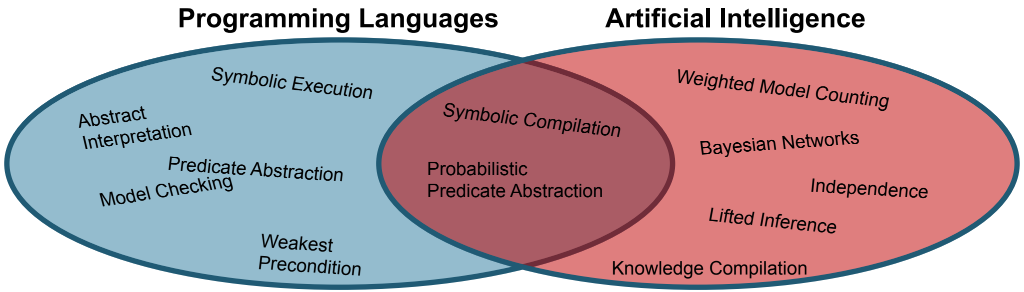
Compiler Optimizations (sneak preview)

Benchmark	Naive compilation	determinism	flip hoisting + determinism	Eager + flip lifting	*Ace baseline*
alarm	156	140	83	69	422
water	56,267	65,975	1509	941	605
insurance	140	100	100	128	492
hepar2	95	80	80	80	495
pigs	3,772	2490	2112	186	985
munin	>1,000,000	>1,000,000	109,687	16,536	3,500

Inference time in milliseconds

Conclusions

- Are we already in the age of computational abstractions?
- Probabilistic circuits for learning deep tractable probabilistic models
- Probabilistic programs as the new probabilistic knowledge representation language



Thanks