Probabilistic Reasoning with PCE

S. Owre, N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

What is PCE?

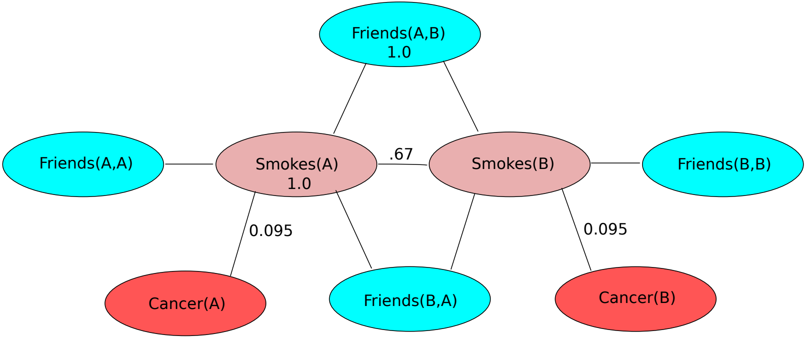
- PCE stands for Probabilistic Consistency Engine
- It is used for probabilistic inference with Markov Logic as a formal framework
- PCE can infer the marginal probabilities of formulas based on facts and rules.
- Facts and rules are presented in an order-sorted first-order logic.
- Inference is carried out using sampling-based methods in order to achieve scale
- PCE is general enough to capture other graph-based formalisms for probabilistic reasoning

Overview

- Small Example
- Probability Basics
- Graphical Models
- PCE Background
- PCE MCSAT Algorithm
- PCE Example: Boy Born on Tuesday
- PCE Application: Machine Reading
- Conclusions

Small Example - Markov Logic Network

- Smoking causes cancer
- Friends influence smoking behavior



Small Example - PCE Specification

```
sort Person;
const A, B, C: Person;
predicate Friends(Person, Person) hidden;
predicate Smokes(Person) hidden;
predicate Cancer(Person) hidden;

# Smoking causes cancer.
add [x] Smokes(x) => Cancer(x)  0.1;
# If two people are friends, either both smoke or neither does.
add [x, y] Friends(x, y) implies (Smokes(x) implies Smokes(y))  1.1;

add Smokes(A);
add Friends(A, B);

mcsat_params 100000, .5, 5, .05, 100, 5;
mcsat; dumptable atom;
```

Small PCE Example - Results

i	prob	atom
0	0.500	Friends(A, A)
1	1.000	Friends(A, B)
2	0.400	Friends(A, C)
3	0.501	Friends(B, A)
4	0.501	Friends(B, B)
5	0.434	Friends(B, C)
6	0.500	Friends(C, A)
7	0.472	Friends(C, B)
8	0.501	Friends(C, C)
9	1.000	Smokes(A)
10	0.752	Smokes(B)
11	0.616	Smokes(C)
12	0.525	Cancer(A)
13	0.519	Cancer(B)
14	0.516	Cancer(C)

Probability Basics (From Neapolitan)

- Given a sample space Ω of the form $\{e_1, \ldots, e_n\}$.
- An event E is a subset of Ω.
- A probability function P assigns a value in $[0, 1]$ to events such that
  1. $P(\{e_1\}) + \ldots + P(\{e_n\}) = 1$, and
  2. $P(E) = \Sigma_{e \in E} P(\{e\})$.
- Example: For a fair 6-sided dice, the probability $P(i)$ for $1 \leq i \leq 6$ is $\frac{1}{6}$, and the probability of an even number is $\frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}$

Bayes Theorem (Wikipedia)

Bayes' theorem relates the conditional and marginal probabilities of events A and B, where B has a non-vanishing probability:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}.$$

Each term in Bayes' theorem has a conventional name:

- $P(A)$ is the prior or marginal probability of A.
- $P(A|B)$ is the conditional or posterior probability of A, given B.
- $P(B|A)$ is the conditional probability of B given A. It is also called the likelihood.
- $P(B)$ is the prior or marginal probability of B; acts as a normalizing constant.

Logic and Probability (Wikipedia)

- Medical diagnosis offers a simple example of Bayesian reasoning.
- We have a test for a disease that returns positive or negative results.
- If the patient has the disease, the test is positive with probability .99.
- If the patient does not have the disease, the test is positive with probability .05.
- A patient has the disease with probability .001.
- What is the probability that a patient with a positive test has the disease?
- $P(D|pos) = P(pos|D)P(D)/P(pos) = .99 \times .001/(.99 \times .001 + .05 \times .999) = 99/5094 = .0194$

Medical Diagnosis in PCE

```
sort Patient;
const a: Patient;
predicate testedPositive(Patient) hidden;
predicate diseased(Patient) hidden;
add testedPositive(a) or ~diseased(a) 4.6; # 99%
add ~testedPositive(a) or ~diseased(a) .01; # 1%
add testedPositive(a) or diseased(a) .05;   # 5%
add ~testedPositive(a) or diseased(a) 3.0; # 95%
add ~diseased(a) 6.9; # 50%
add diseased(a) .001; # .1%

add testedPositive(a);
mcsat_params 1000000, 0.5, 20.0, 0.5, 30;
mcsat; dumptable atom;
```

Result:

```
| 1 |  0.020 | diseased(a)
```

- The general idea is that you have some joint probability distribution over random variables $X = \{X_1, \ldots, X_n\}$
- From the joint distribution, you would like to compute marginal probabilities $P(X_i = x_i)$ and conditional probabilities $P(Z = z | Y = y)$, where Z and Y are disjoint subsets of X
- *Graphical models* represent the joint distribution $P(X = x)$ as a product $1/Z\Pi_k F_k(X_k = x_k)$, where k is a subset of indices, X_k is the corresponding subset of variables, x_k the corresponding values, and Z is a normalization constant
- *Bayesian networks* and *Markov Logic Networks* are both instances of graphical models
- In a Bayesian network, the joint probabilities at a node are relative to the parent nodes, whereas a Markov logic network is undirected

Bayesian Networks

- Potential functions in a Bayesian Network can be converted to feature function using a log-linear model as $P(X = x) = 1/Ze^{(\Sigma_i w_i f_i(x))}$.
- A Bayesian Network can be used to compute joint probability distributions $P(x_1, \ldots, x_n)$ by taking each individual variable X_i, its parents π_i and the valuation of its parents $X_{\pi_i} = x_{\pi_i}$ as $\Pi_i P(X_i = x_i | X_{\pi_i} = x_{\pi_i})$.
- It can also be used for conditional inference to compute the probability that $\overline{X} = \overline{x}$ given $\overline{Y} = \overline{y}$.

Markov Logic Networks

- Markov Logic Networks were developed by Pedro Domingos et al.
- More details may be found in Pedro's recently published book, available online at `http://dx.doi.org/10.2200/S00206ED1V01Y200907AIM007`
- In a Markov Logic Network, all random variables are Boolean and all feature functions are Boolean formulas
- To compute conditional probabilities $P(Z = z | Y = y)$ over a graphical model, one typically uses Markov Chain Monte Carlo (MCMC) sampling algorithms to marginalize over the variables in $X - (Y \cup Z)$
- A Markov chain is a state machine with probabilities attached to the transitions between states – sum of probabilities of outgoing transitions from each state must add up to 1
- *MCSAT* is used as the MCMC sampling method for Markov logic networks

PCE Background

- PCE is based on Markov Logic, and uses MCSAT to compute marginal probabilities
- The input language consists of sorts, subsorts, constants, observable predicates and hidden predicates, facts, and weighted rules
- The MCSAT inference averages the probabilities over a sequence of random models generated using SampleSAT
- It draws heavily on the work of Pedro Domingos et al. and the Alchemy system
- The main differences are that PCE has subsorts, and is online and can run interactively or as an XML-RPC server
- On the other hand, Alchemy is more mature and includes machine learning components and several other features

MCSAT (Poon & Domingos 2006)

- MCSAT generates a sequence of models $x^{(0)}, \ldots, x^{(i)}, \ldots$
- Hard clauses are facts (and negations, by closed-world assumption) and clauses with maximal weight
- M is a set of satisfied clauses

MCSAT(*clauses*, *weights*, *num-samples*)
  $x^{(0)} \leftarrow$ Satisfy(hard *clauses*)   (WalkSAT)
  for $i \leftarrow 1$ **to** *num-samples* **do**
    $M = \emptyset$
    for all $c_k \in$ *clauses* satisfied by $x^{(i-1)}$ **do**
      with probability $1 - e^{-w_k}$ add c_k to M
    end for
    Sample $x^{(i)} \sim U_{SAT(M)}$ (Random model for set M)
    (SampleSAT)
  end for

WalkSAT Algorithm (Selman et al. 1996)

WalkSAT(*clauses*, *max-tries*, *max-flips*, *p*)
  for $i \leftarrow 1$ **to** *max-tries* **do**
    solution = random truth assignment
    for $j \leftarrow 1$ **to** *max-flips* **do**
      if all *clauses* satisfied **then**
        return *solution*
      $c \leftarrow$ random unsatisfied clause
      with probability *p*
        flip a random variable in *c*
      else
        flip variable in *c* that maximizes
          number of satisfied *clauses*
  return failure

SampleSAT (Wei et al. 2004)

```
SampleSAT(clauses, max-tries, max-flips, p, sa-p, sa-temp)
  for i ← 1 to max-tries do
    solution = random truth assignment
    for j ← 1 to max-flips do
      with probability 1 − sa-p
        if all clauses satisfied then
          return solution
        c ← random unsatisfied clause
        with probability p
          flip a random variable in c
        else
          flip variable in c that maximizes
            number of satisfied clauses
      else
        perform simulated annealing step with sa-temp
  return failure, best solution found
```

PCE Example: Boy Born on Tuesday

An acquaintance tells you she has two children, one is a boy born on tuesday. What is the probability she has two boys?

```
sort Day;
sort Child;
const Mo, Tu, We, Th, Fr, Sa, Su: Day;
const A, B: Child;
predicate boy(Child) hidden;
predicate born_on(Child, Day) hidden;

# Every child must be born on one and only one day
add [c] born_on(c, Mo) or born_on(c, Tu) or born_on(c, We)
     or born_on(c, Th) or born_on(c, Fr) or born_on(c, Sa)
     or born_on(c, Su);
add [c, d1, d2] (born_on(c, d1) and born_on(c, d2)) implies d1 = d2;
add (born_on(A, Tu) and boy(A)) or (born_on(B, Tu) and boy(B));

mcsat_params 100000, .5, 5, .05, 100, 5;
ask (boy(A) and boy(B));
```

PCE Example: Boy Born on Tuesday

An acquaintance tells you she has two children, one is a boy born on tuesday. What is the probability she has two boys?

```
sort Day;
sort Child;
const Mo, Tu, We, Th, Fr, Sa, Su: Day;
const A, B: Child;
predicate boy(Child) hidden;
predicate born_on(Child, Day) hidden;

# Every child must be born on one and only one day
add [c] born_on(c, Mo) or born_on(c, Tu) or born_on(c, We)
     or born_on(c, Th) or born_on(c, Fr) or born_on(c, Sa)
     or born_on(c, Su);
add [c, d1, d2] (born_on(c, d1) and born_on(c, d2)) implies d1 = d2;
add (born_on(A, Tu) and boy(A)) or (born_on(B, Tu) and boy(B));

mcsat_params 100000, .5, 5, .05, 100, 5;
ask (boy(A) and boy(B));
```

Which bowl is the cookie from?

Two bowls: A and B, A has 10 chocolate and 30 plain cookies, B has 20 of each. A bowl is picked at random, followed by a cookie picked at random, which is plain. Which bowl is the cookie from?

```
sort Bowl;
const A, B: Bowl;
sort Color;
const plain, choc: Color;
predicate bowl(Bowl) hidden;
predicate cookie(Color) hidden;
add ~bowl(A)  or ~bowl(B);
add ~cookie(plain) or ~cookie(choc);
add bowl(A) or bowl(B);
add bowl(A) .693;
add bowl(B) .693;
add bowl(A) => cookie(plain) 1.386; # 75%
add bowl(A) => cookie(choc)   .288; # 25%
add bowl(B) => cookie(plain)  .693; # 50%
add bowl(B) => cookie(choc)   .693; # 50%
add cookie(plain);
mcsat_params 200000, .5, 5, .05, 100, 5;
mcsat; dumptable atom;
```

Which bowl is the cookie from?

Two bowls: A and B, A has 10 chocolate and 30 plain cookies, B has 20 of each. A bowl is picked at random, followed by a cookie picked at random, which is plain. Which bowl is the cookie from?

```
sort Bowl;
const A, B: Bowl;
sort Color;
const plain, choc: Color;
predicate bowl(Bowl) hidden;
predicate cookie(Color) hidden;
add ~bowl(A)  or ~bowl(B);
add ~cookie(plain) or ~cookie(choc);
add bowl(A) or bowl(B);
add bowl(A) .693;
add bowl(B) .693;
add bowl(A) => cookie(plain) 1.386; # 75%
add bowl(A) => cookie(choc)   .288; # 25%
add bowl(B) => cookie(plain)  .693; # 50%
add bowl(B) => cookie(choc)   .693; # 50%
add cookie(plain);
mcsat_params 200000, .5, 5, .05, 100, 5;
mcsat; dumptable atom;
```

Monty Hall Problem

Three doors, behind one of which is a car. Contestant chooses door A, and is shown another door, which is empty. She is then offered the opportunity to switch. Should she?

```
sort Door;
const a, b, c: Door;
sort Switch;
const switch: Switch;
predicate car(Door) hidden;
predicate win(Switch) hidden;
add car(a) or car(b) or car(c);
add ~car(a) or ~car(b);
add ~car(b) or ~car(c);
add ~car(c) or ~car(a);
add car(a) implies ~win(switch);
add car(b) implies win(switch);
add car(c) implies win(switch);
mcsat_params 200000, .5, 5, .05, 100, 5;
mcsat; dumptable atom;
```

Monty Hall Problem

Three doors, behind one of which is a car. Contestant chooses door A, and is shown another door, which is empty. She is then offered the opportunity to switch. Should she?

```
sort Door;
const a, b, c: Door;
sort Switch;
const switch: Switch;
predicate car(Door) hidden;
predicate win(Switch) hidden;
add car(a) or car(b) or car(c);
add ~car(a) or ~car(b);
add ~car(b) or ~car(c);
add ~car(c) or ~car(a);
add car(a) implies ~win(switch);
add car(b) implies win(switch);
add car(c) implies win(switch);
mcsat_params 200000, .5, 5, .05, 100, 5;
mcsat; dumptable atom;
```

PCE Application: Machine Reading

- PCE is currently being used in a machine-reading project managed by SRI
- FAUST (Flexible Acquisition and Understanding System for Text)
- Large project involving SRI, Stanford, PARC, UMass, UIUC, and UWashington
- PCE will act as a “harness”, accepting weighted assertions and rules from various classifiers and learners

PCE Application: NFL Articles

An early use case for FAUST is a corpus of NFL newspaper articles

> *With Favre throwing the best-looking pinpoint passes this side of Troy Aikman and with receiver Robert Brooks doing a great impression of Michael Irvin and with the Packers' defense playing like, well, like themselves, Green Bay routed Philadelphia, 39 to 13.*

Task is to determine which teams played, who won and who lost, and what was the final score

Related Work on MLNs

- Alchemy: Markov logic networks in C++ (P. Domingos, Univ. of Washington)
- Markov TheBeast: Markov logic networks in Java (S. Riedel)
- ProbCog: Probabilistic Cognition for Cognitive Technical Systems
  Markov logic networks in Python and Java (Dominik Jain)
- PyMLNs: inference and learning tools for MLNs in Python (Dominik Jain)
- Incerto: A Probabilistic Reasoner for the Semantic Web based on Markov Logic (Pedro Oliveira)

Future Work

- Lazy MCSAT is an algorithm developed by Domingos that only creates instances as needed
- Make use of conditionals to make the conditional probability calculation more convenient
- Allow relational constraints, for example, that a predicate is functional
- Extend rules to work with fully observable predicates
- Allow probabilities to be given in place of weights - both are useful
- Non-Boolean Random Variables
- Weight and rule learning from training data

Bibliography

- Matthew Richardson and Pedro Domingos.
  Markov logic networks.
  Mach. Learn., 62(1-2):107–136, 2006.

- Bart Selman, Henry Kautz, and Bram Cohen.
  Local search strategies for satisfiability testing.
  In *DIMACS Ser. in Discr. Math. and Theor. Comp. Sci.*, pages 521–532, 1995.

- Wei Wei, Jordan Erenrich, and Bart Selman.
  Towards efficient sampling: exploiting random walk strategies.
  In *AAAI'04*, pages 670–676. AAAI Press, 2004.

- Hoifung Poon and Pedro Domingos.
  Sound and efficient inference with probabilistic and deterministic dependencies.
  In *AAAI'06*, pages 458–463. AAAI Press, 2006.

- Parag Singla and Pedro Domingos.
  Memory-efficient inference in relational domains.
  In *AAAI'06*, pages 488–493. AAAI Press, 2006.

- Hoifung Poon, Pedro Domingos, and Marc Sumner.
  A general method for reducing the complexity of relational inference and its application to MCMC.
  In *AAAI'08*, pages 1075–1080. AAAI Press, 2008.

Conclusions

- Probabilistic inferencing has always been important for domains with uncertainty: planning, robotics, natural language processing, etc.
- It is becoming important even for formal methods, as a part of certifying reliability.
- PCE is a robust and scalable tool for probabilistic inference
- We invite you to try it out and give us feedback
- PCE is open source, and will be available soon as a component of the Personal Assistant that Learns (PAL) Framework at `http://pal.sri.com`

Bruno Dutertre was involved in early design and implementation