# Efficient Inference in Probabilistic Consistency Engine (PCE)

Shangpu Jiang
Supervised by Dr. Natarajan Shankar

September 26, 2013

## Markov Logic

- A probabilistic first-order logic (FOL)
- Knowledge Base (KB) is a set of weighted FOL formulas $W = \{\ldots(w_i, F_i)\ldots\}$
- The probability of a truth assignment $x$ to the ground atoms:

$$\Pr(X = x|w) = \frac{1}{Z(w)}\exp(\sum_i w_i n_i(x))$$

where $w_i$ is the weight of $F_i$ (the $i$th formula in the KB) and $n_i(X)$ is the number of true groundings of $F_i$

## MAP/MPE Inference: MaxWalkSAT (Kautz et al. 1997)

- MAP inference is a weighted satisfiability problem (MAX-SAT), which can be solved by MaxWalkSAT
- Tseitin-Transformation for CNF formulas with weight, e.g.,
  $a \wedge (\neg b \vee c) : w$
  $\Rightarrow (r : w) \wedge (\neg r \vee a) \wedge (\neg r \vee \neg b \vee c)$

## MAP/MPE Inference: MaxWalkSAT (Kautz et al. 1997)

```
MaxWalkSAT(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 minimizes the total cost
      if solution
  return failure, best solution found
```

## Probabilistic Inference: MC-SAT (Poon & Domingos 2006)

```
MCSAT(*clauses*, *weights*, *num-samples*)
    $x^{(0)} \leftarrow$ Satisfy(hard *clauses*)
    **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 $M$)
    **end for**
```

## Problem with MC-SAT

- $(P(x) : 10) \wedge (\neg P(x) : 9)$
- Proposed solution: Add a small perturbation to every sample before choosing the formulas for the next sample
- Still does not reflect the difference of the weights

## Problem with MC-SAT

- $(P(x) : 10) \wedge (\neg P(x) : 9)$
- Proposed solution: Add a small perturbation to every sample before choosing the formulas for the next sample
- Still does not reflect the difference of the weights
- Other solutions: a mixture of MC-SAT and Gibbs sampling

## Lazy MaxWalkSAT

- Most groundings of a predicate take a default truth value, only a few need to be instantiated into memory. e.g., $Sm(x)$
- Most groundings of a clause is satisfied by default. e.g.,

$$Fr(x, y) \wedge Sm(x) \Rightarrow Sm(y)$$

- Only unsatisfied clauses are instantiated, e.g., $Fr(A, B)$, given $Sm(A)$, or $Sm(B)$
- No need for initial randomization

## Lazy MC-SAT

- Calls Lazy WalkSAT and Lazy SampleSAT
- In Lazy SampleSAT, we need to randomize 1-hop neighborhood of active atoms (different from Lazy WalkSAT)
- When a clause is activated, we need to determine its membership of the current set of formulas immediately

## Problems with Lazy MC-SAT

- Sometimes everything will be grounded finally, e.g., $Fr(x, y) \wedge Sm(x) \Rightarrow Sm(y)$
- Default value assumption is violated: the default probability is usually not $0/1$
  - When a prior is imposed, e.g., for $\neg Sm(x) : 1$ everything needs to be instantiated to achieve sufficient accuracy.
  - The grounding strategy for MaxWalkSAT is not correct. e.g., when $Fr(A, B)$, given $Sm(A)$ or $Sm(B)$

## 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?*

- $\Pr(boy(A) \land boy(B))$ doesn't converge to the correct answer (0.481) as the number of samples goes up
- This suggests SampleSAT does not generate absolutely uniform models
- sa_probability influences the uniformity

```
|  0 | 0.5030 | (boy(A)) & (boy(B))
|  1 | 0.0340 | (boy(A)) & (boy(B)) & (born_on(A, Tu)) & (born_on(B, Tu))
|  2 | 0.0388 | (boy(A)) & (boy(B)) & (born_on(A, Tu)) & (born_on(B, Mo))
|  3 | 0.0383 | (boy(A)) & (boy(B)) & (born_on(A, Tu)) & (born_on(B, We))
|  4 | 0.0385 | (boy(A)) & (boy(B)) & (born_on(A, Tu)) & (born_on(B, Th))
|  5 | 0.0384 | (boy(A)) & (boy(B)) & (born_on(A, Tu)) & (born_on(B, Fr))
|  6 | 0.0364 | (boy(A)) & (boy(B)) & (born_on(A, Tu)) & (born_on(B, Sa))
|  7 | 0.0402 | (boy(A)) & (boy(B)) & (born_on(A, Tu)) & (born_on(B, Su))
|  8 | 0.0390 | (boy(A)) & (boy(B)) & (born_on(A, Mo)) & (born_on(B, Tu))
|  9 | 0.0381 | (boy(A)) & (boy(B)) & (born_on(A, We)) & (born_on(B, Tu))
| 10 | 0.0396 | (boy(A)) & (boy(B)) & (born_on(A, Th)) & (born_on(B, Tu))
| 11 | 0.0417 | (boy(A)) & (boy(B)) & (born_on(A, Fr)) & (born_on(B, Tu))
| 12 | 0.0393 | (boy(A)) & (boy(B)) & (born_on(A, Sa)) & (born_on(B, Tu))
| 13 | 0.0407 | (boy(A)) & (boy(B)) & (born_on(A, Su)) & (born_on(B, Tu))
| 14 | 0.0353 | (boy(A)) & (~boy(B)) & (born_on(A, Tu)) & (born_on(B, Mo))
| 15 | 0.0366 | (boy(A)) & (~boy(B)) & (born_on(A, Tu)) & (born_on(B, Tu))
| 16 | 0.0333 | (boy(A)) & (~boy(B)) & (born_on(A, Tu)) & (born_on(B, We))
| 17 | 0.0366 | (boy(A)) & (~boy(B)) & (born_on(A, Tu)) & (born_on(B, Th))
| 18 | 0.0335 | (boy(A)) & (~boy(B)) & (born_on(A, Tu)) & (born_on(B, Fr))
| 19 | 0.0361 | (boy(A)) & (~boy(B)) & (born_on(A, Tu)) & (born_on(B, Sa))
| 20 | 0.0361 | (boy(A)) & (~boy(B)) & (born_on(A, Tu)) & (born_on(B, Su))
| 21 | 0.0339 | (~boy(A)) & (boy(B)) & (born_on(A, Mo)) & (born_on(B, Tu))
| 22 | 0.0376 | (~boy(A)) & (boy(B)) & (born_on(A, Tu)) & (born_on(B, Tu))
| 23 | 0.0357 | (~boy(A)) & (boy(B)) & (born_on(A, We)) & (born_on(B, Tu))
| 24 | 0.0371 | (~boy(A)) & (boy(B)) & (born_on(A, Th)) & (born_on(B, Tu))
| 25 | 0.0355 | (~boy(A)) & (boy(B)) & (born_on(A, Fr)) & (born_on(B, Tu))
| 26 | 0.0346 | (~boy(A)) & (boy(B)) & (born_on(A, Sa)) & (born_on(B, Tu))
| 27 | 0.0351 | (~boy(A)) & (boy(B)) & (born_on(A, Su)) & (born_on(B, Tu))
```

## Example: Social Network

```
sort Person;
const Ann, Bob, Carl, Dee, Earl, Fran: Person;
predicate Fr(Person, Person) direct;
predicate Sm(Person) indirect;

assert Fr(Ann, Bob);
assert Fr(Bob, Carl);
assert Fr(Carl, Dee);
assert Fr(Dee, Earl);
assert Fr(Earl, Fran);
add [x, y] Fr(x, y) and Sm(x) implies Sm(y) 5;

ask Sm(Fran);
mcsat; dumptable atom;
```

## Example: Social Network

Eager inference:

```
| 36 |     3316 |     10000 | 0.3316 | Sm(Ann)
| 37 |     4213 |     10000 | 0.4213 | Sm(Bob)
| 38 |     4826 |     10000 | 0.4826 | Sm(Carl)
| 39 |     5296 |     10000 | 0.5296 | Sm(Dee)
| 40 |     5717 |     10000 | 0.5717 | Sm(Earl)
| 41 |     6731 |     10000 | 0.6731 | Sm(Fran)
```

Lazy inference:

```
| 5 |     5037 |     10000 | 0.5037 | Sm(Fran)
```

## Text Classification (WebKB)

```
sort word;
sort page;
sort class;

predicate HasWord(word,page) direct;
predicate Topic(class,page) indirect;

const 'abstract', ...: word;
const 'http://ccwf.cc.utexas.edu/~hksa/', ...: page;
const Course, Department, Faculty, Person, ResearchProject, Staff, Student : class;

assert HasWord('abstract','ftp://ftp.cs.utexas.edu/pub/bshults/ATP-tech-reports/INDEX.html');
...

add [a1] HasWord('abstract',a1) implies Topic(Course,a1) -0.192907;
add [a1] HasWord('academ',a1) implies Topic(Course,a1) 0.24151;
...

ask [a1,a2] Topic(a1,a2);
mcsat; dumptables qinst;
```

- A mixture of MC-SAT and Gibbs sampling to alleviate the problem of Markov logic with opposite high determinism
- Correct activation of associated rules of a given atom in Lazy MC-SAT
- A scheme of choosing a neighborhood network of *evidence* and *queries* for efficient inference. e.g., $Fr(A, B) : 100$, $Fr(A, C) : 0.01$, $\Pr(Sm(A)) = ?$