CS267A: Probabilistic Programming and Relational Learning **Spring 2020**

Homework 6: Due Monday May 25, 11:59PM

Instructions: Upload one file to CCLE: a PDF typeset using LaTeX containing your solutions. No late submissions will be accepted. See the syllabus for policies about collaboration and academic honesty.

Problem 1

Assume we have a probabilistic database with four relations: $R(r)$, $S(s_1, s_2)$, $U(u)$, and $T(t)$. Compute each of the following queries symbolically using the lifted inference rules as we did in class, or state that it is not possible. You can find the lifted inference rules on Page 266 of [1].

For example, if the query were $\Pr(\exists x.R(x))$, you would write:

$$\begin{aligned} P(Q) &= 1 - P(\neg Q) \\ &= 1 - P(\neg \exists x R(x)) \\ &= 1 - P(\forall x \neg R(x)) \\ &= 1 - \prod_x P(\neg R(x)) \end{aligned}$$

The final answer is the most important, but you can include intermediate steps (which correspond to individual applications of the lifted inference rules) for partial credit.

1. $\Pr(\exists x.R(x) \wedge T(x))$
2. $\Pr(\exists x.\exists y.S(x,y) \wedge R(x))$
3. $\Pr((\exists x.\exists y.R(x) \wedge S(x,y) \wedge T(y)) \vee (\exists x.U(x)))$
4. $\Pr(\exists x_1.\exists x_2.\exists y_1.\exists y_2.R(x_1) \wedge S(x_1,y_1) \wedge T(x_2) \wedge S(x_2,y_2))$
5. $\Pr((\exists x_1.\exists y_1.R(x_1) \wedge S(x_1,y_1)) \vee (\exists x_2.\exists y_2.T(y_2) \wedge S(x_2,y_2))$
6. $\Pr((\exists x_1.\exists y_1.R(x_1) \wedge S(x_1,y_1)) \vee (\exists x_2.\exists y_2.S(x_2,y_2) \wedge T(y_2)) \vee (\exists x_3.\exists y_3.R(x_3) \wedge T(y_3))$
   (*Hint*: you may need to symbolically cancel some queries)

Problem 2

Recall the following query from class:

$$H_0 = \exists x.\exists y.S(x) \wedge F(x,y) \wedge R(y) \tag{1}$$

In class, we showed that evaluating H_0 for an arbitrary database is $\#P$-hard in the size of the database. We proved this by reduction to the positive partitioned 2-CNF (#PP2CNF) counting problem, for which we now give a formal definition.

Definition 1. *A 2-CNF is a CNF where each clause has exactly 2 literals. A* positive partitioned *2-CNF is a 2-CNF where variables are partitioned into 2 sets* $\mathbf{X}, \mathbf{Y}, \mathbf{X} \cap \mathbf{Y} = \emptyset$*, and every clause is of the form* $(x \vee y)$ *with* $x \in \mathbf{X}$ *and* $y \in \mathbf{Y}$*. Finally, #PP2CNF is the problem of counting how many models a PP2CNF formula has.*

For each of the following PP2CNF formulae, give three tables $S(x)$, $F(x, y)$, and $R(y)$ such that evaluating H_0 on these tables can be used to compute the model count of the formula:.

1. $f = x_1 \vee y_1$.

   Hint: What is the model count for this formula? Since you can compute it by hand, use it to test your answer.

2. $f = (x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3) \wedge (x_1 \vee y_3)$.

Hint: Represent the x_i variables as tuples in table $S(x)$, the y_i variables as tuples in table $R(y)$, and each clause as a tuple in $F(x, y)$. How do you relate "a variable is true" to the presence of a tuple in S or R? How do you relate "a clause is true" to the presence of a tuple in F? How can you choose the weights of each tuple so that the query can be used to compute (but *does not necessarily equal*) the model count?

Problem 3

It is well-known that solving the satisfiability problem is NP-complete. Suppose we show that a SAT-solver can be used to solve a Sudoku puzzle. Does this mean that solving Sudoku puzzles is NP-hard? Why or why not?

Problem 4

The #PP3CNF counting problem is defined as follows:

Definition 2. *A 3-CNF is a CNF where each clause has exactly 3 literals. A* positive partitioned *3-CNF is a 3-CNF whose variables, labeled* $\mathbf{V}$*, are partitioned into 3 disjoint sets* $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$*, i.e.:*

$$\mathbf{V} = \mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}, \quad \mathbf{X} \cap \mathbf{Y} = \mathbf{Y} \cap \mathbf{Z} = \mathbf{X} \cap \mathbf{Z} = \emptyset$$

and each clause contains exactly one positive literal from $\mathbf{X}$*,* $\mathbf{Y}$*, and* $\mathbf{Z}$*. Finally,* #PP3CNF *is the problem of counting the number of satisfying assignments (models) of a PP3CNF formula.*

Is the #PP3CNF problem #P-hard? Explain why or why not.

References

[1] Van den Broeck, Guy and Suciu, Dan. *Query Processing on Probabilistic Data: A Survey*. 2017. `http://web.cs.ucla.edu/~guyvdb/papers/VdBFTDB17.pdf`