Capita Selecta AI (Probabilistic Programming) 2016-2017

Inference for SRL

The goal of the Probabilistic Programming assignment is to build your own state-of-the-art inference engine that can answer probabilistic queries for probabilistic graphical models such as Bayesian networks and statistical relational models such as ProbLog.

- You can work alone or in teams of two students. Do not share results between teams.
- Send a <yourlastnames>.zip file containing (1) a report describing your approach and results in pdf and (2) executable code with a readme file to wannes.meert@cs.kuleuven.be by March 31, 2017.
- Grades for this part are fully determined by the submission (no presentation).
- Questions can be asked in the Toledo discussion forum.

1 Probabilistic Inference Using Weighted Model Counting (4/10)

One of the most performant techniques to compute the marginal or conditional probability of a query given a probabilistic graphical model (PGM) is to reduce the problem to weighted model counting (WMC). This entails that a PGM such as a Bayesian network is represented as a propositional knowledge base in conjunctive normal form (CNF) with weights associated to the propositional variables.

1.1 PGM to CNF (1/4)

Read about this approach in Chavira and Darwiche [1] and familiarize yourself with ENC1 (the encoding of Chavira and Darwiche) and ENC2 (the encoding of Sang, Beam and Kautz).

In this task you are asked to encode the earthquake Bayesian network on `https://dtai.cs.kuleuven.be/problog/tutorial/basic/02_bayes.html#using-multi-valued-annotated-disjunctions` as a CNF (the final example for the given queries and including the evidence). You can do this manually or write an automatic convertor.

Task 1.1.1 Write the encoding for the earthquake network as CNF and associated weights using ENC1.

Task 1.1.2 Write the encoding for the earthquake network as CNF and associated weights using ENC2.

1.2 SRL to CNF (1/4)

ProbLog, a Statistical Relational Learning formalism, is a generalization of PGM that allows one to express complex relations. Similar to PGMs, probabilistic inference for ProbLog can be reduced to a Weighted Model Counting task. Read about this approach in Fierens et al. [3] (you can ignore cyclic rules).

In this task you will encode the probabilistic graph on `https://dtai.cs.kuleuven.be/problog/tutorial/basic/04_pgraph.html` as a CNF (after grounding for both queries). You can do this manually or write an automatic convertor.

Task 1.2.1 Write the encoding for the probabilistic graph as CNF and associated weights using the ProbLog encoding.

1.3 Weighted Model Counting (1/4)

WMC can be performed by applying a search algorithm on the CNF or by compiling the CNF into a structure on which WMC can be performed in polynomial time with respect to the size of the structure. An advantage of the standardization to CNF is that multiple model counters can be applied, each with their own advantages and disadvantages. An exhaustive overview of model counters is avialable on `http://beyondnp.org/pages/solvers/model-counters-exact/`.

Task 1.3.1 Select two or three exact weighted model counters and apply them to the three CNFs you have built.

Task 1.3.2 Explain briefly the main theoretical differences between the three weighted model counters.

Task 1.3.3 Create an overview of the computational requirements (e.g. runtime, memory).

1.4 Knowledge Compilation (1/4)

In this task we will focus on compilation of the CNF to an SDD structure as explained in Oztok and Darwiche [4] (or alternatively Choi, Kisa, and Darwiche [2]). SDDs make use of the concept *vtree* to figure out the order in which variables are dealt with. The choice of this vtree has a large impact on the size of the circuit and thus the cost of performing inference. You can make use of the miniC2D package[1] to translates a CNF into an SDD and perform weighted model counting (or alternatively the SDD package itself[2]).

Task 1.4.1 For each of the CNFs you build, decribe the vtree that, in your experiments, gives the most compact circuit. You can use the heuristics available in miniC2D (flags `-m/-t/...`) or SDD (flags `-m/-t/-r/...`) to generate different vtrees or build your own (flag `-v`).

Task 1.4.2 Is there a pattern in what makes a good vtree (e.g. is minfill always better than balanced or right-linear).

2 Build an Inference Engine (4/10)

In this part you are tasked with implementing your own pipeline to perform inference on a ProbLog program. The challenge is to outperform the ProbLog implementation available on the website[3] (it is possible). Your pipeline should implement the following steps (you are allowed to deviate as long as you can handle the tasks):

1. If the input is a Bayesian network, translate it to a ProbLog program. You can use the conversion scripts that are available as part of the ProbLog distribution or write your own.
2. Ground the ProbLog program. You are allowed to use the `ground` command and its options available as a command in the ProbLog implementation.
3. Transform the ground ProbLog program to a CNF (or propositional formula if you prefer this) and list of weights for the binary variables in the CNF. You can use any encoding or variation you wish. You can assume that the ground ProbLog has only conjunctions of literals in the body (no complex parsing required).
4. Feed the CNF to the miniC2D (or SDD) toolbox. You can use any set of options or vtree.
5. Compute the (conditional) probability of a given query.
6. Report (a) the probability, (b) total runtime and runtime of the separate parts, (c) number of variables and lines in the CNF, (d) statistics on the depth and branching factor of the vtree, (e) number of edges and nodes in the circuit.

Task 2.1 Implement the pipeline in your preferred programming language (if no preference, use Python).

Task 2.2 Apply your pipeline to the examples used in previous tasks.

Task 2.3 Apply your pipeline to the example on `https://dtai.cs.kuleuven.be/problog/tutorial/tutslides/04_bayesian_learning.html`.

Task 2.4 Apply your pipeline to the alarm Bayesian network on `http://www.bnlearn.com/bnrepository/#alarm`. The public version of ProbLog uses an inefficient compilation strategy for this type of Bayesian networks, can you beat ProbLog? Try to compute the marginal probability of as many nodes in the network as possible and report timings (since there is no evidence, you can drop leaf nodes incrementally until your solution works).

[1] `http://reasoning.cs.ucla.edu/minic2d/`
[2] `http://reasoning.cs.ucla.edu/sdd/`
[3] `https://dtai.cs.kuleuven.be/problog`

3 Comparison with Approximate Inference (2/10)

In some situations it is infeasible to find an exact solution for a probabilistic query. In such cases an approximate model counter can be used. A list of tools is available at http://beyondnp.org/pages/solvers/model-counters-exact/.

Task 3.1 Alter your inference pipeline to allow for approximate inference (support at least two approximate model counters). Explain your changes.

Task 3.2 Perform an exhaustive comparison between your exact inference strategy and the approximate inference approach on the examples of the previous tasks. Look at accuracy of the results and the computational aspects.

References

[1] Mark Chavira and Adnan Darwiche. "On probabilistic inference by weighted model counting". In: *Artificial Intelligence* 172.6 (2008), pp. 772–799.

[2] Arthur Choi, Doga Kisa, and Adnan Darwiche. "Compiling Probabilistic Graphical Models using Sentential Decision Diagrams". In: *Proceedings of the 12th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*. 2013.

[3] Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. "Inference in probabilistic logic programs using weighted CNFs". In: *Theory and Practice of Logic Programming* 15 (2015).

[4] Umut Oztok and Adnan Darwiche. "A top-down compiler for sentential decision diagrams". In: *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*. 2015.