Introducing Probabilistic Programming Languages to a Computer Science Bachelor Course

Bart Stam
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
b.stam@student.utwente.nl

ABSTRACT
This paper is about probabilistic programming languages and focuses on the question if probabilistic programming languages would be a useful addition to an artificial intelligence course for bachelor computer science students. Specifically, it focuses on reasoning under uncertainty using Bayesian networks. The author, who was unfamiliar with this type of languages up to this point, has followed the artificial intelligence course in questions a few years ago.

Keywords
Probabilistic Programming, Bayesian Networks, Education

1. INTRODUCTION
The use of probabilistic models in artificial intelligence has been much discussed and built upon in the past years. Many different techniques are utilized to model virtual agents, many of which are very complex and difficult to implement. Fairly recently, a new method to combat this problem has made an appearance, in the form of probabilistic programming languages.

Probabilistic Programming Languages are languages designed to describe probabilistic models and then perform inference on those models[4]. In theory, this can be done using a regular programming language. However, correctly and efficiently describing these models can be an extremely delicate process that the average programmer cannot realistically accomplish. This is where these new probabilistic languages may prove to be an important asset, as they allow everyone to use preprogrammed models by simply inserting variables.

The main focus of this paper is to determine whether or not these new probabilistic programming languages would be a good addition to an artificial intelligence course for bachelor students at The University of Twente. The course teaches about reasoning under uncertainty using Bayesian Networks. During the course, a practical about modeling naive Bayesian classifiers is offered to the students. In this paper, the possibility of an expansion to this practical or perhaps a new practical entirely is explored. Two probabilistic programming languages are selected and compared to each other as well as to one of the regular programming languages often used in the computer science bachelor.

The author, who was unfamiliar with these probabilistic programming languages before writing this paper, first does some research into what they offer in addition to regular programming languages. Then, a Bayesian network is programmed in all three, which is used to reason about if there would be any merit in constructing a practical exercise for the students using these languages. The goal of this new practical would be to teach students about the use of probabilistic programming languages, as well as make the artificial intelligence course more enjoyable for them. The findings of the author will partially be motivated by his personal experience of the artificial intelligence course.

1.1 Bayesian Networks
A Bayesian network is a probabilistic model that represents the conditional dependencies between certain variables using a directed acyclic [7][2][3]. These Bayesian networks are used often in real world applications. A popular use for them is email spam filters. Given a set of spam emails and a set of regular emails, an analysis is done on the most used words in either. This yields a probability that, given a spam email or regular email, a certain word is used. Using a Bayes network it is possible to now, given a certain word, calculate the probability of the email being either a spam email or a regular email. Another example would be diseases and their symptoms, and calculating the probability of a patient suffering from a certain disease given their symptoms.

1.2 Probabilistic Programming Languages
A probabilistic programming language can be a standalone programming language or an augmentation to an already existing programming language. A probabilistic programming language in itself cannot necessarily achieve things that would be unachievable in a regular language. However, it does provide efficient and easy to use implementation of certain statistical models such as Bayesian networks. In essence, probabilistic programming languages were introduced to allow more programmers to utilise these models without having to understand their inner workings.

The exact definition of what counts as a probabilistic programming language is still debated. Some say any language that provides any sort of implementation of a statistical model qualifies as probabilistic. Even a simple function that generates random numbers would be sufficient. In this paper however, we will only be looking at languages that explicitly provide means to work with Bayesian networks.
1.3 The University Programme
The rationale behind this research paper is the intention to implement a course on probabilistic programming language into a computer science university programme. To fully understand where the focus will be when testing these probabilistic languages, a brief description of the university programme is required.

The specific programme that we refer to is the 6th quartile of the technical computer science bachelor at The University of Twente. It is a 10 week long course named Intelligent Interaction Design. In it, several subjects are taught, among which an intro to artificial intelligence, machine learning, and statistics.

In the past, the Artificial Intelligence part of the programme was larger than it is currently. At present, the only inference method taught in the programme is the enumeration method. A number of years ago, the Markov Chain Monte Carlo method was also taught[10].

At the end of the artificial intelligence course in this module, a small project on naive Bayesian classifiers is offered. The students of the course implement such a network to determine whether a given e-mail is spam or not spam.

1.4 Problem Statement
The University of Twente has expressed desire to educate students of their technical computer science bachelor about modern probabilistic programming languages. Which probabilistic programming languages the students should be taught about however is not clear and depends on several factors. Furthermore, the exact advantages and disadvantages of using a probabilistic programming language over regular language in this university programme are still unclear.

If probabilistic programming languages turn out to be a suitable addition to the Intelligent Interaction Design module, the idea is to form a practical exercise to introduce the students to them. The criteria of this practical exercise are that it must offer new subject matter, outside of just being about probabilistic programming languages, and ideally it would offer the students a enjoyable way to learn more about Bayesian networks.

Introducing a programming language designed for a specific task to introduce students to a certain subject is not a new idea for the computer science bachelor. In the Programming Paradigms module, prolog is used to introduce students to logical knowledge models and deductive reasoning. Prolog is a language in which logical models are very straightforward to model, and using them for deductive reasoning is relatively effortless. The question is whether probabilistic programming language can fulfill the same purpose in the Intelligent Interaction Design module as prolog does in the Programming Paradigms module.

1.5 Research Questions

1. What are the key differences between the chosen probabilistic programming languages, and does this make them equally or more qualified to be used in education than a regular programming language?

2. Based on the author’s personal experience with the artificial intelligence course, which probabilistic programming languages offers the most enjoyable opportunities for a new practical exercise?

3. Which probabilistic programming language would best fit into a practical exercise for the Intelligent Interaction Design module?

4. Can a probabilistic programming languages fulfill the same purpose as prolog does in the Programming Paradigms module?

2. METHOD OF RESEARCH
The answer to the first research question will serve as one of the means to answer the rest of the research questions. Probabilistic Programming languages in the form we see today are relatively a very recent technology, and so the answer to the first question will also be a means for the author to familiarize himself with them. This will be mostly be done by implementing an example Bayesian network in each of the languages and arguing about the process and result. The network that is implemented in each of the chosen languages, the sprinkler network, is described in this section below. The answer to this question specifically can mostly be found in sections 5, 6, and 7 of this paper.

For the answer to the second question, first a number of criteria are established that are necessary for a successful practical exercise. These criteria will be based on past practicals as well as the author’s personal experience with the artificial intelligence course and technical computer science bachelor as a whole. The probabilistic programming languages as well as the regular programming language are tested against these criteria. A description of the current final practical exercise of the artificial intelligence course is given in section 8 of this paper. The answer to this question specifically can mostly be found in section 8 of the paper.

The answer to the first question in combination with the answer to the second question will be the main reasons used to argue about the third research question. Both new subject matter as well as enjoyment of the practical are very important aspects. Based on the answer to this question, possible practical exercises are discussed that may not have been possible before the introduction of probabilistic programming languages. In the end, the author recommends one of the possibilities supported by his personal experience and findings in the paper. The answer to this question specifically is given in section 9.1 of this paper.

The answer to the fourth question is given by means of the answers to all the prior research questions. It is essentially the answer to the question if introducing probabilistic programming languages to the Intelligent Interaction Design course would be a good idea. The answer to this question specifically is given in section 9 of this paper, the conclusions.

The tested probabilistic programming languages in this paper are PyMC and BayesPy, both extensions to the regular programming language python. The regular programming languages used most in the computer science bachelor at The University of Twente are Java and python, but in recent years it has shifted more towards python. Because the possible expansion to the artificial intelligence course would comes in the form of a single practical, it would cost too much time to make students learn the syntax of a different language first. These are the reasons behind choosing languages that are extensions to python. PyMC and BayesPy both use a different inference method that could possibly influence the opportunities they offer.
for a new practical. Both their inference methods differ from the inference method currently applied in the artificial intelligence course, the enumeration method.

### 2.1 Sprinkler Network

The sprinkler network is a very simple model that is often used to familiarize students with the concept of Bayesian Networks[1]. It models the relation between a sprinkler, rain, and whether or not the grass is wet. The rain influences whether or not the sprinkler is on, and both the rain and the sprinkler influence whether or not the grass is wet. This model is also used in the *Intelligent Interaction Design* module to give an introduction to Bayesian Networks. Below is a figure depicting said network.

![Sprinkler Network](image)

To get an impression of the differences between the regular programming language Python and the probabilistic programming languages PyMC3 and BayesPy, implementations and application of inference for all three will be shown. The standard probabilities for this network that will be used in this paper are shown in tables 1 and 2.

While this network may seem very simple, the architecture is not much more complicated than that of the naïve Bayes classifier that students of the *Intelligent Interaction Design* module are currently tasked with implementing in the final practical. A description of the current final practical exercise can be found in section 8 of this paper.

### 3. RELATED WORK

First and foremost, *Artificial Intelligence: A Modern Approach* by Peter Norvig and Stuart J. Russell is the central reference work in the *Intelligent Interaction Design* module. This book describes all three of the enumeration methods applied by the chosen programming languages, as well as the sprinkler network and the naïve Bayesian classifiers. This book has an excellent explanation on reasoning under uncertainty, and also clearly exemplifies why the enumeration method currently used in the *Intelligent Interaction Design* module restricts the range of practical exercises that can be done severely. A quick summary of the reasoning under uncertainty chapter in the book is given in section 4 of this paper.

For *PyMC*, there exist numerous resources detailing its use. One of such is Cameron Davidson’s *Bayesian Methods for Hackers*[9]. Davidson’s book is mostly freely available and describes the use of Bayesian Networks as well as how to implement them using *PyMC*. Furthermore, *Bayesian Methods for Hackers* serves as a central resource for examples and explanations in the *PyMC* universe. This book gives a student friendly explanation of how to model simple networks in *PyMC*. Furthermore, it is freely available. It would be a good addition to the *Intelligent Interaction Design* module should *PyMC* be chosen as an expansion to it. For this paper, instructions on how to work with *PyMC* were taken from this book in addition to the *PyMC* documentation website.

*PyMC* uses Markov Chain Monte Carlo as its inference method. To get more insight into the workings of this particular algorithm, an explanation of it by Jeremy Kun is referenced[10]. *BayesPy* is not as developed as *PyMC* is, so related work about it is a lot more scarce. Instructions on how to work with *BayesPy* and implement a simple Bayesian network in it were mostly taken from its documentation website.

*BayesPy* uses an inference method unique compared to that of *PyMC* and the inference method used in the *Intelligent Interaction Design* module. The inference method used by *BayesPy* is called Variational Bayesian Approximation. Information about the specific use of this inference method was learned from the paper by the developer of *BayesPy* himself, Jaakko Luttinen: *Fast Variational Bayesian Linear State Space Model*[13].

Lastly, the paper *Variational Inference: a Review for Statisticians* by David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe compares Markov Chain Monte Carlo and Variable Bayesian Approximation as inference methods[8]. These are the inference methods used by *PyMC* and *BayesPy* respectively. This paper is used to make some arguments about which probabilistic programming language would be better suited for a possible new practical.

The above papers are used to argue about what the particular probabilistic programming languages would add to the artificial intelligence module. Their inference methods differ from each other, and in combination with their ease of use will most likely form the decisive factors in deciding which are fit for the *Intelligent Interaction Design* module.

### 4. REASONING UNDER UNCERTAINTY

In this section, a short summary of the *Reasoning under Uncertainty* chapter in Norvig and Russel’s *Artificial Intelligence: A Modern Approach* is given[15]. The purpose of this section is to outline why the enumeration method that is used exclusively in the *Intelligent Interaction Design* module inhibits its choice of practical exercises for the students, and why a different inference method could be a good addition.

A problem with first-order logic is that agents hardly ever have access to the full truth about their environment. Using their perception, they may not be able to determine certain factors about the environment that could be vital to determining the best choice of actions. Agents must therefore act under uncertainty. The qualification problem states that listing all conditions to guarantee for a
real world action to have its intended effect is an impossibility[14].

The example given in Norvig and Russel’s book is that of an agent wanting to drive someone to the airport to catch a flight. The agent considers a plan that involves them leaving the house 90 minutes before the flight departs and driving at a reasonable speed. Even though the airport is only 15 minutes away, the agent cannot say with certainty that they will arrive on time. A conclusion such as ‘we will arrive on time if we leave 90 minutes prior to take off’ cannot be made. Rather, only a weaker conclusion like ‘we will arrive on time if we leave 90 minutes prior to take off, there is traffic holding us up, we do not get into an accident, the car will not run out of gas, etc.’ can be made. Therefore, a logical agent would be unlikely to pick this option as it is not guaranteed to achieve its goal.

Nonetheless, let us assume the agent’s plan is in fact the best thing to do. This means that out of all the possible plans that may be executed by the agent, this particular plan maximizes the expected payoff. Calculating which plan is the best becomes more accurate the more (correct) prior probabilities are introduced. However, calculation speeds may drastically suffer from introducing more priors and relationship between probabilities.

The enumeration method used in the Intelligent Interaction Design method aims to precisely determine the probabilities of each possible plan given all the prior probabilities. The Markov Chain Monte Carlo and Variational Bayesian Approximation methods used by PyMC and BayesPy merely aim to make an accurate estimation of the probabilities, and thus finish their job much faster. Relating this to practical exercises that can be done in a reasonable time span, the enumeration method is not very feasible for larger networks. This is a big reason why the final practical exercise in the artificial intelligence course is implementing a naive Bayesian classifier rather than a more complex and more accurate Bayesian classifier.

In section 8 of this paper, the current final practical exercise of the artificial intelligence course is described, and the exercise of the artificial intelligence course is implemented in Python. As such, given the priors, it is completely accurate.

5. PYTHON

5.1 Introduction

Python is the standard programming language in the Computer Science bachelor at The University of Twente. As such, it is also used in the Intelligent Interaction Design module. Python in itself does not offer any support for Bayesian networks or performing inference on them. As such, it is all done manually by the students.

5.2 Sprinkler Network Example

As Python does not have default support for modeling networks, the sprinkler network must be modeled manually by the students. This can be done in many ways, as long as the prior probabilities are clear. One example would be to make a class representing a probability function that holds tuples of variables and their values. Its fields could look like this:

```python
event = ((W, True))
evidence = ((R, True), (S, False))
probability = 0.8
```

5.3 Inference Method

During the Intelligent Interaction Design module, inference in Python is done through a method called enumeration. Enumeration is essentially a brute force method that enumerates every possible situation to compute the probabilities. For example, the probability that the sprinkler was on given that the grass is wet would be computed as follows (using the probability and variables in table 1 and 2):

\[ P(R|W) = \frac{P(R,W)}{P(W)} \]

Computation of the numerator:

\[ P(R, W) = \sum_S P(R, W, S) \]
\[ P(R, W, S, R) = 0.8 \cdot 0.99 \cdot 0.2 = 0.1584 \]
\[ P(R, W, S, !R) = 0.99 \cdot 0.01 \cdot 0.2 = 0.00198 \]
\[ P(R, W, S) = 0.1584 + 0.00198 = 0.16038 \]

Computation of the denominator:

\[ P(W) = \sum_{S} \sum_{R} P(W) \]
\[ \sum_{S} \sum_{R} P(W) = P(W, S, !R) + P(W, !S, R) + P(W, S, R) + P(W, !S, !R) \]
\[ P(W, S, !R) = 0.00198 \]
\[ P(W, !S, R) = P(W|S, !R) \cdot P(1|R) = 0.0 \]
\[ P(W, !S, !R) = P(W|S, !R) \cdot P(S|R) \cdot P(!R) = 0.288 \]
\[ \sum_{S} \sum_{R} P(W) = 0.1584 + 0.00198 + 0.288 = 0.44838 \]

And finally:

\[ P(R|W) = \frac{P(R,W)}{P(W)} = \frac{0.16038}{0.44838} = 0.3577 = 35.77% \]

This enumeration method has taken into account all possible scenarios and their corresponding prior probabilities. As such, given the priors, it is completely accurate.

5.4 Discussion

Clearly, even for such a small network, computing certain probabilities given some evidence is still very computationally intensive. Still, when the networks do not have too many complex dependencies, this method of inference is still feasible, and has the upside that it guarantees completely accurate results.

Another advantage of this method is that it is easily understandable from a mathematics point of view, and can be written out relatively easily by hand. For educational purposes, this is of course a big plus.

6. PYMC

6.1 Introduction

PyMC is an open-source probabilistic programming framework with an intuitive and easily readable syntax that closely resembles the syntax statisticians use to describe models. The framework is fiscally sponsored by NumFOCUS.

PyMC is a widely used probabilistic programming language, which is a big plus when using it in educational programmes. Not only will aid in using the language be broadly available on the internet, it also increases the chances of the taught knowledge being applicable later on.
6.2 Sprinkler Network Example

Below is the code for the initialization of the probabilities for the network[6]. The probabilities are set through lambda functions because they are traced during inference. With PyMC, values not wrapped in lambda functions cannot be tracked for inference.

```python
# Initialization
observed_values = [1.]

rain = pymc.Bernoulli('rain', .2,
                         value=np.ones(len(observed_values)))
p_sprinkler = pymc.Lambda('p_sprinkler',
                          lambda rain: np.where(rain, .01, .4))

sprinkler = pymc.Bernoulli('sprinkler',
                          p_sprinkler,
                          value=np.ones(len(observed_values)))

p_grass_wet = pymc.Lambda('p_grass_wet',
                          lambda sprinkler, rain: np.where(rain, .99, .9)
                                      .where(rain, .8, 0.2))

grass_wet = pymc.Bernoulli('grass_wet',
                          p_grass_wet,
                          value=observed_values, observed=True)

model = pymc.Model([grass_wet, p_grass_wet, sprinkler, p_sprinkler, rain])
```

6.3 Inference Method

PyMC uses Markov Chain Monte Carlo, or MCMC for short, as its inference method[5]. MCMC is used for sampling probability distributions, and if often applied to Bayesian networks in order to approximate probabilities. In essence, it first makes an estimation of the dependencies between nodes in the network, and then gradually refines its estimation until it finds a fitting model.

Unlike the method used in the Intelligent Interaction Design module, Markov Chain Monte Carlo does not necessarily provide a completely accurate model. Either a maximum number of iterations, or the minimal acceptable error is given. The precision of the model depends on the number of steps that the algorithm is allowed to make.

Performing inference on the model we specified earlier with PyMC goes as follows.

```python
mcmc = pymc.MCMC(model)
mcmc.sample(2000)
```

The above snippet of code samples the network using 2000 iterations of the MCMC algorithm. After 2000 iterations, the results can be found below.

The probability that it rained, given that the grass is wet:
\[ P(R|W) = 0.345809971208 \]

The probability that the sprinkler was turned on, given that the grass is wet:
\[ P(S|W) = 0.658887710259 \]

The probability that the sprinkler was turned off and it did not rain, given that the grass is wet:
\[ P(\neg S, \neg R|W) = 0.212304894681 \]

6.4 Discussion

Setting up a model and performing inference on it using PyMC is extremely simple. After defining the prior probabilities it is merely a matter of configuring the amount of iterations the Markov Chain Monte Carlo algorithm must complete. The results, however, become more accurate with the number of iterations. With a limited number of iterations, the result will most likely be slightly off. This is evident when comparing the probability for \( P(R|W) \) found by the Markov Chain Monte Carlo algorithm to the probability for \( P(R|W) \) found by the enumeration method. Although the results are close, they are not equal.

However, the Monte Carlo Markov Chain method of inference is guaranteed to get closer to the correct probabilities the longer it is allowed to perform iterations[8]. This is an important aspect of the inference method, seeing as we do not want the algorithm to give incorrect answers in any situation. This is not always the case for Variational Bayesian Approximation, as is mentioned in the next section.

7. BAYESPY

7.1 Introduction

BayesPy provides tools for Bayesian inference in python. BayesPy in an open-source software developed between 2011 and 2016. Its functionality is somewhat more limited than that of PyMC, but it is still sufficient for what is on the schedule for the university module.

7.2 Sprinkler Network Example

Currently, the sprinkler network or any other network with probabilities close to 0 or 1 cannot be correctly implemented into BayesPy. The reason for this is that the method of inference used does not lend itself well to computing probabilities in such networks. However, the developer of BayesPy, Jaakko Luttinen, has said that the BayesPy team is currently working on exact inference for discrete graphs[12]. This would make the implementation of networks like the sprinkler network possible. The suggested syntax for such a network would according to Jaakko Luttinen would look like the following snippet of code[12].

```python
X = CategoricalGraph(
    { ('rain',): [0.8, 0.2],
      ('sprinkler', 'rain'): [(0.6, 0.4), (0.99, 0.01)],
      ('grass wet', 'sprinkler', 'rain'): [[1.0, 0.0], [0.2, 0.8]],
      (0.1, 0.9), [0.01, 0.99]]
)
```

7.3 Inference Method

BayesPy uses variational Bayesian approximation as its inference method[13]. This differs from PyMC, which uses Monte Carlo Markov Chains as its inference method. According to Jaakko Luttinen, the creator of BayesPy, it does not make sense to use variational Bayesian approximation on small networks[11].

Variational Bayesian approximation however is extremely efficient and quick at doing inference on larger networks. Supposedly, it is much quicker than Monte Carlo Markov
7.4 Discussion

Given the fact that networks handled in the artificial intelligence module are very unlikely to approach a complexity where variational Bayesian approximation is required to perform inference on them, and the fact that variational Bayesian inference is not guaranteed to approach the correct probabilities in a network, it seems inadvisable to introduce BayesPy to the module.

8. ADDITION OF A NEW PRACTICAL EXERCISE

8.1 Naive Bayesian Classifier

Currently, the final practical exercise that combines what the students have learned during the artificial intelligence course consists of implementing a naive Bayesian classifier. The purpose of this naive Bayesian classifier is to determine whether a given e-mail is spam or not spam. The students are given a database holding a large amount of E-mails that are already classified as either spam or not spam. Their job is to create a program to "feed" these e-mails to, that then uses the frequency of certain words in e-mails to assess if it is spam. The program must do so by constructing a Bayesian network with probabilities based on word frequency. The Bayesian network for such a naive e-mail classifier would look as follows:

In the imagine above, the C would be the class of the file, in this case spam or not spam. The nodes connected to the class would be words with a certain probability of being present in the e-mail. Words that have a high probability of being present in a spam e-mail and low probability of being present in a regular e-mail (or vice versa) can tell the classifier a lot about the e-mails class. For example, if an e-mail would contain the word "viagra", it is very likely to be spam.

Because all nodes in the network aside from the class node are only connected to a single other node, the enumeration method explained earlier in this paper can relatively quickly calculate the probability of an e-mail being spam or not spam. However, some words such as "not" do not make much sense without the words surrounding them, and as such should have connections to other words in the network for them to be meaningful. This is why this particular classifier is called "naive": there are no relationships between tokens used to classify an e-mail.

8.2 Criteria for a Successful Practical

This section is mostly written from the author’s personal experience with the Intelligent Interaction Design module and the bachelor programme as a whole.

In general, practicals that leave the students to use their own creativity are the most enjoyable. A good example of this would be the practical that is in place for the second module of the computer science bachelor, where the students are allowed to program an agent for a given computer game. The students are given some hints, but are completely free to choose how to implement the agent.

In a way, the naive Bayesian classifier praticum also allows this since students are free to temper with the network architecture to make it more accurate (and in a sense, less naive). However, how the network is made up was not the goal of the practical. Rather, the goal was to implement the enumeration inference method.

Furthermore, a practical should of course either strengthen the students’ knowledge about a portion of the subject matter, or it should expand the subject matter with new theory. The naive Bayesian classifier practical mainly does the first. After the students have made several exercises about performing inference using the enumeration method, they are tasked with actually implementing it and using it in practice.

If a new practical would be offered to students introducing new theory to the already existing subject matter, perhaps a new inference method or a more creative practical about network architecture could be made. Especially the latter seems to work well with the idea of allowing students to use their creativity and come up with their own ideas rather than following a set of instructions.

Lastly, from my experience with past projects in the computer science bachelor, practicals where students are tasked to create a program that competes with other students' programs are always seen as very motivating. Thankfully, Bayesian networks have many applications for artificial intelligence in games, and a possibility for a Bayesian game agent competition definitely exists.

8.3 Naive Bayesian Classifier Expansion

A possible expansion to implementing the naive Bayesian classifier would be to let students link certain adjectives to nouns, or use certain groups of words to classify e-mails and bring back the rate of false-positives or false-negatives. Moreover, if the students are allowed to design the architecture of the network rather than just calculate the probabilities, it would be a very interesting addition to the already present subject matter. The focus would then be on how Bayesian networks are actually structured rather than just the inference methods.

The above idea could also be applied to a Bayesian network practical about games, where the students would design a Bayesian game agent for a game and ultimately have their agents compete with each other. If the layout of the network is the focus rather than the inference method, many different applications for Bayesian network open themselves up to a small project or practical.

To make such a practical feasible, an inference method other than the enumeration methods has to be used. As can be seen from the example in section 5.3, inference calculations using the enumeration method can quickly become very computationally intensive. If students were allowed to design their own network architecture, chances are that a faster inference method would be required. This is where Markov Chain Monte Carlo or Variational Bayesian
Approximation could come in. The enumeration method may still be used for certain small sub tasks in the students’ implementation to make sure the answer is completely accurate, leaving them free to choose their inference method would add to the creative aspect of the practical.

Of course, both these inference methods, like the enumeration method, could be programmed in python by the students themselves. However, they are not as intuitive as the enumeration method and a significant portion of the students’ time would go into learning how the methods work and programming them. If the focus of the practical is on the layout of the Bayesian network, it would be a great help if the inference methods were already baked into the language as is the case with PyMC and BayesPy.

This is a strong argument for why probabilistic programming languages could be a good addition to the Intelligent Interaction Design module.

9. CONCLUSIONS

The answers to research questions 1 and 2 were mostly given in the sections 5 through 8. The key differences between python, PyMC, and BayesPy mostly lie in the way they allow the user to define a network, and their method of inference. In this paper, we have talked about about all three methods of inference used in these languages, and came to the conclusion that Markov Chain Monte Carlo and Variational Bayesian Approximation (used in the probabilistic programming languages) may be of use when looking to expand the practicals offered in the artificial intelligence course.

Three main criteria for a good practical exercise for the computer science bachelor were formulated. These three criteria were:

1. The practical must build upon subject matter previously taught in the course; or the practical must introduce some new theory or challenge not previously explored in the course
2. The practical must allow the students to be creative and use their own ideas rather than follow a set of guidelines. This mainly serves to make it more enjoyable.
3. The practical may be set up as a competition where students’ implementations of the agent can be measured against each other. This works to motivate the students to do well.

These criteria were used to answer the second research question on which probabilistic programming language offers an opportunity for an enjoyable new practical.

A new type of practical that was suggested in section 8 is one that is focused on the architecture of Bayesian networks rather than performing inference on them. In such a scenario, the networks produced by the students could be of such complexity that the enumeration inference method is no longer suitable. The inference methods used by PyMC and BayesPy, Monte Carlo Markov Chain and variational Bayesian approximation respectively, are much faster at performing inference on a complex network than the enumeration method is.

Of the three inference methods mentioned above, variational Bayesian approximation is the fastest by a margin, especially the version used in BayesPy. However, it is unlikely that the networks in the artificial intelligence course grow to a complexity where it would be required over say, Monte Carlo. If we consider the fact that variational Bayes is not guaranteed to approach the correct probabilities for a network, PyMC seems to be the better option of the two.

As for the final research question, it seems probabilistic programming languages could serve the same purpose for the Intelligent Interaction Design module as prolog does for the Programming Paradigms module in the sense that it takes the focus away from the implementation and the logic behind the inference method and allows the students to focus on a different aspect of the whole. In this case, that would be the architecture of the Bayesian networks and how that influences the results and computation times.

10. REFERENCES