Implicative Bayesian Reasoner for Java

Implementing a Bayesian Reasoning tool to support Classical Implication statements

Luke Neville
Computer Science Honours
University of Cape Town
Cape Town, South Africa
nvlluk001@myuct.ac.za

ABSTRACT

Implicative Bayesian Networks (IBNs) allow classical and defeasible implication statements to be added to a Bayesian Network. This allows additional dependencies to be added between variables in the network, increasing its expressivity [1]. A software tool, the Implicative Bayesian Reasoner for Java (IBRJ) partially implements the theory of IBNs, giving researchers the ability to use and understand how these modified networks function. IRBJ allows for classical implication statements to be added to the network while maintaining the ability to draw inference. A Graphical User Interface is also provided that shows how the network changes given the additional information. An extension of the BayesNets Interchange Format is proposed that allows IBNs to be saved in a standard format. This paper details the development of the IBRJ tool, and explains the implementation process followed. A background on IBNs is also given as an explanation of what is being implemented, including algorithms that define how the network reacts to the introduction of implication statements.

Keywords

Bayesian Networks; Formal Logic; Knowledge Representation; Propositional Logic; Java

1. INTRODUCTION

Both Bayesian Networks and Formal Logic have been successfully used to model and come to conclusions about a wide variety of domains. However, the union of these two fields has been overlooked. Introducing logic statements into a Bayesian network increases the expressiveness and thus usefulness of the network. This allows a richer model to be constructed that more accurately reflects any given input domain.

This paper details the construction of a tool, the Implicative Bayesian Reasoner for Java (IBRJ), which allows a user to supplement a Bayesian Network with limited forms of logical implication. This follows from the theoretical insight into Implicative Bayesian Networks (IBN) presented by Roussos [1]. The tool is aimed at researchers in the fields of Bayesian networks and formal logic, and demonstrates how IBNs function.

The tool allows for classical implication statements to be added to the network. The results of the addition of these new dependencies are shown by the tool. Inference can still be drawn on the network, demonstrating how the new network dependencies effect the conclusions that can be drawn on the network. Due to the nature of Bayesian Networks, variables in the network are largely independent. Supplementing the network with classical implication produces additional dependencies between variables or modifies existing dependencies. This allows the network user to introduce stronger dependencies between network variables than the typical causal relationships found in Bayesian Networks. This paper also presents a possible extension to the common BayesNets Interchange Format that would allow for IBNs to be saved and loaded in a standard format.

The research sets out to determine if a tool could be developed that easily allows a user to add classical implication statements into a Bayesian reasoner while maintaining the ability to make meaningful queries to the network. The IBRJ software tool achieves this and is presented alongside this paper. This allows a researcher in the field to easily use and understand how IBNs function. Included in the tool is a Graphical User Interface that demonstrates how the network structure changes as the new dependencies between network variables are introduced. We briefly discuss the theory behind the IBRJ tool followed by a discussion on the design, implementation and results achieved from the development of the software.

The addition of logical implication statements into Bayesian Networks gives the network designer finer control over what the network describes and how it does so. The additional dependencies can be added to a network after it has been constructed as more knowledge becomes available, without the network having to be completely redesigned. The tool allows for this process to be visualised and the network to be interacted with. Both Bayesian Networks and formal logic find many applications in artificial intelligence, and thus the union of these two fields may yield ramifications in the way these two research areas are applied.

2. BACKGROUND

2.1 Propositional Logic

Logic, as a mathematical tool, provides a basis for evaluating and reasoning about the world around us. Unambiguous systems are defined that allow precise, logical conclusions to be drawn from a collection of statements. Propositional logic is one of the most common formal logic systems due to its applicability to modelling philosophical theory and na-

ture[13].

Propositional Logic, also known propositional calculus, is a branch of logic that deals with singular truth-bearing variables and how these variables can be connected to create logical arguments [14]. These truth-bearing variables are known as 'atoms' and are the smallest unit of knowledge in a propositional knowledge base [12]. They can take on one of two states - true or false. Logical connectives can be used to extend an atom or join multiple atoms together in order to create composite atoms. These composite atoms can in turn only be true or false, though they are composed of one to many true or false atoms. Atoms and composite atoms are known as sentences, and a collection of sentences makes up a knowledge base [12][14]. As sentences are precise in their truth value, we can use them to reason about the world, or the state of a given system. In natural language, we use declarative sentences to describe what is going on around us. For example, we can state that *"it is raining"*. This is a propositional atom - it holds a singular value of either true or false and tells us something about the current state of the world.

2.1.1 Classical Implication

There are five connectives in Propositional logic: $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and $\leftrightarrow$. We will only look at implication, or the $\Rightarrow$ connective, as it is what this research is based on. For a discussion on the other connectives, see *A Concise Introduction to Logic (DeLancey, C)* [12]. This connective, often referred to as *classical implication*, represents an *if... then...* relation between the two atoms it connects. For example, $\mathbf{P} \Rightarrow \mathbf{Q}$ would be read as *"if **P** then **Q**"*. The first sentence, **P**, is known as the antecedent, and the second, **Q**, is known as the consequent. The truth values of an implication statement are given in table 1. Note that the value of the implication sentence relies mainly on the value of the antecedent.

P	Q	$P \Rightarrow Q$
1	1	1
1	0	0
0	1	1
0	0	1

Table 1: Truth table for $\mathbf{P} \Rightarrow \mathbf{Q}$

2.2 Bayesian Networks

Bayesian Networks, also known as Belief Networks [17], are a class of probabilistic graphical models that have been used in many fields for use as causal modelling and probabilistic inference [16]. They allow for a coherent method of structuring probabilistic information about a system in a way that makes it is easy to logically reason about the system and its potential states [10]. A Bayesian Network is a double, $BN =< DAG, CPT >$ where DAG is a Directed Acyclic Graph and CPT is a set of Conditional Probability Tables [17]. Nodes in the DAG correspond to the variables that are being modelled, while the edges are interpreted as the probabilistic independence relationship between these variables [10] [17]. We denote this relationship as $\mathbf{N} \rightarrow \mathbf{M}$ where **N** and **M** are variables in the network. The conditional probability table specifies the probabilistic relationship between the given variable and its parents. This assigns an individual probability of a variable being either true or false, based on its parents truth value. Bayesian Networks are powerful in part due to the key notion that a variable is independent of its children once the values of its parents are known [18].

Once a Bayesian Network has been constructed, it can be used to come to conclusions about network variables - the specific probability of an event occurring can be determined. Variables can be 'observed' - this is the processes of providing evidence of a nodes' value. Given these values, the probability of other events can be determined. For example, given a network that models the reasons for having wet grass, we can query *Probability(Wet Grass = True | Raining = False, Sprinkler = On, Water fight = Unknown (True or False))*.

There exist many types of queries and algorithms for providing answers to these queries. The IBRJ tool is concerned primarily with a probability-of-evidence query, where the probability of a specific variable is requested - P(e). When this is computed with no provided evidence it is known as the prior marginal. This is opposed the posterior marginal, which is computed with some evidence - P(e|b). As mentioned, there are many algorithms for determining the marginals. In IRBJ, the variable-elimination algorithm is used for its simplicity and ease of understanding. For a more in depth look at queries and algorithms used, see *Modelling and Reasoning with Bayesian Networks (Darwiche, A)* [19].

2.2.1 The Interchange Format for Bayesian Networks

A standardized format for representing Bayesian Networks was first proposed by members of the AUAI [15] community in order to foster the exchange of knowledge between researchers using Bayesian Networks. The format would define a strict grammar that could be used to save and exchange Bayesian Networks. It was decided that an XML-based format would be used to *"represent directed acyclic graphs that can be associated to conditional probability measures for discrete variables, with the possibility that decision and utility variables be present in the graph"* [11].

The following is an overview of the format. The network is enclosed in `network` tags, defined by a `name` tag and a sequence of `property` tags. Within the network tags, a sequence of `variable` tags are defined that represent each variable or node in the network. The variable contains a `name` tag that defines the name of the node, two `outcome` tags that show the possible states the variable can be in, and again a sequence of `property` tags. Conditional Probability Tables are represented by a sequence of `definition` tags. Each definition references which variable it is associated with via a `for` tag, followed by `given` tags and the `table` tag which contains the CPT values.

The .BIF format is now a widely used standard for saving Bayesian Networks. For more information on the format refer to Fabio Gagliardi Cozman's website [11].

3. THEORETICAL ANALYSIS

3.1 Supplementing Bayesian Networks with Propositional Inference statements

An Implicative Bayesian Network (IBN) is an extension of a Bayesian Network that contains a Propositional knowledge base that holds implication statements. The implications can be classical ($\mathbf{P} \Rightarrow \mathbf{Q}$) or defeasible ($\mathbf{P} \rightsquigarrow \mathbf{Q}$), though

this tool only implements the classical case. Negation is also defined, allowing for the negation of implications, for example, $\neg\mathbf{P} \Rightarrow \mathbf{Q}$. This merges the two fields of Bayesian Networks and Propositional Logic to create a model that is more expressive than each of its components in isolation.

This is possible because the variables in the Bayesian Network can be viewed as propositional atoms connected to each other via probabilistic, or causal, relationships. By adding an implication statement between two network variables (or network atoms) the relationship between them is modified to be an implication relation instead of the prior causal relationship. New relationships can also be added between network atoms, as long as this does not create a cycle in the graph.

If two network atoms have a causal relationship between them, $\mathbf{A} \rightarrow \mathbf{B}$, and then the classical implication $\mathbf{A} \Rightarrow \mathbf{B}$ is added to the IBN knowledge base, the causal relation is replaced by the implication relation for the cases where $\mathbf{A}$ holds. If the defeasible implication $\mathbf{A} \rightsquigarrow \mathbf{B}$ is added however, this may or may not over-write the causal relation, depending on a required observation on the network.

An Implicative Bayesain Network is defined as a 3-tuple:

$$(DAG_{\perp}, CPT_{\perp}, KB),$$

where $DAG_{\perp}$ is a directed acyclic graph connected via causal relations, $CPT_{\perp}$ is a set of conditional probability tables associated with $DAG_{\perp}$ and KB is a set of implications between nodes in $DAG_{\perp}$.

As implication statements are added to KB, causal relations in $DAG_{\perp}$ and $CPT_{\perp}$ are modified to create a $DAG_{\top}$ and $CPT_{\top}$ which represent the new network with these added implications. The basics of IBNs are presented below as a background to this tool, however for a more in depth discussion of IBNs, see Roussos [1].

3.2 Classical Implication

Adding a classical implication statement, $\mathbf{A} \Rightarrow \mathbf{B}$, to the network makes the relation between the two network atoms more specific - it says that the atom $\mathbf{B}$ is dependent on the atom $\mathbf{A}$ more than just causally or by some probability. The dependence is strict - if $\mathbf{A}$ happens then so must $\mathbf{B}$. The converse is not true however - if $\mathbf{A}$ doesn't happen, we cannot make any conclusions about the state of $\mathbf{B}$.

In order to implement this change of relationship in the Bayesian Network, we modify the CPT's to reflect the stricter influence $\mathbf{A}$ has over $\mathbf{B}$. This process varies depending on the relationship the two network atoms have in the Bayesian Network. There are three relation types - Direct, Indirect, and None. The Direct type specifies a relation where $\mathbf{A}$ is a parent of $\mathbf{B}$ in the graph. Indirect type is the opposite - $\mathbf{A}$ is a child of $\mathbf{B}$. The None case handles all other relations between network atoms - nodes that are not relatied, or related via one or more intermediate nodes.

The algorithms for how the network should be modified are given in section 5.2.1. The Direct and Indirect case simply change the current CPT, while the None case adds a new edge in the graph between the two nodes, and then modifies the CPTs to reflect the implication.

As edges can be added to the network, cycles could be created. This is termed a 'clash', and is not allowed in an IBN. If an implication is added to the network that would cause a clash, the implication is rejected. This ensures that the network cannot have cycles.

Despite this, a cycle of implication statements can legally be added to the network, forming an implication cycle. In implication cycle is a cycle formed between nodes in the network where every edge in the cycle is an implication edge. In this case, all events within the implication cycle can be viewed as one event and thus can collapse into one node. This process is described in section 5.2.1.

3.3 Other IBN features

IBNs also define how a Bayesian Networks should respond under the negation of a network variable. The introduction of negation also allows for a new type of implication to be added to the network - defeasible implication. This allows for some notion of exceptionality to be introduced in the network. This requires the use of addtional algorithms such as ranking and rational closure [8][9].

3.4 Querying an Implicative Bayesian Network

The strength of IBNs is that they can be queried in the same way that standard Bayesian Networks can be. This allows all the same algorithms to be used, without any added complexity. Probabilistic observations can still be made and thus prior and posterior marginals can still be computed. Another type of observation is also available - the logical observation. This is a classical or defeasible sentence that can be added to the network as an observation. This does not effect the network graph or CPT's, but rather acts as a specifier for which world the knowledge base is in. When inference is drawn, the knowledge base will be checked to see if it is entailed by the logical observation. If it is, then the knowledge base is applied as usual; however if it isn't, then the $DAG_{\perp}$ network will be used as the observation made shows that the knowledge base is not consistent.

4. REQUIREMENT ANALYSIS

The goal of this project was to design and develop a software tool that allows a user to specify additional dependencies between variables in a Bayesian Network while maintaining the ability to draw inference on the network. Specifically, this is the creation of a tool that allows researchers to use and understand Implicative Bayesian Networks (IBN). This is the implementation and practical execution of the work presented by Roussos [1]. The tool only implements classical implication, as a demonstration of how a Bayesian Reasoner might implement implication.

Requirements were identified that, given full implementation, would result in a completed tool that fully demonstrates IBNs. These requirements are presented below as a set of use cases.

The user needs to be able to:

1. Run the program through a Command Line Interface or a Graphical User Interface.
2. Load or import a Bayesian Network saved in any of the standard formats - *.bif*, *.ibif*, *.xml* or *.net*.
3. View the Bayesian Network graph - the nodes and edges of the graph - as well as the variable CPTs.
4. Make observations on network variables, setting them to a known true/false value, or making a logical observation on the network.

5. Add classical implication statements to the network, and view how this changes graph structure and CPTs.

6. Compute prior and posterior marginals before and after implication statements have been added.

The software tool should:

1. Store an internal representation of an Implicative Bayesian Network that contains a Bayesian Network and a knowledge base of implication statements that have been added to the network.

2. Apply classical implication sentences to the network as defined in IBNs.

3. Find any cycles in the graph and adjust the graph when an implication cycle is found.

4. Parse an extension of the BayesNet Interchange Format, the *.ibif*, to allow users to load and save IBNs.

These use-cases allow a user to perform all necessary tasks to observe how implication statements effect a Bayesian Network, and thus demonstrates how IBNs can be used.

The tool is intended to be used to demonstrate and explain how IBNs function. It should also provide a basis and example for further research into introducing formal logic into Bayesian Networks. Due to this, the software tool is aimed at researchers in the field of Bayesian Networks, formal logic, and knowledge representation. This effected the requirement specifications in that the tool is intended to be a proof-of-concept, rather than a consumer ready product.

As there has been no previous example of introducing implication statements into Bayesian Networks, these research goals and use-cases are important as they provide a novel insight into this field.

5. DESIGN AND IMPLEMENTATION

As the main focus of this project was the development of a software tool, a well planned design was important to ensure successful implementation of the tool. The program was designed to reflect the use-cases documented in section 4. A well-planned design, though one that was open to changes, allowed for an efficient and effective implementation of the IBRJ tool.

5.1 Design

The IBRJ tool is an extension of a Bayesian Reasoner - a tool that can draw inference on a Bayesian network - that introduces the additional options to add implication statements. As such, IBRJ is built on top of a preexisting Bayesian Reasoner. The IBRJ tool will handle the modifications on the Bayesian network, and then make calls to the underlying Bayesian reasoner. This is done so a Bayesian reasoner need not be implemented, as there are many suitable solutions available. This leads to two options; an open-source Bayesian reasoner is extended to add implication support, or a wrapper is built around a Bayesian reasoner library. Both have advantages and disadvantages, discussed below.

Modifying a preexisting Bayesian reasoner means that the base program is already fully implemented. This means that a GUI and UX does not need to be designed and built. This is a major advantage as a well designed UX is important so as to allow the user to fully experiment with and understand how Implicative Bayesian Networks function (IBN). However, extending a preexisting program means that the new code is placed alongside the original code. This results in no modularity and complete dependency on the original program.

Building IBRJ as a wrapper around a Bayesian reasoner removes this high dependency by using the Bayesian reasoner as a library. This means that the reasoner could be swapped for another reasoner should a better option be made available with minimal interference to the software. This does however mean the UX must be designed and built.

It was decided that building the tool as a wrapper would be preferable. This detaches the new implication part of the program from the Bayesian reasoner, removing this dependency. This also means that the choice of Bayesian reasoner is less relevant, as it can be changed with minimal effort.

An appropriate language was needed to implement the software tool. There was a large restriction placed on which languages could be used by which Bayesian reasoner IBRJ was built as a wrapper around. Additionally, the program should be written in a language that is common and used by as many people as possible, allowing users to easily inspect the code to better understand how IBNs were implemented. Following from these considerations, Java was chosen. The Bayesian reasoner chosen was written in Java. Java also has the advantage of being easily portable, and is a widely used and understood language.

The software tool is designed to follow standard Object Orientated Programming principles. This means that code is modular and compartmentalized - only code that relates to a specific class should be found in that class. This ensures that code is well organized and coherent. The package that acts as an interface to the Bayesian reasoner library should be the only place where references to the API are found. This is important as it represents a singular point of contact with the reasoner, ensuring minimal changes needed if the reasoner is changed. An agile development method was applied so as to ensure that rapid changes in the definition of IBNs could be accommodated.

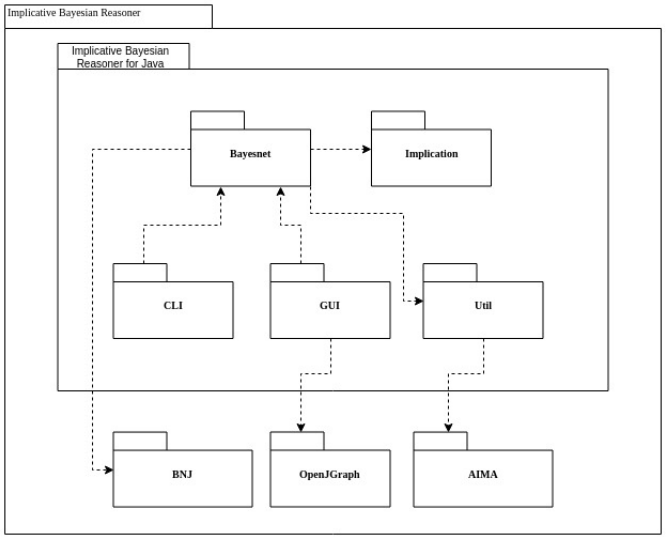
The Bayesian reasoner on top of which this program is built has an internal representation of a graph. This representation needed to be extended to contain a knowledge base of classical implication statements. This is identified as a key area that needed to be implemented. The tool should have internal representations of implication that are coherent within the context of the program. Additionally, algorithms for identifying cycles in the graph need to be implemented. A fully optimized solution is not the goal for this project, and as such this is not a major factor in design. That said, an efficient solution is still considered in the design to ensure that the tool is usable without any performance issues.

As stated above, the design of IBRJ aims to be modular and follow good OOP practices. The package diagram in figure 1 shows the structure of the software tool. Note there are minimal dependencies between packages, ensuring maximum decoupling.

5.2 Implementation

A survey of the available Bayesian reasoners was carried out. The reasoner needed to have a free use licence such as the GNU GPL [2], so that it could be used by IBRJ.

Figure 1: IBRJ Package Diagram



Ideally, the reasoner would also be open source to allow for inspection of the underlying tool and easier development. This was however not essential, as long as the reasoner had a well documented API.

The reasoners identified include Bayesian Networks Tools for Java (BNJ)[3], JavaBayes [4] and Banjo [5]. A number of Bayesian reasoners were not considered as they required a licence to use. Of those identified, BNJ and Banjo were the main candidates. These are completely free and open source, and designed specifically for research purposes. JavaBayes was discredited as it was developed using Java Version 1, which was is not fully compatible with the modern Java compiler. Both BNJ and Banjo are very similar in functionality, however BNJ offers a more comprehensive API to access the underlying classes used for managing the Bayesian Network and drawing inference. Due to this, BNJ was chosen as the underlying Bayesian reasoner. The API for BNJ is well documented, and most methods required by IBRJ were already implemented and easy to use.

After a package and class structure was identified, as shown in figure 1, development of the tool began. Initially, a simple wrapper that added no extra Bayesian reasoning functionality was build around BNJ. This was done in conjunction with building a Command Line Interface (CLI) for IBRJ. Once the simple wrapper been completed, implication was added to the program. Classical implication was added iteratively, implementing the simple *direct* case first, followed by the *indirect* and *none* cases. These cases were implemented using the `supplementClassical` algorithm outlined in section 5.2.1. Subsequently, cycle detection was added in order to identify clashes and implication cycles. This represented the complete implementation of classical implication.

In parallel to this development, a Graphical User Interface was built. This was done to fulfill the requirements of allowing a user to fully understand, via visualizations, how the network graph was effected by supplementing the network with implication statements.

Subsequent to completing the addition of classical implication to the network, a parser for the Implicative BayesNet Interchange format was implemented, allowing saving and loading of IBN's.

Defeasible implication was not implemented in IBRJ as in order for any specific functionality to be drawn from it, propositional negation also needed to be implemented. This pushed the implementation of defeasibility out of the scope of this project.

5.2.1 Algorithms

The following algorithm describes how a classical implication statement, $A \Rightarrow B$, should effect a Bayesian Network.

Algorithm 1: Supplementing a Bayesian Network with classical implication

```
1  function supplementClassical (g, a, b);
   Input  : Graph g, Antecedent node a, Consequent
            node b
   Output: Graph g'
2  if relation(a,b) = DIRECT then
3  |  cpt ← b.CPT;
4  |  foreach entry in cpt do
5  |  |  if cpt.a = 1 and cpt.b = 1 then
6  |  |  |  set P(B|A) = 1;
7  |  |  if cpt.a = 1 and cpt.b = 0 then
8  |  |  |  set P(B|A) = 0;
9  |  end
10 else if relation(a,b) = INDIRECT then
11 |  cpt ← a.CPT;
12 |  foreach entry in cpt do
13 |  |  if cpt.b = 1 and cpt.a = 1 then
14 |  |  |  set P(B|A) = 0;
15 |  |  if cpt.b = 1 and cpt.a = 0 then
16 |  |  |  set P(B|A) = 1;
17 |  end
18 else if relation(a,b) = NONE then
19 |  Add edge from a to b in g;
20 |  cpt ← b.CPT;
21 |  foreach entry in cpt do
22 |  |  if cpt.a = 1 and cpt.b = 1 then
23 |  |  |  set P(B|A) = 1;
24 |  |  if cpt.a = 1 and cpt.b = 0 then
25 |  |  |  set P(B|A) = 0;
26 |  end
```

Once an implication cycle is found in the knowledge base, the variables involved in the cycle are assumed to be equal, and thus those nodes can be reduced to a singular node that represents the whole cycle. This is performed using the `collapseNode` algorithm. The construction of the cycle node CPT is given in Roussos [1].

Entailment checking, used to apply logical observations, is performed using the algorithm presented by Russell and Norvig [7].

5.2.2 User Experience

In order to create a tool that allows researchers to use and understand IBNs effectively, the tool needed to have a clear, simple and easy to use user experience. This would ensure that the user could focus on understanding how the model works, and not on using the tool. A Command Line Interface (CLI) as well as a Graphical User Interface (GUI) were built to cater to all needs and environments. The Model-View-Controller pattern was used in order to maintain a clear

Algorithm 2: Collapsing an implication cycle of nodes to a singular node

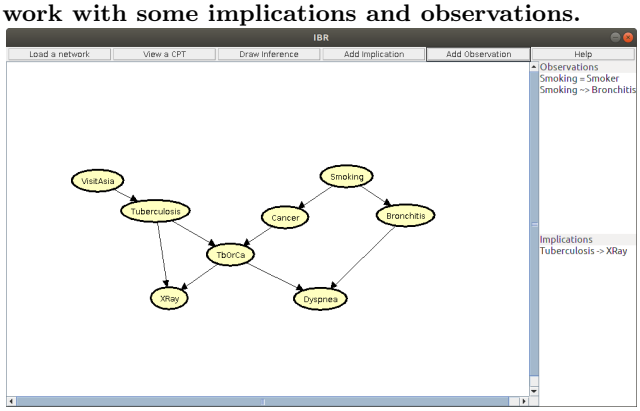
```
1  function collapseNodes (g, cycleNodes);
   Input  : Graph g, Set of cycle nodes cycleNodes
   Output: Graph g'
2  cycleParents ← new empty set;
3  cycleChildren ← new empty set;
4  foreach node in cycleNodes do
5  |  if node.parents not in cycleNodes then
6  |  |  cycleParents.add(node.parents) ;
7  |  if node.children not in cycleNodes then
8  |  |  cycleChildren.add(node.children);
9  |  graph ← graph.remove(node);
10 end
11 foreach parent in cycleParents do
12 |  graph ← graph.addEdge(parent to CycleNode);
13 end
14 foreach child in cycleChildren do
15 |  graph ← graph.addEdge(CycleNode to child);
16 end
17 construct CycleNode.CPT;
18 foreach child in cycleChildren do
19 |  modify child.CPT to represent new parent;
20 end
```

separation between the functionality of the software, and the interface to this functionality. This ensured that maximum code could be reused between the CLI and the GUI.

The CLI is a simple repeating menu style interface. Simple navigation, with shallow menu options, ensure the user will not get lost in menu navigation. The CLI allows the user to perform all functions on an IBN, however the graph and CPTs cannot be viewed graphically. This is especially limiting as it does not allow for one of the main goals of this research to be fulfilled: to easily view how adding implication statements effect the structure of the graph.

This issue is rectified with the introduction of the GUI, which allows users to visualize the graph structure. The GUI, built using the Java Swing framework, provides a simple interface to IBNs. It makes use of the OpenJGraph library for graph visualization, and allows the user to perform all IBN functions. The GUI is shown in figure 2, with the common 'Asia' network visualized.

Figure 2: IBRJ GUI showing the popular *Asia* network with some implications and observations.



5.2.3 *Challenges*

One of the main challenges during development was that of limitations of the API for the underlying Bayesian reasoner. Often the API would not provide the functionality needed by IBRJ. This would mean a wrapper function that made best of the API would have to be written, which often resulted in an undesirable approach, though the only one available. These issues were as a result of BNJ not being built with the features that an IBN requires, such as the heavy modification of CPTs. Another challenge was the changing results presented in theoretical component of this project. This meant that development was often stunted while a solution was worked on. This slowed down progress on the tool.

5.3 Testing

Unit tests were developed alongside development of each package. This ensured that each class and method was producing the correct results throughout development. Integration tests were also set up that tested the functionality of the program as a whole. These tests checked that adding certain implication types had the intended effect on the network. These tests were especially useful for refactoring of classes and packages.

5.4 Software Documentation and Maintenance

The IBRJ software is fully documented, with a README and JavaDocs supplied in both the program directory as well as on the project website. This should provide thorough insight into how to run the program and how to use the programs API. This should make it simple for developers to understand the structure and purpose of each class, making it easy to continue development on IBRJ.

The software package should not require any maintenance as all dependencies are static - that is, they will not change. The only foreseeable maintenance that is potentially required would be updating the Java code should it become unsupported by future versions of the Java compiler.

As the program is coded in Java, portability is ensured via the JVM. Thus, the code should run on any system that supports the JVM.

6. AN EXTENSION TO THE BAYESNET INTERCHANGE FORMAT

The following describes a potential extension to the BayesNet Interchange Format (*.bif*) - the Implicative BayesNet Interchange Format (*.ibif*). This format will allow Bayesian networks to be saved in an XML based file-type that retains the knowledge base of added implication statements. This allows for IBNs to be saved and reloaded, facilitating continuous development on the network, as well as encouraging exchange of research.

6.1 Implicative BayesNet Interchange Format

The *.ibif* format retains all components of the *.bif* format. The new format adds tags that store propositional sentences. The `sentence` tag begins an implication statement. Within a sentence, a `connective` must be defined that specifies the type of implication the sentence represents - classical or defeasible. The sentence must also specify an `antecedent` and a `consequent`. There is also an optional sequence of `prop-`

erty tags. These can be used to add additional information such as if the sentence is part of the knowledge base or an observation on the network.

Listing 1: Example *.ibif* file

```
<?xml version="1.0"?>
<BIF version="0.3">
<network>
 <name>Example Network</name>

 <!-- Variables -->
  ... some entries ...

 <!-- Probability distributions -->
  ... some entries ...

 <!-- Knowledge base -->
 <sentence>
  <connective>classical</connective>
  <antecedent>some-node</antecedent>
  <consequent>another-node</consequent>
 </sentence>

 </network>
</BIF>
```

7. RESULTS AND EVALUATION

The development of the Implicative Bayesian Reasoner (IBN) for Java was a success. The final version is a completed tool that can fulfills all use-cases that were identified. The tool has two interfaces, both of which are fully developed. The tool implements classical implication fully, as described in an IBN. When development began, implementing defeasible implication was an intention. However, as progress on the project continued it was made evident that this would be out of the scope for this project and the implementation of defeasible implication was abandoned.

The tool is easy to use with simple navigation system for both the GUI and the CLI. The main aim of building a tool that allows researchers to use and understand Implicative Bayesian Networks (IBN) is fulfilled. The tool is deemed a success due to it's effectiveness of demonstrating the semantics of IBNs, and achieving all the goals layed out in the project proposal.

All unit and integration tests have been passed, showing that the tool's functionality is correct.

The outline of an updated *.bif* format, *.ibif*, allows for both classical and defeasible implications to be saved as part of a knowledge base component of the interchange format. This successfully allows networks to be saved and reloaded.

7.1 Expressiveness of Implicative Bayesian Networks

Everything that is specified by an IBN can be expressed in a standard Bayesian Network. This is proven implicitly in the definition of how implication statements are added to the network - they modify the network but it remains a Bayesian Network; no Bayesian Network rules are broken in the modified version. Due to this, the IBN is essentially a syntactic modification of Bayesian Networks that semantically may represent something different, but mathematically is identical to Bayesian Networks. This does not mean the model is useless however. The semantic difference is highly valuable as it allows the user to make logical modifications to the network in an intuitive way without having to modify the whole structure of the network. This allows for easier, faster modeling.

8. SOFTWARE LICENCE

The IBRJ software is fully open-source. It is licensed under the GNU General Public Licence (GPL) version 3 [2]. This allows the software and code to be used, run, shared, studied and modified, while ensuring that the software and all reproduced versions of it must remain free and open-source.

9. USAGE

The IBRJ tool can be found and downloaded from the project GitHub repository[1]. After running the *build* script, the *ibr* program script can be run. For further explanation as to how to run the program, please see the README.

10. LIMITATIONS

The IBRJ tool does not implement defeasible implication or negation, two components of Implicative Bayesian Networks. This limits the extent of which IBNs are demonstrated in the IBRJ tool.

A limitation of IBRJ are the available Bayesian Network tools. The software has no support for constructing networks - nodes and edges cannot be added to the network (except for those made as a result of adding implication statements). The only method of opening a Bayesian Network is via loading a pre-constructed network saved in a *.bif*, *.ibif*, *.xml* or *.net* format. This is due to the focus of the software being on observing how Bayesian Networks are modified when implication statements are added, and not on the construction of the Bayesian Networks themselves. Additionally, the IBRJ tool has no Bayesian learning support.

Once an implication statement has been added to the network, it cannot be removed. Additionally, observations made on the network cannot be removed. The network should be reloaded to reset these attributes of the IBN.

The software is developed and tested in a LINUX environment. The JVM should allow IBRJ to work on any OS that supports the JVM, however no guarantee is made that this will be the case.

11. CONCLUSIONS

The Implicative Bayesian Reasoner for Java (IBRJ) is a tool that implements Implicative Bayesian Networks (IBN) [1] in a simple, easy to use way. This allows researchers in the fields of Bayesian Networks, logic and knowledge representation and other related Artificial Intelligence fields to use and understand how IBNs function. This paves the way for further research into IBNs, and sets a basis for how these networks could be implemented.

A proposed modification of the standard *.bif* format is presented that allows for knowledge base statements, such as classical implication, to be saved in an Implicative BayesNet Interchange format, *.ibif*. This successfully allows IBNs to be saved and reloaded.

[1] https://github.com/Nevter/IBRJ

IBNs provide an extension to Bayesian Networks that only extend the semantic meaning of the network. Equal conclusions can be drawn from an IBN and a Bayesian Network. Despite this, the IBN is valuable as it allows researchers an easy way of modifying networks in a logical way. This makes the network more expressive from a users persepective.

12. FUTURE WORK

Continuing development, the next step is to implement negation and subsequently defeasible implication statements. This will result in a fully implemented Implicative Bayesian Network model.

Further work can be done on the IBRJ tool to extend and improve its functionality. The algorithms and methods implemented can be optimized to increase performance of the tool. The Bayesian Reasoning capabilities of IBRJ could also be extended to implement the limitations outlined in section 10.

This research could be continued to look at implementing other Propositional Logic connectives, or alternatively other logic systems in Bayesian Networks. This has the potential to further increase the expressiveness of Bayesian Networks.

13. ACKNOWLEDGEMENTS

Thanks are due to Tommie Meyer for supervising, guiding and providing invaluable input to this project. Additional thanks to Deshen Moodley for being a second set of eyes and providing insight and suggestion towards completion of the project. Thanks are also due to Elijah Roussos for spending countless hours on the theoretical component of this research and providing the methods and algorithms needed to implement IBRJ. Finally, the team that developed the Bayesian Network Tools for Java is thanked for providing a well designed and robust library on top of which this tool is built.

14. REFERENCES

[1] Roussos, E. (2018). A Model for Implicative Bayesian Networks. (Unpublished). University of Cape Town, Cape Town, South Africa.

[2] Gnu.org. (2018). The GNU General Public License v3.0- GNU Project - Free Software Foundation. [online] Available at: https://www.gnu.org/licenses/gpl-3.0.en.html [Accessed 15 Sep. 2018].

[3] Bnj.sourceforge.net. (2018). Bayesian Network tools in Java (BNJ) - Kansas State University Lab for Knowledge Discovery in Databases. [online] Available at: http://bnj.sourceforge.net/ [Accessed 15 Sep. 2018].

[4] cs.cmu.edu. (2018). JavaBayes 0.346. [online] Available at: https://www.cs.cmu.edu/ javabayes/Home/ [Accessed 15 Sep. 2018].

[5] Hartemink, A. (2018). Banjo: Bayesian Network Inference with Java Objects. [online] users.cs.duke.edu. Available at: https://users.cs.duke.edu/ amink/software/banjo/ [Accessed 15 Sep. 2018].

[6] Sharir, M., 1981. A strong-connectivity algorithm and its applications in data flow analysis. Computers & Mathematics with Applications, 7(1), pp.67-72.

[7] Russell, S., Norvig, P., & Davis, E. (2010). Artificial intelligence: a modern approach (3rd ed., International ed.). Upper Saddle River: Prentice Hall.

[8] Lehmann, D. and Magidor, M., 1992. What does a conditional knowledge base entail?. Artificial intelligence, 55(1), pp.1-60.

[9] Booth, R., Casini, G., Meyer, T. A., & Varzinczak, I. J. (2015,June). On the Entailment Problem for a Logic of Typicality. InIJCAI (pp. 2805-2811)

[10] Darwiche, A. (2010). Bayesian networks. Communications of the ACM, 53(12), 80–90. doi:10.1145/1859204.1859227

[11] Cozman, F.G., 1998. The interchange format for Bayesian networks. URL http://www. cs. cmu. edu/afs/cs/user/fgcozman/www/Research-/InterchangeFormat.

[12] Delancey, C. (2017). A concise introduction to logic. Geneseo, NY: Published by Open SUNY Textbooks, Milne Library, State University of New York at Geneseo. Retrieved from https://open.umn.edu/opentextbooks/BookDetail.aspx?bookId=452

[13] Ferreirós, José (2001), "The Road to Modern Logic-An Interpretation", Bulletin of Symbolic Logic, 7 (4): 441–484, doi:10.2307/2687794, JSTOR 2687794.

[14] Bealer, G. (1998). Propositions. (philosophy of language, traditional proposition theory). Mind, 107(425), 1. doi:10.1093/mind/107.425.1

[15] UAI. (2018). Uncertainty in Artificial Intelligence. [online] Available at: http://www.auai.org/ [Accessed 16 Sep. 2018].

[16] Hadlock, C. (2005, September 1). Causality: Models, Reasoning, and Inference. Journal of the American Statistical Association. Taylor & Francis. doi:10.1198/jasa.2005.s38

[17] Zhao, Y., Chen, Y., Tu, K., & Tian, J. (2017). Learning Bayesian network structures under incremental construction curricula. Neurocomputing, 258, 30–40. doi:10.1016/j.neucom.2017.01.092

[18] Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (2013). Context-Specific Independence in Bayesian Networks.

[19] Darwiche, A. (n.d.). Modeling and reasoning with Bayesian networks (First paperback edition.). New York: Cambridge Univ Press.

[20] Kraus, S., Lehmann, D., & Magidor, M. (2002). Nonmonotonic Reasoning, Preferential Models and Cumulative Logics.