NIKKARILA Juha-Pekka
KANGASNIEMI Ilmari
VALTONEN Janne

BAYESIAN NETWORKS – AN EXAMPLE OF SOFTWARE AND SOME DEFENCE APPLICATIONS

Abstract

In this article, publicly available information and software on Bayesian networks is reviewed from the point of view of military applications. The article contains a brief introduction to Bayesian networks and influence diagrams, and lists a number of existing and potential military applications for them. Existing free Bayesian network software is reviewed and their properties are compared. The most suitable one for military defence applications is selected, and its features are assessed in detail. Alongside the features, observed weaknesses and common pitfalls in use are reported. In addition to this, time requirements of network inference are explored, and the available algorithms in our selected software are compared.

Keywords
Military computing, military systems, Bayesian networks, Dynamic Bayesian Networks
Introduction to Bayesian networks

The following section outlines the basics of Bayesian networks and influence diagrams. A more in-depth overview of the subject can be found for example in (Charniak, 1991). For a detailed mathematical description, see for example the book (Neapolitan, 2004).

Bayesian networks

A Bayesian network is a representation for probability-based models. It describes the model through direct dependencies between random variables. This eases the computational handling of the model and gives it a simple graphical presentation. Bayesian networks are applied to a wide range of subjects due to their generality and other useful properties.

Bayesian networks consist of nodes, which represent random variables, and directed arcs, which represent direct dependences among random variables, (Figure 1). It is also required that the resulting network structure is acyclic. In other words, there must be no directed paths returning to their origin point in the network. The terminology used to describe Bayesian networks resembles that of family trees: Nodes can for instance have parents, children, descendants, and ancestors.

Besides the network structure, the state of every node in a Bayesian network has a probability distribution, which depends on the states of the node’s parents. Most Bayesian networks found in practical applications have finite sets of node states. In this case, the distribution is given as a conditional probability table, which lists the probabilities of every state for every combination of parent node states. The probability distributions of node states define the network’s joint probability distribution, which describes the probability of every possible combination of node states in the network.

Solution algorithms for Bayesian networks are generally concerned with computing marginal probabilities. A solution algorithm combines the probability distributions in the network and produces a single distribution for each node. Furthermore, one can fix the states of a subset of nodes, due to for example observational data. These nodes are then treated as evidence nodes, which serve as initial conditions, and the algorithm computes the conditional marginal probabilities through use of Bayes’ theorem.

Influence diagrams

Influence diagrams are a generalization of Bayesian networks used in decision analysis. In an influence diagram, the parts of the model representing the decisions of a given party are not random. The general goal of the model is to find an optimal decision policy for the party in question.
The structure of influence diagrams is similar to that of Bayesian networks. However, two new node types are added. **Decision nodes** describe the decisions under analysis. Like chance nodes, decision nodes have a predetermined, commonly finite set of available states. However, the states do not have a conditional probability distribution. Instead, the parent nodes of a decision node represent the set of nodes, the states of which are known when the decision is made. As a consequence, arcs among decision nodes are used to represent the order of decisions made.

Another new node type in influence diagrams is **utility nodes**. They are used to measure the utility gained from the outcome of the decisions. The state of a utility node is a numerical value conditional on its parents. Utility nodes commonly have no children, and even if they do, the children are usually other utility nodes. The solution algorithm for an influence diagram is concerned with the expected values of utility nodes. Example solution types are finding the decision with the best expected utility, or producing a list of expected utilities for all possible combinations of decisions. (Figure 2).

**Defence applications of Bayesian networks**

**Direct applications**

Bayesian networks have multiple useful properties in defence-related modelling. A vast majority of the systems and scenarios of interest in defence are chaotic, and information is often imprecise. Hence the probability-based approach of Bayesian networks is a clear advantage. The structure of Bayesian networks combines multiple individual probability estimates into a unified picture of the whole model. This is an advantage due to the interdisciplinary nature of defence, as the networks can be based on probability assessments elicited from a multitude of experts of various fields. Furthermore, the clear graphical representation associated with the model helps communicate the model structure to experts with less background in probability theory. In addition to this, the support for decision analysis provided by influence diagrams is a major advantage.

The article (Starr & Shi, 2004) lists a multitude of possible defence applications of Bayesian networks: Modelling combat and base defence scenarios, comparing relative strengths of military units, assessing possible enemy objectives based on observed actions, etc. The article (Falzon, 2006) considers the use of Bayesian networks in operations analysis –based centre of gravity analysis. One of the applications outlined in the book (Pourret, Naïm, & Marcot, 2008) is using Bayesian networks to assess the threat of potential terrorist attacks. There are also applications of Bayesian networks to situation assessment, for example in (Das, Grey, & Gonsalves, 2002) and (Wright, Mahoney, Laskey, Takikawa, & Levitt, 2002).

**Applications in fields related to defence**

As previously mentioned, Defence is a highly interdisciplinary process. Therefore, applications of Bayesian networks in other fields can be of use in defence. An example of this is the field of
medicine, in which Bayesian networks are widely used in diagnosis. Applications in medicine become relevant to defence through, for example, combat first aid and health care of military units. Diagnosis tools based on Bayesian networks could be used, for example, to rapidly screen large military units for potential infectious diseases, or to direct treatment in urgent field situations where a professional doctor is not immediately available.

Similar to diagnosis of diseases, Bayesian networks have been used in troubleshooting devices, vehicles, and software. Military equipment is highly standardised, so potential troubleshooting software has a wide range of applicability. In addition to this, similarly to the diagnosis scenario, the equipment may break down in field conditions far from the nearest maintenance expert. In this case, an automated troubleshooting system could solve a portion of equipment malfunctions without having to transport either the device or the maintenance personnel.

A major application of Bayesian networks is artificial intelligence. Bayesian networks and influence diagrams are used in AI as a possible basis for logical reasoning. For example, the article (Charniak, 1991) mentions route planning and semantic understanding of language as possible AI applications of Bayesian networks. As modern AI develops, it is estimated to become an increasingly significant factor in future warfare. One reason for this is the possible use of autonomous vehicles in dangerous missions, eliminating the threat of personnel casualties. The most visible modern example of this is Unmanned Aerial Vehicles (UAV). Bayesian networks have been used as one of the multitude of tools in UAV development: See for example (Duan, Shao, Su, & Zhang, 2010) and (Ren, Gao, Zheng, & Zhang, 2010).

Bayesian networks have been used in image recognition and fusion of information from multiple sensor systems. These applications are outlined for example in (Luo, Savakis, & Singhal, 2005) and (Singhal, Luo, & Brown, 2000). This has direct applications to defence through target recognition, base surveillance systems, and situation assessment from aerial or satellite images.

**Bayesian network software**

**Desired features**
There exists a wide variety of existing software for Bayesian network inference. Hence, an important part of the software selection process is assessing, which features are the most important.

A major strength of Bayesian networks is their graphical presentation. Precisely due to the clarity of this presentation, one can obtain a surface level understanding of Bayesian network–based models without much education in statistics. Since defence is highly interdisciplinary, the quality of graphical presentation is a significant factor in software selection.

Another key property of Bayesian networks is support for decision analysis. In defence scenarios, singular decisions can have a major impact, so setting aside time to analyze the possible outcomes can result in major improvements. Therefore, support for influence diagrams was given high value in software selection.

Other features considered were among others ease of use and interfaces with widely used systems, programming languages and software. An open source code was considered an advantage, as it
permits editing the program according to possible extra requirements. The documentation quality and availability was also considered.

The selection process focused on free software. A significant portion of available software was immediately cut on the basis of a lacking feature set, the most common reason of exclusion being a complete lack of influence diagram support. Although not the focus of this study, it is worth mentioning that there also exists a multitude of known commercial software, such as BayesiaLab, Netica, and Hugin Expert.

Software under consideration

BUGS

Unlike other software presented here, BUGS (*Bayesian inference Using Gibbs Sampling*) does not focus solely on Bayesian networks. Instead, BUGS is a modelling language for general models in Bayesian statistics, which include Bayesian networks as a special case. The common thread in BUGS is the use of MCMC-sampling (*Markov Chain Monte Carlo*) to evaluate said models.

The BUGS language can be used through multiple existing programs. WinBUGS is an older stable program for Windows. The source code for it is not open, but the program is currently available free of charge. WinBUGS is however not under active development. Instead, the vast majority of development has moved on to its open source equivalent, OpenBUGS. OpenBUGS is significantly more experimental than WinBUGS, but has already achieved some degree of stability.

JAGS (*Just Another Gibbs Sampler*) is an alternative open source program for the BUGS language. JAGS has mostly similar properties to the preceding software, although it has no available GUI. Its greatest advantage is its platform independence: Getting JAGS to work on a platform other than Windows is significantly easier than in the case of other alternatives.

Although there is a GUI for WinBUGS and OpenBUGS, it is significantly limited compared to the actual BUGS language. The greatest assets of BUGS software are the ability to represent a multitude of models and a large list of interfaces to known software and programming languages. However, the inference algorithm for Bayesian networks is limited to MCMC sampling, and the language presents a steep learning curve for potential users. It is also worth mentioning that the language supports network nodes with a non-discrete set of states.

BNT

BNT (*Bayes Net Toolbox*) is an open-source MATLAB plug-in for Bayesian network inference. It has support for influence diagrams, dynamic networks, NoisyOR-nodes, both exact and approximate inference, structure learning, and parameter learning. The development of the plug-in has unfortunately mostly ceased.

The GUI of BNT is limited to that of ordinary MATLAB editors. Visual representations are limited to static drawings, and all editing is script based. A large emphasis is placed on its extendibility: It is relatively easy to implement new model types to the existing framework. Only a limited set of non-discrete node distributions is supported, with the supported ones being based on normal distributions. It is also noteworthy that BNT is reliant on MATLAB, which limits its computation speed, and makes it relatively expensive for users without an existing MATLAB licence.
OpenMarkov
OpenMarkov is a Bayesian network program developed in the Spanish University of Distance Education. It is licensed under the EUPL (European Union Public Licence) and its source code is open. The program is written in Java 7, and therefore its open source code acts as a Java interface of some degree. The file format OpenMarkov uses is a program-specific format based on XML.

The program supports influence diagrams, NoisyAnd and NoisyOr -nodes, and structure learning. OpenMarkov has dynamic model support for MPADs (Markov Process with Atemporal Decisions). MPADs are mainly focused on cost-effectiveness analysis for a single decision, the consequences of which evolve through time. The program also has incomplete support for many other model types – one can create and edit such models with the program, but there is no inference support for them.

GeNIe & SMILE
GeNIe (Graphical Network Interface) and SMILE (Structural Modeling, Inference, and Learning Engine) are a software pair developed in the Decision Systems Laboratory of the University of Pittsburgh. SMILE is a Bayesian network inference engine commonly used through a C++ interface. GeNIe is a graphical user interface built on the SMILE engine, covering the vast majority of the features of SMILE. Besides the inbuilt C++ interface, SMILE also has interfaces in Java (jSMILE) and .NET-languages (SMILE.NET). SMILE is platform independent, but GeNIe is limited to Windows due to its Visual C++ components.

Both GeNIe and SMILE support influence diagrams, dynamic networks, NoisyMax-nodes, structure and parameter learning, and multiple inference algorithms. In addition to these, there are a couple special features for diagnosis and estimating strengths of influence. There is limited support for non-discrete nodes. The file format of GeNIe and SMILE is another program specific XML format. There is, however, also support for some of the common file formats of commercial Bayesian network software.

The source code of GeNIe and SMILE is closed. The programs are available free of charge in exchange for registration. The only limitation on their use is that publications using GeNIe or SMILE have to explicitly mention their use. Unlike GeNIe and SMILE, both SMILE.NET and jSMILE are of open source. The interest in the software in the defence sector is also apparent, as a significant portion of the organizations supporting the development of GeNIe and SMILE is from the defence sector (among others DARPA, NATO, and The Air Force Office of Scientific Research).

Conclusions on software
The comparison ended with GeNIe and SMILE taking the lead. The pair provides a combination of GUI support and programming language interfaces. Most of the desired features are present, with the only major setbacks being the closed source code, and the limited and poorly documented support for non-discrete distributions. The apparent prior support from the defence sector was also taken as a positive indicator.

Further analysis of selected software
GeNIe

Dynamic Bayesian networks
A dynamic Bayesian network is a Bayesian network which takes into account the time dependence of variables. In practice, dynamic networks are simply Bayesian networks with a specific type of structure. Therefore, they can be analysed with any Bayesian network inference software. Specific support for dynamic networks is commonly understood to mean features, which ease the creation and analysis of dynamic networks, (Figure 3).

When creating a dynamic network, one first creates a Bayesian network describing the dependencies among variables on a given time step. Afterwards, the time interval under observation is divided to a discrete set of times, and the network is copied for each element in the discrete time set. Finally, arcs are added between time steps describing the effect of the previous state on the next one. Dynamic networks can also have nodes describing the initial parameters or final results, and nodes outside the time progression that are constantly in effect.

In GeNIe, dynamic networks are created by enabling the Temporal Plate. The dependency network is built in the main Temporal Plate portion. As one creates arcs in the Temporal Plate, the program asks, how many time steps there are between the child and the parent. Unlike other networks, one can connect a node to itself, as long as the connection has at least a one time step interval. If, for example, the size of a probability table depends on the time step, there is an additional menu in the editor for determining the time step under edition.

The Init Conditions field is for nodes affecting only the initial state. Connections can be made from the Init Conditions field to the Temporal Plate, in which case the connection leads only to the first of the time step networks. Similarly, the Term Conditions field is for connections where the parent is in the final time step of the network. Nodes outside the temporal plate can be parents to nodes inside the temporal plate. In this case, the arc is applied to every time step in the network. Unfortunately, there is no support for arcs in the opposite direction, that is, arcs from the Temporal Plate to a node outside it.

Inference for dynamic networks in GeNIe means solving the time evolution of variables. For every time dependent node, a table is compiled containing the marginal probabilities of that node on every time step. Several graphical presentations for the information are also available. Alternatively, the network can be unrolled, which results in a copy of the network that the program treats as a regular Bayesian Network. The unrolled version is freely editable, but cannot be converted back to the Temporal Plate form.

Figure 3. GeNIe and SMILE support dynamic networks, which can encode time-dependent phenomena. The above network represents a simple predation model. The number one on the arcs indicates that the end of the arc is in the next time step.
GeNIe does not support dynamic networks with decision or utility nodes. While one can add decision and utility nodes to a dynamic network, the resulting network is not solved correctly without unrolling, and some of the dependencies may disappear upon unrolling the network.

**NoisyMax-nodes**

Commonly in Bayesian networks, a node’s parents are either causes or requirements which act independently of each other. To ease the creation of such conditional probability tables, most Bayesian network software implements some version of NoisyOr and NoisyAnd nodes. In GeNIe and SMILE these have been implemented with the NoisyMax node type.

NoisyOr nodes are used to model independent causes. For example in a diagnostic network, multiple different diseases may have a probability to cause a certain symptom, and the diseases have no significant effects on each other’s symptom probabilities. When modelling a case like this in GeNIe, the node can be turned into a NoisyMax node by changing its node type. In both the cause and effect nodes, the states are assumed to be in a descending order of intensity, with the last state being the regular state.

The conditional probability table of a NoisyMax node has a probability distribution for each state of each cause node. These distributions are the probabilities that the cause node causes the effect in the absence of any other causes. The lowest state of every cause node is automatically locked to result in the lowest state of the effect node with probability 1. The resulting probability distribution for the effect node corresponds to the scenario, where every cause node results in some state of the effect node according to the given distribution, and the states are combined so that the highest effect wins.

Besides the probability tables for parent nodes, there is also an additional leak probability distribution in the definition of a NoisyMax node. This scenario describes the probability distribution of the effect in absence of all the parent causes. An alternative description for this is the probability that the effect will be caused by effects not taken into account by the model. GeNIe also contains the option to show the regular conditional probability table of a NoisyMax node.

NoisyAnd nodes, on the other hand, describe independent requirements. This can be thought as the parent nodes being factors, the absence of which prevents the effect from occurring. NoisyAnd nodes can also be modelled via NoisyMax nodes. This is achieved by reversing the state order of the effect and parents in the NoisyMax probability table. The leak field is now the probability distribution of the effect when all requirements are fully present. In addition to this, the probability distributions of the requirements represent the reduction in the probability of the effect occurring, when said requirement alone is not present.

When applicable, the use of NoisyMax nodes greatly reduces the size of the conditional probability tables involved. If the $i$:th parent of a given node has $S(i)$ states, a traditional conditional probability table would require the definition of $\prod_i S(i)$ conditional probability distributions. The additional structural assumptions of NoisyMax nodes reduce the amount of probability distributions to $1 + \sum_i (S(i) - 1)$. For example, when a node has $n$ binary parents, the improvement is from $2^n$ distributions to $n + 1$ distributions. Do note though, that the advantage gained is limited to the network construction phase. While there exist computational algorithms which take advantage of
the NoisyMax structure, such as the one outlined in (Zhang & Poole, 1996), GeNIe and SMILE do not as of now implement any of these algorithms.

**Value of information**
Calculating the value of information is a useful basic operation in analyzing influence diagrams. If one is using software which gives the resulting utilities as expected values of optimal policy, this is easy to compute via adding the relevant informational arcs and comparing results. As GeNIe and SMILE give the results of influence diagram inference as a list of expected utilities for every relevant state combination, this approach is not applicable. Due to this, the programs come with a specialized feature for computing values of information.

In GeNIe, the calculator for values of information is found under the Network menu. It has three parameters: A set of chance nodes, a point of view node, and the decision node before which the information is attained. The result is the gain in expected utility when the states of the given chance nodes are known at the decision node of interest. The utility gains are indexed by the information known at the point of view node. Therefore, the value of information represents the expected advantage gained by precise knowledge of the states of the given chance nodes.

**Diagnostic mode**
GeNIe and SMILE have extensive support for Bayesian network –based diagnosis. Diagnostic networks have two special categories for nodes. Target nodes represent the possible faults under diagnosis. Observation nodes, on the other hand, are information obtainable through tests or questioning. When using diagnostic networks, the main goal is finding out the faults the system has.

In GeNIe, diagnostic features are off by default, but can be activated from the Network menu. In diagnosis mode, the node properties of every chance node gain a field for defining, whether it is a target node, an observation node, or an auxiliary node used to simplify network structure. In addition to this, one can define the faulty states for each target node.

The diagnosis window of GeNIe contains a list of faulty states with a probability for each one of them. Selecting a faulty state gives a list of unobserved observation nodes with a diagnostic value for each one of them. The diagnostic value estimates, how much the probability estimate of the faulty state in question would improve by finding out the state of the observation node. One can fix the states of the observation nodes through the list in the diagnosis window and with every change the fault probabilities and diagnostic values are updated. The same network can be used for multiple diagnostic cases, and the program supports saving of case data.
There is also support for costs of observation. For each observation node, one can define a node-specific cost of observation. In this case, the diagnostic window replaces the diagnostic values with cost-effectiveness values. The cost-effectiveness values are defined as the difference of the diagnostic value and a weighted cost of observation. The weight for costs of observation can be adjusted in the diagnosis window. Setting the weight to zero ignores the costs entirely. One can also define more complicated network-based costs of observation. For example, one can define a cost of observation, where observing one node reduces the observation cost of some other nodes, or eliminates the cost entirely.

Strength of influence
GeNIe has a feature for visualizing the strengths of influence among nodes. The feature is activated from the Network menu or from the top toolbar.

The feature takes advantage of node state ordering. From the node properties page of a given node, one can set the states to be in an ascending or descending order of strength. If the orders of strength have been set, GeNIe colours the arcs in the network according to the type of the influence. Green arcs represent strengthening effects, red arcs weakening effects, and purple arcs are used for effects, where a clear direction of influence cannot be determined. If one of the nodes adjacent to a given arc has no state order, the arc in question is coloured blue.

The actual strength of influence is represented via arc thickness. An accurate numerical value can also be seen by placing the mouse cursor over the tip of the arc. The strength of influence toolbar gives four different measures of strength: Euclidean, Hellinger, J-Divergence and CDF. The user manual for GeNIe does not really make the differences between different measures clear. However, detailed information on the measures can be found in (Koiter, 2006), which outlines the development of the feature.

The measures are defined through the following formulae:

\[
\text{Euclidean: } \sqrt{\frac{\sum (p_i - q_i)^2}{2}}
\]
The Euclidean measure treats the distributions as points in an $n$-dimensional space, and gives the Euclidean distance between these two points. The Hellinger measure is similar, but the coordinates are the square roots of probabilities instead. Taking the square roots of the coordinates raises the sensitivity of the measure to changes in probabilities near zero. This is a useful property, since for example a change in probability from 0.5% to 1% can be of significantly greater importance in results than a change from 45% to 45.5%.

Due to its logarithmic definition, J-Divergence gives an even further weight to changes in probabilities close to zero. In fact, if any probability of zero becomes nonzero or vice versa, the influence strength immediately maximizes. Furthermore, the user can fix the normalizing parameter of J-Divergence, where higher parameter values further increase relative differences when dealing with probabilities close to zero.

The last measure, CDF, is made to take into account the amount of states a probability distribution moves by. For the last three measures, it is for example irrelevant whether in the shift from $P$ to $Q$ the main probability mass shifts from the first to the second state or from the first to the fifth state. CDF on the other hand takes into account the portion of states the probability mass moves. Note that the state movements are considered proportional: A shift of one state in a two state node is considered similarly large as a shift of four states in a five state node, as both are shifts from one endpoint to the other.

Besides these, there are two further setting which affect the influence strength calculation. The first one is the method of combining results. In the influence strength computation, the previously outlined distance is calculated for every parent state. This setting defines how the overall distance is formed from these parent state distances. The default setting is Average, in which the combination
method is the common arithmetic mean. Alternative options are Maximum, in which the maximum distance is used, and Weighted, in which the distances are summed with the marginal probabilities of parent states as weights.

The second setting is the direction of influence. In dynamic mode, the influence strength can be calculated in either direction. Under the default settings, if one of the nodes adjacent to the arc is a target node, the influence is calculated from the non-target node to the target node. If both of the adjacent nodes are target nodes, the influence is calculated both ways and the results averaged. The directions can be freely changed by the user. It is noteworthy that a bidirectional influence can change the colour of the arc. An arc where the parent-to-child effect is clearly strengthening may turn purple in dynamic mode, if the influence is calculated bidirectionally and the child-to-parent influence is not unquestionably strengthening.

The graphical representation for influence strengths is a useful way of gaining a general picture of the network structure. When using it, however, one has to be careful, as incomplete knowledge of the available settings can easily result in misunderstandings. Lastly, the feature has only partial support for influence diagrams. Arcs from decision nodes to chance nodes work as they should, but the influence strength for arcs into utility nodes is not calculated, even though a method for it is outlined in (Koiter, 2006).

SMILE, jSMILE and SMILE.NET
The SMILE C++ library is the primary programming interface of the software pair. The other interfaces, jSMILE and SMILE.NET are wrappers made on a higher level of abstraction. The jSMILE interface is based on JNI, which allows Java code to call C++ code. SMILE.NET on the other hand is made in Managed C++, a similar bridge between C++ and .NET-languages, which is currently deprecated. Although the source code for SMILE is not open, the source code for the interfaces is available. Therefore, if the higher abstraction level of the interfaces makes some feature difficult to implement, a user with knowledge in the requisite language may add features to the interface as required.

The features of GeNIe presented above are mostly present in SMILE and its interfaces. For example there exist classes for NoisyMax nodes and value of information computations, and diagnostic properties have multiple classes devoted to them. The strength of influence values are however a part of GeNIe, and therefore are not present in any available version of SMILE.
SMILE differentiates node types based on the formats of their definitions. For example the definition of a decision node is a list of choices, so the node type is DSL_LIST. Chance nodes, which are defined through conditional probability tables, have the node type DSL_CPT. Utility nodes are defined as a table with a utility value for each combination of parent states, so their node type is DSL_TABLE. Similar names are used in the interface libraries. The actual definitions and inference results are saved as one-dimensional arrays. The library however contains utility classes for more complicated structures, such as DSL_Dmatrix, and also provides methods for converting indices from one form to another.

The update logic for networks is completely internal. The programmer can influence the same parameters which are available in GeNIe, such as algorithm choice and the sample size of sampling algorithms. However, the actual computation of marginal probabilities is done with a single update call on the network.

**Inference complexity and algorithm choice**

**Computational complexity**

Exact inference on Bayesian networks has been shown to be NP-hard (Cooper, 1990). This holds even in a very limited special case, where a node can have at most 4 parents and 5 children. Known polynomial solutions for special cases are scarce. The most significant one is Pearl’s message passing algorithm, which solves polytrees in polynomial time. A network is a polytree if the corresponding undirected network has a tree structure.

In addition to this, even approximate inference is NP-hard. This fact has been shown in (Dagum & Luby, 1993) for both deterministic approximation algorithms, and approximation algorithms based on random numbers. The article does mention one special case not in the NP-hard complexity class: Random number based approximate inference with no fixed evidence is in the RP complexity class. RP stands for Randomized Polynomial time, and the RP complexity class is a class in between the P and NP classes. Its exact size with respect to P and NP is currently unknown in the same vein as the sizes of P and NP with respect to each other.

There is no single best algorithm for Bayesian network inference. The run times of algorithms, and in the approximate case the accuracy of results, may depend on completely different aspects of the network. For example, sample based algorithms scale with the amount of nodes and commonly have trouble with very rare evidence, search based methods are not suited for networks with distributions close to uniform, and the speed of exact inference is heavily reliant on the arc structure. Therefore, it is recommended to choose the inference algorithm based on the type of network it will be used on.

**Available algorithms in GeNIe and SMILE**

Many of the algorithms of GeNIe and SMILE are simple and mainly intended for educational and comparison purposes. Out of the available algorithms, two can be considered to be intended for actual fast inference: *Clustering* and *EPIS-sampling*. 
Clustering, also known as clique tree propagation or the junction tree algorithm, is the main exact solution algorithm in GeNIe and SMILE. It originates in the 1988 paper (Lauritzen & Spiegelhalter, 1988), and has seen several improvements and variations over time. For an exact algorithm, it is relatively fast, and it can be used to do inference even in moderately large networks. However, in the case of an extremely large network of chance nodes, approximate algorithms are generally more practical.

Clustering is based on transforming the network to a tree of node clusters following certain properties, also called a clique tree. The clique tree is then solved using a method similar to Pearl’s message passing. The main measure of the algorithm’s runtime is treewidth. Treewidth is a measure describing the size of the largest node in the optimal clique tree. Solving a clique in the tree increases exponentially in difficulty with respect to the size of the clique. Therefore, the computational complexity of solving the clique tree is primarily determined by the size of the largest clique. Merely determining the treewidth of a clique tree is NP-hard. Therefore, clustering algorithms rarely search for the optimal clique tree, and instead use heuristic methods to find a tree of sufficiently low computational complexity.

EPIS-sampling (Evidence pre-propagation importance sampling algorithm) is a hybrid algorithm based on multiple approximation algorithms. A detailed description of the method can be found in (Yuan & Druzdzel, 2002). The article contains tests, in which the algorithm was found to surpass previously known approximate algorithms on the given test networks.

EPIS-sampling uses importance sampling, which is based on using a weight distribution for sampling. The method increases in efficiency the closer the weight distribution is to the actual distribution. Therefore, importance sampling is particularly effective in refining existing approximations. The weight distribution is determined through loopy belief propagation. Loopy belief propagation refers to the use of Pearl’s message passing algorithm on networks other than polytrees. Although loopy belief propagation does not yield exact inference results, practical tests have shown the iterative use of it to give surprisingly useful approximate results.

**Test results on computation times**

In order to test the inference speed of GeNIe and SMILE, the authors conducted a set of speed tests on randomly generated nets. The generated networks were influence diagrams divided in three layers: The bottom layer consisted of a set of decision nodes, the middle layer of chance nodes, and the top layer of utility nodes. Arcs were directed upwards, and no evidence was fixed. The measurements were performed using jSMILE.

Next, the random generation algorithm used to generate the networks is outlined. At first, the previously described sets of nodes were created, with every set having a given size, and the amount of states on a single node having a given upper and lower bound. To ensure that the resulting network is acyclic, ancestral ordering was used. In ancestral ordering, the nodes are given an integer order, and the parents of every node are lower in the order than the node itself. It is a widely known fact that a directed network is acyclic if and only if it has an ancestral order.

The nodes were ordered so that the decision nodes were first in the order, and the utility nodes were last. The algorithm formed the arcs by selecting a random endpoint, and then selecting another
random endpoint from the set of nodes, which according to its type can be connected to the first node. The main limitation on connections was for connections between two chance nodes or two utility nodes to be prohibited. The connections representing the decision order were added separately. Once the algorithm had generated a node pair, it attempted to create an arc between them in the direction given by the ancestral ordering. There were three possible scenarios: the creation succeeded, there was already an arc between said nodes, or a predetermined maximum amount of parent for a given node was surpassed. This operation was then repeated until a given amount of arcs was reached.

The conditional probability tables were formed in the following way: Assume the algorithm is given a chance node with \( n \) states. For every state combination of its parents, the algorithm generates \( n - 1 \) random numbers in the interval \([0, 1]\). These numbers divide the interval to \( n \) subintervals, the lengths of which are the state probabilities. For utility nodes, the utility values were generated simply by selecting random integers from the interval \([0, 1000]\).

Tests indicated that the average computation length increased rapidly with the amount of decision nodes. This was observed through repeated sets of tests, in which the network was expended whenever the next set of tests was undertaken. The expansion included adding a new decision node with three states, and adding enough arcs to keep the network arc density approximately constant. The average computation time ratios of successive sets were found to be rather close to three, the amount of states each new decision node had. Similar tests were made with chance and utility nodes. The slowest increase obtained was with the amount of utility nodes.

While decision nodes had the greatest effect on the average computation time, chance nodes had the greatest effect on computation time deviations. A large number of chance nodes resulted in an increase in special cases with significantly greater computation times than average. These observations coincide with what we know of the algorithms GeNiE and SMILE use: For decision nodes, the programs compute each combination of decisions separately. This results in a direct correlation among decision combinations and computation time. On the other hand, the difficulty induced by chance nodes is mainly defined by the largest node of the clique tree. Therefore, network difficulty varies alongside with the complexity of the clique tree generated by the clustering algorithm.

The exact clustering algorithm solved most of the networks tested relatively quickly. A random network with 6 decision nodes, 6 utility nodes, 20 chance nodes, 100 arcs, and 3 states per node, took on average less than 2 second to solve with clustering. However, there is a possibility that the chance nodes form a hard clique tree, and the computation is orders of magnitude more difficult.

The authors have only preliminary results on EPIS-sampling. Since the expected utilities of randomly generated nets are generally very close to each other, the approximation errors in the results given by EPIS result in the optimal decisions differing heavily from those given by the exact results. In a practical situation, where the decisions made have a greater impact on the outcome, the results obtained by EPIS may be of greater use. The sample size of EPIS is user adjustable: This is done from the Network Properties window in GeNiE, or with the SetNumberOfSamples method of the Network class in SMILE. The runtime of EPIS grows linearly with the sample size, with a fixed initial time requirement caused by the loopy belief propagation step.
The networks tested were in a size category where clustering still outperforms EPIS. It is also notable that both algorithms scale equally with additional decision nodes. Therefore, EPIS is the recommended algorithm only for networks with a highly complex chance node structure.

Conclusions

In this article, Bayesian networks and influence diagrams were reviewed from the point of view of defence applications. Various applications of Bayesian networks and influence diagrams in military related fields were enumerated. Examination of publicly available source material yields a multitude of existing defence applications for the models of interest.

Existing free software for Bayesian network and influence diagram inference was compared. It was determined that from a defence point of view, a combination of graphical representation features and programming interfaces is recommended. Most available software with the required features was heavily focused on only one of these aspects. The most effective combination of these two was found in GeNIe and SMILE.

An examination of the features of GeNIe and SMILE was performed. Within this examination, capabilities of the two programs were reported, as were encountered issues and deficiencies. In addition to this, inference complexity and the available algorithms of GeNIe and SMILE were investigated. The investigation concluded with the exact algorithms being preferable for most influence diagrams, with approximate algorithms being of use only for models with a large and highly complicated chance node structure.
References


CISIAD, UNED. (2013). OpenMarkov Tutorial Version 0.1.3.


In addition to the applications of GeNIE and SMILE mentioned in the text, the graphical models presented in the images of this work have been created with GeNIE.