BN-Tools: A Software Toolkit for Experimentation in BBNs

Benjamin Perry ~ Julie Stilson

Laboratory for Knowledge Discovery in Database, Kansas State University
234 Nichols Hall
Manhattan, KS 66506-2302
bbp9857@ksu.edu ~ jas3466@ksu.edu

Abstract

In this paper, I describe BN-Tools, an open-source, Java-based library for experimental research in Bayesian belief networks (BBNs) that implements several popular algorithms for estimation (approximate inference) and learning along with a graphical user interface for interactive display and editing of graphical models.

Introduction

BBNs are one of the most widely-studied and applied family of knowledge representations for learning reasoning in uncertain environments. As a result, we are interested in developing efficient software toolkits in Java that implement known algorithms for exact and approximate inference, in a modular library that facilitates experimentation. Over the last two years, I have developed several tools in Java that help manipulate BBNs and extrapolate important information from them. Our toolkit consists of core BBN representation classes, the Lauritzen-Spiegelhalter (LS) [LS88] algorithm, an optimized implementation of the *K2* algorithm for structure learning [CH92], a genetic algorithm (GA) wrapper for *K2*, a genetic algorithm to find the *most probable explanation* (MPE), an adaptive importance sampling (AIS) [CD00] algorithm for approximate inference, and several other format conversion, viewing, and editing utilities, including a data simulator.

Implementation

Core classes

The core classes for dealing with a BBN consist of ways to represent each individual node, the network itself, and instantiation managers that allow manipulation of node values while protecting the internal network from changes. These classes are used throughout all of our BBN tools and provide a standard interface for manipulating the network. The network can be built at run-time or loaded from the XML *Bayesian Network Interchange Format* developed by Microsoft (or one of several commercial formats including those for the BBN software packages *Hugin*, *Netica*, *SPI*, and *IDEAL*).

Lauritzen-Spiegelhalter algorithm.

The purpose of the LS algorithm is to find the posterior probabilities of nodes in a BBN with or without some nodes being set to a pre-defined value (evidence nodes). The LS first triangulates the BBN into cliques based on the existing arcs. These cliques are then used in message propagation to pass various probability values to each clique's children and parent(s). Once the algorithm has completed its initial run, all instantiated nodes are removed and the algorithm is repeated, starting with the values computed from the first run. When the algorithm is finished, we will be able to query any node to see how likely any state is given the evidence we specified. Our implementation is multi-threaded for parallel computation of the probability propagation. Due to the overhead of threads with JAVA, the performance takes a hit on multi-processor machines with smaller networks. Larger networks (100+ nodes) do see a slight performance boost.

Adaptive importance sampling algorithm.

Approximate inference algorithms that use sampling techniques to determine the posterior probabilities of nodes are extremely important because they directly model the real-life situations that BBNs represent better than any inference algorithm. One such approximation algorithm is Adaptive Importance Sampling (AIS), where the probabilities of each node are updated several times during probabilistic sampling of the network. These repeated updates help find the importance function for the network, which is an updated version of the conditional probability tables that corrects probabilities of states given that the states of certain nodes have already been set as evidence. This importance function is described as:

$$\begin{aligned} P^{k+1}(x_i \mid Pa(X_i), \mathbf{e}) = \\ P^k(x_i \mid Pa(X_i), \mathbf{e}) + \\ \eta(k) \cdot \left(P'(x_i \mid Pa(X_i), \mathbf{e}) - P^k(x_i \mid Pa(X_i), \mathbf{e})\right) \end{aligned} \quad (1)$$

where P^{k+1} is the new importance function, P^k is the importance function in the last step, P' is an estimated probability based on the samples taken in the last step, η is the pre-determined learning rate, and k is the number of updates that have been made to the importance function.

e corresponds to the pre-defined evidence you set prior to running the network through AIS. AIS also employs two heuristics which adapt the conditional probability tables for unlikely evidence and extremely low probabilities by setting the parents of unlikely evidence nodes to a uniform distribution and by raising low probabilities to a set threshold. These two heuristics allow the posterior probabilities be learned properly despite very low prior probabilities. Because the heuristic initializations and the learned importance function allow AIS to learn the true probabilities better than traditional sampling, we currently use it as a fitness function for evaluating learned structures of BBNs based on different orderings.

K2 algorithm

The *K2* algorithm, developed by Cooper and Herskovits [CH92], learns the structure of a BBN based on a collection of training examples and a specific ordering of the nodes. It is a greedy-search algorithm and only considers nodes previous to the current node when building the list of parents (thus, the order is very important). To evaluate the output of *K2*, the LS algorithm is used to compute each node's probabilities. We then compute root-mean-square-error (RMSE) between the probabilities learned from the samples and the probabilities computed by some inference method with the K2-produced network. RMSE values closest to 0.0 are the best outcomes of the network. We are using other means of judging the fitness of the network as well: adaptive importance sampling and forward sampling. Both are approximates to LS, which will be a little less accurate, but (theoretically) much faster for large networks.

K2 Genetic Algorithm Wrapper.

Finding the optimal ordering is *NP*-hard, so we have designed a genetic algorithm to avoid exhaustively searching all possibilities. RMSE values of the various orderings are used as the fitness. Each chromosome in our genetic algorithm corresponds to an ordering used in K2. We use an order-crossover operator as well as an order swap operator for mutation. The order-crossover's purpose is to maintain the order of the nodes while still swapping out with another chromosome. There are other methods of crossover that we will implement and test in order to improve the performance of our genetic algorithm. We have utilized *GAJIT* as the main genetic algorithm driver. GAJIT is an elitist GA. It allows the user to specify the cull rate (the percentage of chromosomes that make the cut to the next generation). It also allows the user to add self-designed crossover and mutation operators. There are some issues within GAJIT that we will attempt to address in the future, most notably the crowding affect. When selecting two chromosomes to crossover, GAJIT does not care if a chromosome crossovers with itself. Applying the order-crossover technique on two identical chromosomes will yield two identical children, thus filling the population with clones, making it difficult to get out of a local optimum at times. We have developed a job-farm that allows several computers to work on the same generation simultaneously. The job farm utilizes TCP to communicate and is tolerant to lag or dropped connections. Because we use java, we can employ several machines, regardless of their platform. The server manages the genetic algorithm. When a client connects, the server sends important session parameters. Once the client is initialized, the server simply sends an ordering to the client. The client then runs the K2 algorithm on the ordering, runs the specified fitness function on the learned network, and finally sends back to the server the final fitness. The job farm is also capable of running an exhaustive permutation run on smaller networks. We have begun experiments with the K2 GA wrapper by comparing the fitness values among the gold standard Asia and Alarm networks, the canonical ordering (or any topologically sorted ordering), and the best ordering obtained through the wrapper. Our initial findings are very encouraging; the GA wrapper usually produces a network that differs from the gold standard by a small percentage of graph errors (missing, added, or reversed edges).

Genetic algorithm for finding the MPE.

Finding the Most Probable Explanation (MPE) of a network is very useful, although it is *NP*-hard. We have employed a GA to speed up the process by 'stacking the deck' as each generation progresses. A chromosome represents node value instantiations for all nodes in the network. Using *GAJIT* [Fa00] as our main GA driver, we have developed a Markov-blanket crossover and chromosome (sparse state coding) rotation as a mutation operator. The Markov blanket of radius *n* consists of a node's children, mates, parents, and the node itself, recursively computed *n* times for all nodes. We use a default radius of 1 in our GA. Although it is not guaranteed to find the MPE, the GA performs well given a sufficient number of generations and population size.

References

[CD00] Cheng, J. and Druzdel, M. J. 2000. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research (JAIR)*, 13:155-188

[CH92] Cooper, G. F. and Herskovits, E, 1992.. A Bayesian Method for the Induction of Probabilistic Networks from Data. Machine Learning, **9**(4):309-347.

[LS88] Lauritzen, S. L. and Spiegelhalter , D. J, 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B* 50.