A Permutation Genetic Algorithm for Variable Ordering in Learning Bayesian Networks from Data

William H. Hsu **Haipeng Guo** **Benjamin B. Perry** **Julie A. Stilson**

Laboratory for Knowledge Discovery in Databases, Kansas State University
234 Nichols Hall, Manhattan, KS 66506
{bhsu | hpguo | bbp9857 | jas3466}@cis.ksu.edu http://www.kddresearch.org

Abstract

Greedy score-based algorithms for learning the structure of Bayesian networks may produce very different models depending on the order in which variables are scored. These models often vary significantly in quality when applied to inference. Unfortunately, finding the optimal ordering of inputs entails search through the permutation space of variables. Furthermore, in real-world applications of structure learning, the gold standard network is typically unknown. In this paper, we first present a genetic algorithm (GA) that uses a well-known greedy algorithm for structure learning (*K2*) and approximate inference by importance sampling as primitives in searching this permutation space. We then develop a flexible fitness measure based upon inferential loss given a specification of evidence. Finally, we evaluate this GA wrapper using the well-known networks *Asia* and *ALARM* and show that it is competitive with exhaustive enumeration in finding good orderings for *K2*, resulting in structures with low inferential loss under importance sampling.

Keywords: Bayesian networks, genetic algorithms, permutation problems, probabilistic reasoning, machine learning, wrappers

1 INTRODUCTION

Learning the structure, or causal dependencies, of a graphical model of probability such as a Bayesian network (BN) is often a first step in reasoning under uncertainty. In many machine learning applications, it is therefore referred to as a method of *causal discovery* [PV91]. Finding the optimal structure of a BN from data has been shown to be *NP*-hard [HGC95], even without considering latent (unobserved) or irrelevant (extraneous) variables. Therefore, greedy *score-based* algorithms [FG98] have been developed to provide more efficient structure learning at an accuracy tradeoff. In this paper we examine a general shortcoming of greedy structure learning – sensitivity to variable ordering – and develop a genetic algorithm to mitigate this problem by searching the permutation space of variables using a probabilistic inference criterion as the fitness function.

We make the case in this paper that the probabilistic inference performance element, **in the absence of a known gold standard network** or any explicit constraints, can provide the feedback needed to search for a good ordering. We then derive a heuristic based on validation by inference (exact inference [LS88, Ne90] for small networks, approximate inference by stochastic sampling [CD00] for larger ones). Our primary objective is inferential accuracy *using* the learned structure.

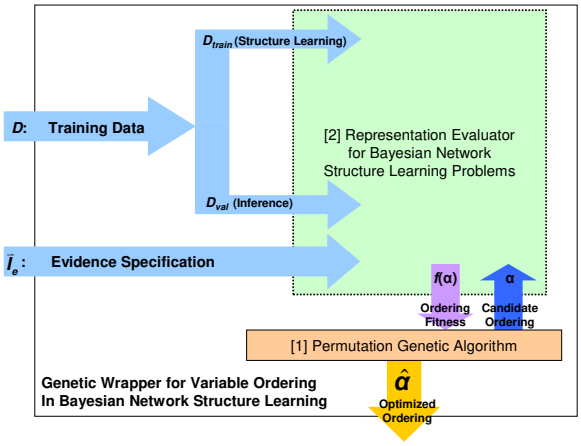

Figure 1. System Design Overview.

Toward this end, we adapt a flexible, composite fitness measure used in other machine learning systems called *wrappers* [KJ97], which automatically tune hyperparameters of the learning system such as the ordering of input variables. We present the system shown in Figure 1, a genetic algorithm-based wrapper [CS96, RPG+98, HWRC01], and show how it provides a parallel stochastic search mechanism for inferential loss-minimizing variable orderings. We demonstrate that, used in tandem with *K2*, it produces structures whose loss under importance sampling is nearly as low as any found by exhaustive enumeration of orderings. Finally, we discuss how this wrapper provides a flexible method for tuning *representation biases* [Mi97] in Bayesian network structure learning using different fitness criteria.

2 VALIDATION OF STRUCTURES

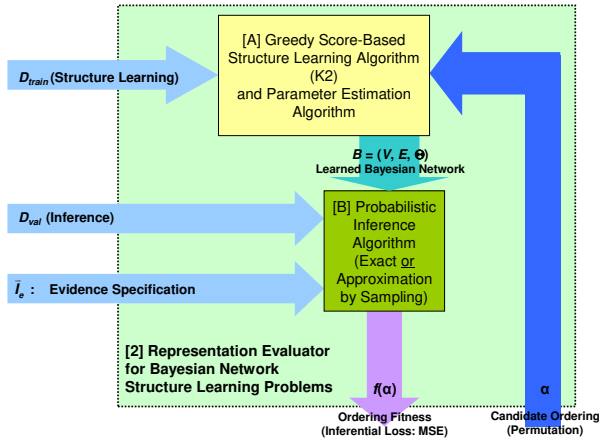

Figure 2. Probabilistic reasoning environment, Module [2] from Figure 1.

Consider a typical probabilistic reasoning environment, as shown in Figure 2, where structure learning [A] is a first step. The input to this system includes a set D of training data vectors $\mathbf{x} = (x_1, \ldots, x_n)$ each containing n variables. If the structure learning algorithm is greedy, an ordering $\boldsymbol{\alpha}$ on the variables may also be given as input. The structure learning component of this system produces a graphical model $B = (V, E, \Theta)$ that describes the dependencies among X_i, including the conditional probability functions. The inferential performance element [B] of this system takes B and a new data set D_{test} of vectors drawn from the desired inference space, where only a subvector $\mathbf{E}$ of $\mathbf{X} = (X_1, \ldots, X_n)$ is observable, and infers the remaining unobserved values $\mathbf{X} \setminus \mathbf{E}$. We denote the indicator bit vector for membership in $\mathbf{E}$ by $\mathbf{I_e}$. The performance criterion f is the additive inverse of the (inferential or utility) loss of [B].

This section specifies the functionality of [A] and [B] and explains the derivation of f as a function of the ordering $\boldsymbol{\alpha}$. In the next section, we show how the environment depicted in Figure 2 is used as the fitness evaluation module [2] of the overall GA-based system (Figure 1).

2.1 Learning Bayesian Network Structure

Consider a finite set $\chi = \{X_1, \ldots, X_n\}$ of discrete random variables. A *Bayesian network* is an annotated directed acyclic graph $G = (V, E)$ that encodes a joint probability distribution over χ. The nodes of the graph correspond to the random variables $X_1, \ldots, X_n$. Each node is annotated with the conditional probability distribution (CPD) that represents $P(X_i \mid Pa_{xi})$, where Pa_{xi} denotes the parents of X_i in G. A Bayesian network B specifies the unique joint probability distribution over χ given by:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P\left(X_i \mid Pa_{x_i}\right) \quad (1)$$

The graph G represents conditional independence properties of the distribution. These are the *Markov independencies*: each variable X_i is independent of its non-descendants, given its parents, in G. [EF01] We denote the annotating CPD parameters of B by Θ; thus, $B = (V, E, \Theta)$.

We are interested in learning B from training data D consisting of examples $\mathbf{x}$. For simplicity, we assume that there are no variables that are latent or completely irrelevant (not weakly relevant [KJ97]). The objective of structure learning is then to find the arcs E for $V = \chi$. Some structure learning algorithms, such as *K2* [CH92], are greedy in that they add arcs based upon the incremental gain that each single arc induces in a global score, such as the Bayesian (Dirichlet) score. [CH92, FG98]. We use *K2* for structure learning – module [A] of Figure 2 – because it finds structures quickly *if* given a reasonable ordering $\boldsymbol{\alpha}$. Variables must occur "upstream" from one another (or "downstream" in $\boldsymbol{\alpha}$, i.e., have a higher index) to be considered as candidate parents. If the number of parents per variable is constrained to a constant upper bound, *K2* has worst-case polynomial running time in the number n of variables.

Two clear limitations of greediness are inability to backtrack (i.e., undo the addition of an arc) or consider the joint effects of adding multiple arcs (parents). This is why greedy structure learning algorithms are sensitive to the presence of irrelevant variables in the training data, a pervasive problem in machine learning [KJ97]. Additionally, *K2* is particularly sensitive to the variable ordering because arcs fail to be added, resulting in unexplained correlations, whenever candidate parents are evaluated in any order that precludes a causal dependency. Were a gold standard structure $G^* = (V, E^*)$ available, this would be seen as an inversion in the partial ordering induced by E^*. Preventing missing arcs – i.e., "false negatives for causality" – is a challenge in structure learning as applied to causal discovery [PV91, FG98].

Unfortunately, just as finding the optimal structure is itself intractable [HGC95], so is finding the optimal ordering of inputs for a given structure learning algorithm. Searching the space of permutations of variables is prohibitive, and defeats the purpose of using a greedy algorithm. In this paper, we focus on *K2* and the problem of optimizing the variables to be given as its input. To specify the optimization of variable order as a search problem, we must define the states (permutations), operators (re-ordering), initial candidates, and evaluation criterion.

2.2 Validation by Inference

A desired joint probability distribution function $\mathbf{P}(\mathbf{X})$ can be computed using the chain rule for Bayesian networks, given above in Equation (1). The *most probable*

explanation (*MPE*) is a truth assignment, or more generally, value assignment, to a *query* $\mathbf{Q} = \mathbf{X} \setminus \mathbf{E}$ with maximal posterior probability given evidence **e**. Finding the MPE directly using Equation (1), requires enumeration of exponentially many explanations. Instead, a family of exact inference algorithms known as *clique-tree propagation* (also called *join tree* or *junction tree* propagation) is typically used in probabilistic reasoning applications. The first of these algorithms was developed by Lauritzen and Spiegelhalter [LS88, Ne90]. Although exact inference is important in that it provides the only completely accurate baseline for the fitness function *f*, the problem for general BNs is *#P*-complete (thus, deciding whether a particular truth instantiation is the MPE is *NP*-complete) [Co90, Wi02].

Approximate inference refers to approximation of the posterior probabilities given evidence. One stochastic approximation method called *importance sampling* [CD00] estimates the evidence marginal by sampling query node instantiations:

$$\mathbf{P(E = e)} = \sum_{\mathbf{X \setminus E}} \mathbf{P}\left(\mathbf{X \setminus E} \mid \mathbf{E = e}\right) \tag{2}$$

[CD00] discusses basic variants of importance sampling. These include *probabilistic logic sampling* [He86], whose importance function is the joint distribution function $\mathbf{P}(\mathbf{X})$. By sampling from the network as if no evidence were given, the priors on source or root nodes are emphasized, resulting in a possibly suboptimal importance function as the authors point out. The source priors are similarly emphasized in *forward simulation* by likelihood weighting [SP89, CD00], which samples using the joint probability of query nodes as the importance function:

$$\mathbf{P(X \setminus E)} = \sum_{x \notin e} P\left(x_i \mid Pa_{x_i}\right) \tag{3}$$

Welch demonstrates [We96] that even a moderately complex binary network with deterministic nodes, approximately the size of *ALARM*, can be difficult to sample from by pure forward sampling if there are enough query nodes (evidence) – the author instantiates 4 of 32 binary nodes with a moderately unlikely evidence vector, $\mathbf{P(e)} = 6.5 * 10^{-4}$.

One way of scaling up to large networks in a realistic probabilistic reasoning application is to dynamically adapt the importance function. [CD00] presents a solution of this type called *adaptive importance sampling* (*AIS*), where a dynamic importance function is first initialized using structural heuristics, then empirically trained in each of several training steps. This is similar to the hyperparameter sampling stages in Markov chain Monte Carlo (**MCMC**) methods [Ne93]. The key issue is whether we have any prior knowledge regarding the estimators (e.g., heuristic importance functions).

We have implemented five variants of importance sampling: forward simulation, logic (*aka* rejection) sampling, backward sampling, self and heuristic importance sampling, and adaptive importance sampling. Because adaptive importance sampling has been empirically shown [CD00] to be more robust in the presence of unlikely evidence **e**, and because we have found it to converge quickly in independent experiments, we use it in our evaluation component, module [B] in Figure 2 above.

2.3 Deriving Fitness

To optimize the ordering, we considered fitness functions with three objective criteria. In this paper, however, we focus *solely* on the first:

1. **Inferential loss**: Quality of the network produced by *K2* as detected through inferential loss evaluated over a holdout validation data set $D_{val} \equiv D \setminus D_{train}$ (see Figure 1) – requires modules [A] and [B] in Figure 2
2. **Model loss**: "Size" of the network under a specified representation – requires module [A] only and is independent of [B]
3. **Ordering loss**: Inference and model-independent measure of data quality given only *D* and $\boldsymbol{\alpha}$ – independent of both modules [A] and [B]

$$f\left(\boldsymbol{\alpha}, D, \vec{\mathbf{I}}_{\mathbf{e}}\right) = a \cdot f_a\left(\boldsymbol{\alpha}, D, \vec{\mathbf{I}}_{\mathbf{e}}\right) + b \cdot f_b\left(\boldsymbol{\alpha}, D\right) + c \cdot f_c\left(\boldsymbol{\alpha}, D\right) \tag{4}$$

$$f_a\left(\boldsymbol{\alpha}, D, \vec{\mathbf{I}}_{\mathbf{e}}\right) = 1 - \sqrt{\frac{1}{\sum_{X_i \in \mathbf{X} \setminus \mathbf{E}} a_i} \sum_{X_i \in \mathbf{X} \setminus \mathbf{E}} \sum_{j=1}^{a_i} \left(P'\left(x_{ij}\right) - P\left(x_{ij}\right)\right)^2} \tag{5}$$

$$f_b\left(\boldsymbol{\alpha}, D\right) = 1 - \frac{\sum_{i=1}^{n} \left(a_i \cdot max\left(\prod_{\left|X_j \in \mathbf{Pa}_{X_i}\right|} a_j, 1 \right) \right)}{\prod_{i=1}^{n} a_i} \tag{6}$$

$$\textbf{\textit{where}} \quad a_i \equiv arity\left(X_i, B = \left(\chi, E, \Theta\right)\right)$$
$$\left(E, \Theta\right) = K2\left(\boldsymbol{\alpha}, D_{train}\right)$$
$$a + b + c = 1 \tag{7}$$

In related work on genetic wrappers for variable selection in supervised inductive learning, Hsu *et al* adapted Equation (4) [HWRC00, HWRC01] from similar fitness functions developed by Cherkauer and Shavlik for decision tree pre-pruning [CS96], Raymer *et al* for similarity-based learning (*k*-nearest neighbor regression) [RPG+97], and Whitley and Guerra-Salcedo for connectionist learning [GW99]. This breadth of applicability demonstrates the generality of simple genetic algorithms as wrappers for performance tuning in supervised inductive learning.

Recently, Hsu *et al* automatically validated the coefficients *a*, *b*, and *c* for several individual data sets on a supervised learning task. [HWRC02] Results were positive in that this approach found application-specific values for these *hyperparameters*, and the GA achieved better generalization accuracy than search-based feature selection wrappers [KJ97] for a real-world test bed (prediction of loss ratio in automobile insurance risk

analysis). Controlling the values of a, b, and c simultaneously proved to be difficult in that large amounts of validation data were required, and the authors report that experiments did not indicate conclusively whether the GA performed better with this single composite-objective fitness function or a multi-objective one (i.e., Pareto optimization). Therefore, for clarity, **we set b and c to 0 to ignore f_b and f_c** in the experiments reported in this paper. In the last section, we discuss the ramifications of this design choice and possible future work using the full f.

We now focus on the first term, f_a. This fitness function computes inferential loss by measuring the predictive power of the Bayesian network on the data set given a specification of evidence, $\mathbf{I_e}$. The specific f_a we use is the normalized additive inverse of the root mean squared error (RMSE), which is the square root of the sum of squared differences betweeen the sampled, approximate probabilities $P'(x_{ij})$ and exact probabilities $P(x_{ij})$, over states x_{ij} of variables X_i. [CD00] Note that f_a is the only term that depends on which variables are *observable*, i.e., members of **E**. We consider this the most important term just as validation set classification error is considered a typical estimator of generalization error in supervised classification learning [Mi97]. Ultimately, a BN B is only as good as the inferences it can produce on real-world data given realistic evidence **e**, and an ordering $\boldsymbol{\alpha}$ is only as good as the BN that it can induce given a specific structure learning algorithm. In the next section, we explain why this is a motivation for GA wrappers in general.

3 SEARCH-BASED ENHANCEMENT OF LEARNING

Figure 1 indicates the role of a combinatorial optimization system for controlling $\boldsymbol{\alpha}$, in context: a probabilistic reasoning system based on greedy structure learning can use an optimized ordering $\hat{\boldsymbol{\alpha}}$ to enhance structure quality. This is done by searching for a good $\boldsymbol{\alpha}$ using a "realistic" inferential criterion and a fixed, greedy structure learning algorithm such as *K2*. We now explore this combinatorial optimization problem and the design of our specific GA.

3.1 Wrapper Approaches to Optimizing Input

Tuning machine learning algorithms for large, complex data sets is an expensive and difficult task. In addition to identifying the appropriate inputs for a particular classification or inference performance element, the system designer must find a representation for hypotheses, i.e. the language for expressing the target concept, and a suitable performance measure by which to evaluate hypotheses. Making appropriate decisions regarding the input specification is crucial for tractable learning, because these determine part of the *inductive bias* [Be90, Mi97] of the learning system. *Bias*, the preferences of a learning system for one hypothesis over another other than those dictated by consistency with the training data, determines how the space of hypotheses (in our application, BN structures) is to be searched and can radically affect the tractability of this search. Unfortunately, effective decisions often depend in subtle ways upon the learning algorithm, training data, and their interaction. A mechanism for systematically identifying good inputs should take the performance element of the system input into account.[1] It must have the ability to tune the learning system by automatically adjusting the some aspect of the input specification (e.g., selected variables, *aka feature subsets*, or variable orderings $\boldsymbol{\alpha}$) and coefficients for quantitative inductive bias such as those discussed previously. Controlling all of these parameters, while keeping the machine learning system efficient and manageable, is not easy.

We approach this problem in BN structure learning by applying search-based combinatorial optimization and use *validation by inference* (presented in the previous section) as a search heuristic. The high-level mechanisms that determine a learning system's representation and preference biases can be expressed using learning *hyperparameters* [Ne93], such as $\boldsymbol{\alpha}$. Just as a learning parameter denotes a trainable component of a pattern detector or classification function, a learning hyperparameter denotes a controllable component of the organization, representation, or search algorithm for a learning problem. Inductive learning systems, or *inducers*, are built with such hyperparameters and the ability to tune them using combinatorial search, based upon evaluation metrics over validation data. The benefits to probabilistic learning and reasoning are the potential for greater flexibility in learning processes, an increase in generalization quality, and the ability to make the learning component more automatic and transparent.

3.2 GA-Based Wrappers

A GA is ideal for implementing wrappers where hyperparameters are naturally encoded as chromosomes such as bit strings or permutations. This is precisely the case with variable (feature subset) selection, where a bit string can denote membership in the subset, and with variable ordering, where a permutation denotes $\boldsymbol{\alpha}$, the order in which nodes are added to the BN. Both of these are forms of *constructive induction* where the input representation is changed from the default [Be90] – here, the full subset χ or an arbitrary ordering $\boldsymbol{\alpha_0}$.

With a GA-based wrapper, we seek to evolve hyperparameter values using the performance criterion of

[1] The term *wrapper* as used in machine learning [Ko95, KJ97] simply refers to this property, wherein the combinatorial optimization system "wraps around" a specific inductive learning and classification or inference ensemble such as the one shown in Figure 2. In the genetic and evolutionary computation literature, as we note below, wrappers for tuning GA hyperparameters have been in use for quite some time. [BGH89, DSG93, HL99]

the overall learning system as fitness. In learning to classify, this may simply mean validation set accuracy. However, as we have noted, many authors of GA-based wrappers have independently derived criteria that resemble *minimum description length* (*MDL*) estimators – that is, they seek to minimize model size and the sample complexity of input as well as maximize generalization accuracy. [CS96, RPG+97, GW99, HWRC00]

An additional benefit of GA-based wrappers is that it can automatically calibrate "empirically determined" constants such as the coefficients *a*, *b*, and *c* introduced in the previous section. As we noted, this can be done using individual training data sets rather than assuming that a single optimum exists for a large set of machine learning problems. This is preferable to empirically calibrating hyperparameters as if a single "best mixture" existed. Even if a very large and representative corpus of data sets were used for this purpose, there is no reason to believe that there is a single *a posteriori* optimum for hyperparameters such as weight allocation to inferential loss, model complexity, and sample complexity of data in the constructive induction wrapper.

Finally, GA wrappers can "tune themselves" – for example, the *GA-Based Inductive Learning* (*GABIL*) system of Dejong *et al* [DSG93] learns propositional rules from data and adjusts constraint hyperparameters that control how these rules can be generalized. Mitchell notes that this is a method for evolving the learning strategy itself. [Mi97] Many classifier systems also implement performance-tuning wrappers in this way. [BGH89] Finally, population size and other constants for controlling elitism, niching, sharing, and scaling can be controlled using *parameterless GAs*. [HL99]

We adapted *GAJIT* [Fa00], a Java shell for developing genetic algorithms, to implement a GA for the permutation problem of ordering variables for Bayesian network structure learning (using *K2*) and inference (using the Lauritzen-Spiegelhalter algorithm [LS88, Ne90] and *forward simulation* [SP89, CD00]). We now specify the ordering problem and, in the next section, present the permutation GA design.

3.3 Ordering and Structure Learning Problems

The ordering problem itself is a straightforward search in permutation space $\mathbf{A}$ for a value of $\boldsymbol{\alpha}$ that minimizes the inferential loss or maximizes its normalized, additive inverse, f_a. Some simple combinatorial analysis illustrates the relative complexity of the ordering and structure learning problems.

Clearly $|\mathbf{A}| = n!$ if we suppose that there are no latent or irrelevant variables. From Stirling's approximation, we can estimate that $|\mathbf{A}| \approx 2^{n \lg n}$. Meanwhile, we know that all elements of structure space are directed acyclic graphs, containing some subset of the n^2 possible directed edges. The size of structure space is thus in $O\left(2^{n^2}\right)$. Note that this includes all directed graphs and is therefore an overestimate. Taking the asymptotic ratio of these two counting functions, however, we see that in the limit, there are infinitely many possible structures *for each* ordering. *K2*, which is deterministic, finds just one such structure, so it is not guaranteed that finding a loss-minimal ordering $\boldsymbol{\alpha}$ will cause it to produce a loss-optimal network *B*, particularly for very large *n*. However, Friedman conjectures [FLNP00] that searching ordering space provides a useful change of representation [Be90] that tends to admit smoother interpolation than in structure space. In evolutionary computation terms, this would mean that ordering space is less *deceptive* [Go89] than structure space.

4 GA FOR VARIABLE ORDERING

4.1 Searching Ordering Space

The criterion f_a is computed by actually learning a BN $B = K2(\boldsymbol{\alpha}, D_{train})$ – more precisely $(E,\Theta) = K2(\boldsymbol{\alpha}, D_{train})$.

E is computed by *K2*, which makes a single pass through $\boldsymbol{\alpha}$ (a permutation of $\chi = \{X_1, \ldots, X_n\}$) and, for each X_i, considering only X_j where $\boldsymbol{\alpha}(j) > \boldsymbol{\alpha}(i)$ as a potential parent of X_i in *E*. It then adds X_j to Pa_{xi} by adding (X_j, X_i) to *E* if and only if this increases the Dirichlet score of Pa_{xi}, evaluated over D_{train}. This continues until: the set of X_j is exhausted, no single parent can be added to incrementally increase the score, or a preset (or automatically calibrated) limit on the size of Pa_{xi} in *E* is reached. For discrete BNs, Θ is computed simply by populating the specified conditional probability tables (CPTs) with frequencies computed using D_{train}.

Once *B* is fully learned, each example in $D_{val} \equiv D \setminus D_{train}$ is masked with $\mathbf{I_e}$ and its complement to obtain separate evidence and query data. The inferential loss f_a is computed as specified in the previous section. The ordering problem is a combinatorial search in $\mathbf{A}$ using f_a as a heuristic.

4.2 Permutation Genetic Algorithm Design

Application of genetic algorithms to permutation problems is discussed in [Go89] and [HH99]. The design of the *GAJIT* wrapper illustrated in Figure 1 is as follows.

We implemented an elitist permutation GA purely by extending the *GAJIT* classes using order crossover (OX) [HH99]. OX exchanges subsequences of two permutations, displacing duplicate indices with holes. It then shifts the holes to one side, possibly displacing some indices, and replaces the original subsequence in these holes. If two parents p_1 = [3 4 6 2 1 5] and p_2 = [4 1 5 3 2 6] are recombined using OX, with the crossover mask underlined, the resulting intermediate representation is i_1 = [- - 5 3 1 4] and i_2 = [- - 6 2 4 1], and the offspring are o_1 = [6 2 5 3 1 4] and o_2 = [5 3 6 2 4 1]. Mutation is implemented by swapping uniformly selected indices. *Cataclysmic mutation* [GW99] can easily be implemented using a *shuffle* operator, but we did not find this necessary.

The *master controller* for our GA runs in a Java virtual machine. It manages slaves that concurrently evaluate members of its population **α**. Each individual is encoded as a permutation of the indices {1, …, *n*}. Slave processes distributed across (4-48 processors) of a distributed-shared memory (DSM) compute cluster run identical copies of the *K2* and inference-based application depicted in Figure 2. Each evaluates the ordering it is given by learning B from D_{train}, a holdout segment of D (2/3 by default) and returns f_a for the validation set $D_{val} \equiv D \setminus D_{train}$. The master GA collects the fitness components for all members of its population and then computes f (here, $f = f_a$).

5 EXPERIMENTAL RESULTS AND EVALUATION

We experimented using the GA on data simulated from the well-known toy BN *Asia* [Ne90], which has 8 nodes. This is a very simple network to perform inference on when the structure is known *a priori*, but the permutation space – which we are searching using only f and the synthetic data – has 8! = 40320 orderings. We also performed exploratory experiments using two versions of the *ALARM* network: a subgraph of 13 nodes and the full 37-node network.

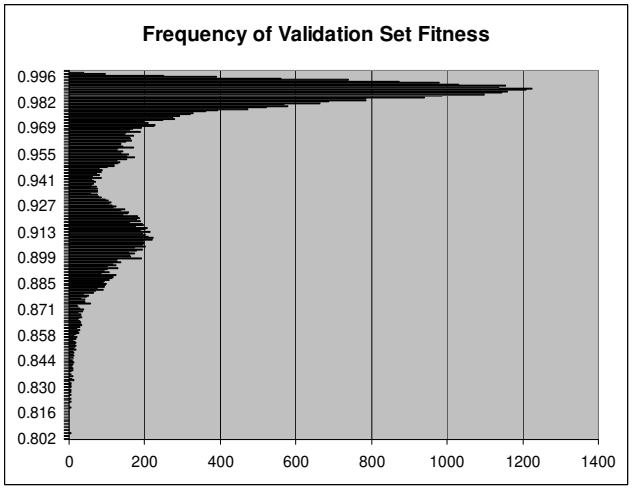

Figure 3. Histogram of estimated fitness for all 8! = 40320 permutations of *Asia* variables.

Figure 3 depicts the histogram of validation set fitness as measured **exhaustively** using Equation 5 and forward simulation [SP89, CD00]. Each of the 8! = 40320 fitness evaluations was made by running *K2* on D_{train} (as shown in Figure 2), consisting of 20000 stochastically-generated samples, and then evaluating the resulting BN using forward simulation on D_{test} (a *holdout test set* **not** used by the GA as D_{val} in Figure 2) and an evidence bit vector $\mathbf{I_e}$= (1 0 0 0 0 0 0 1). The histogram shown corresponds to data generated from the evidence instantiation *Visit-to-Asia* = *true* ∧ *Dyspnoea* = *false*. We note that this is just one evidence specification among many plausible ones that might occur in "real" applications of this consultative BN. The mean fitness is 0.958, the range is [0.0802, 0.999], and the standard deviation is 0.039.

Table 1 summarizes experimental specifications using the experimental platform described in the previous section. Figure 4 shows the average-fitness curve for *Asia* using the *GAJIT* wrapper. Using forward simulation [SP89, CD00], we generated 20000 samples for D_{train}, 5000 for D_{val} (used to evaluate fitness in the GA), and 5000 for D_{val} (used to evaluate "generalization fitness" on the ordering returned by the GA). The number of stochastic samples used to perform inference on D_{val} is given in Table 1; for all runs, 15000 samples were used to perform inference on D_{train}. The GA uses OX (order crossover), swap-mutation, and a population of size 10, and was run for 100 generations.

K2	FS Samples	Best *f* of final gen
5000	1500	0.944
10000	1500	0.960
20000	*150*	*0.935*
20000	*450*	*0.977*
20000	***1500***	***0.978***

Table 1. Results for *Asia* (5000 samples per fitness evaluation in D_{val} and D_{test})

The results for the last line were averaged over 3 trials but Figure 4 depicts the median result. Starting from a test fitness of 0.4 (inferential loss of 0.6), it improves the test fitness to 0.98. This is only about slightly above the mean fitness but it is noteworthy that the gold standard network achieves fitness of only 0.98 as well. We validated this using exact inference (the Lauritzen-Spiegelhalter algorithm [LS88, Ne90]) to compute the marginals on the data and our forward simulation function itself converges to negligibly low relative loss.

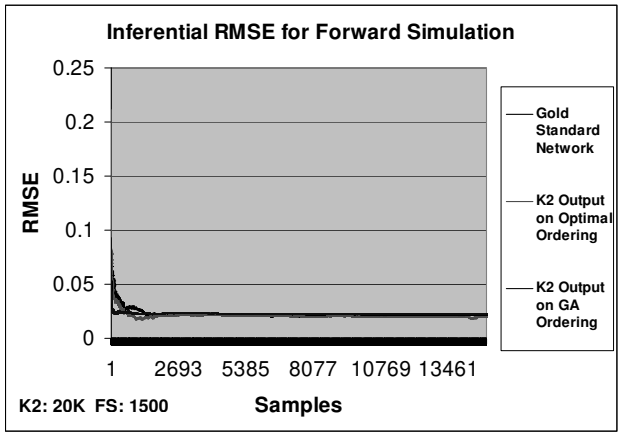

Figure 4. Fitness curve for last run in Table 1

As the fitness curve shows, the *GAJIT* wrapper reaches 0.98 rather quickly. The highest fitness achieved by the wrapper on any run is 0.99, and inspection shows that the corresponding ordering has only one inversion from the canonical one given by Neapolitan [Ne90]. This inversion is consistent with the partial ordering of the

canonical *B*, which means that *K2* can still produce the best possible structure from it.

Experiments using *ALARM-13* and *ALARM-37* indicated that although *K2* is capable of recovering a graph (*V*, *E*) close to the gold standard network (Cooper and Herskovits report only two graph errors using only 20000 training examples [CH92], as we used), its algorithm for estimating conditional probability tables results in high relative inferential loss. We hypothesize that this is due to the skewness of some conditional probability tables (CPTs) in both versions of *ALARM*. The fitness evaluation procedure depicted in Figure 2 is therefore is less effective than on *Asia*. In continuing work, we are hybridizing *K2* with other CPT learning algorithms.

6 DISCUSSION AND FUTURE WORK

We have considered several continuations of this research, grouped into four categories: validation, scalability, comparison to other structure learning methods, and improvements to the ordering GA.

First, validation is currently performed by running importance sampling for precisely 15000 samples (with an importance function update every 100 samples for AIS), and this is repeated to find the fitness of the best ordering $\hat{\boldsymbol{\alpha}}$ found by the generational GA. Experiments currently in development run *K2* with a range of D_{train} sizes to generate a learning curve, and run AIS longer with $\hat{\boldsymbol{\alpha}}$ to get a more accurate evaluation. Automated convergence analysis can be used to adapt the number of samples and the AIS update rate. Fast exact inference to find the true inferential loss baseline, a topic of a concurrent research project, can test the efficacy of AIS itself. We have focused in this paper on the general case, where the gold standard network may not be known, but when it is, one can use graph edit distance between the BN induced by $\hat{\boldsymbol{\alpha}}$ and the gold standard as a validation measure [CH92].

Second, we plan to explore the scalability of the GA wrapper by experimenting with larger networks (such as *ALARM* and *Pathfinder*) with which we have already tested AIS and *K2* as individual components. When used in a GA, which may evaluate fitness thousands to millions of times for this problem, these primitives to become bottlenecks. To make the wrapper feasible, it will be necessary to parallelize *K2* and AIS.

Third, there are several algorithms besides greedy search for structure learning, such as deterministic score-based (sparse candidate, Tabu search) methods, constraint-based methods, stochastic sampling in structure space by direct (non-greedy) global optimization and stochastic sampling in ordering space (to determine structure, without using a greedy algorithm such as K2 as an intermediary). These are often less sensitive to variable ordering but may still be affected by it. In continuing work, we plan to compare our GA wrapper to these techniques.

Fourth, the following are promising variants of the GA that are high experimental priorities: Pareto optimization of (f_a, f_b, f_c) and experimentation with other permutation mutation and crossover operators (partially matched and cycle crossover).

Acknowledgements

Support for this research was provided in part by the Army Research Lab under grant ARL-PET-IMT-KSU-07 and by the Office of Naval Research under grants N00014-00-1-0769 and N00014-01-1-0917.

References

[Be90] D. P. Benjamin, editor. *Change of Representation and Inductive Bias*. Kluwer Academic Publishers, Boston, 1990.

[BGH89] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 40:235-282, 1989.

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

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

[Co90] G. F. Cooper. The computational complexity of probabilistic infernece using bayesian belief networks. *Artificial Intelligence,* 42(2-3):393-405. Elsevier, 1990.

[CS96] K. J. Cherkauer and J. W. Shavlik. Growing Simpler Decision Trees to Facilitiate Knowledge Discovery. In *Proceedings of the Second International Conference of Knowledge Discovery and Data Mining (KDD-96)*, Portland, OR, August, 1996.

[DSG93] K. A. DeJong, W. M. Spears, and D. F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161-188, Kluwer Academic Publishers, 1993.

[EF01] G. Elidan and N. Friedman. Learning the Dimensionality of Hidden Variables. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-2001),* Morgan-Kaufmann, 2001.

[FG98] N. Friedman and M. Goldszmidt. *Learning Bayesian Networks From Data*. Tutorial, American National Conference on Artificial Intelligence (AAAI-98), Madison, WI. AAAI Press, San Mateo, CA, 1998.

[FLNP00] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, Using Bayesian networks to analyze expression data. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology (RECOMB 2000)*, ACM-SIGACT, April 2000.

[Go89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[GW99] C. Guerra-Salcedo and D. Whitley. Genetic Approach to Feature Selection for Ensemble Creation. In *Proceedings of the 1999 International Conference on*

Genetic and Evolutionary Computation (GECCO-99). Morgan-Kaufmann, San Mateo, CA, 1999.

[HGC95] D. Heckerman, D. Geiger, and D. Chickering, Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197-243, Kluwer, 1995.

[HH98] R. L. Haupt and S. E. Haupt. *Practical Genetic Algorithms*. Wiley-Interscience, New York, NY, 1998.

[HL99] G. Harik and F. Lobo. *A parameter-less genetic algorithm.* Illinois Genetic Algorithms Laboratory technical report 99009, 1999.

[HWRC00] W. H. Hsu, M. Welge, T. Redman, and D. Clutter. Genetic Wrappers for Constructive Induction in High-Performance Data Mining. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Las Vegas, NV. Morgan-Kaufmann, San Mateo, CA, 2000.

[HWRC02] W. H. Hsu, M. Welge, T. Redman, and D. Clutter. Constructive Induction Wrappers in High-Performance Commercial Data Mining and Decision Support Systems. *Knowledge Discovery and Data Mining*, Kluwer, 2002.

[KJ97] R. Kohavi and G. H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence, Special Issue on Relevance, 97*(1-2):273-324, 1997.

[Fa00] M. Faupel. GAJIT genetic algorithm package. URL: http://www.angelfire.com/ca/Amnesiac/gajit.html, 2000.

[Mi97] T. M. Mitchell. *Machine Learning.* McGraw-Hill, New York, NY, 1997.

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

[Ne90] R. E. Neapolitan. Probabilistic Reasoning in Expert Systems: Theory and Applications. Wiley-Interscience, New York, NY, 1990.

[Ne93] R. M. Neal. Probabilistic Inference Using Markov Chain Monte Carlo Methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993.

[PV91] J. Pearl and T. S. Verma, A theory of inferred causation. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*. Morgan Kaufmann, San Mateo, CA, 1991.

[RPG+97] M. Raymer, W. Punch, E. Goodman, P. Sanschagrin, and L. Kuhn, Simultaneous Feature Extraction and Selection using a Masking Genetic Algorithm, In *Proceedings of the 7th International Conference on Genetic Algorithms*, pp. 561-567, San Francisco, CA, July, 1997.

[SP89] R. D. Schacter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence 5*, p. 221-231, Elsevier Science Publishing Company, New York, NY, 1989.

[We96] R. L. Welch. Real-Time Estimation of Bayesian Networks. In *Proceedings of UAI-96*, Morgan-Kaufmann, 1996.

[Wi02] Wikipedia Online Encyclopedia, *Sharp-P*. URL: http://www.wikipedia.com/wiki/Sharp-P, 2002.