Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Sciences 163 (2004) 103–122
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Genetic wrappers for feature selection in decision tree induction and variable ordering in Bayesian network structure learning

William H. Hsu

Department of Computing and Information Sciences, Kansas State University, Manhattan, KS 66506, USA
National Center for Supercomputing Applications (NCSA), University of Illinois, Champaign, IL 61820, USA

Received 8 May 2002; accepted 17 March 2003

Abstract

In this paper, we address the automated tuning of *input specification* for supervised inductive learning and develop combinatorial optimization solutions for two such tuning problems. First, we present a framework for selection and reordering of input variables to reduce generalization error in classification and probabilistic inference. One purpose of selection is to control overfitting using validation set accuracy as a criterion for relevance. Similarly, some inductive learning algorithms, such as greedy algorithms for learning probabilistic networks, are sensitive to the evaluation order of variables. We design a generic fitness function for validation of input specification, then use it to develop two genetic algorithm *wrappers*: one for the variable selection problem for decision tree inducers and one for the variable ordering problem for Bayesian network structure learning. We evaluate the wrappers, using real-world data for the selection wrapper and synthetic data for both, and discuss their limitations and generalizability to other inducers.

Keywords: Bayesian networks; Decision trees; Feature selection; Genetic algorithms; Inductive bias; Permutation problems; Structure learning; Variable ordering; Wrappers

E-mail address: bhsu@cis.ksu.edu (W.H. Hsu).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2003.03.019

1. Introduction

This paper presents an input-driven, genetic search-based approach towards automatically tuning the representation bias of a supervised inductive learning system. Specifically, we formulate the tasks of selecting and ordering the input variables as high-level search, then develop accuracy-based evaluation measures for classification and probabilistic inference that we can apply toward heuristic search. The combinatorial growth of the search space in the number of input variables n (2^n for variable selection, $n!$ for the permutation problem of variable ordering) requires an informative heuristic. We therefore examine criteria such as accuracy, model complexity, and task-specific measures, to develop a flexible fitness function that can express a linear combination of these criteria. We also consider how coefficients for these criteria can be empirically calibrated for specific learning and inference problems. Having developed such a flexible fitness function and a validation-based evaluation method, we seek to incorporate these into an efficient, parallel, search-based wrapper.

We justify the implementation of this wrapper using a genetic algorithm (GA) as follows. The size of the search spaces indicates a need for parallel search. The deceptiveness [13] of the fitness function indicates a need for stochastic search and for a global optimization technique in the general case. Given these desiderata, the breadth of the feasible search frontiers in practice suggests that a way to recombine abstractions of good solutions may further improve efficiency. Finally, the coding of these specifications, as bit vectors for variable subsets and permutations for variable orderings, is a natural representation for GAs that makes it easy to define and compare search operators.

To enable a GA or other combinatorial optimization system to search the bias space, the space of input specifications such as variable subsets or orderings, we adapt a flexible, composite fitness measure. Inductive learning systems that search bias space or otherwise control high-level parameters using validation performance of a primitive inducer are called *wrappers* [21]. Recent research applying GA-based wrappers to feature selection for overfitting control in decision trees [5,20], instance-based learning using k-nearest neighbors [28], and multilayer perceptrons [15] has shown that many inducers can be "wrapped" using this methodology. We present a template for a generic, GA-based wrapper as shown in Fig. 1 and show how it provides a parallel stochastic search mechanism for loss-minimizing input specifications—specifically, variable subsets and orderings.

In this paper, our aim is to generalize loss beyond classification accuracy-based fitness. We shall do this first by examining a GA-based selection wrapper and its adaptation to structure learning in Bayesian networks, where loss is measured as error in probabilistic inference rather than classification error. Although the fitness function we evaluate in this work is accuracy-driven, we consider its possible extension to minimum description length (MDL) criteria.

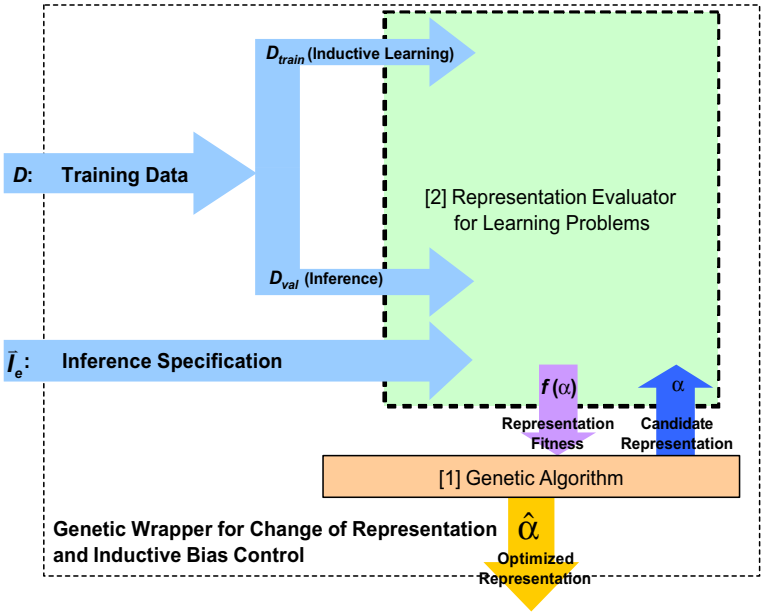

Fig. 1. Abstract system design overview.

2. Background

Consider a typical classification or probabilistic reasoning environment, as shown in Fig. 2, where inductive learning of a classifier or graphical model $[A]$ 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. An input specification α may also be given as input, indicating which variables are to be considered by the inductive learning algorithm and in what order they are to be scored. The structure learning component of this system produces a hypothesis h that classifies instances or describes the dependencies among partially observed X_i (in the case of a graphical model). The inferential performance element $[B]$ of this system takes h and a new data set D_{val} of vectors drawn from the desired inference space and applies h to produce the output. In classification, this is a single prediction $h(x)$ per instance x; in probabilistic inference, only a subvector $\mathbf{E}$ of $\mathbf{X} = (X_1, \ldots, X_n)$ is observable, and h is applied to infer the remaining unobserved values $\mathbf{X} \setminus \mathbf{E}$. We denote the indicator bit vector for membership in $\mathbf{E}$ by $\mathbf{I_e}$—in classification this consists of all but one variable, the target output $c(\mathbf{x})$. The performance criterion f is the additive inverse of the (classification, inferential, or utility) loss of $[B]$.

In this section we specify the functionality of $[A]$ and $[B]$ in a selection wrapper for decision tree induction and an ordering problem for Bayesian

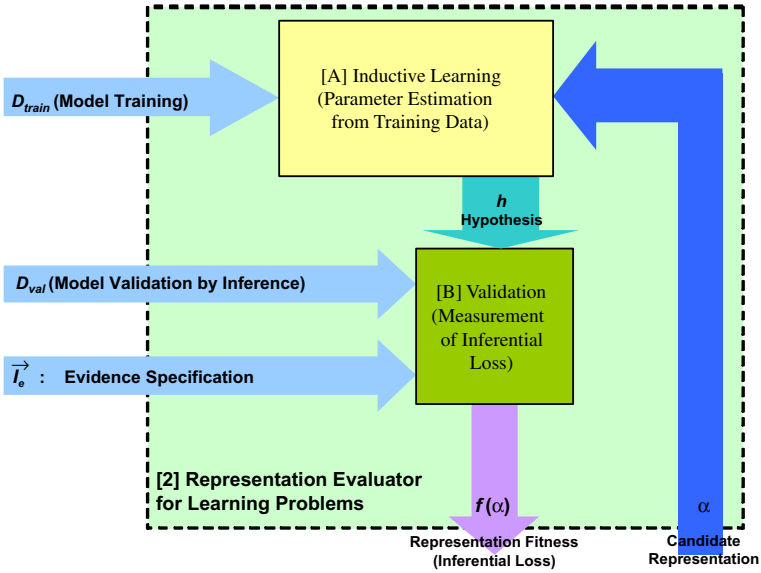

Fig. 2. Model evaluation environment, module {2} from Fig. 1.

network structure learning. We then explain the derivation of a generic fitness function f as a function of the input specification α. In the next section, we show how the environment depicted in Fig. 2 is used as the fitness evaluation module {2} of the overall GA-based system (Fig. 1). The overall output $\widehat{\alpha}$ of Fig. 1—a set of selected variables or a reordering—is evaluated using a second holdout segment, D_{test}.

2.1. Variable selection in overfitting control

The variable selection problem is alternatively known as that of attribute subset selection [19], feature subset selection, and variable elimination; it is one case of relevance determination [21]. Our bias space is simply the power set of variables. Kohavi [21] developed a wrapper based upon deterministic best-first search, using validation set accuracy, that is used in the machine learning library *MLC++* to select variables for many inducers (ID3, C4.5, CN2, Naïve Bayes, IBL, PEBLS, etc.). One purpose of variable selection is to prevent overfitting by using the validation set accuracy of an inducer to pre-prune variables that are irrelevant (not weakly relevant) [21]. As we document in our experimental section, we implemented two GA wrappers for variable selection: Grefenstette's simple GA [14] and Guerra-Salcedo and Whitley's CHC [15]. We observed that these wrappers are competitive with deterministic best-first

search-based wrappers and in addition are less likely to over-prune, but are also less stable and require many more fitness evaluations.

2.2. 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 $P(X_i \mid Pa_{x_i})$, where Pa_{x_i} denotes the parents of X_i in G. A Bayesian network B specifies a unique joint probability distribution over χ given by

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

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 [9]. 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 $\mathbf{x}$. 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_{val} 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 $[B]$. For simplicity, we assume that there are no variables that are latent or completely irrelevant (not weakly relevant [21]). The objective of structure learning is then to find the arcs E for $V = \chi$. Some structure learning algorithms, such as $K2$ [6], 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. [1] We use $K2$ for structure learning—module $[A]$ of Fig. 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

[1] The definition and properties of the Dirichlet scoring function are beyond the scope of this paper; for brevity, we refer the interested reader to [6,11].

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 [21]. 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 [11,27].

Unfortunately, just as finding the optimal structure is itself intractable [18], 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.

Previous work on using genetic algorithms for Bayesian network structure learning includes that of Larrañaga et al., who represented network structure using adjacency matrix (bit strings) with ordering constraints. In this work, Larrañaga et al. [22] noted the sensitivity of greedy score-based methods and GAs to variable ordering. Our adaptation wraps the score-based approach within a permutation GA but focuses on the ordering problem. This approach admits other fitness measures (e.g., marginal likelihood scores) besides the inferential loss estimators discussed in the following section.

2.3. 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 Eq. (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 $\mathbf{e}$. Finding the MPE directly using Eq. (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 [23,26]. 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) [7,30].

Approximate inference refers to approximation of the posterior probabilities given evidence. One stochastic approximation method called *importance sampling* [4] estimates the evidence marginal by sampling query node instantiations:

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

Chen and Druzdzel [4] discuss basic variants of importance sampling. These include *probabilistic logic sampling*, 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, which samples using the joint probability of query nodes as the importance function

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

Welch [29] demonstrates 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, $\mathbf{P}(\mathbf{e}) = 6.5 * 10^{-4}$.

One way of scaling up to large networks in a realistic probabilistic reasoning application is to dynamically adapt the importance function. Cheng and Druzdzel [4] 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 [25]. 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 [4] 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 Fig. 2.

2.4. Deriving fitness

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

Inferential loss. Quality of the network produced by $K2$ as detected through inferential loss evaluated over a holdout validation data set $D_{val} \equiv D \setminus D_{\text{train}}$ (see Fig. 1)—requires modules $[A]$ and $[B]$ in Fig. 2.

Model loss. "Size" of the network under a specified representation—requires module $[A]$ only and is independent of $[B]$.

Ordering loss. Inference and model-independent measure of data quality given only D and $\boldsymbol{\alpha}$—independent of both modules $[A]$ and $[B]$.

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

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

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

$$f_a^{DT}(\boldsymbol{\alpha}, D) = 1 - \frac{m_{correct}}{m_{val}} \tag{5b}$$

where

$$m_{correct} \equiv h.classification\text{-}accuracy(D_{val}.select(\boldsymbol{\alpha}))$$
$$h \equiv h_0.train(D_{train}.select(\boldsymbol{\alpha}))$$
$$m_{val} \equiv |D_{val}|$$

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

where

$$a_i \equiv arity(X_i, B = (\chi, E, \Theta))$$
$$(E, \Theta) = K2(\boldsymbol{\alpha}, D_{train})$$

$$f_b^{DT}(\boldsymbol{\alpha}, D) = 1 - \frac{h.size()}{s_{\max}} \quad \text{e.g., } s_{\max} = m \tag{6b}$$

$$f_c^{DT}(\boldsymbol{\alpha}) = 1 - \frac{|\boldsymbol{\alpha}|}{n} \tag{7}$$

$$a + b + c = 1 \tag{8}$$

In related work on genetic wrappers for variable selection in supervised inductive learning, we adapted Eq. (4) [19,20] from similar fitness functions

developed by Cherkauer and Shavlik [5] for decision tree pre-pruning, Raymer et al. [28] for similarity-based learning (k-nearest neighbor regression), and Guerra-Salcedo and Whitley [15] for connectionist learning. This breadth of applicability demonstrates the generality of simple genetic algorithms as wrappers for performance tuning in supervised inductive learning.

Recently, we automatically validated the coefficients a, b, and c for several individual data sets on a supervised learning task [20]. Results were positive in that this approach found application-specific values for these GA parameters, [2] and the GA achieved better generalization accuracy than deterministic best-first search-based feature selection wrappers [21] for a real-world test bed (risk category classification and loss prediction in commercial data mining). 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 between the sampled, approximate probabilities $P'(x_{ij})$ and exact probabilities $P(x_{ij})$, over states x_{ij} of variables X_i [4]. Note that f_a is the only term that depends on which variables are *observable*, i.e., members of $\mathbf{E}$. 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 [24]. Ultimately, a BN B is only as good as the inferences it can produce on real-world data given realistic evidence $\mathbf{e}$, 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. Searching for variable subsets and orderings in learning

Fig. 1 indicates the role of a combinatorial optimization system for controlling $\boldsymbol{\alpha}$, in context: a probabilistic reasoning system based on greedy

[2] We distinguish between parameters of the *genetic algorithm*, such as these fitness function coefficients, and those that are learned by the *wrapped inducers*. Neal [25] refers to parameters governed by an outer stochastic proposal distribution as *hyperparameters*, but for simplicity we use the term (GA) parameter.

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: controlling input to enhance supervised learning

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* [1,24] 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. [3] It must have the ability to tune the learning system by automatically adjusting 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 GA parameters such as $\boldsymbol{\alpha}$. Just as a parameter of an inducer denotes a trainable component of a pattern detector or classification function, a parameter of the GA denotes a controllable component of the organization, representation, or search algorithm for a learning problem. Inductive learning systems, or *inducers*, are built with such parameters and the ability to tune

[3] The term *wrapper* as used in machine learning [21] 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 Fig. 2. In the genetic and evolutionary computation literature, as we note below, wrappers for tuning GA parameters have been in use for quite some time [2,8,16].

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 parameters 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 α, the order in which nodes are added to the BN. Both of these are methods for *inductive bias control* where the input representation is changed from the default [1]—here, the full subset χ or an arbitrary ordering α_0.

With a GA-based wrapper, we seek to evolve parameter values using the performance criterion of 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 [5,15,20,28].

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 parameters 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 GA parameters such as weight allocation to inferential loss, model complexity, and sample complexity of data in the variable selection wrapper.

Finally, GA wrappers can "tune themselves"—for example, the *GA-Based Inductive Learning* (GABIL) system of DeJong et al. [8] learns propositional rules from data and adjusts constraint parameters that control how these rules can be generalized. Mitchell [24] notes that this is a method for evolving the learning strategy itself. Many classifier systems also implement performance-tuning wrappers in this way [2]. Finally, population size and other constants for controlling elitism, niching, sharing, and scaling can be controlled using *parameterless GAs* [16].

We adapted *GAJIT* [10], 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 junction tree algorithm [23,26] and AIS [4]). 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 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 $|A| = n!$ 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 $O(2^{n^2})$. 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 et al. [12] hypothesize that searching ordering space provides a useful change of representation [1] that tends to admit smoother interpolation than in structure space. In evolutionary computation terms, this would mean that ordering space is less *deceptive* [13] than structure space.

4. Permutation GA for ordering

4.1. Selection GA: searching the power set

The coefficients a, b, and c have been hand-calibrated in several previous GA selection wrappers [5,28]. There is, however, no evidence to indicate that these coefficients should be constants for any particular inducer over all data sets, nor even that keeping them constant throughout the execution of a GA results in an effective fitness criterion.

We reimplemented Grefenstette's simple GA *Genesis* and Guerra-Salcedo and Whitley's CHC [15] in Java and used them to drive wrappers as shown in Fig. 1, where α is coded as a bit string denoting inclusion of a variable in the input schema of an inducer (that is, the data set D is projected to include only columns of data indicated by α). Selection is fitness-proportionate, crossover is uniform in *CHC* and single-point *Jenisis* (the Java port of Grefenstette's simple GA *Genesis* [14]), and mutation is single-bit inversion. No niching, sharing, or elitism is applied. For several individual data sets on a supervised learning task,

we experimented with a range of preset values for a, b, and c, finding data sets where the published defaults of $a = 0.75$, $b = 0.125$, and $c = 0.125$ outperformed all other weights and some where they did not [20]. As summarized below, this simple GA achieved better generalization accuracy than search-based feature selection wrappers [21] for a real-world test bed.

We pause to discuss the choice of crossover operator. Caruana et al. [3] discuss the ramifications of positional bias in single-point crossover, which tends to preserve locality of bits. In variable selection, the original variables are best regarded as an unordered set of variables. This bias is mitigated by uniform and shuffle crossover, hence the use of uniform crossover in CHC [15] and majority of our experiments.

4.2. Permutation GA: searching ordering space

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

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_{x_i} by adding (X_j, X_i) to E if and only if this increases the Dirichlet score of Pa_{x_i}, 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_{x_i} 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 A using f_a as a heuristic.

Application of genetic algorithms to permutation problems is discussed in [13] and [17]. The design of the *GAJIT* wrapper illustrated in Fig. 1 is as follows.

We implemented an elitist permutation GA purely by extending the *GAJIT* classes using order crossover (OX) [17]. 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 $p_1 = [3\,4\,\underline{6\,2}\,1\,5]$ and $p_2 = [4\,1\,\underline{5\,3}\,2\,6]$ are recombined using OX, with the crossover mask underlined, the resulting intermediate representation is $i_1 = [-\,-\,\underline{5\,3}\,1\,4]$ and $i_2 = [-\,-\,\underline{6\,2}\,4\,1]$, and the offspring are $o_1 = [6\,2\,\underline{5\,3}\,1\,4]$ and $o_2 = [5\,3\,\underline{6\,2}\,4\,1]$. Mutation is implemented by swapping uniformly selected indices. Cataclysmic mutation 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 $\boldsymbol{\alpha}$. Each individual

is encoded as a permutation of the indices $\{1, \ldots, n\}$. 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 Fig. 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 developed a real-world data set as part of a commercial data mining test bed [20]. This data set consists of 350 aggregate training examples (divided into 5 folds of 70 examples each) representing 350 000 customer records and containing 100 input attributes. Some feature construction steps reported in [20] were applied to reduce an original 471 attributes to these 100.

Results for this data set using ID3 are shown in Table 1, for 5-fold cross validated runs. Although the average prediction accuracy for the simple GA is higher than those for FSS and the unwrapped inducer, we found that the actual subsets found by FSS and simple GA (SGA) across folds are not stable (that is, they overlap in only 5–6 variables between any two subsets). The effectiveness of the GA wrapper approach for inducing decision trees, is therefore not conclusive, though it is still possible to adapt bagging or rule post-pruning to obtain a coherent classifier as output.

We have ported the *MLC++* base classes and the ID3 and Naïve Bayes inducer into a Java edition called MLJ and incorporated Quinlan's C4.5 into this code base. Continuing with experimentation using GA selection wrappers,

Table 1
5-fold cross-validation error for simple GA versus FSS, on commercial data set

		Cross-validation segment						
		0	1	2	3	4	Mean	S.D.
Training set accuracy (%)	ID3	100.0	100.0	100.0	100.0	100.0	100.0	0.00
	FSS-ID3	55.00	54.29	67.86	50.36	60.71	57.64	6.08
	Jenesis	65.71	67.14	71.43	71.43	55.71	66.29	5.76
Test set accuracy (%)	ID3	41.43	42.86*	28.57	41.43	44.29	39.71	5.67
	FSS-ID3	48.57*	35.71	24.29	47.14	54.29	44.00	7.74
	Jenesis	41.43	42.86*	31.43*	52.86*	55.71*	44.86	8.69
Attributes selected	ID3	35	35	37	40	35	36.40	1.96
	FSS-ID3	7	8	7	13	18	10.60	4.32
	Jenesis	20	19	22	20	23	20.80	1.47

we ported the CHC wrapper of Guerra-Salcedo and Whitley [15] into Java and adapted it to perform selection. CHC is a more sophisticated GA, featuring options such as cataclysmic mutation (where diversity is maintained through population-wide perturbation).

The CHC wrapper also achieves results comparable to Kohavi's FSS when wrapped around both ID3 and C4.5, and outperforms it on some data sets from the Irvine Machine Learning Database repository, as shown in Table 2. We observed that in two of the data sets—*Anneal* and *Credit*—the CHC wrapper consistently outperformed the *MLC++* FSS wrapper using both ID3 and C4.5. Five of the cases were not entirely conclusive: *Breast*, *Hypothyroid*, *Mushroom*, *Pima* and *Solar*. In two cases (*Hypothyroid* and *Solar*), CHC showed an insignificant advantage; and in one (*Mushroom*), the outcome was tied. For *Mushroom*, the FSS wrapper is (notoriously) known to hurt performance slightly due to overselection; CHC simply achieves 0% test set error and is competitive with the unwrapped inducer with slightly higher (nonzero) variance. It appears that all variable selection tends to hurt generalization accuracy in *Anneal*. Finally, performance by CHC is markedly worse than that of both unwrapped and FSS-wrapped ID3 on the toy problems *Monk1* and *Monk3*. Though this record is mixed, is important to note that except for the Monk's problems, the CHC wrapper tends to boost or maintain the performance of at least one of the inducers.

Generally, the CHC wrapper is competitive with best-first search-based FSS using population size 100 after 10–100 generations. We did note instability (i.e., failure to converge) in nearly all cases where CHC underperforms an unwrapped or FSS-wrapped inducer.

An example curve for validation set accuracy is shown in Fig. 3, showing the fitness (dominated by the training error term) for CHC-ID3. This plot depicts a run with population size 100 and 100 generations. We note that the cases where

Table 2
Comparison of unwrapped inducer (ID3 and C4.5), feature subset selection (FSS) wrapper from *MLC++*, and CHC genetic wrapper (population size 100, 100 generations)

Data set	ID3	C4.5	FSS-ID3	FSS-C4.5	CHC-ID3	CHC-C4.5
Anneal	0.00 ± 1.24	11.33 ± 1.83	0.67 ± 0.47	10.33 ± 1.76	0.00 ± 0.45[a]	1.00 ± 1.45
Breast	5.58 ± 1.51	4.29 ± 1.33[a]	5.58 ± 1.51	4.29 ± 1.33[a]	5.15 ± 0.78	4.72 ± 1.15
Credit (CRX)	27.50 ± 3.17	17.50 ± 2.69	17.50 ± 2.69	18.00 ± 2.72	17.00 ± 1.73	15.50 ± 1.68[a]
Hypothyroid	0.95 ± 0.30	0.76 ± 0.27	1.23 ± 1.34	0.76 ± 0.06	0.94 ± 0.05	0.75 ± 0.06[a]
Monk1	18.98 ± 1.89	24.31 ± 2.07	2.78 ± 0.79[a]	11.11 ± 1.51	27.77 ± 0.90	25.00 ± 7.51
Monk3	8.33 ± 1.33	2.78 ± 0.79[a]	2.78 ± 0.79[a]	2.78 ± 0.79[a]	25.00 ± 3.31	2.78 ± 11.38
Mushroom	0.00 ± 0.00[a]	0.00 ± 0.00[a]	0.19 ± 0.00	0.19 ± 0.83	0.00 ± 0.03	0.00 ± 0.03
Pima	29.30 ± 0.85	23.83 ± 2.67	29.69 ± 2.86	20.70 ± 2.54[a]	28.90 ± 0.98	24.60 ± 3.89
Solar	27.78 ± 4.33	26.85 ± 4.28	27.78 ± 4.33	31.48 ± 4.49	26.85 ± 2.65[a]	26.85 ± 2.95

[a] Denotes best result.

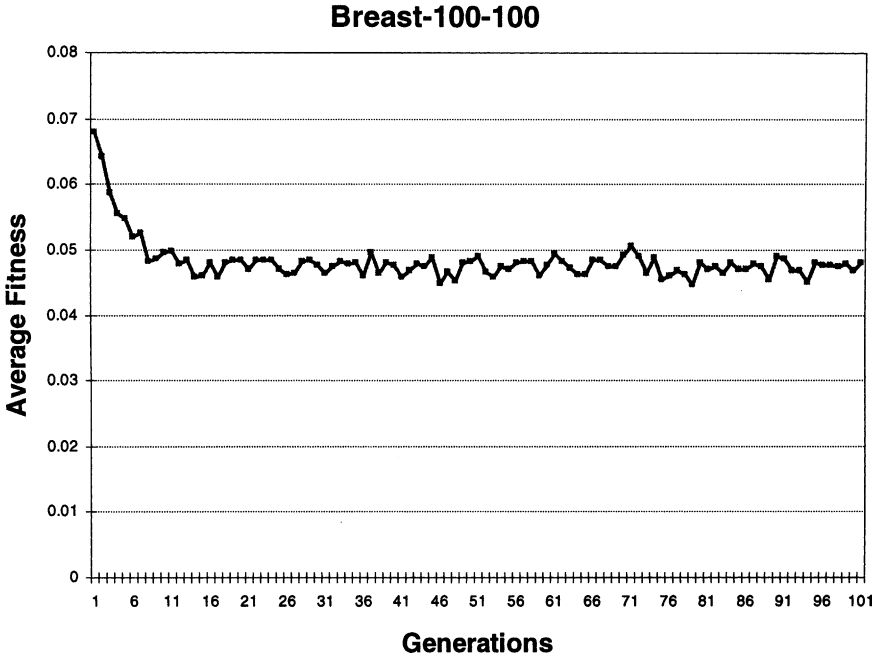

Fig. 3. Training error curve plotted over 100 generations of CHC-ID3 run, population size 100, for the Wisconsin breast cancer set. *Lower is better*.

CHC-wrapped decision tree inducers outperform or are competitive with the FSS-wrapped ones include larger real-world data sets such as *Breast*, *Credit* (CRX), and *Hypothyroid* for knowledge discovery in databases (KDD), by contrast with the synthetic data sets. In the case of *Credit*, we found that the fitness improved steadily and was already outperforming that of FSS, but cataclysmic tended to wipe out this progress.

For the ordering GA, we conducted a series of experiments using data simulated from the well-known toy BN *Asia* [26], which has eight 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.

Table 3 summarizes experimental results (validation set accuracy) achieved using the experimental platform described in the previous section. Fig. 4 shows the average-fitness curve for *Asia* using the *GAJIT* wrapper. We generated 5000 samples using forward sampling for D_{train} and 1000 for D_{val}. The GA with OX and swap-mutation improves the ordering to within 0.01 of the optimum RMSE (about 0.95, calculated using exact inference to compute the marginals on the data), which is the average of best results achieved by AIS, over 10 trials.

Table 3
Results for *Asia* data set (5000 training, 1000 validation samples per f, AIS update every 100 samples)

Population size	Generations	Average f over run	Best f, final gen
10	10	0.8864	0.9313
10	100	0.9116	0.9316

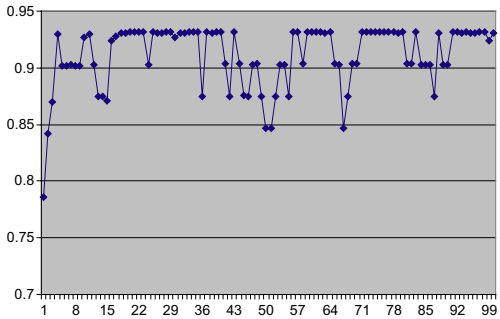

Fig. 4. GA fitness curve over 100 generations for *Asia* data set (5000 training, 1000 validation samples per f, AIS update every 100 samples).

As the fitness curve shows, the *GAJIT* wrapper reaches 0.932 rather quickly. The highest fitness achieved by the wrapper on any run is 0.964, and inspection shows that the corresponding ordering has only one inversion from the canonical one given by Neapolitan [26]. 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.

6. Discussion and future work

Slightly positive results for the SGA and positive results for CHC indicate that an accuracy-based fitness function can be used to drive a GA wrapper for variable selection. We are continuing to collect results to test scalability to data sets with more variables, a known issue in GA-based optimization, and to tune the fitness coefficients. The key remaining goal, however, is to better understand how possibly differing input specifications produced by a variable selection module on different cross-validation folds or samples can be combined.

Positive preliminary results for the Bayesian network ordering GA indicate that for small networks, the ordering can indeed be optimized. Scalability is a very significant concern here but is currently limited by severe computational

bottlenecks in the module for validation by inference. We have considered several continuations of this research: validation, scalability, and comparison to other structure learning methods and permutation GAs.

Validation is currently performed by running AIS for precisely 1000 samples with an importance function update every 100 samples, and this is repeated to find the fitness of the best ordering $\hat{\boldsymbol{\alpha}}$ found by the generational GA. Future experiments shall 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. 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 [6].

We plan to explore the scalability of the GA wrapper by experimenting with larger networks (such as *ALARM*, *Pathfinder*, and *CPCS*) 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 become bottlenecks. To make the wrapper feasible, it will be necessary to parallelize $K2$ and AIS.

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. Finally, 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 US National Science Foundation (NSF) under cooperative agreement 9874732 (NSF EPSCoR First Award) and by the US Office of Naval Research (ONR) under grant N00014-01-1-0917. The author thanks Haipeng Guo, Roby Joehanes, Benjamin B. Perry, and Julie A. Thornton, for development assistance with the Bayesian network structure learning, inference, and variable ordering GA components; Cecil P. Schmidt, James A. Louis, Matt A. Durst, and James W. Plummer, for reimplementing the *ID3*, *C4.5*, *Naïve Bayes* inducers and *CHC* wrapper in Java; and Thomas Redman and David Clutter for implementing *NCSA D2K* and assisting with development of the SGA wrapper. Finally, the author

thanks the anonymous reviewers for helpful comments and for suggestions regarding GA design and experiments.

References

[1] D.P. Benjamin (Ed.), Change of Representation and Inductive Bias, Kluwer Academic Publishers, Norwell, MA, 1990.
[2] L.B. Booker, D.E. Goldberg, J.H. Holland, Classifier systems and genetic algorithms, Artificial Intelligence 40 (1989) 235–282.
[3] R.A. Caruana, L.A. Eshelmann, J.D. Schaffer, Representation and hidden bias II: eliminating defining length bias in genetic seach via shuffle crossover, in: N.S. Sridharan (Ed.), Eleventh International Joint Conference on Artificial Intelligence, vol. 1, Morgan Kaufmann Publishers, San Mateo, CA, 1989, pp. 750–755.
[4] J. Cheng, M.J. Druzdzel, AIS-BN: an adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks, Journal of Artificial Intelligence Research (JAIR) 13 (2000) 155–188.
[5] K.J. Cherkauer, J.W. Shavlik, Growing simpler decision trees to facilitate knowledge discovery, in: Proceedings of the Second International Conference of Knowledge Discovery and Data Mining, (KDD-96), Portland, OR, AAAI Press, Menlo Park, CA, 1996, pp. 315–318.
[6] G.F. Cooper, E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, Machine Learning 9 (4) (1992) 309–347.
[7] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artificial Intelligence 42 (2–3) (1990) 393–405.
[8] K.A. DeJong, W.M. Spears, D.F. Gordon, Using genetic algorithms for concept learning, Machine Learning 13 (1993) 161–188.
[9] G. Elidan, N. Friedman, Learning the dimensionality of hidden variables, in: A. Darwiche, N. Friedman (Eds.), Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence, (UAI-2001), Seattle, WA, Morgan Kaufmann, San Francisco, CA, 2001, pp. 144–151.
[10] M. Faupel, GAJIT genetic algorithm package, Available from <http://www.angelfire.com/ca/Amnesiac/gajit.html> 2000.
[11] N. Friedman, M. Goldszmidt, Learning Bayesian networks from data Tutorial, in: American National Conference on Artificial Intelligence, (AAAI-98), Madison, WI, AAAI Press, San Mateo, CA, 1998.
[12] N. Friedman, M. Linial, I. Nachman, D. Peér, Using Bayesian networks to analyze expression data, in: R. Shamir, S. Miyano, S. Istrail, P. Pevzner, M. Waterman (Eds.), Proceedings of the Fourth Annual International Conference on Computational Molecular Biology, (RECOMB 2000), Tokyo, Japan, ACM Press, New York, NY, 2000, pp. 127–135.
[13] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[14] J.G. Grefenstette, Genesis 5.0, in: A.C. Schulz (Ed.), NRL GA Archives Source Code Collection, Available from <ftp://ftp.aic.nrl.navy.mil/pub/galist/src/genesis.tar.Z>, 1994.
[15] C. Guerra-Salcedo, L.D. Whitley, Genetic approach to feature selection for ensemble creation, in: W. Banzhaf, J. Daidam, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith (Eds.), Proceedings of the 1999 International Conference on Genetic and Evolutionary Computation, (GECCO-99), Orlando, FL, Morgan Kaufmann, San Mateo, CA, 1999, pp. 236–243.

[16] G. Harik, F. Lobo, A parameter-less genetic algorithm, Technical Report 99009, Illinois Genetic Algorithms Laboratory (IlliGAL), 1999.
[17] R.L. Haupt, S.E. Haupt, Practical Genetic Algorithms, Wiley-Interscience, New York, NY, 1998.
[18] D. Heckerman, D. Geiger, D. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, Machine Learning 20 (3) (1995) 197–243.
[19] W.H. Hsu, M. Welge, J. Wu, T. Yang, Genetic algorithms for selection and partitioning of attributes in large-scale data mining problems, in: Proceedings of the Joint AAAI-GECCO Workshop on Data Mining with Evolutionary Algorithms, Orlando, FL, July 1999.
[20] W.H. Hsu, Control of inductive bias in supervised learning using evolutionary computation: a wrapper-based approach, in: J. Wang (Ed.), Data Mining: Opportunities and Challenges, Idea Group Publishing, Hershey, PA, 2003, pp. 27–54.
[21] R. Kohavi, G.H. John, Wrappers for feature subset selection, Artificial Intelligence (Special Issue on Relevance) 97 (1–2) (1997) 273–324.
[22] P. Larrañaga, M. Poza, Y. Yurramendi, R.H. Murga, C.M.H. Kuijpers, Structure learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters, IEEE Journal on Pattern Analysis and Machine Intelligence 18 (9) (1996) 912–926.
[23] S.L. Lauritzen, 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) 157–224.
[24] T.M. Mitchell, Machine Learning, McGraw-Hill, New York, NY, 1997.
[25] 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.
[26] R.E. Neapolitan, Probabilistic Reasoning in Expert Systems: Theory and Applications, Wiley-Interscience, New York, NY, 1990.
[27] J. Pearl, T.S. Verma, A theory of inferred causation, in: J.F. Alien, R. Fikes, E. Sandewall (Eds.), Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann, San Mateo, CA, 1991, pp. 441–452.
[28] M.L. Raymer, W.F. Punch, E.D. Goodman, P.C. Sanschagrin, L.A. Kuhn, Simultaneous feature extraction and selection using a masking genetic algorithm, in: T. Back (Ed.), Proceedings of the Seventh International Conference on Genetic Algorithms, (ICGA-97), San Francisco, CA, Morgan Kaufmann Publishers, San Mateo, CA, 1997, pp. 561–567.
[29] R.L. Welch, Real-time estimation of Bayesian networks, in: Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence, (UAI-96), Portland, OR, Morgan Kaufmann Publishers, San Mateo, CA, 1996, pp. 533–544.
[30] Wikipedia Online Encyclopedia, Sharp-P, Available from <http://www.wikipedia.com/wiki/Sharp-P> 2003.