Empirical Behavior of Bayesian Network Structure Learning Algorithms

Brandon Malone(✉)

Max Planck Institute for the Biology of Ageing, Cologne, Germany
brandon.malone@age.mpg.de

Abstract. Bayesian network structure learning (BNSL) is the problem of finding a BN structure which best explains a dataset. Score-based learning assigns a score to each network structure. The goal is to find the structure which optimizes the score. We review two recent studies of empirical behavior of BNSL algorithms.

The score typically reflects fit to a training dataset; however, models which fit training data well may generalize poorly. Thus, it is not clear that finding an optimal network is worthwhile. We review a comparison of exact and approximate search techniques. Sometimes, approximate algorithms suffice; for complex datasets, the optimal algorithms produce better networks.

BNSL is known to be NP-hard, so exact solvers prune the search space using heuristics. We next review problem-dependent characteristics which affect their efficacy. Empirical results show that machine learning techniques based on these characteristics can often be used to accurately predict the algorithms' running times.

Keywords: Bayesian networks · Structure learning · Algorithm selection · Empirical hardness

1 Introduction

Bayesian networks (BNs) (Pearl 1988) are a widely-used formalism for representing uncertain relationships among variables in a domain of interest. In some cases, domain experts can specify these relationships as a BN structure; however, when they are unknown, we must learn the structure from data.

In the commonly-used *score-based* framework (Heckerman et al. 1995), a score is assigned to each structure. The score is typically a penalized log-likelihood which trades off the fit of a BN to the data with the complexity of the structure. The BN structure learning problem (BNSL) is then cast as an optimization problem in which the goal is to find a BN structure with an optimal score.

BNSL is known to be NP-hard (Chickering 1996), so early optimization algorithms (such as Cooper and Herskovits (1992), Heckerman et al. (1995),

B. Malone—This paper is based on Malone et al. (2014, 2015), with co-authors Matti Järvisalo, Petri Myllymäki, Kusta Kangas and Mikko Koiviso from HIIT and the Department of Computer Science at the University of Helsinki.

J. Suzuki and M. Ueno (Eds.): AMBN 2015, LNAI 9505, pp. 105–121, 2015.
DOI: 10.1007/978-3-319-28379-1_8

Friedman et al. (1999), Chickering (2002), Moore and Wong (2003), Teyssier and Koller (2005), Tsamardinos et al. (2006)) used local search techniques. However, these algorithms suffer from the same problem faced by all local search techniques: the quality of the found solution relative to an optimal one is unknown. Consequently, a variety of algorithms have been proposed which solve the problem exactly (Ott et al. 2004; Koivisto and Sood 2004; Silander and Myllymäki 2006; Parviainen and Koivisto 2009; de Campos and Ji 2011; Yuan and Malone 2013; Bartlett and Cussens 2015; van Beek and Hoffmann 2015).

Since BNSL is NP-hard, the exact algorithms have exponential worst-case behavior. Nevertheless, many of the algorithms employ sophisticated heuristics, such as branch-and-bound techniques, to provably rule out many possible structures. In practice, these algorithms can learn provably optimal networks for modestly-sized datasets; in general, optimal networks on the order of 50 variables can be learned with reasonable resources (Malone et al. 2014).

The score of a BN structure is ideally a reflection of how well it models a training dataset. The general assumption has been that networks which model the training data well also accurately reflect new data. However, it is well-known that a model can describe a training set very well, yet generalize poorly to new data (Mitchell 1997). Thus, there is no guarantee that a network which optimizes a score for a training set will generalize well to new data.

Until a recent study (Malone et al. 2015), there was no clear empirical evidence on whether the increased computational efforts required by exact approaches to BNSL are justifiable in terms of generalization to unseen testing data. As the first half of this paper, we review that work, which shows that for some datasets, simple strategies such as greedy hill climbing can provide good generalization. However, the simple strategies fail to generalize well on other datasets. Predictive likelihood results show that the optimal algorithms consistently generalize well.

Because of their guarantees, all of the exact algorithms find optimal, equivalent networks. So, in terms of generalization, these algorithms are equivalent. As previously mentioned, though, the algorithms use sophisticated, and very different, heuristics to find the optimal network and prove its optimality. In terms of resource requirements, then, specific implementations of these algorithms, *solvers*, are very different (van Beek and Hoffmann 2015).

For the second half of this work, we review a study (Malone et al. 2014) which shows that machine learning techniques can learn a simple, yet nontrivial, model that accurately predicts the fastest solver for a given instance. Additional features are shown to capture the hardness of an instance more accurately. Models with the additional features significantly improve prediction accuracy.

The rest of this paper is structured as follows. In Sect. 2, we formally introduce Bayesian networks and BNSL. Section 3 provides an overview of the specific solvers used in this work, while Sect. 4 outlines the datasets used. Generalization of learned networks is reviewed in Sect. 5, and Sect. 6 reviews results on exact solver behavior. Finally, Sect. 7 concludes the paper.

2 Background

A Bayesian network (Pearl 1988) is a compact representation of a joint probability distribution over the random variables $\mathbf{V} = \{X_1, \ldots, X_n\}$. It consists of a directed acycic graph (DAG) in which each vertex corresponds to one of the random variables; a directed edge indicate direct dependence between two variables. Additionally, each variable X_i has an associated probability distribution, conditioned on its parents in the DAG, PA_i. The joint probability distribution given by the network is

$$P(\mathbf{V}) = \prod_{i=1}^{n} P(X_i|PA_i). \tag{1}$$

Given a dataset $\mathcal{D} = \{D_1, \ldots D_N\}$, where each D_i is a complete instantiation of $\mathbf{V}$, the goal of structure learning is to find a Bayesian network $\mathcal{N}$ which best fits $\mathcal{D}$. The fit of $\mathcal{N}$ to $\mathcal{D}$ is quantified by a scoring function s. Many scoring functions have been proposed in the literature, including Bayesian scores (Cooper and Herskovits 1992; Heckerman et al. 1995), MDL-based scores (Suzuki 1999; Silander et al. 2008), and independence-based scores (de Campos and Huete 2000), among others. The scoring functions can typically be interpretted as penalized log-likelihood functions. All commonly used scoring functions are *decomposable* (Heckerman et al. 1995); that is, they decompose into a sum of *local scores* for each variable, its parents, and the data,

$$s(\mathcal{N}; \mathcal{D}) = \sum_{i=1}^{n} s_i(PA_i; \mathcal{D}), \tag{2}$$

where $s_i(PA_i)$ gives the score of X_i using PA_i as its parents and is non-negative. We omit $\mathcal{D}$ when it is clear from context.

A variety of pruning rules (Suzuki 1999; Tian 2000; Teyssier and Koller 2005; de Campos and Ji 2011) can be used to demonstrate that some parent sets are never optimal for some variables. Additionally, in practice, large parent sets are often pruned *a priori*. We refer to parent sets remaining after all pruning as *candidate parent sets* and denote all candidate parent sets of X_i as $\mathcal{P}_i$.

The *Bayesian network structure learning* problem (BNSL) is defined as follows.

The BNSL Problem
Input: A set $\mathbf{V} = \{X_1, \ldots, X_n\}$ of variables and a local score $s_i(PA_i)$ for each $PA_i \in \mathcal{P}_i$ for each X_i.
Task: Find a DAG N^* such that

$$N^* \in \arg\min_{N} \sum_{i=1}^{n} s_i(PA_i),$$

where PA_i is the parent set of X_i in N and $PA_i \in \mathcal{P}_i$.

3 Solvers

This section describes all *solvers* (algorithm implementations) used in this work. **Hill climbing with a tabu list and random restarts** (TABU, http://www.bnlearn.com). Hill climbing is a widely-used local search technique in discrete optimization (Russell and Norvig 2003) that typically finds local optima for an objective function f by maintaining a *current* solution and applying *search operators*. At each step, all search operators are tentatively applied to the current solution to find its *neighborhood*. The member of the neighborhood which results in the biggest improvement to f is selected as the new current solution. This process is repeated until a local optimum is found. *Random restarting* is a strategy to escape from a local optimum by randomly changing a locally optimal solution and restarting the search from the new random solution. The *tabu list* strategy (Glover 1990) augments random restarts by keeping track of recently visited solutions; solutions in the tabu list are ignored when considering new neighborhoods. Even with random restarts and a tabu list, the algorithm provides no guarantees on the proximity of local optima to globally optimal solutions.

In the context of BNs, each solution corresponds to a network; the search operators considered here are edge addition, deletion and reversal (as long as the resulting structure is a DAG). The objective function f is exactly the scoring function s.

Max-min hill climbing (MMHC, http://www.bnlearn.com)**.** Max-min hill climbing (Tsamardinos et al. 2006) is a two-phase hybrid learning algorithm. During the first phase, it uses a set of statistical independence tests to identify arcs that are forbidden from appearing in the learned network. The second phase uses TABU to find local optima within this restricted space. Here we use a mutual information statistical test during the first phase. The first phase of MMHC is similar to constraint-based methods such as PC (Spirtes et al. 2000). Empirically, MMHC has been shown to outperform several other state-of-the-art algorithms, including PC, sparse candidate, three phase dependency analysis, optimal reinsertion and greedy equivalence search (Tsamardinos et al. 2006). While MMHC does guarantee to recover BN structures when the data are faithful to a DAG in the large sample limit (Tsamardinos et al. 2006), it does not offer any non-trivial guarantees about the generalization quality of the learned network for unfaithful, finite datasets.

Chow-Liu (CL)**.** The Chow-Liu algorithm (Chow and Liu 1968) is an exact, polynomial-time algorithm for finding an optimal tree-structured BN. The algorithm calculates the mutual information between all pairs of variables to form a weighted graph. The maximum spanning tree through the graph corresponds to the optimal tree-structured BN.

A* (A*, http://www.urlearning.org)**.** State space search using A* (Yuan and Malone 2013) is a provably optimal algorithm which is guaranteed to optimize s. It is based on casting BNSL as a shortest-path finding problem; A* is then

used to solve the shortest path problem, which gives the optimal network for the given local scores. For the exact solver comparisons, we refer to a variant of A* which uses multiple pattern databases as A*EC.

Integer linear programming (ILP, http://www.cs.york.ac.uk/aig/sw/gobnilp/)**.** Another approach to solving BNSL optimally is based on integer linear programming (ILP) (Bartlett and Cussens 2015). In ILP, BNs are defined as vertices on a particular polytope, and a cutting plane approach is used to find the vertex corresponding to the optimal BN.

Branch and Bound (BnB, http://www.ecse.rpi.edu/~cvrl/structlearning.html)**.** The branch-and-bound search algorithm (de Campos and Ji 2011) searches for optimal networks in a relaxed space of directed graphs that may contain cycles. Found cyclic solutions are iteratively ruled out by removing one arc in it and branching over the possible choices of the arc to remove.

Provably optimal (OPT)**.** All optimal algorithms (including A*, ILP, BnB, and their variants) find equivalent networks[1]. Thus, in the context of the generalization analysis, they are equivalent and only one of the optimal algorithms is used for each dataset.

Solver resource constraints. For running the experiments we used a cluster of Dell PowerEdge M610 computing nodes equipped with two 2.53-GHz Intel Xeon E5540 CPUs and 32-GB RAM. For each individual run, we used a timeout of 2 h and a 28-GB memory limit. We treat the runtime of any instance as 2 h if a solver exceeds either the time or memory limit.

4 Datasets

We used a similar set of benchmark datasets for both studies; in total, we used 48 distinct datasets[2]:

- Datasets sampled from benchmark Bayesian networks. 19 datasets, SAMPLED.
- Datasets from the UCI repository. 19 datasets, UCI.
- Datasets sampled from random Bayesian networks. 7 datasets, SYN.
- Datasets we compiled by processing log files. 3 datasets, LOG.

We preprocessed each dataset by removing all continuous variables, variables with very large domains (e.g., unique identifiers), and variables that take on only one value. Other than preprocessing, the datasets were used slightly differently in the generalization study compared to the exact solver analysis; the relevant sections discuss exactly how the datasets were used.

[1] This work assumes s is score-equivalent (Heckerman et al. 1995).
[2] The datasets are available at http://bnportfolio.cs.helsinki.fi/.

5 Generalization of Learned Networks

Our aim in the first part of this work is to shed light on the relationship of different learning strategies, based on the solvers discussed in Sect. 3, and the unknown discrepancy between training set scores and generalization. In particular, we address the following research questions for different fixed learning algorithms and training sets.

Q1 How do hard constraints on the number of parents in learned structures affect their generalization?
Q2 How does the amount of training data affect the generalization of learned structures?
Q3 Which learning strategies result in networks with the best generalization?

Our main findings, based on a rigorous experimental setup, are the following. With respect to Q1, we show that for small datasets, hard constraints limiting the maximum number of parents to 2 improves generalization on a few datasets for local search algorithms; however, optimal algorithms usually benefit from a higher limit. We answer Q2 by using increasingly large subsets of available training data. Regardless of the algorithms' guarantees, more training data results in more accurate predictions on testing data. Finally, we address Q3 by considering all of the data collected during the evaluation. For some datasets, simple strategies such as the tractable Chow-Liu algorithm can provide good generalization. However, the simple strategies fail to generalize well on other datasets. Predictive likelihood results show that OPTconsistently generalizes well.

5.1 Experimental Setup

Datasets. We used 29 datasets from the UCI and SAMPLED categories; the number of variables in the datasets ranges from 17 to 60, and the number of records ranges from about 30 to 20 000. We used standard 10-fold cross-validation in order to evaluate the learning strategies.

Parent limit. For all algorithms except CL, we used hard limits of 2 and 8 on the number of parents. When discussing algorithms, we use a subscript to indicate the maximum number of parents, such as OPT_8.

Scoring function. We selected the commonly-used Bayesian Dirichlet with score equivalence and uniform structure prior (BDeu) scoring function (Heckerman et al. 1995) with an equivalent sample size (ESS) of 1 as the scoring function.

Inference. For all learned structures, parameter values were set using a symmetric Dirichlet prior with a concentration parameter of 1 (which is equivalent to Laplacian smoothing). All testing likelihood calculations were performed by multiplying relevant family factors.

Evaluation. In order to address our research questions, we use the predictive likelihood to evaluate the generalization capability of the learned networks. In

particular, for a dataset d and learning strategy l, we calculate the per-prediction-likelihood, $\ell_{pp}^{d,l}$, which is the likelihood of each prediction on the test set,

$$\ell_i^{d,l} = \sum_{r=1}^{N} \log P(d_r|\mathcal{N}) = \sum_{r=1}^{N}\sum_{i=1}^{n} \log P(X_i^r|PA_i^r) \tag{3}$$

$$\ell_{pp}^{d,l} = -\frac{\sum_{i=1}^{10} \ell_i^{d,l}}{N_d \cdot n_d}, \tag{4}$$

summing over the folds $i = 1..10$, where $\ell_i^{d,l}$ is the predictive likehood on the test set for fold i using learning strategy l, N_d is the number of records in the test set, and n_d is the number of variables in the dataset.

The numerator of Eq. 4 is the sum over all of the test set predictive likelihoods for learning strategy l and dataset d. Each $\ell_i^{d,l}$ term comprises $\frac{N_d}{10} \cdot n_d$ terms. In total, the sum in the numerator includes $N_d \cdot n_d$ terms, each of which corresponds to the log probability of one variable of one record from the test set. Consequently, the denominator serves as a normalizing constant, and $\ell_{pp}^{d,l}$ is the average log probability of each prediction.

In order to compare learning strategies, we normalize the $\ell_{pp}^{d,l}$ values for each dataset between 0 and 1 to obtain

$$\hat{\ell}_{pp}^{d,l} = 1 - \frac{\ell_{pp}^{d,l} - \min_{l'}\{\ell_{pp}^{d,l'}\}}{\max_{l'}\{\ell_{pp}^{d,l'}\} - \min_{l'}\{\ell_{pp}^{d,l'}\}} \tag{5}$$

where l' ranges over all learning strategies. Note that, after normalization, the learning strategy with the best $\ell_{pp}^{d,l}$ has $\hat{\ell}_{pp}^{d,l} = 0$ while the worst learning strategy has $\hat{\ell}_{pp}^{d,l} = 1$.

It is important to note that $\ell_{pp}^{d,l}$ and $\hat{\ell}_{pp}^{d,l}$ consider all variables equally. In particular, they do not consider a special "class" variable.

5.2 Impact of Restricting Parent Set Size

We study question Q1 by comparing the $\hat{\ell}_{pp}^{d,l}$ among datasets when using $k = 2$ and $k = 8$ as the maximum number of parents for each learning algorithm. The BDeu score implicitly restricts the maximum number of selected parents as a soft constraint by integrating over all parameterizations of parent instantiations. Other scores, such as MDL, explicitly incorporate a complexity penalty to discourage large parent sets. In both cases, though, this restriction is a soft constraint. Here, we consider the maximum number of parents as a *hard constraint.*

Optimal. Figure 1 (left) shows the performance (in terms of $\hat{\ell}_{pp}^{d,l}$) of generalization using OPT_k for parent limits $k = 2, 8$. The (left, top) and (left, bottom) plots show distinctly different patterns. Figure 1 (left, top) clearly shows that OPT_2 results in better generalization for SAMPLED datasets with 100 records. However, as the number of records increases, OPT_8 yields better performance. In contrast, for UCI datasets, OPT_8 is almost always better.

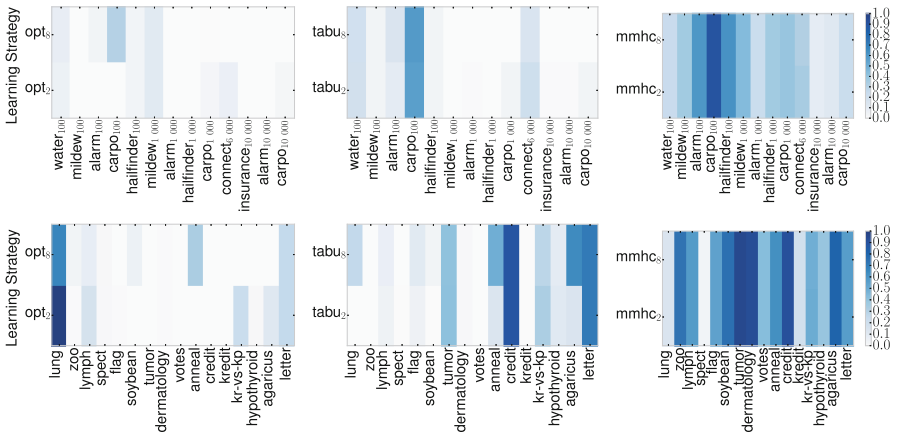

Fig. 1. The $\hat{\ell}_{pp}^{d,l}$ values for OPT (left), TABU (center) and MMHC (right) with a hard limit of $k = 2$ and $k = 8$ for SAMPLED (top) and UCI (bottom) datasets. The datasets are sorted in ascending number of records. Lighter colors indicate better performance. Close inspection of the MMHC strategies show some slight difference; however, they are difficult to discern in the scaled image.

Tabu. In contrast to the results for OPT, Fig. 1 (center, bottom) shows that TABU_2 generalizes better than TABU_8 for UCI datasets. One possible explanation for this difference is that the greedy strategy of TABU_8 favors structures which improve the likelihood while increasing the complexity of the learned structures. Thus, the learned structure overfits the training data and does not generalize well to testing data. In contrast, as OPT is guaranteed to find the best-scoring structure, it finds structures which better balance training set likelihood and complexity. The hard constraints on the number of parents for TABU_2 forbid it from selecting the complex structures. Both TABU_2 and TABU_8 typically generalize well on SAMPLED datasets.

MMHC. Figure 1 (right) shows that the hard parent limit has little effect on $\hat{\ell}_{pp}^{d,l}$ for MMHC. The first phase of MMHC uses a set of statistical independence tests to restrict the learned network structures. For many of the datasets, the relatively small number of records restricts the power of these tests and leads to a very small search space in the second phase, despite initially allowing many more structures for the 8-parent space.

In summary, the answer to Q1 clearly depends both on the training datasets and learning algorithm; the global guarantees of OPT allow it to fully take advantage of the larger $k = 8$ search space, but the local search strategy of TABU performs better in the more restricted $k = 2$ space.

More data is required to accurately estimate the conditional probability distributions for complex structures (with more parameters). This may explain why OPT_2 generalizes better than OPT_8 for datasets with a small number of records.

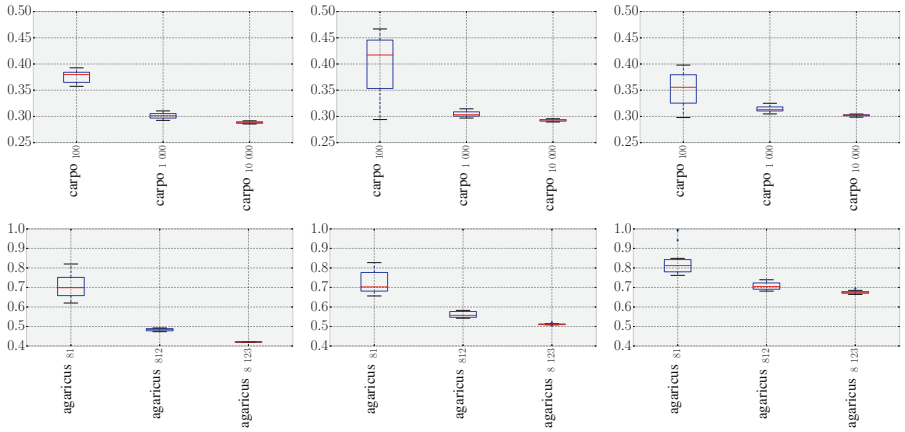

Fig. 2. The $\hat{\ell}_{pp}^{d,l}$ values for using the OPT_8 (left), TABU_2 (center), and CL (right) learning strategies as the number of records increases. The top row is for the *carpo* dataset (SAMPLED); the bottom row is for the *agaricus* dataset (UCI). Note the different y-axes for the plots. Lower values and smaller boxes are better.

5.3 Impact of Amount of Training Data

To investigate the impact of the amount of available training data, to answer Q2 we compared how $\ell_{pp}^{d,l}$ of OPT_8, TABU_2 and CL behave as the number of records available for training increases. Figure 2 shows that for all algorithms on both SAMPLED and UCI datasets, more records lead to better $\ell_{pp}^{d,l}$. Furthermore, the plots also show that with more records, the variance of $\ell_{pp}^{d,l}$ decreases. Interestingly, the plot also shows that CL performs better than OPT_8 and TABU_2 on *carpo*, a SAMPLED dataset, when only 100 records are available. This again highlights that restricted model classes can generalize better than those which allow more parameters, especially when little data is available to estimate the parameter values. Despite the differences in guarantees, OPT_8, TABU_2 and *cl* perform similarly for $carpo_{1\ 000}$ and $carpo_{10\ 000}$.

As with *carpo*, for the UCI*agaricus* dataset, the likelihood improves and variance decreases as the number of records increases. However, OPT_8 improves from $\ell_{pp}^{d,l} \approx 0.7$ for 81 records to $\ell_{pp}^{d,l} \approx 0.48$ with 812 records. In contrast, TABU_2 only improves from $\ell_{pp}^{d,l} \approx 0.7$ to about $\ell_{pp}^{d,l} \approx 0.55$, and CL exhibits even less improvement. For *agaricus*, OPT_8 using only 812 records results in better generalization than TABU_2 or CL with all 8 123 records.

We observed similar behavior on other SAMPLED and UCI datasets as the amount of training data was varied. The same general trends hold for all algorithms and datasets with respect to Q3. Namely, the predictive likelihood improves and variance decreases as the size of the training set increases.

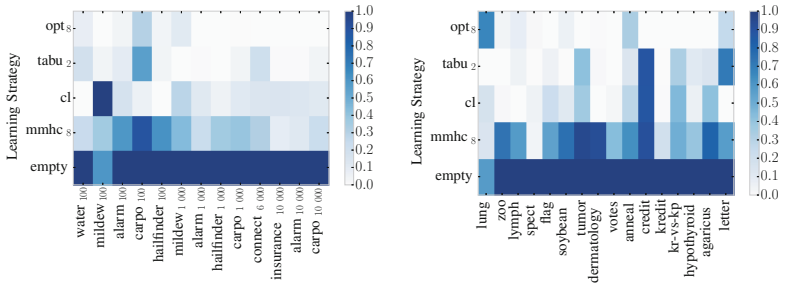

Fig. 3. The $\hat{\ell}_{pp}^{d,l}$ values for the best learning strategies. The empty network is included as a baseline. The SAMPLED datasets are shown in the left heatmap, and UCI datasets are in the right. The datasets are sorted in ascending number of records. Lighter colors indicate better performance (Color figure online).

5.4 Comparison Across Learning Strategies

Finally, based on the previous results, we studied Q3 by choosing the best learning strategies and comparing their $\hat{\ell}_{pp}^{d,l}$ across all of the datasets. In essence, we fix the training set while varying the learning strategy. Additionally, EMPTY (with no edges) was included as a baseline. The results in Fig. 3 show several expected trends and a few surprises. As expected, EMPTY is the worst on almost all of the datasets. For the reasons mentioned in Sect. 5.2, MMHC_8 was typically worse than the other strategies. These trends are consistent for both SAMPLED and UCI datasets. For SAMPLED datasets, TABU_2 and OPT_8 have very similar $\hat{\ell}_{pp}^{d,l}$ for most datasets; the $\hat{\ell}_{pp}^{d,l}$ of CL is also surprisingly similar to that of the two more "sophisticated" strategies.

For UCI datasets, OPT_8 continues to consistently have good $\hat{\ell}_{pp}^{d,l}$. On the other hand, CL and TABU_2 exhibit much more inconsistency in their generalization relative to OPT_8. For some datasets, such as *dermatology* and *kredit*, they match OPT_8; on others, such as *credit* and *tumor*,CL and TABU_2 do not generalize well. Surprisingly, CL exhibits the best $\hat{\ell}_{pp}^{d,l}$ for *letter*, the UCI dataset with the most records.

For Q3, OPT guarantees consistently translate into networks with good generalization. Algorithms with weaker guarantees produce networks with inconsistent generalization.

Comments on Datasets. Besides the behavior of the learning algorithms, these results also suggest differences in the datasets themselves. In particular, it seems that SAMPLED datasets are "easier," in the sense that many learning strategies find networks which generalize well. On the other hand, only the strategy with strong guarantees consistently generalizes well on UCI datasets. In some sense, this result is not surprising. The SAMPLED data is by construction accurately modeled by a BN, while it is very unlikely that UCI datasets are faithful to any BN. These caveats are important for future evaluations.

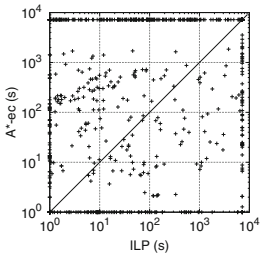

Fig. 4. Comparison of two state-of-the-art algorithms for finding an optimal Bayesian network. Runtimes below 1 or above 7200 s are rounded to 1 and 7200, respectively. See Sect. 5 for descriptions of the solvers and the datasets.

6 Exact Solver Empirical Hardness Models

As shown in Sect. 5, exact algorithms often lead to networks which generalize better than those found with approximation algorithms. Due to the intrinsic differences between the algorithmic approaches underlying the solvers, it is not surprising that their relative efficiency varies on a per-instance basis. To exemplify this, a comparison of the runtimes of ILP and A*EC is illustrated in Fig. 4 using typical benchmark datasets. Evidently, neither of these two solvers dominates the other, as there clearly are instances on which one solver is much more efficient than the other.

To explain the observed orthogonal performance characteristics shown in Fig. 4, it has been suggested, roughly, that typical instances can be solved to optimum by A* if the *number of variables* n is at most around 50 (Fan et al. 2014), and by ILP if the number of *candidate parent sets* m is not very large (Bartlett and Cussens 2015).

Unfortunately, beyond this rough characterization, the practical time complexity of the fastest algorithms is currently poorly understood. The gap between the analytic worst-case and best-case bounds is very large, and typical instances fall somewhere in between. Moreover, the sophisticated search heuristics employed by the algorithms are quite sensitive to small variations in the instances, which results in somewhat chaotic looking behavior of runtimes. Even the following basic questions are open:

Q4 Are the simple features, the number of variables n and the number of candidate parent sets m, sufficient for determining which of the available solvers is fastest for a given instance?

Q5 Are there other efficiently computable features that capture the hardness of the problem significantly more accurately than n and m alone?

In this section, we answer both these questions in the affirmative. We answer Q4 by learning a simple, yet nontrivial, model that accurately predicts the fastest solver for a given instance based on n and m only. We show how this yields an algorithm portfolio that almost always runs as fast as the fastest algorithm, thus

significantly outperforming any fixed algorithm on a large collection of instances. To address this issue and answer Q5, we introduce and study several additional features that potentially capture the hardness of the problem more accurately for a given solver. In particular, we show that learning models with a much wider variety of features yields significant improvement in the prediction accuracy.

Related Work. The idea of learning to predict an algorithm's runtime from empirical data is not new. Rice (1976) proposed feature-based modeling to facilitate the selection of the best-performing algorithm for a given problem instance, considering various example problems. More recently, machine learning and empirical hardness models (Leyton-Brown et al. 2002) have been used for solver portfolios in several domains.

6.1 Capturing Hardness

The *hardness* of a BNSL instance, relative to a given solver, is the runtime of the solver on the instance. We aim to find a *model* that approximates the hardness and is efficient to evaluate for any given instance from a small set of efficiently computable *features* of BNSL instances. We can then learn the model by computing the feature values and collecting empirical runtime data from a set of BNSL instances. We first introduce several candidate features that are potentially informative about the hardness of BNSL instances for one or more solvers. We then explain how we learn a hardness model and estimate its prediction accuracy.

Features for BNSL. We consider several features which naturally fall into four categories, explained next, based on the strategy used to compute them: **Basic**, **Basic extended**, **Lower bounding**, and **Probing**. Due to space constraints, please refer to the original paper for the complete list of features.

The **Basic** features include the number of variables n and the mean number of candidate parent sets per variable, m/n. The features in **Basic extended** summarize the size distribution of the collections $\mathcal{P}_i$ and the parent sets PA_i in each $\mathcal{P}_i$. In the **Lower bounding** category, the features reflect statistics from a directed graph that is an optimal solution to a relaxation of the original BNSL problem. In the **Simple LB** subcategory, a graph is obtained by letting each variable select its best parent set according to the scores. Many solvers use this lower bounding technique. In the **Pattern database LB** subcategory, the features are the same but the graph is obtained from a more sophisticated relaxation using pattern databases (Yuan and Malone 2013).

Probing refers to running a solver for several seconds and collecting statistics about its behavior during the run. We consider three probing strategies: TABU, an anytime variant of A* (Malone and Yuan 2013), and ILP (Cussens et al. 2013). Probing is implemented by running each algorithm for 5 s and collecting several features of the learned structure.

Model Training and Evaluation. Based on the features discussed in the previous section, we trained reduced error pruning trees (REP trees) (Quinlan 1987)

to predict the runtime of an instance of BNSL for each solver. We chose these decision tree models because of their interpretability, compared to techniques such as neural networks or support vector machines, and because of their flexibility, compared to linear regression and less expressive model classes.

6.2 Experiment Setup

We used all of the datasets mentioned in Sect. 4. We considered 5 different scoring functions[3]: BDeu with the Equivalent Sample Size selected from $\{0.1, 1, 10, 100\}$ and BIC. For each dataset and scoring function, we generated scores with parent limits ranging from 2 to up to 6. The size of the datasets ranged from about 100 records to over 60,000 records. For portfolio construction we removed very easy instances (solved within 5 s by all solvers) as uninteresting, and instances on which all solvers failed, leaving 586 instances. We evaluated the portfolios using 10-fold cross-validation.

6.3 Portfolios for BNSL

This section focuses on the construction of practical BNSL solver portfolios in order to address question Q4. Optimal portfolio behavior is to always select the best-performing solver for a given instance. As the main results, we will show that, perhaps somewhat surprisingly, it is possible to construct a practical BNSL solver portfolio that is close-to-optimal using only the **Basic** features.

As the basis of this work, we ran all the solvers and their parameterizations on all the benchmark instances. Figure 5 (left) shows the number of instances for which each solver was the fastest. The performance of BnB is in general inferior to the other solvers; in the following we will focus on ILP and A*ec. However, recall Fig. 4: while ILP is clearly best measured in the number of instances solved the fastest, the performance of ILP on a per-instance basis is very much orthogonal to that of A*. We now show that a simple BNSL solver portfolio can capture the best-case performance of *both* of these approaches.

A Very Simple Solver Portfolio. We found that using only the **Basic** features are enough to construct a highly efficient BNSL solver portfolio. While on an intuitive level the importance of these two features may be to some extent unsurprising, such intuition does not directly translate into an actual predictor that would close-to-optimally predict the best-performing solver.

Figure 5 (right) shows the performance of each individual solver variant, as well as the Virtual Best Solver (VBS), which is the theoretically optimal portfolio which always selects the best algorithm, constructed by selecting *a posteriori* the fastest solver for each input instance. "portfolio" is our simple portfolio which uses only the **Basic** features. As the figure shows, the performance of our simple portfolio is very close to the theoretically optimal performance of VBS and greatly outperforms the individual solvers.

[3] Our results were not very sensitive to the scoring function, except its effect on the number of CPSs, so our results generalize to other decomposable scores.

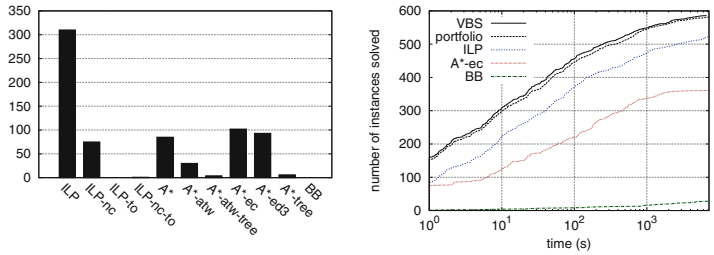

Fig. 5. (left) VBS contributions of each solver, i.e., the number of instances for which a solver was fastest. Several variants of ILP and A* were used. Please see the original paper for more details. (right) Solver performance: VBS, our simple portfolio, ILP, A*EC, and BnB.

6.4 Predicting Runtimes

To address Q5, we investigate the effect of the feature sets on prediction accuracy.

As just shown, the **Basic** features can effectively distinguish between solvers to use on a particular instance of BNSL. However, notable improvements in runtime prediction accuracy are gained by employing a much wider range of features. Figure 6 (left) compares the actual runtimes for A*EC to the predictions made by the REP tree model trained using only the **Basic** features. The model clearly splits the instances into only a few bins and predicts the same runtime for all instances in the bin. The actual runtime ranges are quite wide within each bin. For example, for the bin with predictions near 80 s, the actual runtimes span from around 5 s to about an hour. Even though these predictions allow for good portfolio behavior, they are not useful to estimate actual runtime.

On the other hand, Fig. 6 (right) shows the same comparison for models learned using **A* probing** features (1–38, 51–62). Many more of the predictions fall along or near the main diagonal. That is, the larger, more sophisticated feature set results in more accurate runtime predictions. We observed similar, though less pronounced, trends for ILP.

6.5 REP Tree Characteristics

For additional insight, we considered how often specific features were selected (Table 1). A feature is rarely selected for predicting both solvers. This further confirms that the solver runtimes are influenced by different structural properties of instances. Nevertheless, **Simple LB** features were helpful for both algorithms. Somewhat surprisingly, the **Pattern database LB** features were more useful for ILP, even though A*EC directly uses the pattern database in its search. For all of the graph-based features (node degree and non-trivial SCCs), the standard deviation was always selected over the maximum and mean. This suggests that systematic variations between nodes are important for determining the hardness of an instance. The table also shows that a small number of features were consistently selected for most of the cross-validation folds for any particular solver.

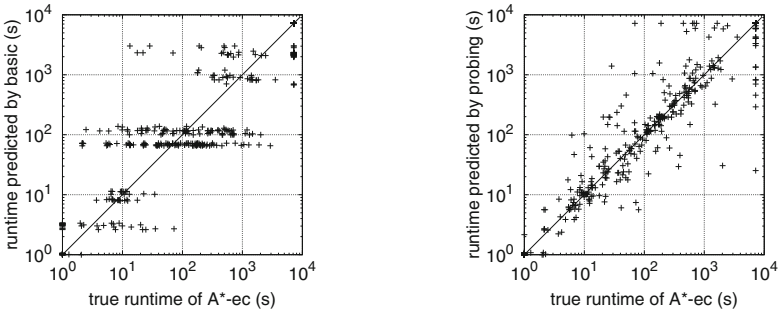

Fig. 6. Predicted vs. actual runtimes for A*EC using **Basic** features only (left) and all features up to A* probing (right).

Qualitatively, this implies that most of the trees were based on the same small set of features. Developing a more in-depth understanding of these instance characteristics in light of solver performance is an important aspect of future work.

Table 1. Features used for A*EC and ILP in more than 5 of the 10 cross-validation folds. For each solver, the set of possible features consisted of non-probing features (1–38) and the relevant probing features.

	Feature	A*-ec	ILP
(1)	Number of variables, n	10	0
(2)	Number of CPS, mean	0	7
(3)	Number of CPS, sum, m	2	10
(4)	Number of CPS, max	0	7
(8)	CPS cardinalities, sd	0	8
(11)	Simple LB, Node in-degree, sd	0	7
(14)	Simple LB, Node out-degree, sd	8	0
(17)	Simple LB, Node degree, sd	10	0
(26)	Pd LB, Node in-degree, sd	1	9
(38)	Pd LB, Size of non-trivial SCCs, sd	0	8
(62)	A* probing, Error bound	10	0
(68)	ILP probing, Node out-degree, sd	0	10
(74)	ILP probing, Error bound	0	10

7 Discussion

Bayesian network structure learning (BNSL) continues to be an area of very active research. In this review, we have presented two orthogonal studies of

BNSL algorithms. The first demonstrated that, whenever possible, exact learning algorithms should be used for finding structures. The second study showed that it is typically possible to not only select the best exact learning algorithm for a given dataset but also predict how long it will take to find the optimal structure.

These studies suggest a variety of future investigations. For example, the most "Bayesian" approach to generalization should be a model averaging strategy, but the current work considers only a single structure. In light of the generalization results, empirical hardness models could be built for different dataset categories.

References

Bartlett, M., Cussens, J.: Integer linear programming for the Bayesian network structure learning problem. In: Artificial Intelligence (2015)

Chickering, D.M.: Learning Bayesian networks is NP-complete. In: Fisher, D., Lenz, H.-J. (eds.) Learning from Data: Artificial Intelligence and Statistics V. Lecture Notes in Statistics, vol. 112, pp. 121–130. Springer, New York (1996)

Chickering, D.M.: Learning equivalence classes of Bayesian-network structures. J. Mach. Learn. Res. **2**, 445–498 (2002)

Chow, C., Liu, C.: Approximating discrete probability distributions with dependence trees. IEEE Trans. Inf. Theory **14**(3), 462–467 (1968)

Cooper, G.F., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. Mach. Learn. **9**, 309–347 (1992)

Cussens, J., Bartlett, M., Jones, E.M., Sheehan, N.A.: Maximum likelihood pedigree reconstruction using integer linear programming. Genet. Epidemiol. **37**(1), 69–83 (2013)

de Campos, C.P., Ji, Q.: Efficient learning of Bayesian networks using constraints. J. Mach. Learn. Res. **12**, 663–689 (2011)

de Campos, L.M., Huete, J.F.: A new approach for learning belief networks using independence criteria. Int. J. Approximate Reasoning **24**(1), 11–37 (2000)

Fan, X., Yuan, C., Malone, B.: Tightening bounds for Bayesian network structure learning. In: Proceedings of the 28th AAAI Conference on Artificial Intelligence (2014)

Friedman, N., Nachman, I., Peer, D.: Learning Bayesian network structure from massive datasets: the "sparse candidate" algorithm. In: Proceedings 13th Conference on Uncertainty in Artificial Intelligence (1999)

Glover, F.: Tabu search: a tutorial. Interfaces **20**(4), 74–94 (1990)

Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: the combination of knowledge and statistical data. Mach. Learn. **20**, 197–243 (1995)

Koivisto, M., Sood, K.: Exact Bayesian structure discovery in Bayesian networks. J. Mach. Learn. Res. **5**, 549–573 (2004)

Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: the case of combinatorial auctions. In: Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 556–572. Springer, Heidelberg (2002)

Malone, B., Järvisalo, M., Myllymäki, P.: Impact of learning strategies on the qual-packing Bayesian networks: an empirical evaluation. In: Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (2015)

Malone, B., Kangas, K., Järvisalo, M., Koivisto, M., Myllymäki, P.: Predicting the hardness of learning Bayesian networks. In: Proceedings of the 28th AAAI Conference on Artificial Intelligence (2014)

Malone, B., Yuan, C.: Evaluating anytime algorithms for learning optimal Bayesian networks. In: Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (2013)

Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)

Moore, A., Wong, W.-K.: Optimal reinsertion: a new search operator for accelerated and more accurate Bayesian network structure learning. In: Proceedings of the 20th International Conference on Machine Learning, pp. 552–559 (2003)

Ott, S., Imoto, S., Miyano, S.: Finding optimal models for small gene networks. In: Proceedings of the Pacific Symposium on Biocomputing (2004)

Parviainen, P., Koivisto, M.: Exact structure discovery in Bayesian networks with less space. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, Quebec, Canada. AUAI Press (2009)

Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Mateo (1988)

Quinlan, J.R.: Simplifying decision trees. Int. J. Man Mach. Stud. **27**, 221–234 (1987)

Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)

Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson Education, Upper Saddle River (2003)

Silander, T., Myllymäki, P.: A simple approach for finding the globally optimal Bayesian network structure. In: Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (2006)

Silander, T., Roos, T., Kontkanen, P., Myllymäki, P.: Factorized normalized maximum likelihood criterion for learning Bayesian network structures. In: Proceedings of the 4th European Workshop on Probabilistic Graphical Models (2008)

Spirtes, P., Glymour, C., Schemes, R.: Causation, Prediction, and Search, 2nd edn. MIT Press, Cambridge (2000)

Suzuki, J.: Learning Bayesian belief networks based on the MDL principle: an efficient algorithm using the branch and bound technique. IEICE Trans. Inf. Syst. **E82–D**(2), 356–367 (1999)

Teyssier, M., Koller, D.: Ordering-based search: a simple and effective algorithm for learning Bayesian networks. In: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (2005)

Tian, J.: A branch-and-bound algorithm for MDL learning Bayesian networks. In: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (2000)

Tsamardinos, I., Brown, L., Aliferis, C.: The max-min hill-climbing Bayesian network structure learning algorithm. Mach. Learn. **65**, 31–78 (2006)

van Beek, P., Hoffmann, H.-F.: Machine learning of Bayesian networks using constraint programming. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 429–445. Springer, Heidelberg (2015)

Yuan, C., Malone, B.: Learning optimal Bayesian networks: a shortest path perspective. J. Artif. Intell. Res. **48**, 23–65 (2013)