Eliminating Variable Order Instability in Greedy Score-Based Structure Learning.

Neville K. Kitson N.K.KITSON@QMUL.AC.UK
Anthony C. Constantinou A.CONSTANTINOU@QMUL.AC.UK

Machine Intelligence and Decision Systems (MInDS) Group,
Queen Mary University of London, Mile End Road, London, E1 4NS, United Kingdom

Editors: J.H.P. Kwisthout & S. Renooij

Abstract

Many Bayesian Network structure learning algorithms are unstable in that the learnt graph is sensitive to arbitrary artefacts of the dataset, such as the ordering of columns (i.e., variable order). PC-Stable, developed by Colombo and Maathuis (2014), attempts to address this issue for the widely-used PC algorithm, prompting researchers to use the 'stable' version instead. However, this problem seems to have been overlooked for score-based algorithms. In this study, we show that some widely-used score-based algorithms suffer from the same issue and that PC-Stable, although less sensitive than most of the score-based algorithms tested, is not completely stable. We also present a solution to score-based greedy hill-climbing that completely eliminates this instability, and provide two implementations: the HC-Stable and Tabu-Stable algorithms, the latter of which learns more accurate graphs than all the well-known algorithms we compared it to.

Keywords: Bayesian Networks; algorithm stability; hill-climbing; Tabu algorithm

1. Introduction

Bayesian Networks (BNs) (Koller and Friedman, 2009) are an approach for modelling complex probabilistic relationships in diverse domains such as healthcare (Kyrimi et al., 2021), fault diagnosis (Cai et al., 2017) and the environment (Vitolo et al., 2018). They can be used to answer *probabilistic queries* which predict the probability distribution of a subset of variables conditional on the values of other variables, and so can answer questions such as *if these symptoms are present, what is the probability the patient has disease Y?*. Moreover, if one additionally assumes that the relationships are *causal*, then the resulting *Causal* BN can be used to answer *interventional queries* such as *if the patient is given this treatment, what is the likely outcome?* (Pearl, 2009). Thus, BN's have an important potential role as A.I. decision support systems.

Because BNs are probabilistic graphical models, one key challenge is to specify the graphical structure underlying them. Using machine learning to infer this structure from observational data is an active research area. Recent work by Kitson and Constantinou (2024) shows that many algorithms are *unstable*; that is, sensitive to artefacts of the data, such as the ordering of columns in the data. The principal contribution of this paper is to describe an approach which eliminates this instability for greedy score-based hill-climbing algorithms. We show that one of our implementations, Tabu-Stable, is completely stable *and* learns more accurate graphs than all of the well-known algorithms that we compare it to. The approach described in this paper also has wider applicability to many algorithms that make use of greedy hill-climbing, such as hybrid algorithms.

2. Background

2.1. Bayesian Networks

The key element of a Bayesian Network is a Directed Acyclic Graph (DAG) where each node represents a variable and the directed edges, or *arcs*, represent a dependence relationship between the two variables. We denote the n variables in the BN as $X_1, ..., X_n$. If there is an arc $X_A \longrightarrow X_B$, X_A is termed the *parent* of X_B. The DAG is constructed so that a node is conditionally independent of all variables except its descendants given its parents. This is called the *Local Markov Property* and allows the global probability distribution to be expressed compactly as:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)) \tag{1}$$

where $\mathbf{Pa}(X_i)$ are the parents of X_i.

A key result flowing from the Local Markov Property is that a graphical property of the DAG, *d-separation*, is equivalent to variables being conditionally independent of one another (Pearl, 1988). D-separation can be used to infer the graph structure from the independence and dependence relationships present in the data. In general, however, more than one DAG is consistent with the independence relationships (Verma and Pearl, 1990). These *Markov Equivalent* graphs belong to a *Markov Equivalent Class (MEC)*. A MEC is usually represented by a *Completed Partially Directed Acyclic Graph (CPDAG)*, where directed edges indicate edges where all DAGs in the MEC have that same orientation, and undirected edges indicate that some DAGs have one orientation and the rest the other.

The other component of a BN is the specification of the probabilistic dependence relationship between adjacent variables and the probability distributions assumed. For the discrete variable networks considered here, this takes the form of *Conditional Probability Tables (CPTs)* which define a multinomial distribution for the child values for each combination of parental values.

2.2. Structure Learning Algorithms

The specification of a BN's DAG structure may be undertaken using human expertise, using a structure learning algorithm to learn it from data, or a combination of both. Structure learning algorithms usually learn from observational data, since it is more readily available. The algorithms typically make assumptions that often do not hold in practice. For example, that there is no missing data, measurement error or latent confounders.

Constraint-based algorithms such as PC (Spirtes and Glymour, 1991), GS (Margaritis and Thrun, 1999) and Inter-IAMB (Tsamardinos et al., 2003) use statistical conditional independence (CI) tests to identify the independence relationships in the data, and use the d-separation principle to infer the DAG structure. Constraint-based algorithms usually assume *faithfulness*, which states that there are no independence relationships in the data which are not implied by the DAG. Variants of the approach such as FCI (Spirtes et al., 2001) can account for latent confounders.

Score-based algorithms represent the second major class of algorithms. These follow a more traditional machine-learning approach of using an objective function to assign a

score to each graph and then employ some strategy to find a high-scoring graph. The Hill-Climbing (HC) (Herskovits, 1990) and Tabu (Bouckaert, 1995) algorithms are two simple score-based algorithms that remain competitive and commonly-used. Other score-based approaches search through MEC space, for example, GES (Chickering, 2002) and FGES (Ramsey et al., 2017). *Exact* score-based algorithms, such as GOBNILP (Bartlett and Cussens, 2017) guarantee to return the highest scoring graph, though typically with limits placed on the number of parents of any node.

Other classes of algorithms include *hybrid* ones such as MMHC (Tsamardinos et al., 2006) and H2PC (Gasse et al., 2014) which use a mix of score and constraint-based methods. More recent developments include algorithms which make additional assumptions about the functional relationships between variables to identify arc orientations (Peters et al., 2014), and algorithms such as NOTEARS (Zheng et al., 2018) which treat structure learning as a continuous optimisation problem. Kitson et al. (2023) provide a comprehensive survey across the different classes of algorithms.

The objective function in score-based approaches usually includes the log-likelihood of the data being generated from the graph, with two commonly-used scores being BIC (Suzuki, 1999) and BDeu (Heckerman et al., 1995). The BIC score, S_{BIC}, for a graph G with n variables and dataset D is computed as follows:

$$S_{BIC}(G, D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} \left[N_{ijk} \log \frac{N_{ijk}}{N_{ij}} \right] - \frac{\log N}{2} \cdot F \tag{2}$$

The first term on the right hand side of Equation 2 is the log-likelihood and is based on counts of values in the dataset. Specifically, N_{ijk} is the number of rows where node X_i has the k^{th} out of the r_i possible values and its parents $\mathbf{Pa}(X_i)$ have the j^{th} combination of values out of the q_i possible combinations, and N_{ij} is the total number of rows where the parents have that j^{th} combination of values. The second term is a model complexity penalty, where N is the total number of rows in D, and F is the number of free parameters in the CPTs of the model.

BDeu, S_{BDeu}, is a Bayesian score representing the posterior probability of graph G given the data D assuming some prior beliefs about the probability of each graph and set of parameter values. If all graphs are assumed equally probable, then BDeu is given by:

$$S_{BDeu}(G, D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \left[\log \frac{\Gamma(\frac{N'}{q_i})}{\Gamma(N_{ij} + \frac{N'}{q_i})} + \sum_{k=1}^{r_i} \log \frac{\Gamma(N_{ijk} + \frac{N'}{r_i q_i})}{\Gamma(\frac{N'}{r_i q_i})} \right] \tag{3}$$

where Γ is the Gamma function, N' assigns a weight to the prior parameter beliefs, and other symbols take the same meanings as in Equation 2. BIC and BDeu are *decomposable* scores, since they represent the sum of individual scores for each node. This property facilitates the efficient re-computation of the graph score as the graph is modified. Both are also *score equivalent* which means they assign the same score to all DAGs in a particular MEC.

2.3. Related Work

The instability of structure learning algorithms, and specifically their sensitivity to arbitrary dataset artefacts such as *variable order*, seems to have attracted relatively little attention.

Variable order in this paper means the order of the columns in the dataset, something which is arbitrary and ideally ought to have no effect on the learnt graph. Exact algorithms such as A-Star (Yuan et al., 2011) which implicitly determines the best node order, and those that search over MECs such as GES and FGES should, *in principle*, be insensitive to dataset artefacts. Algorithms that rely on a topological order being specified such as K2 (Cooper and Herskovits, 1992), or ones that generate an order themselves (Larranaga et al., 1996; Behjati and Beigy, 2020), should also be stable. Approaches that average over several learnt graphs, either those sampled from a posterior distribution of graphs such as Order-MCMC (Friedman and Koller, 2003), or which use different sub-samples of data (Broom et al., 2012) or different classes of learner (Constantinou et al., 2023) might be less sensitive since any effect of dataset artefacts may tend to 'cancel out' over the population of learnt graphs.

Nonetheless, the stability of algorithms is rarely explicitly considered or evaluated. An exception is PC-Stable (Colombo and Maathuis, 2014) where the authors strive to minimise the effect of node processing order in the PC algorithm which they find has a considerable effect on how errors propagate throughout the learning process. PC-Stable offers better accuracy and lower sensitivity to variable order than PC, and it is generally chosen over PC for that reason. However the results in Subsection 5.3 demonstrate that it retains a considerable amount of instability. Kitson and Constantinou (2024) demonstrate that variable order can impact the ranking of algorithms, but it is not usually considered when comparing algorithms. Scutari et al. (2019) do try a small number of different orderings in order to improve arc orientations as part of a comparative benchmark, but sensitivity to ordering is not reported.

3. Eliminating Instability in Hill-Climbing

This section discusses the source of instability within hill-climbing algorithms and how it can be addressed. The focus is on the Tabu algorithm since it is commonly used and is competitive in benchmarks, but we apply the same approach to the simpler HC algorithm. The resulting two new algorithms, Tabu-Stable and HC-Stable, are described.

3.1. How Instability Arises in Hill-Climbing Algorithms

HC is a greedy, score-based hill-climbing algorithm. It typically starts exploring the search-space of graphs from an empty DAG, and searches for the single arc addition, deletion or reversal which increases the score of the DAG the most at each iteration. Changes that would create a cycle are not considered. The algorithm terminates when there are no further changes that would increase the score, and the resulting DAG generally has only a locally-maximum score.

Tabu is a higher-performing variant of HC that allows iterations where the score stays the same or decreases allowing the algorithm to escape some local maxima. Tabu maintains a fixed-length list of recently visited DAGs, *tabulist*, to prevent the algorithm from repeatedly considering previously-visited DAGs. The black-coloured pseudo-code in Algorithm 1 shows the main elements of Tabu. The *deltas* variable holds the score change associated with every possible change to the DAG. Lines 5 to 25 form the main iterative loop, with the highest-scoring change for each iteration identified in the **foreach** loop, and applied to the DAG

at line 19. *UpdateDeltas* updates *deltas* appropriately following the change; for example, adding arc $A \longrightarrow B$ would mean that deltas for adding arcs which point to B must be recalculated to take into account that B now has a new parent. The *stop_condition* for the main loop is that none of the last *noinc* (a hyperparameter) changes have increased the score. HC is similar to Algorithm 1 except that *tabulist* is not required, and the *stop_condition* is that there are no further changes which will *increase* the score.

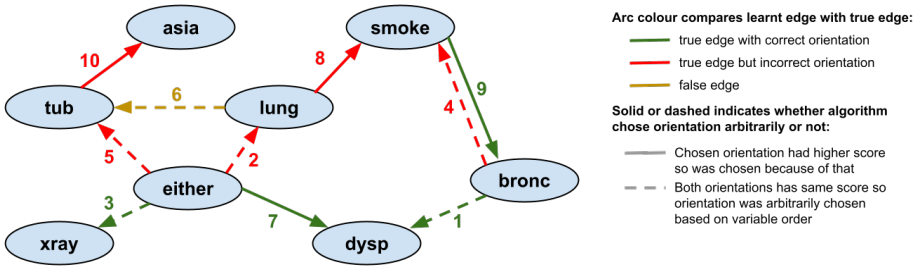

Figure 1: The sequence of DAG changes when HC learns the Asia network from 10,000 samples. The numbers beside each arc show the iteration at which it is added. Arc colours compare the learnt arc against the true arc, and whether it is solid or dashed indicates whether its orientation was arbitrary. Variable order within the dataset is alphabetic.

Figure 1 illustrates the source of instability in hill-climbing by showing the sequence in which HC conventionally learns the Asia network from 10^4 rows of data, when using the BIC score as the objective function. Both orientations of the edge *bronc*—*dysp* give the same maximal score improvement at the first iteration. If HC is implemented to use the variable order to orientate the arc in this situation, and that order is alphabetic, then orientation *bronc* $\longrightarrow$ *dysp* would be chosen. This *arbitrary* orientation happens to agree with the true graph. The orientation of the second arc added, *either* $\rightarrow$ *lung*, is similarly arbitrary, but in this case, incorrect. Just like in the case of constraint-based learning with the PC algorithm, **this instability propagates to subsequent iterations**. For example, if the variable order had been such that the second iteration correctly added *lung* $\rightarrow$ *either*, then edge *tub*—*either* would also have been orientated correctly. This, in turn, would have stopped the extraneous arc *lung* $\rightarrow$ *tub* being added. While the impact of variable order varies from network to network, Kitson and Constantinou (2024) show that it is generally considerable, typically overshadowing the impact of changing objective functions, sample size or hyperparameters.

3.2. Stabilising Hill-Climbing

Algorithm 1 illustrates the key elements, shown in red pseudo-code, of the new algorithm Tabu-Stable that avoids the arbitrary orientations and hence, becomes completely insensitive to the ordering of the variables as read from data. The key is to determine a *stable order* at line 4 that is not dependent on dataset artefacts such as variable order. The DAG change with the highest score improvement at each iteration is determined in the **for each** loop as usual, but this now also records whether there is an equivalent change (*equiv_change*), which adds an arc in the opposite orientation with the same maximum score improvement.

In that case, the one consistent with *stable_order* is added to the DAG. Analogously to the relationship between Tabu and HC, HC-Stable is simply Tabu-Stable with *tabulist* removed and a *stop_condition* that there are no further changes that would *increase* the score.

Algorithm 1: Tabu-Stable (changes to standard Tabu shown in red)

Input: *data*, data set to learn graph from
Output: *best_dag*, highest-scoring DAG found

```
1  stable_order ← GetStableOrder(data) (see Algorithm 2)
2  best_dag ← dag ← empty DAG
3  tabulist ← empty list
4  deltas ← score change for each arc addition
5  repeat
6      max_delta ← None
7      foreach dag_change = AllowedChange(dag, tabulist) do
8          if max_delta = None or delta[dag_change] > max_delta then
9              max_delta ← delta[dag_change]
10             best_change ← dag_change
11             equiv_change ← None
12         else if AddingSameEdgeWithSameDelta(dag_change, best_change) then
13             equiv_change ← dag_change
14         end if
15     end
16     if equiv_change ≠ None and equiv_change consistent with stable_order then
17         best_change ← equiv_change
18     end if
19     dag ← dag + best_change
20     UpdateDeltas(deltas, best_change)
21     insert dag into tabulist
22     if Score(dag, data) > Score(best_dag, data) then
23         best_dag ← dag
24     end if
25 until stop_condition
```

The function *GetStableOrder* shown in Algorithm 2 returns the stable node order used to avoid arbitrary orientations. This order is generated in two stages. Firstly, lines 1-11 of Algorithm 2 produce *dec_score_order*, which contains nodes primarily ordered by the decomposable score, BIC or BDeu for example, that will later be used in the structure learning itself. The sort key for the list has three elements, which in order of precedence are: (1) the score of the node without parents, (2) the mean score of the node where every other node is taken as its single parent (computed in lines 4 to 7), and (3) a textual rendition of the counts of values of the variable, e.g. "{'no': 5, 'yes': 3}". The first two elements of the sort key use scores such as those described in Equation 2 and Equation 3 and are therefore

Algorithm 2: GetStableOrder - determines a stable processing order

```
    Function GetStableOrder(data):
        Input: data, data set learning graph from
        Output: stable_order, stable order for use in Tabu-Stable
1       dec_score_order ← empty_list
2       foreach variable in data do
3           uncond_score = NodeScore(variable)
4           cond_score = 0.0
5           foreach possible single_parent of variable in data do
6               cond_score ← cond_score + NodeScore(variable, single_parent)/(n − 1)
7           end
8           sorted_value_counts ← counts of unique values of variable in data
9           sort_key = (uncond_score, cond_score, sorted_value_counts)
10          dec_score_order ← InsertByKey(variable, sort_key)
11      end
12      inc_score_order ← reverse(dec_score_order)
13      inc_dag ← HC(data, inc_score_order)
14      dec_dag ← HC(data, dec_score_order)
15      if DAGScore(inc_dag, data) > DAGScore(dec_dag, data) then
16          stable_order = TopologicalOrder(inc_dag)
17      else
18          stable_order = TopologicalOrder(dec_dag)
19      end
20      return stable_order
```

determined solely by the *data distribution itself*. Moreover, since the second element of the sort key for a node compares the distribution of combinations of values of that node and every other node, the intuition is that it is very unlikely that two variables will have the same sort key unless they are indeed identical (duplicate). One situation where the first two elements of the sort key are the same for non-identical variables is where the sequences of values for them are isomorphic, e.g. a, a, a, c, c, a and c, c, c, b, b, c, but here the third element of the sort key will differ. If two variables do have the same sort key, then they revert to being ordered by variable order. This will most likely occur because the variables are indeed identical, perhaps because of limited sample size. Thus, the whole algorithm retains some unavoidable sensitivity to variable order when there are identical (or duplicate) variables. This is explored in Subsection 5.2.

It is not expected that *dec_score_order* will necessarily be a ***good*** order, rather, only that it will be insensitive to artefacts of the dataset such as row or column order. Kitson and Constantinou (2024) show that if the node ordering is very different to the topological ordering of the true graph, it will likely adversely affect the accuracy of the learnt graph. To counter this, lines 12-19 of Algorithm 2 attempt to improve the accuracy of the learnt graph by considering both *dec_score_order* and its reverse. It chooses between these two orders by using HC to learn a graph with each order in lines 13 and 14, to test which results in the higher-scoring DAG. The topological order of the higher-scoring learnt graph is returned as

stable_order. The results in Subsection 5.1 show that this empirical approach both stabilises hill-climbing search *and* improves the structural accuracy of the learnt graph.

4. Evaluation

The stability and accuracy of our approach is assessed using synthetic datasets generated from the seventeen discrete variable networks shown in Table 3 of Appendix A. These have between 8 and 109 nodes and are obtained from the Bayesys (Constantinou et al., 2024) and bnlearn repositories (Scutari, 2021b). These networks are commonly used in the literature to assess algorithms, and are largely expert-specified, generally representing causal networks found in the real world. Sample sizes of 10^2, 10^3, 10^4 and 10^5 are used to cover a typical range of sample sizes encountered in practical structure learning, including low-dimensional settings. The BIC score is used throughout. Discrete variable datasets are used since a wide variety of standard datasets are available, but the results here are expected to apply to continuous variable datasets since hill-climbing makes arbitrary orientation choices in that setting too. Further work to confirm this would be valuable.

Since observational data is being used, the CPDAG of the true graph and the learnt graph are compared using the F1 metric, which has the advantage of being comparable between networks of different sizes. The semantics used by the widely-used bnlearn package (Scutari, 2021a) are adopted to compute F1, and are detailed in Appendix B. To assess the stability of algorithms, each combination of sample size and network is repeated 25 times with the variable order, variable names, and row order in the dataset all randomised. The CPDAG F1 is computed for each of these 25 experiments, and the standard deviation (S.D.) of the F1 value reported as an indicator of the sensitivity of the algorithm to these randomised dataset artefacts.

5. Results

5.1. Comparison of Different Orderings

Tabu or HC variant	Precision	Recall	F1	F1 S.D.	Normalised BIC score	Relative Execution time
Tabu - variable order	0.4937	0.4169	0.4426	0.0828	-25.4818	1.00
Tabu - decreasing score order	0.4741	0.3993	0.4236	0.0035	-25.4844	1.16
Tabu - increasing score order	0.5329	0.4487	0.4776	**0.0030**	-25.4849	1.14
Tabu-Stable	**0.5529**	**0.4670**	**0.4976**	0.0035	**-25.4630**	**1.42**
HC - variable order	**0.4281**	**0.3659**	**0.3844**	**0.0930**	**-25.5031**	**0.70**
HC-Stable	0.5029	0.4248	0.4511	0.0032	-25.4684	1.03

Table 1: Mean value of Precision, Recall, F1, S.D. of F1, relative execution time, and BIC score averaged across all networks and sample sizes for standard and stable variants of Tabu and HC. Best values are shown in bold, and worst values in bold red text.

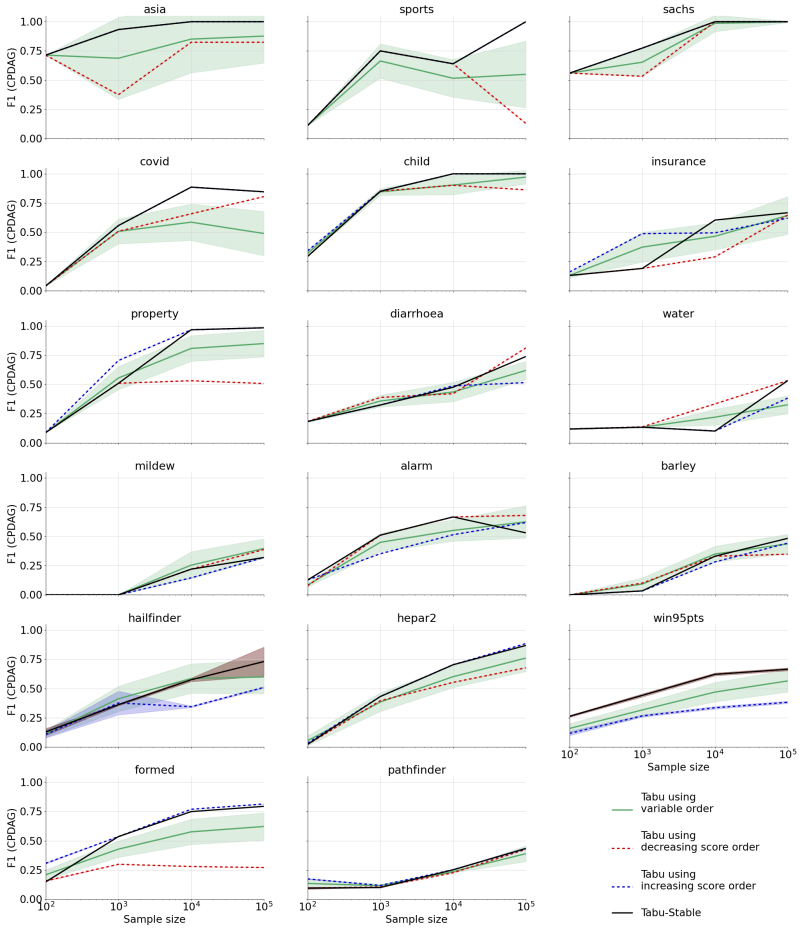

Figure 2: A comparison of the F1 CPDAG of the graph learnt by Tabu using different node orderings: a) variable order, b) simple increasing or c) decreasing score order, and d) Tabu-Stable. 25 experiments with randomised variable names, variable order and row order are conducted for each of the four sample sizes for each network. Shading around lines indicates the standard deviation of F1 values. Note that lines are drawn on the chart in the order shown in the key, so a particular line may be hidden where values are coincident.

This subsection reports the F1 achieved by Tabu using different orderings: a) the conventional approach based on variable order, b) using an increasing or c) decreasing score order, and d) Tabu-Stable incorporating Algorithm 2 which tries both increasing and decreasing score order. Figure 2 shows the F1 achieved with these different orderings, with the shaded area around the lines indicating the S.D. of F1 values at each sample point. This instability is most pronounced for variable order shown in green, being considerable for most networks. The instability is substantially reduced with all three score-based orderings, although some instability is visible for the Hailfinder network which will be discussed in Subsection 5.2.

Table 1 summarises the results from Figure 2 by averaging the F1 and F1 S.D. over all networks and sample sizes, and additionally includes results for HC and HC-Stable, and

Precision and Recall values. The score-based ordering approaches in both HC and Tabu reduces mean F1 S.D. by around thirty and twenty five times respectively, indicating that using a stable order does indeed improve the stability of the learnt graph. Tabu with a decreasing score order worsens the mean F1 value by 0.0190 whereas using an increasing score order improves it by 0.0350 over Tabu using variable order. However, Tabu-Stable produces the largest improvement in F1 of 0.0550, as well as offering the best Precision and Recall values, suggesting that choosing the better of the decreasing and increasing score orders does improve overall accuracy. HC using variable order is more unstable than Tabu, but HC-Stable, again, reduces this instability considerably and increases mean F1 by 0.0667 over conventional HC. However, Tabu-Stable retains most of the accuracy improvement over HC-Stable that Tabu has over HC, suggesting that the accuracy improvement due to Tabu and that using a stable node order are additive in Tabu-Stable. These results confirm the benefits of using Algorithm 2 to determine a stable order.

The penultimate column in Table 1 shows the normalised BIC score averaged across all networks and sample sizes. BIC scales with sample size, so normalised scores obtained by dividing BIC by the sample size are used. The normalised BIC score is characteristic for each network, but this column serves to indicate the overall effect on the BIC score of each approach. It shows that using a simple decreasing or increasing score order tends to worsen the score, whereas Tabu-Stable and HC-Stable which use Algorithm 2 tend to improve the BIC score.

The final column in Table 1 reports the mean execution time relative to Tabu using variable order. HC is substantially quicker than Tabu, since it does not attempt to escape local maxima and generally performs fewer iterations. For Tabu, the increasing and decreasing score orders increase the runtime by around 15%. Tabu-Stable is, on average, 42% slower than conventional Tabu since it performs two additional HC-Stable runs to determine the node order before the final structure-learning process starts. However, we suggest that the benefits of improved accuracy and stability outweigh the increased runtime of this relatively fast algorithm.

5.2. Analysis of the Residual Instability in Tabu-Stable

Sample size	hailfinder	win95pts	formed	pathfinder	hailfinder2	win95pts2
10^2	0.0301	0.0081	0.0041	0.0093	0.0000	0.0077
10^3	0.0081	0.0135	0.0000	0.0000	0.0000	0.0000
10^4	0.0157	0.0123	0.0000	0.0000	0.0000	0.0000
10^5	0.1245	0.0120	0.0000	0.0000	0.0000	0.0000

Table 2: F1 S.D. over 25 random orderings at different sample sizes using Tabu-Stable for the four networks with residual sensitivity to variable order, and for modified versions of Hailfinder and Win95pts where identical (or duplicate) variables are prevented.

Tabu-Stable returns a F1 S.D. of 0.000 at all sample sizes in 13 out of the 17 networks listed in Table 3. Table 2 provides a breakdown of the F1 S.D. by sample size using Tabu-

Stable for the four networks where some instability remains. Formed and Pathfinder only have residual instability at a sample size of 100. However, Hailfinder and Win95pts retain some instability at all sample sizes. This is because both networks have some local structures and CPT values that deterministically create identical values for pairs of variables at all sample sizes. The last two columns in Table 2 show results for versions of these networks that are modified slightly to remove these deterministic relationships. Hailfinder2 and Win95pts2 have had one node and two arcs removed respectively to achieve this. Table 2 shows that instability has been removed completely for Hailfinder2 and only remains at the smallest sample size for Win95pts2.

5.3. Comparing Tabu-Stable and HC-Stable with other algorithms

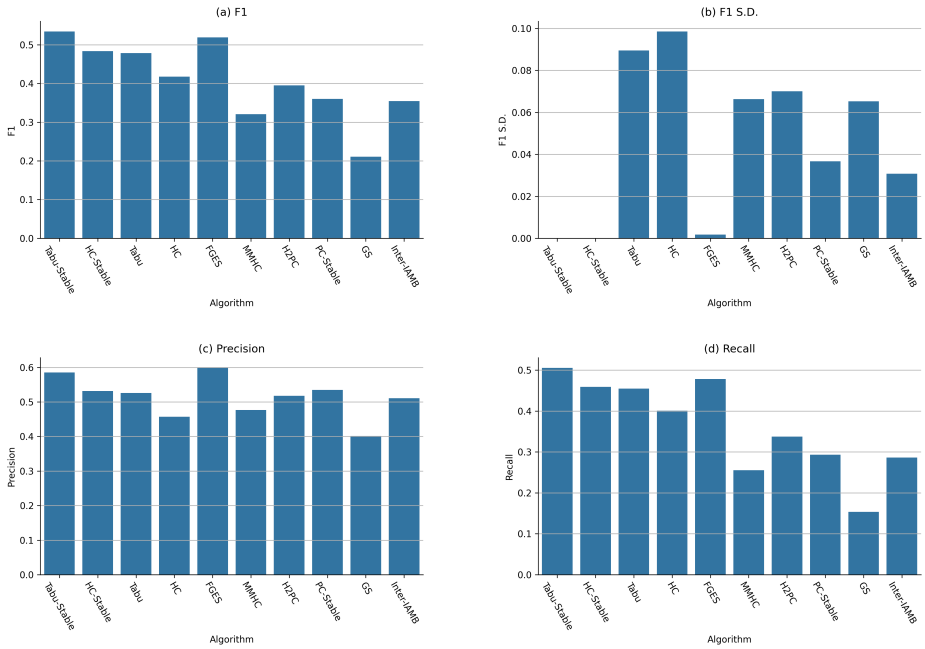

Figure 3: Mean values of F1, F1 S.D., Precision and Recall across networks and sample sizes which do not contain identical (or duplicate) or single-valued variables for different algorithms.

Figure 3 compares the mean F1, F1 S.D., Precision and Recall achieved by different algorithms against HC-Stable and Tabu-Stable. The implementation of FGES from the Tetrad package (Ramsey et al., 2018), and MMHC, H2PC, PC-Stable, GS and Inter-IAMB from the bnlearn package (Scutari, 2021a) are used. Results for Tabu and HC using variable order are also included. The results use the variants of Hailfinder and Win95pts discussed in Subsection 5.2 which avoid identical variables. Additionally, the bnlearn algorithms reject datasets which contain any variable that has the same value for all rows, so the cases where this occurs are also excluded from this comparison. These are the datasets with 100 rows for the Insurance, Water, Barley, Hailfinder, Win95pts, Formed and Pathfinder networks. Excluding experiments where datasets contain identical or single-valued variables means

that the algorithms have potential to achieve full stability, provided they are *truly* stable. Thus, all algorithms are compared across the same experiments, with the exception that FGES failed to complete within 3 hours for sample sizes of 10^4 and 10^5 for Hailfinder and Pathfinder - the mean of the F1s achieved by all the other algorithms is assumed for these cases when computing the mean F1 for FGES.

Tabu-Stable and HC-Stable are the only algorithms to produce a mean F1 S.D. of zero, by completely eliminating instability. FGES is found to be *almost* stable with a mean F1 S.D. of 0.0018. The other algorithms, including PC-Stable which has a mean F1 S.D. of 0.0367, all exhibit considerable instability. Moreover, Tabu-Stable also offers the highest mean F1 of 0.5341 - this value is higher than that quoted in Table 1 because the datasets not considered in this set of experiments, due to duplicate or single-valued variables, tend to be those with lowest sample sizes that also tend to produce lower F1 scores. FGES produces the second highest mean F1 of 0.5188, and would have been 0.5278 had its failure cases been ignored rather than being assigned an average F1 over the other algorithms. FGES provides the best Precision, 0.0129 better than Tabu-Stable, but the latter provides the best Recall, 0.0269 better than FGES.

6. Concluding Remarks

We present a new hill-climbing approach which eliminates instability as long as the datasets do not contain identical (or duplicate) variables. This is in contrast to many well-known algorithms that produce results that are sensitive to arbitrary artefacts of the dataset, typically the order in which columns appear in the dataset.

To achieve stability, an initial phase is used to first determine a stable node order based on the objective function scores of each node. This stable node ordering is then used in subsequent hill-climbing to orientate arcs in iterations where the two highest-scoring changes to the DAG are the addition of an arc with opposing orientations and which have the same score improvement. We describe two implementations that use this approach, HC-Stable and Tabu-Stable.

As well as being completely stable, these algorithms are found to increase the accuracy of the learnt graph considerably. HC-Stable improves mean F1 by around 16% over conventional HC, and Tabu-Stable by 10% over Tabu across a range of commonly-used networks. In particular, Tabu-Stable achieves higher accuracy than eight other well-known hybrid, constraint and score-based algorithms that we compare it to, whilst avoiding the instability that they all, including PC-Stable, demonstrate to some degree.

The methodology allows for alternative ways of determining a node order to be incorporated. For example, that of Behjati and Beigy (2020) which aims to generate the topological order of the true graph. Care would need to be taken that any such method does not use any dataset artefacts to determine part of the ordering, if the resulting algorithm is to be stable.We suggest that Tabu-Stable be considered for practical applications and in algorithm comparisons given its stability, performance, simplicity and robustness. More generally, stability aspects should be considered when designing and evaluating algorithms because instability can bias comparative benchmarks and practical results. Moreover, as we see here, investigating the causes of instability can inform the development of new algorithms.

References

M. Bartlett and J. Cussens. Integer linear programming for the bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017.

S. Behjati and H. Beigy. Improved k2 algorithm for bayesian network structure learning. *Engineering Applications of Artificial Intelligence*, 91:103617, 2020.

R. R. Bouckaert. *Bayesian belief networks: from construction to inference*. PhD thesis, University of Utrecht, 1995.

B. M. Broom, K.-A. Do, and D. Subramanian. Model averaging strategies for structure learning in bayesian networks with limited data. *BMC bioinformatics*, 13:1–18, 2012.

B. Cai, L. Huang, and M. Xie. Bayesian networks in fault diagnosis. *IEEE Transactions on industrial informatics*, 13(5):2227–2240, 2017.

D. M. Chickering. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002.

D. Colombo and M. H. Maathuis. Order-independent constraint-based causal structure learning. *Journal of Machine Learning Research*, 15:3921–3962, 2014.

A. Constantinou, N. K. Kitson, Y. Liu, K. Chobtham, A. H. Amirkhizi, P. A. Nanavati, R. Mbuvha, and B. Petrungaro. Open problems in causal structure learning: A case study of covid-19 in the uk. *Expert Systems with Applications*, 234:121069, 2023.

A. C. Constantinou, Y. Liu, K. Chobtham, Z. Guo, and N. K. Kitson. The Bayesys data and Bayesian network repository v1.5. http://bayesian-ai.eecs.qmul.ac.uk/bayesys/, 2024. Bayesian Artificial Intelligence research lab, Queen Mary University of London, London, UK.

G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

N. Friedman and D. Koller. Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks. *Machine learning*, 50:95–125, 2003.

M. Gasse, A. Aussem, and H. Elghazel. A hybrid algorithm for bayesian network structure learning with application to multi-label learning. *Expert Systems with Applications*, 41 (15):6755–6772, 2014.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243, 1995.

E. Herskovits. Kutato: An entropy-driven system for construction of probabilistic expert systems from databases. In *Proc. 6th International Conference on Uncertainty in Artificial Intelligence, Cambridge, MA, 1990*, pages 117–128, 1990.

N. K. Kitson and A. C. Constantinou. The impact of variable ordering on bayesian network structure learning. *Data Mining and Knowledge Discovery*, pages 1–25, 2024.

N. K. Kitson, A. C. Constantinou, Z. Guo, Y. Liu, and K. Chobtham. A survey of bayesian network structure learning. *Artificial Intelligence Review*, pages 1–94, 2023.

D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

E. Kyrimi, S. McLachlan, K. Dube, M. R. Neves, A. Fahmi, and N. Fenton. A comprehensive scoping review of bayesian networks in healthcare: Past, present and future. *Artificial Intelligence in Medicine*, 117:102108, 2021.

P. Larranaga, C. M. Kuijpers, R. H. Murga, and Y. Yurramendi. Learning bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE transactions on systems, man, and cybernetics-part A: systems and humans*, 26(4):487–493, 1996.

D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. *Advances in neural information processing systems*, 12, 1999.

J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

J. Pearl. *Causality*. Causality: Models, Reasoning, and Inference. Cambridge University Press, 2009. ISBN 9780521895606.

J. Peters, J. M. Mooij, D. Janzing, and B. Schölkopf. Causal discovery with continuous additive noise models. *The Journal of Machine Learning Research*, 15(1):2009–2053, 2014.

J. Ramsey, M. Glymour, R. Sanchez-Romero, and C. Glymour. A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images. *International journal of data science and analytics*, 3:121–129, 2017.

J. D. Ramsey, K. Zhang, M. Glymour, R. S. Romero, B. Huang, I. Ebert-Uphoff, S. Samarasinghe, E. A. Barnes, and C. Glymour. Tetrad—a toolbox for causal discovery. In *8th international workshop on climate informatics*, pages 1–4, 2018.

M. Scutari. bnlearn (Version 4.7) [Computer program], 2021a. https://cran.r-project.org/web/packages/bnlearn/index.html (downloaded: 17 December 2021).

M. Scutari. Bayesian Network Repository, 2021b. https://www.bnlearn.com/bnrepository/.

M. Scutari, C. E. Graafland, and J. M. Gutiérrez. Who learns better bayesian network structures: Accuracy and speed of structure learning algorithms. *International Journal of Approximate Reasoning*, 115:235–253, 2019.

P. Spirtes and C. Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72, 1991.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, prediction, and search*. MIT press, 2001.

J. Suzuki. Learning bayesian belief networks based on the minimum description length principle: basic properties. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 82(10):2237–2245, 1999.

I. Tsamardinos, C. F. Aliferis, and A. Statnikov. Time and sample efficient discovery of markov blankets and direct causal relations. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 673–678, 2003.

I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 65(1):31–78, 2006.

T. Verma and J. Pearl. Equivalence and synthesis of causal models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pages 255–270, 1990.

C. Vitolo, M. Scutari, M. Ghalaieny, A. Tucker, and A. Russell. Modeling air pollution, climate, and health data using bayesian networks: A case study of the english regions. *Earth and Space Science*, 5(4):76–88, 2018.

C. Yuan, B. Malone, and X. Wu. Learning optimal bayesian networks using a* search. In *Twenty-second international joint conference on artificial intelligence*, 2011.

X. Zheng, B. Aragam, P. K. Ravikumar, and E. P. Xing. Dags with no tears: Continuous optimization for structure learning. *Advances in neural information processing systems*, 31, 2018.

Appendix A. Properties of networks used in this study

This appendix provided details of the seventeen discrete variable networks used for evaluation in this study. Sports, Covid, Diarrhoea, Property and Formed are obtained from the Bayesys repository (Constantinou et al., 2024) and the remainder from the bnlearn repository (Scutari, 2021b)

Network	Number of variables	Number of arcs	Mean in-degree	Maximum in-degree	Mean degree	Maximum degree
asia	8	8	1	2	2	4
sports	9	15	1.67	2	3.33	7
sachs	11	17	1.55	3	3.09	7
covid	17	37	2.18	5	4.35	10
child	20	25	1.25	2	2.5	8
insurance	27	52	1.93	3	3.85	9
property	27	31	1.15	3	2.3	6
diarrhoea	28	68	2.43	8	4.86	17
water	32	66	2.06	5	4.12	8
mildew	35	46	1.31	3	2.63	5
alarm	37	46	1.24	4	2.49	6
barley	48	84	1.75	4	3.5	8
hailfinder	56	66	1.18	4	2.36	17
hepar2	70	123	1.76	6	3.51	19
win95pts	76	112	1.47	7	2.95	10
formed	88	138	1.57	6	3.14	11
pathfinder	109	195	1.79	5	3.58	106

Table 3: Properties of networks used in this study

Appendix B. Derivation of F1 used

This appendix explains how the F1 metric used to compare the learnt graph with the data-generating is derived because different authors use slightly different methodologies for calculating the arc differences which underlie the F1 metric. F1 is defined as

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

where Precision and Recall are defined as

$$Precision = \frac{TP}{TP + FP}, \; Recall = \frac{TP}{TP + FN} \tag{5}$$

and TP is the number of True Positives, FP is False Positives and FN is False Negatives. We follow the approach adopted in the bnlearn package (Scutari, 2021a) to compute these counts as shown in Table 4.

Learnt graph	Data-generating graph	True Positive (TP)	False positive (FP)	False Negative (FN)
$\longrightarrow$	$\longrightarrow$	1	0	0
—	—	1	0	0
$\longrightarrow$	no edge	0	1	0
—	no edge	0	1	0
no edge	$\longrightarrow$	0	0	1
no edge	—	0	0	1
$\longrightarrow$	$\longleftarrow$	0	1	1
$\longrightarrow$	—	0	1	1
—	$\longrightarrow$	0	1	1

Table 4: The contribution to the True Positive, False Positive, False Negative counts (and hence F1) and SHD resulting from different combinations of edges in the learnt and data-generating graph used in this study.