Enumerating Equivalence Classes of Bayesian Networks using EC Graphs

Eunice Yuh-Jie Chen and **Arthur Choi** and **Adnan Darwiche**
Computer Science Department
University of California, Los Angeles
{eyjchen,aychoi,darwiche}@cs.ucla.edu

Abstract

We consider the problem of learning Bayesian network structures from complete data. In particular, we consider the enumeration of their k-best equivalence classes. We propose a new search space for A* search, called the EC graph, that facilitates the enumeration of equivalence classes, by representing the space of completed, partially directed acyclic graphs. We also propose a canonization of this search space, called the EC tree, which further improves the efficiency of enumeration. Empirically, our approach is orders of magnitude more efficient than the state-of-the-art at enumerating equivalence classes.

1 INTRODUCTION

Learning the structure of a Bayesian network is a fundamental problem in machine learning and artificial intelligence. Historically, approximate methods, such as Markov Chain Monte Carlo (MCMC) and local search, were used for this task. In the past decade, there has been a surge in interest, in finding *optimal* Bayesian network structures, i.e., learning a single best directed acyclic graph (DAG) from a complete dataset; see, e.g., (Koivisto and Sood, 2004; Singh and Moore, 2005; Silander and Myllymäki, 2006; Jaakkola et al., 2010; Cussens, 2011; Yuan and Malone, 2013).

In some situations, learning a single optimal DAG is not sufficient—a single DAG is subject to noise and other idiosyncrasies in the data. As such, a data analyst would want to be aware of other likely DAGs. Hence, a number of algorithms have been proposed to *enumerate* the k-most likely DAGs from a complete dataset (Tian et al., 2010; Cussens et al., 2013; Chen and Tian, 2014; Chen et al., 2015). Such methods further facilitate approximate Bayesian model averaging (Tian et al., 2010; Chen and Tian, 2014).

Appearing in Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 51. Copyright 2016 by the authors.

There is a fundamental inefficiency in enumerating the k-most likely DAGs, namely that any given DAG may be Markov equivalent to many other DAGs, which are all equally expressive in terms of representing probability distributions. Thus, by enumerating DAGs, one may spend a significant amount of effort in enumerating redundant Bayesian networks. In this paper, we consider instead the enumeration of their equivalence classes, with each equivalence class representing a potentially large (even exponential) number of DAGs, which we show can be the case in practice empirically.

In this paper, we propose a new approach to enumerating equivalence classes that is in practice *orders of magnitude* more efficient than the existing state-of-the-art, which is based on dynamic programming (Chen and Tian, 2014). Our approach is instead based on a framework proposed by Chen et al. (2015), which provides a general approach to a variety of structure learning tasks, such as enumerating the k-best DAGs. This approach is based on navigating an expressive yet seemingly intractable search space, called the BN graph, which represents the space of all DAGs. Chen et al. show that the complexity of the BN graph can be mitigated by exploiting an oracle for optimal structure learning, which in turn can be used to tackle even more computationally challenging tasks (such as enumerating the k-best DAGs).

Here, we propose a specific instance of this framework, where we *specialize* the BN graph to a more compact search space over equivalence classes. In particular, we represent equivalence classes (ECs) using completed partially directed acyclic graphs (CPDAGs) (Chickering, 2002), leading to a new search space called the EC graph. We further propose a *canonization* of the EC graph, leading to the EC tree, whose properties can be exploited by heuristic search methods such as A*, leading to improved time and space efficiency.

This paper is organized as follows. Section 2 reviews Markov equivalence and related concepts. In Section 3, we first review search spaces for learning Bayesian networks, and then propose the EC graph. Section 4 discusses the enumeration of equivalence classes. In Section 5, we canonize the EC graph to the EC tree. We evaluate our approach empirically in Section 6, and conclude in Section 7.

2 TECHNICAL PRELIMINARIES

The structure of a Bayesian network (BN) is given by a DAG, and two given DAGs are considered Markov equivalent iff they encode the same conditional independencies, and consequently, represent the same class of probability distributions. For example, the following three DAGs are Markov equivalent:

$X_1 \rightarrow X_2 \rightarrow X_3$	$X_1 \leftarrow X_2 \leftarrow X_3$	$X_1 \leftarrow X_2 \rightarrow X_3$

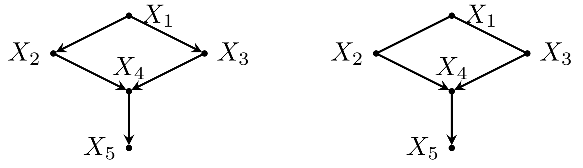
Markov equivalence can be characterized by a graphical criterion, based on the structure of a DAG. First, the *skeleton* of a DAG is the undirected graph found by ignoring the orientation of the edges. Second, a *v-structure* in a DAG is a set of three nodes X, Y, Z with edges $X \rightarrow Y \leftarrow Z$, but with no edge between X and Z. The following theorem characterizes Markov equivalent DAGs.

Theorem 1 (Verma and Pearl, 1990) *Two DAGs are Markov equivalent iff they have the same skeleton and the same v-structures.*

A set of Markov equivalent DAGs can be summarized by a *partially directed acyclic graph* (PDAG), which is a graph that contains both directed and undirected edges, but with no directed cycles; see, e.g., Chickering (2002). Given a PDAG P, we can induce a set of Markov equivalent DAGs by directing the undirected edges of a PDAG, but as long as we introduce no directed cycles and no new v-structures. We use $\mathsf{class}(P)$ to denote this set of Markov equivalent DAGs.

In an equivalence class of DAGs, each edge of their common skeleton can be classified as *compelled*, or *reversible*. An edge connecting X and Y is compelled to a direction $X \rightarrow Y$ if *every* DAG in the equivalence class has the directed edge $X \rightarrow Y$. Otherwise, an edge is reversible, and there exists a DAG in the equivalence class with edge $X \rightarrow Y$, and another DAG with edge $Y \rightarrow X$. In a PDAG, if all compelled edges are directed in the appropriate orientation, and all reversible edges are left undirected, then we obtain a *completed PDAG* (CPDAG).[1] A CPDAG uniquely characterizes an equivalence class of DAGs (Chickering, 2002). That is, there is a one-to-one correspondence between CPDAGs and equivalence classes. Further, a given CPDAG represents all DAGs, and only those DAGs, of a given equivalence class.

As an example, the CPDAG $X_1 - X_2 - X_3$ represents the Markov equivalence class for the three DAGs given in our example from the start of this section. We provide another example below, of a DAG (left), and its corresponding CPDAG (right).



3 SEARCH SPACES OVER BNs

Given a dataset $\mathcal{D}$ over variables $\mathbf{X}$, we want to learn the DAG G of a Bayesian network, but one that minimizes a score that decomposes over families $X\mathbf{U}$ (where X is a variable with parents $\mathbf{U}$):

$$\mathsf{score}(G \mid \mathcal{D}) = \textstyle\sum_{X\mathbf{U}} \mathsf{score}(X\mathbf{U} \mid \mathcal{D}). \qquad (1)$$

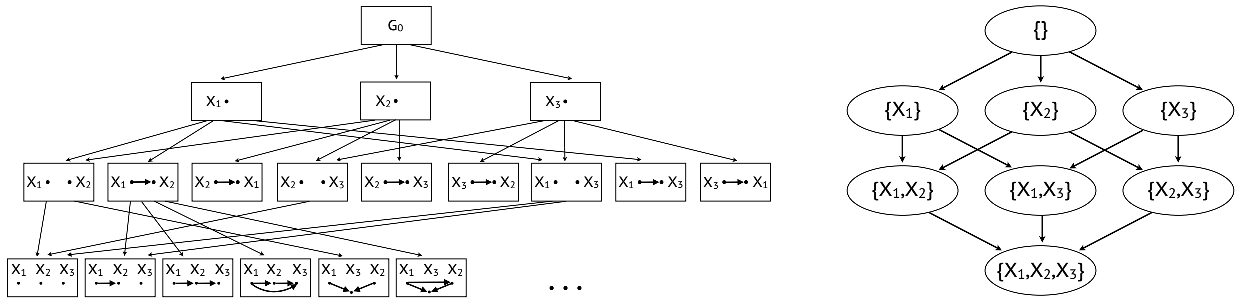
For example, MDL and BDeu scores decompose over families (possibly negated, to obtain a minimization problem). See, e.g., Darwiche (2009); Koller and Friedman (2009); Murphy (2012). MDL and BDeu scores are *score equivalent*: they assign the same score to two Markov equivalent DAGs (Chickering, 1995).

3.1 BN Graphs

Previously, Chen et al. (2015) proposed the *BN graph*, a search space over DAGs, for learning Bayesian networks from data. The BN graph was shown to be an effective framework for certain structure learning tasks, such as enumerating the k-best DAGs, and for learning with certain types of non-decomposable scores.

Figure 1(a) illustrates a BN graph over 3 variables. In this graph, each node represents a DAG over some subset of the variables $\mathbf{X}$. A directed edge $G_i \xrightarrow{X\mathbf{U}} G_j$ from a DAG G_i to a DAG G_j exists iff G_j can be obtained from G_i by introducing variable X as a leaf node with parents $\mathbf{U}$. Thus, the BN graph is a layered graph, where each layer adds one more leaf to a DAG when we walk an edge from one layer to the next. Hence, when we refer to a DAG G_i, we assume it is on the i-th layer, i.e., G_i has i nodes. The top (0-th)

[1]CPDAGs are also sometimes referred to as *essential graphs* or *maximally oriented graphs*.



(a) A BN graph. (b) An order graph.

Figure 1: Bayesian network search spaces for the set of variables $\mathbf{X} = \{X_1, X_2, X_3\}$.

layer contains the root of the BN graph, a DAG with no nodes, which we denote by G_0. The bottom (n-th) layer contains DAGs G_n over our n variables $\mathbf{X}$. A path $G_0 \xrightarrow{X_1\mathbf{U}_1} \cdots \xrightarrow{X_n\mathbf{U}_n} G_n$ from the root to a DAG G_n on the bottom layer, is a construction of the DAG G_n, where each edge $G_{i-1} \xrightarrow{X_i\mathbf{U}_i} G_i$ adds a new leaf X_i with parents $\mathbf{U}_i$. Moreover, each path corresponds to a unique ordering $\langle X_1, \ldots, X_n \rangle$ of the variables.

For example, consider the BN graph of Figure 1(a) and the following path, i.e., sequence of DAGs:

G_0	G_1	G_2	G_3
	X_1	$X_1 \to X_2$	$X_1 \to X_2 \quad X_3$

Starting with the empty DAG G_0, we add a leaf X_1 (with no parents), then a leaf X_2 (with parent X_1), then a leaf X_3 (with no parents), giving us a DAG G_3 over all 3 variables.

If each edge $G_{i-1} \xrightarrow{X_i\mathbf{U}_i} G_i$ is associated with a cost $\mathsf{score}(X_i\mathbf{U}_i \mid \mathcal{D})$, then the cost of a path from the root G_0 to a goal G_n gives us the score of the DAG,

$$\mathsf{score}(G_n \mid \mathcal{D}) = \textstyle\sum_{i=1}^{n} \mathsf{score}(X_i\mathbf{U}_i \mid \mathcal{D})$$

as in Equation 1. Hence, in the BN graph, the DAG G_n with the shortest path from the root G_0 to itself, is an optimal DAG with the lowest cost.

The BN graph has $O(n! \cdot 2^{\binom{n}{2}})$ nodes, leading to a tremendously large and seemingly intractable search space. Classically, only approximate methods such as MCMC and greedy local search were used to navigate search spaces over DAGs. Recently, Chen et al. (2015) showed that a search space as large as the BN graph can in fact be efficiently navigated, by leveraging advances in optimal Bayesian network structure learning.

3.2 EC Graphs

We now propose a new search space for Bayesian network structures, but more specifically, for their equivalence classes. This is a more compact search space, called the *EC graph*, where each node now represents an *equivalence class* of DAGs.

In an EC graph, each node represents a CPDAG P, which denotes a set of Markov equivalent DAGs G. Intuitively, we can obtain an EC graph by aggregating the Markov equivalent DAGs of a BN graph into a single node, and labeling the resulting node with the corresponding CPDAG P.

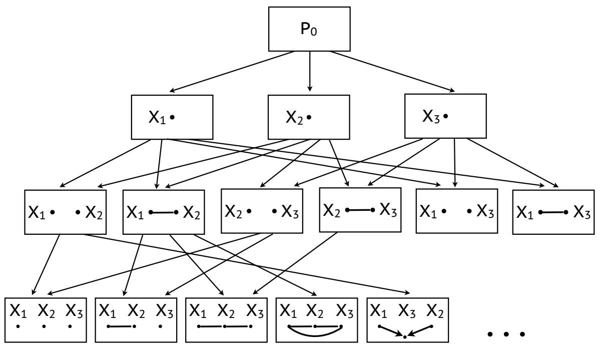
Recall that in a BN graph, a directed edge $G_i \xrightarrow{X\mathbf{U}} G_j$ indicates that DAG G_j can be obtained from DAG G_i by adding to G_i a leaf node X with parents $\mathbf{U}$. In an EC graph, we have a corresponding directed edge $P_i \xrightarrow{X\mathbf{U}} P_j$. Here, the CPDAG P_i represents the equivalence class $\mathsf{class}(P_i)$, containing DAGs G_i. We can view the edge as adding a leaf node X with parents $\mathbf{U}$ to *each* of the DAGs $G_i \in \mathsf{class}(P_i)$. First, we observe that any of the resulting DAGs G_j must belong to the same equivalence class P_j.

Proposition 1 *Let P_i denote a CPDAG and let G_i denote a DAG in $\mathsf{class}(P_i)$. If we add a new leaf X_i with parents $\mathbf{U}_i$ to the DAGs G_i, the resulting DAGs G_j belong to the same equivalence class P_j.*

Figure 2(a) illustrates an EC graph over 3 variables. Consider, as an example, the CPDAG $X_1 - X_2$, which corresponds to the Markov equivalent DAGs $X_1 \to X_2$ and $X_1 \leftarrow X_2$. Adding a new leaf X_3 with parent X_2, we obtain $X_1 \to X_2 \to X_3$ and $X_1 \leftarrow X_2 \to X_3$, with CPDAG $X_1 - X_2 - X_3$. We remark that there is a third DAG $X_1 \leftarrow X_2 \leftarrow X_3$ in this equivalence class, which we did not obtain here since X_3 is not a leaf. This DAG is obtained on a different path of the EC graph, where we add X_1 as a leaf to the CPDAG $X_2 - X_3$.

Proposition 2 *For a given CPDAG P, and any DAG $G \in \mathsf{class}(P)$, there exists a path that constructs G from root P_0 to node P in the EC graph.*

We remark that when we traverse an edge $P_i \xrightarrow{X\mathbf{U}} P_j$, we add a new leaf to the DAGs of $\mathsf{class}(P_i)$, yet the new


(a) An EC graph.

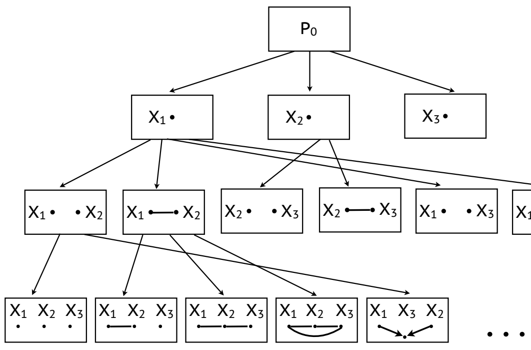

(b) An EC tree.

Figure 2: EC search spaces for the set of variables $\{X_1, X_2, X_3\}$.

edges in the resulting CPDAG P_j may be directed or undirected. Thus, to traverse the EC graph, we need to a way to orient the new edges from parents $\mathbf{U}$ to leaf X in CPDAG P_j. Previously, Chickering (1995) proposed a polytime algorithm that, given a DAG G, finds the corresponding CPDAG P. We next observe that Chickering's algorithm can be run *incrementally*, by labeling only the new edges from $\mathbf{U}$ to X.

Proposition 3 *Consider a DAG G_i, its CPDAG P_i, and a DAG G_j obtained by adding a new leaf X to DAG G_i, with parents $\mathbf{U}$. The CPDAG P_j for G_j can be obtained locally from P_i, by applying Algorithm 1.*

Given a DAG G, Chickering's original algorithm traverses the nodes of a DAG G, in topological order, and labels the edges incoming a node as either compelled or reversible. Hence, running the same algorithm on a DAG G_j will first obtain the sub-CPDAG P_i. The edges incoming to the new leaf X can then be labeled by running an additional iteration of Chickering's algorithm, which is given by Algorithm 1.

As in the BN graph, each edge $P_{i-1} \xrightarrow{X_i \mathbf{U}_i} P_i$ is associated with a cost, $\mathsf{score}(X_i \mathbf{U}_i \mid \mathcal{D})$. For metrics that are *score equivalent*, the cost of a path from the empty CPDAG P_0 to a CPDAG P_n gives us the score of the corresponding DAGs G_n in $\mathsf{class}(P_n)$,

$$\mathsf{score}(G_n \mid \mathcal{D}) = \textstyle\sum_{i=1}^{n} \mathsf{score}(X_i \mathbf{U}_i \mid \mathcal{D}).$$

as in Equation 1. Hence, the CPDAG P_n that has the shortest path from the root P_0 is an optimal equivalence class, whose DAGs have the lowest cost.

Algorithm 1: LabelEdges($P, X\mathbf{U}$)

```
Data: CPDAG $P$, new variable $X$ with parents $\mathbf{U}$ in $P$
Result: Label edges from $X$ to $\mathbf{U}$ as compelled/reversible
begin
    label each edge between $X$ and $\mathbf{U}$ as unknown
    let $G$ be any DAG in $\mathsf{class}(P)$
    while there exists an unknown edge do
        let $X' - X$ be the unknown edge with the greatest $X'$ in a topological ordering of $G$
        foreach $X'' \to X' \in P$ do
            if $X'' \notin \mathbf{U}$ then label all unknown $Y - X$ incident to $X$ as compelled to $X$
            else label $X'' - X$ as compelled to $X$
        if $\exists Z \in \mathbf{U}$ s.t. $Z \to X' \notin G$ then label all unknown $Y - X$ incident to $X$ as compelled to $X$
        else label all unknown $Y - X$ incident to $X$ as reversible
```

3.3 Order Graphs

Yuan and Malone (2013) formulate the structure learning problem as a shortest-path problem on a graph called the *order graph*. Figure 1(b) illustrates an order graph over 3 variables $\mathbf{X}$. In an order graph, each node represents a subset $\mathbf{Y}$ of the variables $\mathbf{X}$. There is a directed edge from $\mathbf{Y}$ to $\mathbf{Z}$ in the order graph iff we add a new variable X to the set $\mathbf{Y}$, to obtain the set $\mathbf{Z}$. We can view the order graph as another compressed form of the BN graph (and the EC graph): if we aggregate all DAGs G over the same subset of nodes $\mathbf{Y}$ in the BN graph, we obtain an order graph.

The principle advantage of the order graph is its size: there are only 2^n nodes in the order graph (which is much smaller than the BN graph). However, certain learning tasks, such as enumerating the k-best Bayesian network structures, can be orders-of-magnitude more effective on the BN graph than on the order graph, as demonstrated by (Chen et al., 2015).

4 ENUMERATION WITH A*

Previously, Chen et al. (2015) demonstrated that enumerating the k-best Bayesian network structures can be done effectively using A* search on the BN graph. This approach was shown to be orders-of-magnitude more efficient than the existing state-of-the-art, which included dynamic programming algorithms based on the order graph (Tian et al., 2010), and methods based on integer linear programming (Cussens et al., 2013). Here, we propose to enumerate equivalence classes of Bayesian network structures, also using A* search, but now on the EC graph.

A* search is a best-first search, for finding shortest paths on a graph. It uses an *evaluation* function f to guide the search process, where we expand first those nodes with the lowest f cost (Hart et al., 1968). Intuitively, nodes with lower f cost are more promising nodes to explore. In an EC graph, the evaluation function for A* takes the form: $f(P) = g(P)+h(P)$, where P is a given CPDAG. Further, function g is the *path cost*, i.e., the cost of the path to reach P from the root node P_0. Function h is the *heuristic function*, which estimates the cost to reach a goal, starting from P. If our heuristic function h is *admissible*, i.e., it never over-estimates the cost to reach a goal, then A* search is optimal. That is, the first goal node P_n that A* expands is the one that has the shortest path from the root P_0. In general, A* search is more efficient, when given a heuristic function h that can more accurately estimate the cost to a goal state.

We are particularly interested in *perfect* heuristic functions $h(P)$, which can predict the optimal path from a given node P to a goal node P_n. In particular, A* search with a perfect heuristic offers a simple approach to enumerating shortest paths. In this case, A* can immediately find the first optimal solution with a perfect heuristic; indeed, it marches straight to a goal node (with appropriate tie-breaking, where we expand the deepest node first). The next best solution can be found by simply continuing A* search. Once we have exhausted all optimal CPDAGs (if more than one exists), a perfect heuristic is no longer perfect, with respect to the next best CPDAGs. Our heuristic is still admissible, however, as it still lower-bounds the cost of a path from a node P to a goal node P_n—it may just report a cost for a goal node that was already enumerated (and hence has a lower cost).

Indeed, this was the strategy underlying the approach proposed by Chen et al. (2015), to enumerate the k-best DAGs, in the BN graph. More abstractly, we can view this strategy as one that first assumes an oracle (the perfect heuristic) that can solve an NP-complete problem, i.e., finding the single best DAG Chickering et al. (2004). We can then use this oracle to find the k-th best DAG, which appears, fundamentally, to be a much more difficult problem. For example, identifying the k-th best most probable explanation (MPE) in a Bayesian network is FP^{PP}-complete (Kwisthout et al., 2011), whereas the MPE problem itself is only NP-complete (Shimony, 1994).

Algorithm 2: $\text{ValidEdge}(P_{i-1}, X_i\mathbf{U}_i)$

```
Data: CPDAG $P_{i-1}$ and candidate family $X_i\mathbf{U}_i$
Result: true if $P_{i-1} \xrightarrow{X_i\mathbf{U}_i} P_i$ is valid, false otherwise
begin
    let $P_i$ be the CPDAG obtained by appending $X_i$ to $P_{i-1}$ (using Algorithm 1)
    foreach node $X_k$ in $P_i$ where $k > i$ do
        if there exists compelled edge $Y \leftarrow X_k$ then
            continue
        let $\mathbf{S} = \{Y \mid Y - X_k$ is reversible in $P\}$
        if variables in $\mathbf{S}$ form a clique then
            return false
    return true
```

As in Chen et al. (2015), we assume an oracle that can find the single-best DAG, but now to find the k-th best CPDAG. For this purpose, any learning system could be used as such, provided that it can accept a DAG G, and find an optimal DAG G_n that extends it. Systems such as URLearning meet this criterion Yuan and Malone (2013), which we use in our subsequent experiments. See also Dechter et al. (2012), for more on using A* search for k-best enumeration.[2]

4.1 Implementation

In our framework, the heuristic function evaluations (made through our oracle) are relatively expensive operations. Hence, we cache heuristic values (some entries can even be primed, by inferring them from other evaluations). Further, we use partial-expansion A* to delay the generation of children (which does not sacrifice the optimality of A* search); see, e.g., Yoshizumi et al. (2000); Felner et al. (2012). In particular, we only generate those children whose heuristic values fall within a certain threshold; this threshold is then increased if a node needs to be re-expanded.

5 EC TREES

For heuristic search methods such as A* search, the structure of the search space has a significant impact on the efficiency of search. Consider the EC graph in particular. A CPDAG node P can be reached from the

[2]We remark that Dechter et al. (2012) is more specifically concerned with the enumeration of the k-shortest paths. Since we are interested in enumerating the k-closest goal nodes, we remark that some, but not all, of their theoretical analyses applies to our problem. In particular, each distinct goal node in the EC graph may have many paths that can reach it. Hence, once we obtain one goal node, many more shortest-paths may be needed to obtain the next closest (and distinct) goal node.

root P_0 via possibly many paths. Further, each path has the same cost (Markov equivalent DAGs have the same score). Thus, after we visit a node for the first time, we do not care about other paths that reach the same node. This redundancy introduces significant memory and computational overheads to A* search. Thus, we propose a *canonization* of the EC graph, that ensures that every CPDAG node P can be reached by a unique, canonical path. Here, each node has a single parent, and hence, we obtain a search tree.[3] Hence, we refer to the canonized space as the *EC tree.* Figure 2(b) depicts an EC tree over 3 variables.

Consider any path $P_0 \xrightarrow{X_1\mathbf{U}_1} \cdots \xrightarrow{X_i\mathbf{U}_i} P_i$ from the root node P_0 to a node P_i, in the EC graph, which corresponds to an ordering of nodes, $\pi_i = \langle X_1, \ldots, X_i \rangle$. To define a canonical path from P_0 to P_i, it thus suffices to define a canonical ordering of the variables of P_i. In turn, to define an EC tree from an EC graph, it suffices to (1) associate each CPDAG node P with a canonical ordering π, and (2) show which edges of the EC graph remain in the EC tree, with respect to the canonical orderings.

First, let us define a canonical ordering for a given DAG G: let us use the largest topological ordering that is consistent with a given DAG G, but in reverse lexicographic order (where the right-most element in an order is the most significant).[4] We can construct such an order, from right-to-left, by iteratively removing the leaf with the largest index. For example, the DAG $X_1 \leftarrow X_2 \rightarrow X_3$ has two topological orderings: $\pi_a = \langle X_2, X_1, X_3 \rangle$ and $\pi_b = \langle X_2, X_3, X_1 \rangle$, where π_a is larger in reverse lexicographic order (we remove X_3 first, then X_1, and then X_2).

We now define a canonical ordering for a CPDAG P: let us use the largest canonical ordering of its Markov equivalent DAGs $G \in \mathsf{class}(P)$. Consider the CPDAG $X_1 - X_2 - X_3$, and its Markov equivalent DAGs:

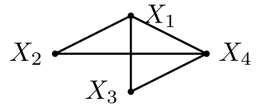
$X_1 \rightarrow X_2 \rightarrow X_3$	$X_1 \leftarrow X_2 \leftarrow X_3$	$X_1 \leftarrow X_2 \rightarrow X_3$

with the canonical orderings: $\pi_a = \langle X_1, X_2, X_3 \rangle$, $\pi_b = \langle X_3, X_2, X_1 \rangle$, and $\pi_c = \langle X_2, X_1, X_3 \rangle$. Among these DAGs, ordering π_a is the largest, and is thus the canonical ordering of CPDAG $X_1 - X_2 - X_3$.

Given a CPDAG P, we can construct its canonical ordering π, again from right-to-left, by iteratively removing the largest leaf among the Markov equivalent DAGs in $\mathsf{class}(P)$. As for obtaining the structure of the EC tree, this iterative process provides a local condition for determining whether an edge $P_{i-1} \xrightarrow{X_i\mathbf{U}_i} P_i$ belongs in the EC tree. That is, variable X_i must be the largest leaf among the Markov equivalent DAGs in $\mathsf{class}(P_i)$. This is summarized by the following result.

Proposition 4 *Let π_{i-1} be the canonical ordering of CPDAG P_{i-1}, and let P_i be the CPDAG found by adding leaf X_i with parents $\mathbf{U}_i$ to the DAGs $G_{i-1} \in \mathsf{class}(P_{i-1})$. In this case, $\pi_i = \langle \pi_{i-1}, X_i \rangle$ is the canonical ordering of P_i iff X_i has the largest index among all leaves in DAGs $G_i \in \mathsf{class}(P_i)$.*

It remains to show how to identify, for a given CPDAG P, the largest leaf among the Markov equivalent DAGs in $\mathsf{class}(P)$. Consider the following CPDAG:



We note that P cannot be obtained by appending X_4 as a leaf. If it could, then the resulting DAG would have a new v-structure $X_2 \rightarrow X_4 \leftarrow X_3$, since there is no edge connecting X_2 and X_3 (i.e., such a DAG would not belong in the equivalence class of P). As we cannot append X_4 as a leaf, it cannot be the last variable of any topological ordering of a DAG in $\mathsf{class}(P)$. However, there is a DAG in $\mathsf{class}(P)$ where X_3 is a leaf. The canonical ordering for P thus mentions X_3 last.

For a given CPDAG P_i, the following theorem allows us to enumerate all leaves among the DAGs in $\mathsf{class}(P_i)$, allowing us to easily test whether a node X appears as the largest leaf in some DAG of a given CPDAG P. Algorithm 2 further provides a polytime procedure for this test.

Theorem 2 *Consider a CPDAG P and variable X, with no compelled edges directed away from X. Let set $\mathbf{S}$ be the nodes adjacent to X through a reversible edge. In this case, there exists a DAG $G \in \mathsf{class}(P)$ where X is a leaf iff nodes $\mathbf{S}$ form a clique in P.*

A proof appears in the supplementary Appendix.

Finally, we remark that the only difference between the EC tree and the EC graph is that each node in the EC tree can be reached through exactly one path, compared to multiple paths in the EC graph. This distinction results in memory and computational savings for A* search, as discussed earlier. Otherwise, A* search in the EC tree proceeds in the same manner as in the EC graph. In particular, a heuristic function is admissible in the EC tree iff it is admissible in the EC graph. Thus, to navigate the EC tree, we use the same heuristic function as discussed in Section 4.1.

[3]In more technical terms, the savings that we obtain are: (1) duplicate detection is no longer needed (i.e., the closed list), and (2) fewer edges in the search space implies fewer heuristic function evaluations.

[4]Here, we assume comparisons are made based on the natural ordering of variables, i.e., by index.

benchmark			10-best				100-best				1,000-best			
			EC tree		KBestEC		EC tree		KBestEC		EC tree		KBestEC	
name	n	N	t	m	t	m	t	m	t	m	t	m	t	m
wine	14	178	0.05	1	15.35	2	0.15	1	270.14	2	0.86	1	5569.34	4
letter	17	20,000	18.11	1	120.26	2	48.31	1	2559.38	2	81.72	1	$\times_t$	
voting	17	435	1.89	1	141.66	2	2.11	1	3289.75	2	6.67	1	$\times_t$	
zoo	17	101	2.89	1	139.37	2	3.59	1	3206.18	2	6.03	1	$\times_t$	
statlog	19	752	29.28	1	618.73	2	41.99	1	$\times_t$		43.89	1	$\times_t$	
hepatitis	20	126	36.33	1	1328.27	2	63.37	1	$\times_t$		101.05	2	$\times_t$	
imports	22	205	174.84	4	$\times_s$		223.78	4	$\times_s$		224.11	4	$\times_s$	
parkinsons	23	195	897.81	8	$\times_t$		897.97	8	$\times_t$		898.68	8	$\times_t$	

Table 1: Time t (in seconds) and memory m (in GBs) used by EC tree and KBestEC. $\times_t$ denotes an out-of-time (2hr), and $\times_s$ denotes a segmentation fault. n is the number of variables in the dataset, and N is the size of the dataset.

benchmark		10-best				100-best				1,000-best			
name	n	gen.	exp.	re-exp.	invoke	gen.	exp.	re-exp.	invoke	gen.	exp.	re-exp.	invoke
adult	14	243	220	630	67	2045	1865	12355	265	15072	13574	163462	465
wine	14	2205	1874	0	39	12691	10256	10156	206	68403	56612	111224	358
nltcs	16	252	249	2151	287	1407	1389	27069	689	11431	11193	356755	1363
letter	17	377	377	2936	324	2705	2704	59915	1562	16057	15979	659076	3373
msnbc	17	555	555	5450	1147	969	965	22490	1269	2662	2561	78050	1313
voting	17	1617	1419	0	147	15971	11613	23026	413	114498	106378	421512	1402
zoo	17	192	151	0	166	1978	1029	929	377	9330	8812	7812	864
statlog	19	393	369	0	212	3379	3065	2965	444	26698	24063	46126	685
hepatitis	20	2431	2281	0	667	17875	15852	15752	2191	121022	99956	197912	6423
imports	22	666	270	0	103	4181	2034	0	244	21544	14064	0	259
parkinsons	23	836	524	0	214	4222	2616	0	237	26390	17134	0	318

Table 2: EC tree: number of nodes (1) generated, (2) expanded, (3) re-expanded, and (4) oracle invocations.

6 EXPERIMENTS

We compare our approach with the recently proposed algorithm for finding the k-best equivalence classes of Bayesian networks, called KBestEC,[5] based on dynamic programming (DP) (Chen and Tian, 2014).

Our experiments were performed on a 2.67GHz Intel Xeon X5650 CPU. We use real-world datasets from the UCI ML Repository (Bache and Lichman, 2013),[6] and assume BDeu scores with an equivalent sample size of 1. We adapt the URLearning structure learning package of Yuan and Malone (2013) and Fan et al. (2014),[7] to serve as our oracle for learning a single optimal DAG, as in Section 4. We pre-compute the scores of candidate parent sets, which we provide as input into each system we evaluate. All timing results are averages over 10 runs.

For each approach, we enumerate the 10-best, 100-best and 1,000-best CPDAGs, with a 2 hour limit on running time. To analyze memory usage, we incrementally increased the amount of memory available to each system (from 1GB, 2GB, 4GB, to 8GB), and recorded the smallest limit that allowed each system to finish. For our A* search, we set the threshold for partial-expansion to the value of the optimal BN plus 1, which increases by 1 for re-expansions.

Table 1 summarizes our results for A* search on the EC tree, and for the DP-based approach of KBestEC. First, we observe that on instances where both A* search and KBestEC are successful, A* search is consistently more efficient, both in terms of computation time and in memory usage. In terms of time, A* search can be *orders of magnitude* more efficient: for example, in the zoo dataset, A* search is over 893 times faster than KBestEC. We further observe that A* search is capable of scaling to larger networks, and to larger values of k. In fact, KBestEC appears to scale super-linearly with k, but A* search appears to scale *sub-linearly* with respect to k. This trend can in part be attributed to the more exhaustive nature of DP, which maintains all partial solutions that can potentially be completed to a k-th best solution.

To gain more insight about the computational nature of A* search on the EC tree, consider Tables 3 & 2, which includes 3 additional datasets, adult, nltcs, and msnbc (we were unable to generate score files for these datasets using KBestEC). In Table 3, we consider the

[5]Open-source, available at `http://web.cs.iastate.edu/~jtian/Software/AAAI-14-yetian/KBestEC.htm`

[6]Discretized, and available at `http://urlearning.org/`

[7]Open-source, available at `http://urlearning.org/`. To share computations among repeated invocations of URLearning, we also cache some intermediate results.

benchmark		10-best		100-best		$1,000$-best	
name	n	T_h	T_{A*}	T_h	T_{A*}	T_h	T_{A*}
adult	14	0.25	0.01	0.49	0.05	0.63	0.56
wine	14	0.03	0.02	0.05	0.10	0.09	0.77
nltcs	16	3.35	0.01	5.44	0.12	8.08	1.50
letter	17	18.07	0.04	47.74	0.57	75.54	6.18
msnbc	17	145.64	0.07	152.87	0.18	154.61	0.46
voting	17	1.88	0.01	1.92	0.19	4.16	2.51
zoo	17	2.88	0.01	3.55	0.04	5.83	0.20
statlog	19	29.27	0.01	41.94	0.05	43.49	0.40
hepatitis	20	36.24	0.09	62.79	0.58	96.82	4.23
imports	22	174.82	0.02	223.71	0.07	223.81	0.30
parkinsons	23	897.74	0.07	897.82	0.15	898.25	0.43

Table 3: Time T_h to compute the heuristic function, and time T_{A*} spent in A* search (in seconds).

benchmark	n	N	10-best	100-best	$1,000$-best
adult	14	30,162	68	1,399	15,572
wine	14	178	60	448	4,142
nltcs	16	16,181	3,324	27,798	248,476
letter	17	20,000	884	15,796	569,429
msnbc	17	291,326	231,840	1,720,560	16,921,080
voting	17	435	30	413	3,671
zoo	17	101	52	377	5,464
statlog	19	752	44	444	4,403
hepatitis	20	126	89	892	8,919
imports	22	205	12	136	1,493
parkinsons	23	195	132	476	3,444

Table 4: Number of DAGs in the k-best equivalent classes.

amount of time T_h spent in evaluating the heuristic function (i.e., invoking our oracle), and the time T_{A*} spent traversing the EC tree in A* search (where total time spent is $t = T_h + T_{A*}$, as reported in Table 1). Table 2 further reports the number of nodes generated (i.e., inserted into the open list), expanded, and re-expanded (by partial-expansion) in A* search. We also report the number of oracle invocations. First, observe that A* search spends almost all of its time in evaluating the heuristic function, which we already know is relatively expensive. Next, observe that the the number of nodes generated is relatively low. This suggests that the oracle is powerful enough to efficiently navigate the search space of the EC tree. Further, we observe that the number of oracle invocations is also low, which is further minimized by caching and inferring heuristic values, as discussed in Section 4.1.

We further count the equivalent number of DAGs represented by the k-best CPDAGs, in Table 4. Previously, Gillispie and Perlman (2001) observed that when the number of variables is small (not greater than 10), a CPDAG represents *on average* 3.7 DAGs. Here, we observe that for a moderate number of variables, a CPDAG may represent a much larger number of DAGs. That is, when we learn equivalence classes, the data may prefer CPDAGs with many reversible

benchmark			$1,000$-best EC			
			EC tree		BN graph	
name	n	N	t	m	t	m
adult	14	30,162	1.19	1	11.87	1
wine	14	178	0.86	1	3.89	1
nltcs	16	16,181	9.58	1	1,126.05	4
letter	17	20,000	81.72	1	4,666.29	4
msnbc	17	291,326	155.07	1	$\times_t$	
voting	17	435	6.67	1	17.89	1
zoo	17	101	6.03	1	10.34	1
statlog	19	752	43.89	1	76.99	1
hepatitis	20	126	101.05	2	284.46	4
imports	22	205	224.11	4	604.14	8
parkinsons	23	195	898.68	8	1,450.46	16

Table 5: Time t (in seconds) and memory m (in GBs) used by EC tree and BN graph. n is the number of variables in the dataset, and N is the size of the dataset. A $\times_t$ corresponds to an out-of-time (2hr).

edges (deviating from the average case). For example, the larger datasets (larger N) tend to have equivalence classes that contain many more DAGs. When we compare the enumeration of the $1,000$-best equivalence classes, with the enumeration of an equivalent number of DAGs, using the system of Chen et al. (2015),[8] we can again see orders-of-magnitude improvements in efficiency; see Table 5. In the supplementary Appendix, we observe similar gains by the EC tree, compared to the BN graph, in terms of nodes explored by A* search, as would be expected (EC trees have smaller search spaces).

7 CONCLUSION

In this paper, we propose an approach for enumerating the k-best equivalence classes of Bayesian networks, from data. Our approach is an instance of a more general framework for the optimal structure learning of Bayesian networks, given by Chen et al. (2015). In particular, we specialize the search space of Bayesian network structures, to a search space of their equivalence classes. We further identify a canonization of this search space, to improve the efficiency of search further. Empirically, our approach is orders-of-magnitude more efficient than the state-of-the-art, in the task of enumerating equivalence classes.

Acknowledgments

We thank Joseph Barker, Brandon Malone, and Ethan Schreiber for helpful comments and discussions. This work has been partially supported by NSF grant #IIS-1514253 and ONR grant #N00014-15-1-2339.

[8]More specifically, we used an updated and better optimized system for the BN graph, based on Chen et al. (2015), which further outperforms their reported results.

References

K. Bache and M. Lichman. UCI Machine Learning Repository, 2013. URL `http://archive.ics.uci.edu/ml`.

E. Y.-J. Chen, A. Choi, and A. Darwiche. Learning optimal Bayesian networks with DAG graphs. In *Proceedings of the 4th IJCAI Workshop on Graph Structures for Knowledge Representation and Reasoning (GKR'15)*, 2015.

Y. Chen and J. Tian. Finding the k-best equivalence classes of Bayesian network structures for model averaging. In *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence*, 2014.

D. M. Chickering. A transformational characterization of equivalent Bayesian network structures. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 87–98, 1995.

D. M. Chickering. Learning equivalence classes of Bayesian network structures. *Journal of Machine Learning Research*, 2:445–498, 2002.

D. M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is np-hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

J. Cussens. Bayesian network learning with cutting planes. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 153–160, 2011.

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

A. Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

R. Dechter, N. Flerova, and R. Marinescu. Search algorithms for m best solutions for graphical models. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence*, 2012.

X. Fan, C. Yuan, and B. Malone. Tightening bounds for Bayesian network structure learning. In *In Proceedings of the Twenty-Eighth Conference on Artificial Intelligence*, 2014.

A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N. R. Sturtevant, J. Schaeffer, and R. Holte. Partial-expansion A* with selective node generation. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence*, 2012.

S. B. Gillispie and M. D. Perlman. Enumerating markov equivalence classes of acyclic digraph dels. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 171–177, 2001.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4 (2):100–107, 1968.

T. Jaakkola, D. Sontag, A. Globerson, and M. Meila. Learning Bayesian network structure using LP relaxations. In *Proceedings of the Thirteen International Conference on Artificial Intelligence and Statistics*, pages 358–365, 2010.

M. Koivisto and K. Sood. Exact Bayesian structure discovery in Bayesian networks. *The Journal of Machine Learning Research*, 5:549–573, 2004.

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

J. Kwisthout, H. L. Bodlaender, and L. C. van der Gaag. The complexity of finding k-th most probable explanations in probabilistic networks. In *Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 356–367, 2011.

C. Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, 1995.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

S. E. Shimony. Finding MAPs for belief networks is NP-hard. *Artif. Intell.*, 68(2):399–410, 1994.

T. Silander and P. Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pages 445–452, 2006.

A. P. Singh and A. W. Moore. Finding optimal Bayesian networks by dynamic programming. Technical report, CMU-CALD-050106, 2005.

J. Tian, R. He, and L. Ram. Bayesian model averaging using the k-best Bayesian network structures. In *Proceedings of the Twenty-Six Conference on Uncertainty in Artificial Intelligence*, pages 589–597, 2010.

T. Verma and J. Pearl. Equivalence and synthesis of causal models. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 220–227, 1990.

T. Verma and J. Pearl. An algorithm for deciding if a set of observed independencies has a causal explanation. In *Proceedings of the Eighth international conference on uncertainty in artificial intelligence*, pages 323–330, 1992.

T. Yoshizumi, T. Miura, and T. Ishida. A* with partial expansion for large branching factor problems. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2000.

C. Yuan and B. Malone. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–65, 2013.

Appendix

Proof of Theorem 2

First recall the following theorem from Verma and Pearl (1992); Meek (1995).

Theorem 3 *Given a PDAG where all the v-structures are oriented, then the CPDAG can be obtained by repeatedly applying the following orientation rules:*

- R_1 *Orient $b - c$ into $b \rightarrow c$ whenever there is an arrow $a \rightarrow b$ such that a and c are nonadjacent.*
- R_2 *Orient $a - b$ into $a \rightarrow b$ whenever there is chain $a \rightarrow c \rightarrow b$.*
- R_3 *Orient $a - b$ into $a \rightarrow b$ whenever there are two chains $a - c \rightarrow b$ and $a - d \rightarrow b$ such that c and d are nonadjacent.*
- R_4 *Orient $a - b$ into $a \rightarrow b$ whenever there are two chains $a - c \rightarrow d$ and $c \rightarrow d \rightarrow b$ such that c and b are nonadjacent and a and d are nonadjacent.*

Proof of Theorem 2 (If:) First, note that there are no compelled edges directed away from X (which would make X a non-leaf). Next, note that we can compel each edge $S - X$ towards X one-by-one, each time checking if Theorem 3 compels another edge in $S' - X$ towards S' instead. If this never happens, then all edges $S - X$ can be oriented towards X (which makes X a possible leaf).

We start by orienting one edge $S - X$ towards S. Consider each of the rules in Theorem 3:

- R_1 cannot be used to orient the edge from $X \rightarrow S$. First, if there were some other compelled edge $Y \rightarrow X$ where $Y \notin \mathbf{S}$ and Y is not adjacent to S, then the edge $S - X$ should already have been compelled (i.e., P was not a CPDAG). Otherwise, we only orient edges from $S \rightarrow X$ and all $S \in \mathbf{S}$ are adjacent.
- R_2 cannot be used to orient the edge from $X \rightarrow S$, since there is no chain from X to S (which would imply X was a non-leaf).
- R_3 cannot be used to orient the edge from $X \rightarrow S$, since all potential neighbors c and d via an unoriented edge must be adjancent.
- R_4 cannot be used to orient the edge from $X \rightarrow S$, since all potential neighbors c and b via an unoriented edge must be adjancent.

Since no rule compels us to orient an edge away from X, we can orient the edges one-by-one to make X a leaf.

(Only if:) We show that if $\mathbf{S}$ is not a clique, then X is not a leaf. If $\mathbf{S}$ is not a clique, then there is a pair S and S' in $\mathbf{S}$ that are non-adjacent. If we orient $S - X$ as $S \leftarrow X$, then X is not a leaf. If we orient $S - X$ as $S \rightarrow X$, then by Theorem 3, Rule R_1, we must orient $S' - X$ as $S' \leftarrow X$. Hence, X is not a leaf. □

Proofs of Propositions

Proof of Proposition 1 Follows from the definition of equivalence classes. □

Proof of Proposition 2 This path can be constructed by following any path to G in the BN graph, and then identifying the corresponding CPDAGs in the EC graph. □

Proof of Proposition 3 Follows from Theorem 10 in Chickering (1995). □

Proof of Proposition 4 Follows from the fact that the canonical ordering is the ordering with the largest reverse lexicographic order. □

EXPERIMENTS: EC TREE VS. BN GRAPH

In this section, we provide additional experimental results on EC trees and BN graphs, to gain a deeper insight into their performances differences. We first enumerate the 10-best, 100-best, and 1000-best equivalence classes using a EC tree. Using the BN graph, we then enumerate an equivalent number of DAGs, per dataset; Table 4 (from the main text) reports these numbers. Table 6 summarizes the time (in seconds) and memory (in GB) used by the EC tree and the BN graph, during A* search. As seen in the paper, the EC tree is more efficient (both in speed and memory consumption) than the BN graph, up to orders of magnitude differences (at least in time).

Table 7 shows, for the BN graph, the number of generated nodes, expanded nodes, re-expanded nodes (by partial-expansion), and the number of invocations of the oracle. In contrast to the EC tree (from Table 2 in the main text), the BN graph usually expands and generates at least one order of magnitude more nodes, and in some datasets, e.g., letter, msnbc and nltcs, more than three orders of magnitudes. In addition, Table 8 shows the time spent on computing the heuristic function T_h and the time spent on traversing the search space T_{A*}. For the data sets where the k-best equivalence classes contains a large number of DAGs, i.e., adult, letter, msnbc and nltcs, the majority of the time

benchmark			10-best EC				100-best EC				1,000-best EC			
			EC tree		BN graph		EC tree		BN graph		EC tree		BN graph	
name	n	N	t	m	t	m	t	m	t	m	t	m	t	m
adult	14	30162	0.26	1	0.47	1	0.54	1	1.49	1	1.19	1	11.87	1
wine	14	178	0.05	1	0.09	1	0.15	1	0.33	1	0.86	1	3.89	1
nltcs	16	16181	3.36	1	11.34	1	5.56	1	46.91	1	9.58	1	1126.05	4
letter	17	20000	18.11	1	20.68	1	48.31	1	84.07	1	81.72	1	4666.29	4
msnbc	17	291326	145.71	1	896.45	2	153.05	1	$\times_t$		155.07	1	$\times_t$	
voting	17	435	1.89	1	2.86	1	2.11	1	3.70	1	6.67	1	17.89	1
zoo	17	101	2.89	1	5.09	1	3.59	1	5.85	1	6.03	1	10.34	1
statlog	19	752	29.28	1	51.89	1	41.99	1	73.77	1	43.89	1	76.99	1
hepatitis	20	126	36.33	1	86.05	2	63.37	1	176.83	2	101.05	2	284.46	4
imports	22	205	174.84	4	455.65	8	223.78	4	603.68	8	224.11	4	604.14	8
parkinsons	23	195	897.81	8	779.90	16	897.97	8	1034.50	16	898.68	8	1450.46	16

Table 6: Time t (in seconds) and memory m (in GBs) used by EC tree and BN graph. n is the number of variables in the dataset, and N is the size of the dataset. A $\times_t$ corresponds to an out-of-time (2hr).

benchmark		10-best EC				100-best EC				1,000-best EC			
name	n	gen.	exp.	re-exp.	invoke	gen.	exp.	re-exp.	invoke	gen.	exp.	re-exp.	invoke
adult	14	1672	1352	3852	173	30619	27015	179312	634	245687	215753	2602353	1050
wine	14	8203	3714	0	107	44209	23572	23124	274	461799	254154	500024	595
nltcs	16	56719	53572	452232	633	326813	314528	6021330	1372	2727808	2605978	82512570	2180
letter	17	16296	16227	153430	678	230931	230931	4948105	2726	5443620	5424968	213643716	4963
msnbc	17	1288695	1288339	10564990	2695	$\times_t$				$\times_t$			
voting	17	5314	4364	346	147	72658	50004	99182	413	114498	106378	421512	3965
zoo	17	1049	704	0	330	12003	3875	3498	539	53562	47162	41698	1695
statlog	19	1915	1558	0	212	19653	12847	12403	628	153048	101570	194334	1029
hepatitis	20	18726	15470	0	4645	167033	105997	105105	13223	1378039	720381	1422924	31854
imports	22	1023	295	0	150	8041	2724	0	404	41007	21475	0	426
parkinsons	23	8233	2732	0	290	32136	10189	0	652	151200	58802	0	1273

Table 7: BN graph: number of nodes (1) generated, (2) expanded, (3) re-expanded, and (4) oracle invocations.

benchmark		10-best EC		100-best EC		1,000-best EC	
name	n	T_h	T_{A*}	T_h	T_{A*}	T_h	T_{A*}
adult	14	0.46	0.02	0.88	0.61	1.11	10.76
wine	14	0.05	0.04	0.06	0.26	0.12	3.77
nltcs	16	9.22	2.11	15.00	31.90	18.23	1107.81
letter	17	19.52	1.16	48.28	35.79	70.22	4596.07
msnbc	17	126.34	770.11	$\times_t$		$\times_t$	
voting	17	2.80	0.06	2.86	0.84	7.32	10.57
zoo	17	5.06	0.02	5.77	0.09	9.69	0.65
statlog	19	51.78	0.11	73.47	0.30	75.31	1.68
hepatitis	20	85.56	0.48	174.07	2.76	263.28	21.18
imports	22	454.71	0.93	602.49	1.19	602.71	1.43
parkinsons	23	778.45	1.45	1032.56	1.94	1447.56	2.89

Table 8: Time T_h to compute the heuristic function and time T_{A*} spent in A* search, in the BN graph (in seconds).

is spent on exploring the search space, rather the computing the heuristic function. This is in contrast to the EC tree, illustrated in Table 3, and the smaller datasets in Table 8, where the heuristic function is the performance bottleneck. However, on the EC tree, for these larger datasets, only a very small amount of time is used to traverse the search space (in Table 3), which shows that the EC tree is a more compact and efficient search space for enumerating equivalence classes.