Journal of Machine Learning Research 2 (2002) 445-498 Submitted 7/01; Published 2/02

Learning Equivalence Classes of Bayesian-Network Structures

David Maxwell Chickering DMAX@MICROSOFT.COM
Microsoft Research
One Microsoft Way
Redmond, WA 98052

Editor: Craig Boutilier

Abstract

Two Bayesian-network structures are said to be *equivalent* if the set of distributions that can be represented with one of those structures is identical to the set of distributions that can be represented with the other. Many scoring criteria that are used to learn Bayesian-network structures from data are *score equivalent*; that is, these criteria do not distinguish among networks that are equivalent. In this paper, we consider using a score equivalent criterion in conjunction with a heuristic search algorithm to perform model selection or model averaging. We argue that it is often appropriate to search among equivalence classes of network structures as opposed to the more common approach of searching among individual Bayesian-network structures. We describe a convenient graphical representation for an equivalence class of structures, and introduce a set of operators that can be applied to that representation by a search algorithm to move among equivalence classes. We show that our equivalence-class operators can be scored locally, and thus share the computational efficiency of traditional operators defined for individual structures. We show experimentally that a greedy model-selection algorithm using our representation yields slightly higher-scoring structures than the traditional approach without any additional time overhead, and we argue that more sophisticated search algorithms are likely to benefit much more.

1. Introduction

Throughout the last decade, there has been an enormous amount of work in the area of learning Bayesian-network structures from data; Buntine (1996) provides a good review of the literature, Heckerman (1995) presents a tutorial on the topic, and Jordan (1998) contains some introductory articles and more recent advances. Contributions to this area of research can typically be classified as addressing one of two general problems: the *evaluation* problem and the *identification* problem.

Work on the evaluation problem concentrates on deriving scoring criteria for network structures given data; that is, given a structure and a set of data, how do we assign a score to the structure that measures how well it fits with the data? Work to develop or extend evaluation criteria for Bayesian-network structures includes Buntine (1991), Cooper and Herskovitz (1992), Suzuki (1993), Bouckaert (1993), and Heckerman, Geiger, Chickering (1995), among many others.

The identification problem is to identify one or more network structures that yield a high value for the scoring criterion. Research contributions to this problem include heuristic search techniques as well complexity results. Note that much of the work on the so-called "independence approach" to learning structure falls into this category. A few of the many contributions to this area include Chow and Liu (1968), Verma and Pearl (1990), Cooper and Herskovitz (1992), and Chickering (1996a).

The notion of *equivalence* plays an important role for both of the learning problems. For any given network structure, there is a corresponding set of probability distributions that can be represented using a Bayesian network with that structure. Two network structures are *equivalent* if the set of distributions that can be represented using one of the structures is identical to the set of distributions that can be represented using the other. Because equivalence is reflexive, symmetric, and transitive, the relation defines a set of equivalence classes over network structures. Many of the evaluation criteria found in the literature are *score equivalent*; that is, these scoring criteria assign the same score to equivalent structures. As we shall argue later, score equivalence is a desirable property of a scoring criterion unless "extra" semantics (such as causality) can be attributed to the edges in a Bayesian network.

The main contribution of this paper is to show how to take advantage of a score-equivalent evaluation criterion when identifying high-scoring network structures. In particular, we formulate a search space that can be used in conjunction with a score-equivalent scoring criterion to search over equivalence classes of Bayesian-network structures, as opposed to the more common approach of searching over network structures themselves. The search space is similar to the one presented by Chickering (1996a); we extend this work by showing how all of the operators can be scored *locally*, and thus we incur overhead comparable to traditional structure search. By locally, we mean that the change in score can be computed using local functions of a subset of the nodes in the network. We provide experimental results using a greedy search algorithm that demonstrate that our approach yields slightly higher-scoring solutions while remaining as fast as the traditional approach. We argue that when a more sophisticated search algorithm is applied to the learning problem, our space should prove to be significantly more advantageous.

Other researchers have designed search algorithms to be used with a score-equivalent scoring criterion. To our knowledge, however, we are the first to use an explicit representation of an equivalence class and yet retain the advantage of local scoring.

This paper is organized as follows. In Section 2 we discuss previous relevant work and describe our notation. In Section 3, we describe two algorithms that are needed to specify our search space. In Section 4, we provide a detailed specification of our search space in which the states are defined to be equivalence classes of Bayesian-network structures. In Section 5, we show how to evaluate each of the operators locally, and provide necessary and sufficient conditions for each of the operators to be valid. In Section 6, we provide experimental results from applying a greedy search algorithm to our search space, using a particular score-equivalent evaluation criterion. Finally, in Section 7, we conclude with a discussion. Detailed proofs for our main results are given in the appendix.

2. Background

In this section, we discuss previous work that is relevant to this paper, and describe some of the notation that we use.

2.1 Bayesian Networks

A Bayesian network B for a set of variables $U = \{x_1, \ldots, x_n\}$ is a pair $(\mathcal{G}, \Theta)$. $\mathcal{G} = (V, E)$ is a directed acyclic graph—or *DAG* for short—consisting of (1) nodes V in one-to-one correspondence with the variables U, and (2) directed edges E that connect the nodes. Θ is a set of conditional probability distributions such that $\Theta_i \in \Theta$ defines the conditional probability of node x_i given its parents in $\mathcal{G}$. A Bayesian network represents a joint distribution over U that factors according to the structure $\mathcal{G}$ as follows:

$$p_B(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | \Pi_{x_i}, \Theta_i) \tag{1}$$

where Π_{x_i} is the set of parents of node x_i in $\mathcal{G}$. As discussed below, there may be additional semantics to the structure of a Bayesian network.

The structure $\mathcal{G}$ of a Bayesian network imposes a set of independence constraints that must hold in any distribution that can be represented by a network with that structure. In particular, it can be shown that Equation 1 imposes the constraint that each node is independent of its non-descendants given its parents. See (for example) Pearl (1988) for a detailed discussion these and other independence constraints that can be derived from a DAG, as well as a thorough introduction to Bayesian networks.

Throughout this paper, we make many comparisons between DAGs (and also between PDAGs, which are defined below). To simplify the discussion, we will assume that when any such comparison is made, the models are defined over the same set of variables. Thus when we say, for example, that two DAGs $\mathcal{G}$ and $\mathcal{G}'$ represent the same independence constraints, we assume that $\mathcal{G}$ and $\mathcal{G}'$ have the same node sets.

Two DAGs $\mathcal{G}$ and $\mathcal{G}'$ are *equivalent* if for every Bayesian network $B = (\mathcal{G}, \Theta)$, there exists a Bayesian network $B' = (\mathcal{G}', \Theta')$ such that B and B' define the same probability distribution, and vice versa.

Note that we have placed no restrictions on the functional form of the conditional probability distributions given in Equation 1. Often in practice, each of these distributions is assumed to be a member of some specific family, where the choice of family depends on the particular variable and parent set. For example, the conditional distributions for continuous variables with continuous parents might be restricted to linear regressions. If such restrictions are included, our current definition of equivalence is inadequate because it ignores any non-independence constraints that the DAG imposes via the choice of distribution family. For the remainder of this paper, we will assume that any distribution restrictions are chosen such that the only (additional) constraints on the joint distribution imposed by the DAG are independence constraints. There are two common examples of such distribution restrictions in practice: (1) when all variables are discrete, the joint distribution is an unrestricted discrete distribution and the conditional probability distributions are independent multinomials for each variable and each parent configuration, and (2) when all variables are continuous, the joint distribution is a multivariate Gaussian and the conditional probability distributions are independent linear regressions for each variable.

2.2 Learning Bayesian Networks

As described in the introduction, there has been an enormous amount of work on learning Bayesian networks from data. Methods to learn Bayesian networks from data typically consist of two components. The first component is a *scoring criterion* that takes as input a Bayesian-network structure, a data set, and possibly some domain knowledge; the scoring criterion returns a value indicating how well the structure fits the data. The second component in a typical learning method is a *search algorithm* that identifies one or more structures with high score. Once the structure of a Bayesian network is identified from data, it is typically straightforward to estimate the parameters of that model.

An important property of any method for learning Bayesian networks from data is the *interpretation* that is given to structures by the scoring criterion. For most criteria, Bayesian-network structures are interpreted as independence constraints in some distribution. We call this the *independence interpretation* of structures. For example, most Bayesian criteria interpret a DAG $\mathcal{G}$ as an assertion about the independence constraints in the distribution from which the data was generated. Penalized-likelihood criteria, such as AIC (Akaike, 1974) and BIC (Schwarz, 1978), interpret $\mathcal{G}$ as a set of independence constraints in the maximum-likelihood estimate, derived from the observed data, of a joint distribution over the domain. The MDL criterion (e.g., Rissanen, 1986 and Bouckaert, 1993) interprets $\mathcal{G}$ as a set of independence constraints in a joint distribution that can be used to compress the observed data.

The most common exception to the independence interpretation of structures is when the scoring criterion attributes additional causal semantics to the edges in a DAG. See, for example, Pearl and Verma (1991), Spirtes, Glymour and Scheines (1993), Druzdzel and Simon (1993), and Heckerman and Shachter (1995) for formal causal semantics to Bayesian networks.

When structures are interpreted as independence constraints in some distribution, all of the DAGs within the same equivalence class are redundant representations of the same constraints. It should therefore not be surprising that many common scoring criteria assign the same score to structures in the same equivalence class, a property we call *score equivalence*. Chickering (1995), for example, shows that the AIC criterion, the BIC criterion, the MDL criterion and the (Bayesian) BDe criterion (Heckerman et al., 1995) are all score equivalent.[1]

Given a scoring criterion, the goal of the search procedure is to identify one or more high-scoring models. In a model-selection scenario, for example, the goal of the search procedure is to identify a single model with the highest possible score. Chickering (1996b) shows that the problem of identifying the Bayesian-network structure with the highest score is NP-hard when using the BDe scoring criterion. Although not yet proven, it is thought by most researchers that this model-selection problem is hard for other common criteria as well. As a result, a large amount of research effort has concentrated on applying various heuristic search algorithms. By far the most common approach is to apply a heuristic search algorithm to the space of DAGs; in particular, these algorithms traverse the space of DAG models by applying local changes to the edges of DAGs.

1. Chickering (1995) shows that the BDe criterion is score equivalent under the independence interpretation of DAGs; the BDe scoring criterion can also be used when DAGs are interpreted causally.

2.3 Learning Equivalence Classes vs. Learning DAGs

The focus of this paper is to improve heuristic search algorithms in the common situation when DAGs are interpreted as independence constraints, and when the scoring criterion is score equivalent. As pointed out by Andersson, Madigan and Perlman (1997), there are a number of potential problems when heuristic search algorithms traverse through the space of DAGs instead of the space of equivalence classes of DAGs in this situation. We now elaborate on some of these problems.

The first potential problem is one of efficiency. Many of the operators used by these search algorithms are defined to move between DAGs within the same equivalence class. Under the independence interpretation of structure, applying such an operator will result in (a new representation of) the same state. As a result, unless the search algorithm explicitly tests for equivalence, the algorithm can waste time re-scoring the same equivalence class to evaluate the merit of each such operator. In many cases, in order for the algorithm to move from one equivalence class to another, it will have to make numerous moves within the same equivalence class. In other words, the local connectivity of equivalence classes in the search space depends on the particular DAG being used to represent the current equivalence class. This can be particularly problematic when applying a greedy search algorithm; in this case, the choice of the representation for an equivalence class can be arbitrary due to ties in the scoring criterion, and because a move within an equivalence class can never increase the score, the connectivity of the search space becomes arbitrary as well. Furthermore, the equivalence classes that are reachable from a given (equivalence class) state will depend on how the search algorithm reached that state.

Consider Figure 1, which shows some DAGs defined over a three-variable domain. Assume that a greedy search algorithm is applied to this space, starting from the graph containing no edges shown in Figure 1(a), and assume that the operators used by the algorithm are directed edge additions, deletions and reversals. As we shall see below, the DAGs in Figure 1(b) and Figure 1(c) are equivalent. Using a score-equivalent criterion, a greedy search algorithm does not prefer either DAG to the other, and assuming the score for these DAGs are better than any other DAG reachable in one move from the initial state, the algorithm will arbitrarily choose one of them. Unless the algorithm recognizes that these two DAGs are equivalent, a greedy search algorithm will waste time evaluating an edge reversal once reaching either of these states because they are reachable from each other by an edge reversal. If the DAG in Figure 1(b) is chosen, the algorithm can move, in one step, to the DAG in Figure 1(d). On the other hand, if the DAG in Figure 1(c) is chosen then there is no single edge addition or reversal that can move the search to the DAG in Figure 1(d), which happens to be the only DAG representation for its corresponding equivalence class. Unfortunately, the score of the DAG identified by a greedy search can be sensitive to these arbitrary choices.

Another potential problem with searching over DAG models instead of equivalence classes has to do with the structure prior provided for Bayesian scoring criteria. If an algorithm interprets each DAG within a given equivalence class as a separate model, the effective prior probability for an equivalence class will be the *sum* of the prior probabilities for all DAGs within that class. This implicit summing of priors can have unintended consequences. For example, assume that we assign a uniform prior over all DAGs. In a

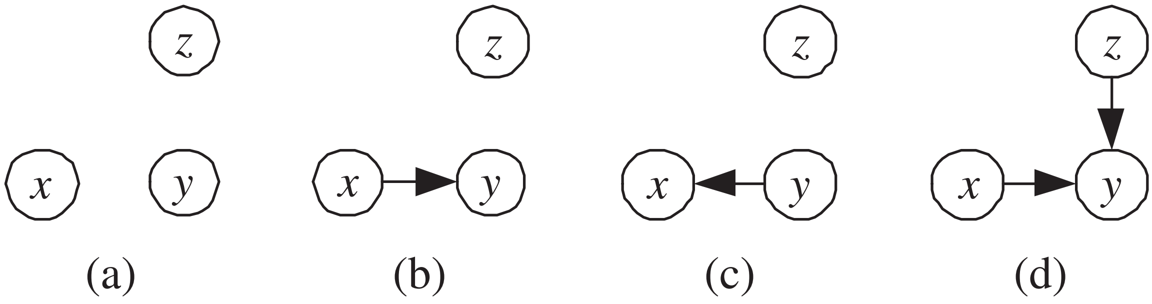

Figure 1: Example of a greedy algorithm applied to the space of DAG models.

domain with n variables, the equivalence class that places no independence constraints on the distribution has $n!$ DAG representations, whereas the equivalence class that imposes the constraint that all variables are mutually independent has exactly one DAG representation. Thus the "uniform" prior specified for the criterion in fact favors the no-independence model over the all-independence model by a factor of $n!$ a priori.

When performing model *selection*, assigning priors to DAGs instead of equivalence classes is not necessarily problematic. In particular, if the algorithm identifies a single high-scoring DAG, that DAG will correspond to a high-scoring equivalence class that has the same prior as the DAG. The main problem with the structure priors comes when performing Bayesian model averaging or selective Bayesian model averaging; in this case, the structure prior affects how learned models are combined together to compute various quantities of interest. Madigan, Andersson, Perlman and Volinsky (1996) recognized this problem and, in the context of Monte-Carlo sampling of structures, devised methods for sampling equivalence classes.

There has been some recent work by Gillispie and Perlman (2001) studying the behavior of the ratio of the number of DAGs to the number of equivalence classes as a function of the size of the domain. Their results suggest that this ratio reaches an asymptote around 3.7 with as few as ten nodes. This result potentially mitigates our worries about searching over DAG models because we expect the number of DAGs for the average equivalence class to be reasonably small. We expect, for example, that a naive exhaustive search algorithm traversing all models will take only four times as long in DAG space than if it enumerated the equivalence classes explicitly. More important, however, is the number of DAG elements contained in the equivalence classes over which our algorithm searches; as we shall see in Section 6, this number can be enormous.

There exist scoring criteria that use the independence interpretation of structure, yet are not score equivalent. The most widely used example of this is the K2 criterion derived by Cooper and Herskovitz (1992). The K2 criterion applied to a DAG $\mathcal{G}$ evaluates the relative posterior probability that the generative distribution has the same independence constraints as those entailed by $\mathcal{G}$. The criterion is not score equivalent because the prior distributions over parameters corresponding to different structures are not always consistent; nevertheless, the criterion is very efficient to evaluate and yields good results in practice. The results of this paper can be applied directly when using criteria of this type by allowing equivalence classes to be evaluated by arbitrary members of that class; this will often be reasonable because the distinction between DAG scores within the same equivalence class under the independence interpretation should have no significance other than computational convenience.

There also exist score-equivalent criteria that do not use the independence interpretation of structure. An example of this is the (Bayesian) BDe scoring criterion derived by Heckerman et al. (1995) when applied to learning causal networks from data. This scoring criterion has the property that the data does not distinguish between equivalent structures. Thus, if all DAGs in an equivalence class are equally likely a priori, they will all get the same score using the criterion, and therefore searching over equivalence classes is appropriate.

Other researchers have explored methods for searching through equivalence classes of Bayesian-network structures. One approach, commonly known as the *independence approach* to learning, uses an independence oracle—implemented with some statistical test with a given level of sensitivity—in conjunction with a set of rules to identify an equivalence class from data. Verma and Pearl (1992), Spirtes, Glymour and Scheines (1993), and Meek (1995) all describe examples of this approach. In this paper, we concentrate instead on model selection and model averaging when using scoring criteria to identify models. See Chickering (1995) for a discussion of the distinction between the independence approach to learning and the so-called *metric approach* that we consider here.

As mentioned above, Madigan et al. (1996) consider searching through the space of equivalence classes using Monte-Carlo sampling, and their operators can be used for general heuristic search algorithms as well. Spirtes and Meek (1995) define a greedy search algorithm that explicitly searches through the space of equivalence classes. Chickering (1996a) defines a set of operators that traverse through the space of equivalence classes, and shows that a greedy search algorithm using these operators generally results in better models than when searching through DAG space. Dash and Druzdzel (1999) apply multiple instances of the independence approach—using random levels of significance for the statistical test that approximates the independence oracle—and evaluate each result by extracting a representative DAG model, retaining the best such DAG at each step.

In this paper, we define a search space that can be used by any heuristic search algorithm, and we show how to score all of the operators using local functions of the nodes in our graphical representations of equivalence classes. The advantage of our approach over previous work is efficiency: because the operators can be scored locally, search algorithms applied to our space can traverse through equivalence classes much faster than previous algorithms. Munteanu and Cau (2000) derive three operators that are a subset of the operators we present in this paper and present local scores for them, but our results provide counter-examples to the validity conditions of all of their operators and to the scores of two of the three. After the original submission of the present paper, Munteanu and Bendou (2001) extended the set of three operators to include two more of the operators that we present here.

2.4 General Notation and Results

We now introduce notation and results that will be used to define our search space. The *skeleton* of any DAG is the undirected graph resulting from ignoring the directionality of every edge. A *v-structure* in a DAG $\mathcal{G}$ is an ordered triple of nodes (x, y, z) such that $\mathcal{G}$ contains the directed edges $x \rightarrow y$ and $z \rightarrow y$, and x and z are not adjacent in $\mathcal{G}$. Verma and Pearl (1990) derive the following characterization of equivalent structures:

Theorem 1 [Verma and Pearl, 1990] *Two DAGs are equivalent if and only if they have the same skeletons and the same v-structures.*

A directed edge $x \rightarrow y$ is *compelled* in $\mathcal{G}$ if for every DAG $\mathcal{G}'$ equivalent to $\mathcal{G}$, $x \rightarrow y$ exists in $\mathcal{G}'$. For any edge e in $\mathcal{G}$, if e is not compelled in $\mathcal{G}$, then e is *reversible* in $\mathcal{G}$; that is, there exists some DAG $\mathcal{G}'$ equivalent to $\mathcal{G}$ in which e has opposite orientation.

A consequence of Theorem 1 is that for any edge e participating in a v-structure in some DAG $\mathcal{G}$, if that edge is reversed in some other DAG $\mathcal{G}'$, then $\mathcal{G}$ and $\mathcal{G}'$ are not equivalent. Thus any edge participating in a v-structure is compelled. Not every compelled edge, however, necessarily participates in a v-structure. For example, the edge from z to w in the DAG shown in Figure 3(a) is compelled.

An acyclic partially directed graph, or *PDAG* for short, is a graph that contains both directed and undirected edges. It is acyclic in the sense that it contains no *directed* cycles. We use PDAGs to represent equivalence classes of Bayesian-network structures. Let $\mathcal{P}$ denote an arbitrary PDAG. We define the equivalence class of DAGs corresponding to $\mathcal{P}$—denoted $Class(\mathcal{P})$—as follows: $\mathcal{G} \in Class(\mathcal{P})$ if and only if $\mathcal{G}$ and $\mathcal{P}$ have the same skeleton and the same set of v-structures.[2] From Theorem 1, it follows that a PDAG containing a directed edge for every edge participating in a v-structure, and an undirected edge for every other edge, uniquely identifies an equivalence class of DAGs. There may be many other PDAGs, however, that correspond to the same equivalence class. For example, any DAG interpreted as a PDAG can be used with our definition of $Class$ to represent its own equivalence class.

If a DAG $\mathcal{G}$ has the same skeleton and the same set of v-structures as a PDAG $\mathcal{P}$, and if every directed edge in $\mathcal{P}$ has the same orientation in $\mathcal{G}$, we say that $\mathcal{G}$ is a *consistent extension* of $\mathcal{P}$. Note that any DAG that is a consistent extension of $\mathcal{P}$ must also be contained in $Class(\mathcal{P})$, but not every DAG in $Class(\mathcal{P})$ is a consistent extension of $\mathcal{P}$. If there is at least one consistent extension of a PDAG $\mathcal{P}$, we say that $\mathcal{P}$ *admits a consistent extension*. Figure 2(a) shows a PDAG that admits a consistent extension, and Figure 2(b) shows one such consistent extension. Figure 2(c) shows a PDAG that does not admit a consistent extension.

The *completed PDAG* corresponding to an equivalence class is the PDAG consisting of a directed edge for every compelled edge in the equivalence class, and an undirected edge for every reversible edge in the equivalence class. Note that for a completed PDAG $\mathcal{P}^c$, unlike arbitrary PDAGs, every DAG contained in $Class(\mathcal{P}^c)$ is a consistent extension of $\mathcal{P}^c$. Figure 3(a) shows a DAG $\mathcal{G}$, and Figure 3(b) shows the completed PDAG for $Class(\mathcal{G})$. PDAGs are called *patterns* by Spirtes et al. (1993) and completed PDAGs are called *essential graphs* by Andersson et al. (1997) and *maximally oriented graphs* by Meek (1995).

Given an equivalence class of Bayesian-network structures, the completed PDAG for that class is unique. We emphasize this result with a lemma.

2. The definitions for the skeleton and set of v-structures for a PDAG are the obvious extensions to these definitions for DAGs.

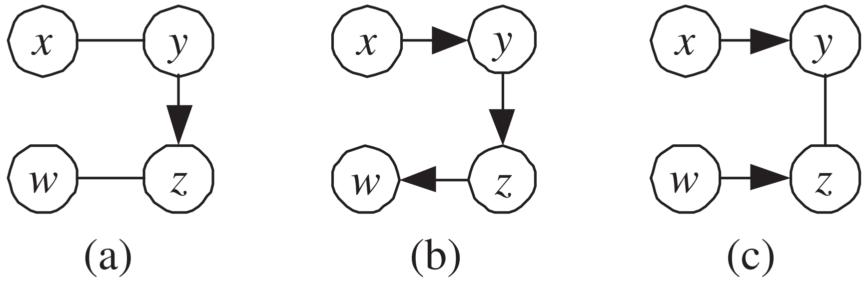

Figure 2: (a) a PDAG that admits a consistent extension, (b) a consistent extension of the PDAG in (a), and (c) a PDAG that does not admit a consistent extension.

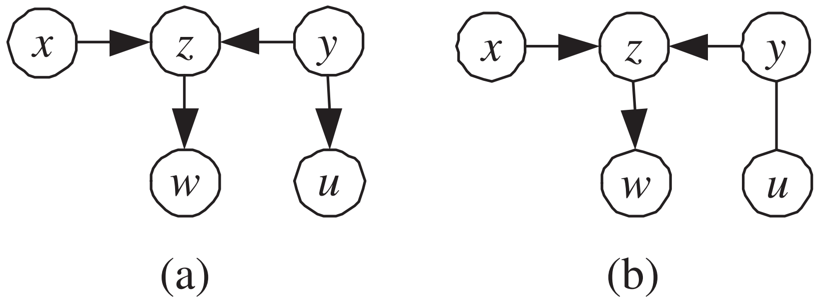

Figure 3: (a) a DAG $\mathcal{G}$ and (b) the completed PDAG for $Class(\mathcal{G})$.

Lemma 2 *Let $\mathcal{P}_1^c$ and $\mathcal{P}_2^c$ denote two completed PDAGs that both admit a consistent extension. Then $\mathcal{P}_1^c = \mathcal{P}_2^c$ if and only if $Class(\mathcal{P}_1^c) = Class(\mathcal{P}_2^c)$.*

Proof: Follows immediately by Theorem 1 and by the definitions of compelled and reversible edges. □

A *clique* in a DAG or a PDAG is a set of nodes for which every pair of nodes is adjacent. A *topological sort* of the nodes in a DAG $\mathcal{G}$ is any total ordering of the nodes such that, for any pair of nodes x_i and x_j in $\mathcal{G}$, if x_i is an ancestor of x_j, then x_i must precede x_j in the ordering.

As in Equation 1 above, we use Π_x to denote the set of parents for node x in both a DAG and a PDAG. We use N_x to denote the set of neighbors of node x in a PDAG. We use $N_{x,y} = N_x \cap N_y$ to denote the set of common neighbors of x and y in a PDAG. We use $\Omega_{x,y} = \Pi_x \cap N_y$ to denote the set of parents of node x that are neighbors of node y in a PDAG.

3. Converting Between DAGs and PDAGs

In this section, we describe two algorithms that we need in order to apply the search operators that we present in the next section. The first algorithm, which we refer to as

DAG-to-CPDAG, takes as input a Bayesian-network structure, and outputs a completed PDAG representation of the equivalence class to which that structure belongs. The second algorithm, which we refer to as PDAG-to-DAG, takes as input a PDAG representation for an equivalence class, and outputs a DAG contained in that class.

Although we briefly discuss the complexity of implementing both of these algorithms, the time most heuristic search algorithms will spend converting between DAGs and PDAGs is insignificant. In particular, most algorithms spend the majority of their time evaluating the score of adjacent states rather than actually traversing states; because we have derived local scores for all of the operators, the conversion algorithms presented in this section need only be applied when the search algorithm commits to moving to a given state. Furthermore, the complexities of the conversion algorithms depend only on the structural complexity (i.e., number of nodes and edges) of the graph, and not on the size of the data.

Several researchers, including Verma and Pearl (1992) and Meek (1995), present *rule-based* algorithms that can be used to implement DAG-to-CPDAG. The idea of these implementations is as follows. First, we undirect every edge in a DAG, except for those edges that participate in a v-structure. Then, we repeatedly apply one of a set of rules that transform undirected edges into directed edges. The rules "fire" by matching local patterns of undirected and directed edges in the given graph. Meek (1995) proves that the transformation rules are sound and complete. That is, once no rule matches on the current graph, that graph must be a completed PDAG. In the appendix, we detail these rules for the purpose of proving some of our results. Andersson et al. (1997), provide a similar rule-based algorithm, except that edges from a DAG are *undirected* by matching patterns of edges.

We now provide the details of an alternative algorithm that can be used to implement DAG-to-CPDAG which is computationally efficient. In particular, Chickering (1995) describes an algorithm that, when given a DAG containing $|E|$ edges, runs in time $O(|E|)$ on average. The algorithm labels all of the edges in a DAG as either "compelled" or "reversible"; given such a labeling, it is trivial to construct the corresponding completed PDAG. The first step of the algorithm is to define a (particular) total ordering over the edges in the given DAG. For simplicity, we present this step as a separate procedure listed in Figure 4. In Figure 5, we show an algorithm of Chickering (1995) that labels the edges. For a detailed study of the correctness of these algorithms, we refer the reader to Chickering (1995).

We now consider a simple implementation of PDAG-to-DAG due to Dor and Tarsi (1992). Recall from Section 2 that we use N_x and Π_x to denote the set of neighbors and parents, respectively, of node x. Given a PDAG $\mathcal{P}$, we create a DAG $\mathcal{G}$ that contains all of the directed edges from $\mathcal{P}$, and no other edges. We then repeat the following procedure: First, select a node x in $\mathcal{P}$ such that (1) x has no outgoing edges and (2) if N_x is non-empty, then $N_x \cup \Pi_x$ is a clique. If $\mathcal{P}$ admits a consistent extension, a node x with these properties is guaranteed to exist. Next, for each undirected edge $y - x$ incident to x in $\mathcal{P}$, insert a directed edge $y \rightarrow x$ in $\mathcal{G}$. Finally, remove x and all incident edges from the $\mathcal{P}$ and continue with the next node. The algorithm terminates when all nodes have been deleted from $\mathcal{P}$.

We now consider a very loose upper bound on the complexity of a simple implementation of PDAG-to-DAG for a PDAG consisting of $|V|$ nodes and $|E|$ edges. For each node x, we can determine if its neighbors and parents form a clique in $O(|N_x \cup \Pi_x|^2)$ time. For

Algorithm ORDER-EDGES($\mathcal{G}$)
Input: DAG $\mathcal{G}$
Output: DAG $\mathcal{G}$ with labeled total order on edges
1. Perform a topological sort on the NODES in $\mathcal{G}$
2. Set $i = 0$
3. **While** there are unordered EDGES in $\mathcal{G}$
4. Let y be the lowest ordered NODE that has an unordered EDGE incident into it
5. Let x be the highest ordered NODE for which $x \rightarrow y$ is not ordered
6. Label $x \rightarrow y$ with order i
7. $i = i + 1$

Figure 4: Algorithm to produce a total ordering over the edges in a DAG. The algorithm is used by Algorithm LABEL-EDGES.

Algorithm LABEL-EDGES($\mathcal{G}$)
Input: DAG $\mathcal{G}$
Output: DAG $\mathcal{G}$ with each edge labeled either "compelled" or "reversible"

1. Order the edges in $\mathcal{G}$ using **Algorithm Order-Edges**
2. Label every edge in $\mathcal{G}$ as "unknown"
3. **While** there are edges labeled "unknown" in $\mathcal{G}$
4. Let $x \rightarrow y$ be the lowest ordered edge that is labeled "unknown"
5. **For** every edge $w \rightarrow x$ labeled "compelled"
6. **If** w is not a parent of y
7. Label $x \rightarrow y$ and every edge incident into y with "compelled"
8. **Goto** 3
9. **Else**
10. Label $w \rightarrow y$ with "compelled"
11. **If** there exists an edge $z \rightarrow y$ such that $z \neq x$ and z is not a parent of x
12. Label $x \rightarrow y$ and all "unknown" edges incident into y with "compelled"
13. **Else**
14. Label $x \rightarrow y$ and all "unknown" edges incident into y with "reversible"

Figure 5: Algorithm to label each edge in a DAG with "compelled" or "reversible", which leads to an immediate implementation of DAG-TO-CPDAG.

each edge that is removed, we need only check the endpoints to see if their neighbors and parents now form a clique; if $N_z \cup \Pi_z$ is a clique for some node z in $\mathcal{P}$ at any step of the algorithm, this set must remain a clique after each edge removal in the algorithm. If $|N_x \cup \Pi_x|$ is bounded above by some constant k then we can implement the algorithm in time $O(|V|k^2 + |E|k^2)$.

Our upper bound may not be very good if k is on the order of the number of variables in the domain. We note, however, that if we select x_i after discovering a clique among its neighbors and parents, then the algorithm results in *every* member of $N_{x_i} \cup \Pi_{x_i}$ being made a parent of x_i in the DAG. Because the number of parameters in the local distribution of a discrete node grows exponentially with the number of parents, it is reasonable to expect the number of *parents* of such a node to be bounded by some reasonably small constant k. This can be enforced explicitly using the scoring criterion, or implicitly by the lack of data needed to support such a complex model. If we are given the upper bound k ahead of time, we need never even *check* for a clique among the neighbors and parents unless the number of these nodes is less than k.

In practice, graphs encountered during search are reasonably sparse, and as will be evident from the timing results in Section 6, the time spent executing either of the two algorithms presented in this section is insignificant.

4. Heuristic Search for Bayesian-network Structures

Given a scoring criterion for evaluating Bayesian-network structures, a typical learning algorithm attempts to identify one or more structures that attain a high score by applying a heuristic search algorithm. In this section, we formulate a *search space* that the heuristic search algorithm can use—in conjunction with a score-equivalent scoring criterion—to search over equivalence classes of Bayesian-network structures. A search space has three components:

1. A set of states
2. A representation scheme for the states
3. A set of operators

The set of states represents the logical set of solutions to the search problem, the representation scheme defines an efficient way to represent the states, and the set of operators is used by the search algorithm to transform the representation of one state to another in order to traverse the space in a systematic way. Once the search space has been defined, any one of a number of well-known heuristic search algorithms can easily be applied to that space.

In perhaps the simplest formulation of a search space for learning Bayesian networks, the states of the search are defined to be individual Bayesian-network structures, the representation of a state is simply a DAG, and the operators are defined to be local changes to those DAGs. For example, Chickering, Geiger and Heckerman (1995) compare various search procedures in the search space of Bayesian-network structures, using the following operators: for any pair of nodes x and y, if x and y are adjacent, the edge connecting them can be either deleted or reversed. If x and y are not adjacent, an edge can be added in either direction. All operators are subject to the constraint that a cycle cannot be formed. We shall use *Bayesian-network space*, or *B-space* for short, to denote the search space defined in this way. Figure 6 shows an example of each operator in B-space.

To appreciate the reasons for the popularity of B-space among practitioners, we need the following definition.

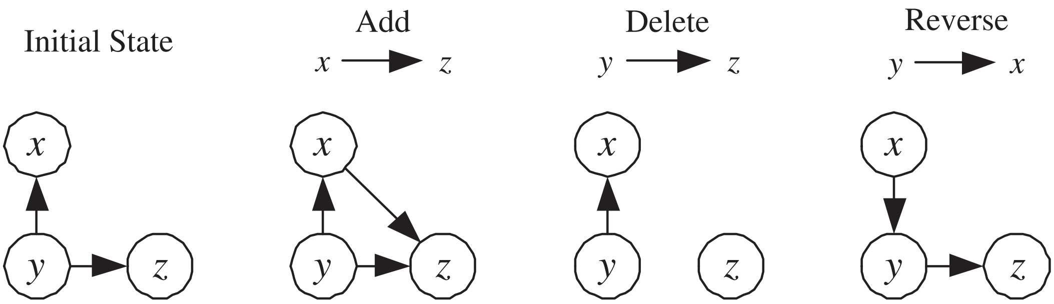

Figure 6: States resulting from the application of a single operator in B-space.

Definition 3 *A Bayesian-network-structure scoring criterion S is* decomposable *if it can be written as a sum of measures, each of which is a function only of one node and its parents.*

In other words, a decomposable scoring criterion S applied to a DAG $\mathcal{G}$ can always be expressed as:

$$S(\mathcal{G}) = \sum_{i=1}^{n} s(x_i, \Pi_{x_i}) \tag{2}$$

where n is the number of nodes in $\mathcal{G}$ and $s(x_i, \Pi_{x_i})$ is a function only of x_i and its parents in $\mathcal{G}$. Note that the data D is implicit in Equation 2. When we say that $s(x_i, \Pi_{x_i})$ is only a function of x_i and its parents, we intend this to also mean that the *data* on which this measure depends is restricted to those columns corresponding to x_i and its parents.[3]

Given a decomposable scoring criterion, B-space is particularly attractive because the change in score that results from the application of an operator can be computed locally. In particular, given the score for some DAG, only those terms in the sum above corresponding to nodes whose parents have changed need be re-computed to derive the resulting score. As discussed at length in Section 2, however, there are some potential problems with using B-space when learned DAGs are interpreted as independence constraints in some distribution.

We now define a search space for which the states are equivalence classes of Bayesian-network structures. We call this search space *equivalence-class space*, or *E-space* for short. We use completed PDAGs to represent the states of the search in E-space. As we saw from Lemma 2 in Section 2, using completed PDAGs instead of general PDAGs (or DAGs in the case of B-space) eliminates the problem of having multiple representations for the same equivalence class. To complete the specification of E-space, we define six simple operators that can be applied to completed PDAGs. The operators are given in Table 1.

All operators are subject to the constraint that the resulting graph is a PDAG (i.e., a graph containing no directed cycles) that admits a consistent extension. If an operator does, in fact, result in a PDAG that admits a consistent extension, we say that the operator is *valid*; otherwise, we say the operator is *not valid*.

3. To be explicit, we could re-write the terms in the sum of Equation 2 as $s(x_i, \Pi_{x_i}, D(\{x_i\}), D(\Pi_{x_i}))$, where $D(X)$ denotes the data restricted to the columns corresponding to the variables in set X. We find it convenient, however, to keep the notation simple.

1. **(InsertU)** For any pair of nodes x and y that are not adjacent in $\mathcal{P}^c$, we can insert an undirected edge between x and y
2. **(DeleteU)** For any undirected edge $x - y$ in $\mathcal{P}^c$, we can delete the edge
3. **(InsertD)** For any pair of nodes x and y that are not adjacent in $\mathcal{P}^c$, we can insert a directed edge in either direction
4. **(DeleteD)** For any directed edge $x \to y$ in $\mathcal{P}^c$, we can delete the edge
5. **(ReverseD)** For any directed edge $x \to y$ in $\mathcal{P}^c$, we can reverse the edge
6. **(MakeV)** For any triple of nodes x, y and z in $\mathcal{P}^c$, such that (1) x and z are not adjacent, (2) $\mathcal{P}^c$ contains the undirected edge $x-y$, and (3) $\mathcal{P}^c$ contains the undirected edge $y - z$, we can insert the v-structure $x \to y \leftarrow z$.

Table 1: E-space operators to be applied to completed PDAGs

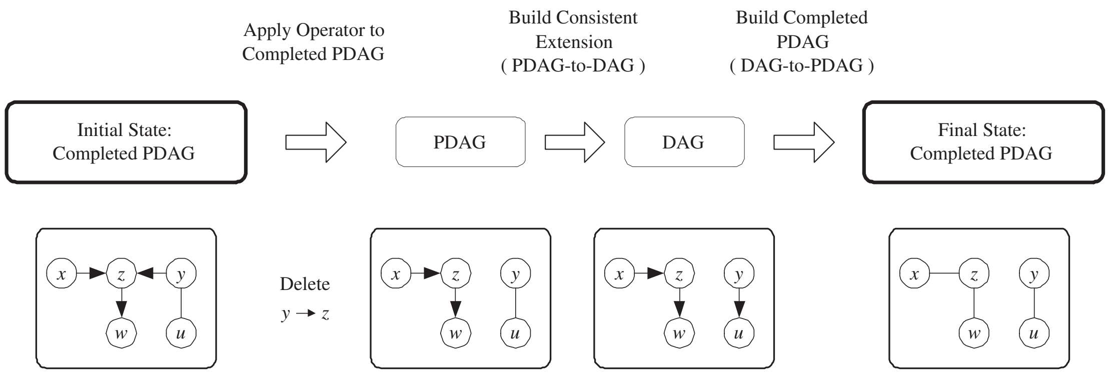

Figure 7: Diagram depicting how operators are applied. The PDAGs on the bottom give an example for every stage of the process.

After applying an operator to a completed PDAG, the resulting PDAG is not necessarily completed. To recover the completed PDAG resulting from the operator, we use the transformation algorithms presented in Section 3. For a given completed PDAG, let $\mathcal{P}$ denote the PDAG—not necessarily completed—that results after directly applying one of the operators. The completed PDAG that results from the operator is obtained as follows. First, we call the algorithm PDAG-TO-DAG with input $\mathcal{P}$ to extract a consistent extension $\mathcal{G}$. If $\mathcal{P}$ does not admit a consistent extension, then the given operator is not valid. To complete the application, the algorithm DAG-TO-CPDAG is called with input $\mathcal{G}$ to build the resulting completed PDAG representation. The process of applying an operator is depicted schematically in Figure 7, and the application of each operator type is illustrated in Figure 8.

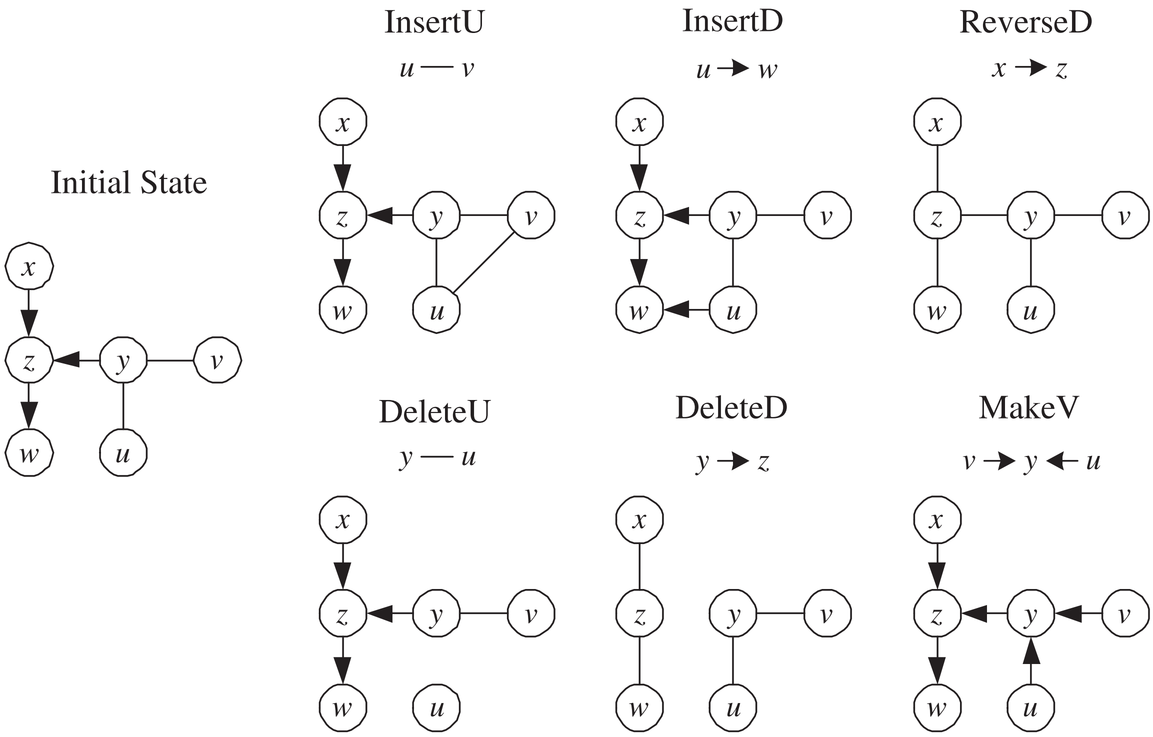

Figure 8: Example of each type of operator.

There is one other restriction that we place on the insertion operators. Namely, we only allow the insertion of an edge if it has the same "directedness" in the resulting completed PDAG. In other words, we cannot insert a directed edge if that edge will be undirected in the resulting completed PDAG, and we cannot insert an undirected edge if the edge will be directed in the resulting completed PDAG. In Figure 8, for example, we are not allowed to insert the directed edge $v \rightarrow u$.

As we state in the following theorem, the operators are *complete* for the search space. That is, given any pair of completed PDAGs $\mathcal{P}_1^c$ and $\mathcal{P}_2^c$ that both admit a consistent extension, there exists a sequence of valid operators that moves from $\mathcal{P}_1^c$ to $\mathcal{P}_2^c$.

Theorem 4 *The operators InsertU, InsertD, DeleteU, DeleteD, and MakeV are complete.*

Note that the ReverseD operator is not needed for the space to be complete; we include it because it adds extra connectivity to the space without adding too many extra operators. We provide a detailed proof of Theorem 4 in the appendix, but the intuition is as follows: we can transform $\mathcal{P}_1^c$ into a completed PDAG with no edges by deleting edges, either directed or undirected, one at a time until there are none remaining. We can then construct each of the v-structures that exist in $\mathcal{P}_2^c$ with (one or) two edge additions followed by (if necessary) a MakeV operator. The remaining directed edges from $\mathcal{P}_2^c$ can then be inserted one at a time such that they are also directed in $\mathcal{P}_1^c$. Finally, all of the undirected edges from $\mathcal{P}_2^c$ can be added to $\mathcal{P}_1^c$.

Having defined the search space, we can now apply heuristic search techniques to identify high-scoring equivalence classes using a scoring criterion that evaluates Bayesian-network structures. In particular, given a current state, an algorithm can evaluate each adjacent state (i.e., the state resulting from applying each operator) by applying the operator to the PDAG and evaluating the score of any consistent extension of the resulting PDAG. Chickering (1996a) provides experimental results that compare E-space[4] to B-space using this exact approach, in conjunction with a greedy search algorithm and a Bayesian scoring criterion. The results show that although the greedy algorithm found slightly better equivalence classes when applied to E-space, the algorithm was much faster when applied to B-space.

The extra overhead Chickering (1996a) encountered for E-space stems from the way operators in E-space are scored. Although these operators are all simple and local changes to the edges in a completed PDAG, as we see in the example of Figure 7, an operator can have "cascading" effects on the PDAG—and consequently the consistent extension that is used to score the equivalence class—that results. To score an operator, it was sometimes necessary to first apply that operator to the current state, then extract a consistent extension using PDAG-to-DAG, and finally score the resulting DAG without being able to exploit locality. The author mentions that most operators can be scored by applying local changes to a corresponding consistent extension; unfortunately the few operators that do need to be applied to the PDAG slow the algorithm enough to raise doubt about the usefulness of the space.

In the next section, we show how to score all of the operators in E-space locally, and thus eliminate the drawback of using the state space.

5. Local Scoring in E-Space

In this section, we present the main result of this paper; we show how to score locally all of the E-space operators presented in the previous section. In particular, we show that given a decomposable, score-equivalent scoring criterion for DAG models, the increase in score that results from applying each of the operators can be calculated by evaluating at most four terms in the sum of Equation 2.

To describe the local score, we need the concept of a *semi-directed* path. This is the same as a directed path except that any of the edges may be undirected. More formally we have the following definition.

Definition 5 *A* semi-directed path *from x to y in a PDAG is a path from x to y such that each edge is either undirected or directed away from x.*

Recall from Section 2 that Π_x is the set of parents of x, N_x is the set of neighbors of x, $N_{x,y}$ is the set of common neighbors of x and y, and $\Omega_{x,y}$ is the set of parents of x that are neighbors of y. The following six theorems and corresponding corollaries demonstrate, for each of the six types of operators, how to test whether or not the operator is valid and how to score the operator. In the statement of these results, we have adopted the following notation to simplify presentation: for a set of nodes M and a node x, we use M^{+x} as shorthand for $M \cup \{x\}$. Similarly, we use M^{-x} as shorthand for $M \setminus \{x\}$.

4. Chickering (1996a) does not require the presence of the two undirected edges in the MakeV operator.

Theorem 6 *Let x and y be two nodes that are not adjacent in $\mathcal{P}^c$. The insertion of the undirected edge $x - y$ is valid if and only if (1) every undirected path between x and y contains a node in $N_{x,y}$, and (2) $\Pi_x = \Pi_y$.*

Corollary 7 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid insertion of an undirected edge $x - y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, N_{x,y}^{+x} \cup \Pi_y) - s(y, N_{x,y} \cup \Pi_y)$$

Theorem 8 *Let $x - y$ be an undirected edge in completed PDAG $\mathcal{P}^c$. The deletion of $x - y$ is valid if and only if $N_{x,y}$ is a clique of undirected edges.*

Corollary 9 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid deletion of an undirected edge $x - y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, N_{x,y} \cup \Pi_y) - s(y, N_{x,y}^{+x} \cup \Pi_y)$$

Theorem 10 *Let x and y be any two nodes that are not adjacent in completed PDAG $\mathcal{P}^c$. The insertion of $x \to y$ is valid if and only if (1) every semi-directed path from y to x contains at least one node in $\Omega_{x,y}$, (2) $\Omega_{x,y}$ is a clique of undirected edges, and (3) $\Pi_x \neq \Pi_y$.*

Corollary 11 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid insertion of a directed edge $x \to y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, \Omega_{x,y} \cup \Pi_y^{+x}) - s(y, \Omega_{x,y} \cup \Pi_y)$$

Theorem 12 *Let $x \to y$ be a directed edge in completed PDAG $\mathcal{P}^c$. The deletion of $x \to y$ is valid if and only if N_y is a clique of undirected edges.*

Corollary 13 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid deletion of a directed edge $x \to y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, N_y \cup \Pi_y^{-x}) - s(y, N_y \cup \Pi_y)$$

Theorem 14 *Let $x \to y$ be a directed edge in completed PDAG $\mathcal{P}^c$. The reversal of $x \to y$ is valid if and only if (1) every semi-directed path from x to y that does not include the edge $x \to y$ contains at least one node in $\Omega_{y,x} \cup N_y$, and (2) $\Omega_{y,x}$ is a clique of undirected edges.*

Corollary 15 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid reversal of a directed edge $x \to y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, \Pi_y^{-x}) + s(x, \Pi_x^{+y} \cup \Omega_{y,x}) - s(y, \Pi_y) - s(x, \Pi_x \cup \Omega_{y,x})$$

Theorem 16 *Let $x - z - y$ be any length-two undirected path in completed PDAG $\mathcal{P}^c$ such that x and y are not adjacent. Replacing the undirected edges with directed edges to create the v-structure $x \to z \leftarrow y$ is valid if and only if every undirected path between x and y contains a node in $N_{x,y}$.*

Corollary 17 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid application of the MakeV operator is:*

$$s(z, \Pi_z^{+y} \cup N_{x,y}^{-z+x}) + s(y, \Pi_y \cup N_{x,y}^{-z}) - s(z, \Pi_z \cup N_{x,y}^{-z+x}) - s(y, \Pi_y \cup N_{x,y})$$

We provide the proofs of these validity conditions and scores in the appendix. The proofs for each of the operators follow the same basic approach, depicted in Figure 9. Given the current state $\mathcal{P}^c$, we construct a consistent extension $\mathcal{G}$ such that the operator can be applied directly to $\mathcal{G}$, resulting in a DAG $\mathcal{G}'$ that is shown to be a consistent extension of the PDAG $\mathcal{P}'$ that results from applying the operator to $\mathcal{P}^c$. Most of the work involved in the proofs lies in showing that the validity conditions for each operator are necessary and sufficient; the local score for each operator is an immediate corollary of each of the corresponding validity proofs.

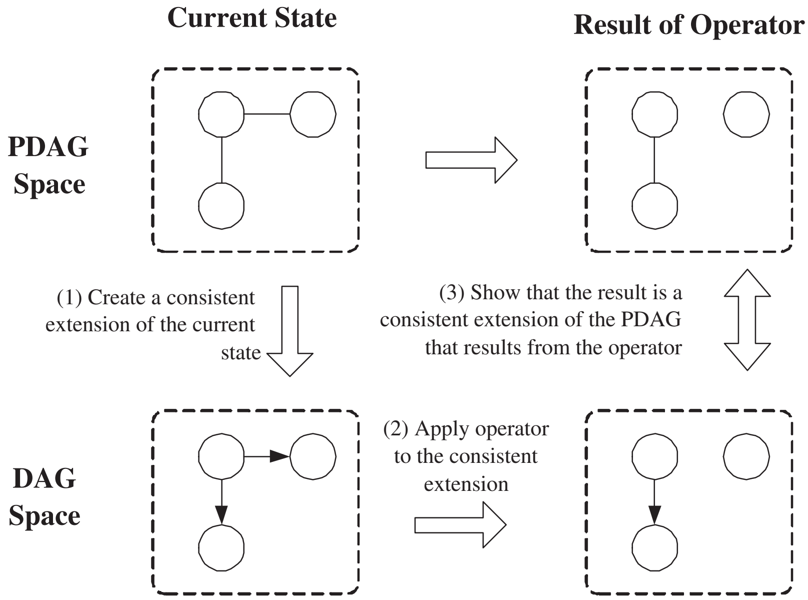

Figure 9: Approach taken to prove each of the operator results.

In Table 2 we summarize the results of the six theorems and the corresponding six corollaries. In particular we provide, for each of the E-space operators, both necessary and sufficient conditions for that operator to be valid, and the increase in score that results from applying that operator. The table, in conjunction with the two algorithms presented in Section 3, should be sufficient for practitioners to implement an efficient search algorithm in E-space.

We now consider how to efficiently test the "path" validity conditions. Consider a data structure for a PDAG that allows for random access to all of the nodes, and direct access to all adjacent nodes (i.e., parents, children and neighbors) of each node. We can test for the presence of semi-directed or undirected paths that do not pass through some set S as follows. First, we mark each node in S, and then we perform a depth-first search from the

Operator	Validity Tests	Change in Score
InsertU $x - y$	(1) Every undirected path from x to y contains a node in $N_{x,y}$ (2) $\Pi_x = \Pi_y$	$s(y, N_{x,y}^{+x} \cup \Pi_y) - s(y, N_{x,y} \cup \Pi_y)$
DeleteU $x - y$	$N_{x,y}$ is a clique	$s(y, N_{x,y} \cup \Pi_y) - s(y, N_{x,y}^{+x} \cup \Pi_y)$
InsertD $x \to y$	(1) Every semi-directed path from y to x contains a node in $\Omega_{x,y}$ (2) $\Omega_{x,y}$ is a clique (3) $\Pi_x \neq \Pi_y$	$s(y, \Omega_{x,y} \cup \Pi_y^{+x}) - s(y, \Omega_{x,y} \cup \Pi_y)$
DeleteD $x \to y$	N_y is a clique	$s(y, N_y \cup \Pi_y^{-x}) - s(y, N_y \cup \Pi_y)$
ReverseD $x \to y$	(1) Every semi-directed path from x to y that does not include the edge $x \to y$ contains a node in $\Omega_{y,x} \cup N_y$ (2) $\Omega_{y,x}$ is a clique	$s(y, \Pi_y^{-x}) + s(x, \Pi_x^{+y} \cup \Omega_{y,x})$ $- s(y, \Pi_y) - s(x, \Pi_x \cup \Omega_{y,x})$
MakeV $x \to z \leftarrow y$	Every undirected path between x and y contains a node in $N_{x,y}$	$s(z, \Pi_z^{+y} \cup N_{x,y}^{-z+x}) + s(y, \Pi_y \cup N_{x,y}^{-z})$ $- s(z, \Pi_z \cup N_{x,y}^{-z+x}) - s(y, \Pi_y \cup N_{x,y})$

Table 2: Necessary and sufficient validity conditions and (local) change in score for each operator in E-space

source node (following the appropriate links, depending on the type of path) looking for the destination node, where the search is not allowed to pass through any marked node (or any previously-visited node). This algorithm takes time $O(|S| + |E|)$ in the worst case, where $|E|$ is the number of edges in the graph. If the algorithm ever becomes a bottleneck, a search algorithm can postpone checking the path conditions until the operator is tentatively chosen. There is a tradeoff with this trick because typically search algorithms need the score of an operator to determine whether or not it is chosen; if we postpone the validity test, we may end up unnecessarily scoring invalid operators.

In the experiments presented in Section 6, the time taken to test the validity conditions for the operators was insignificant.

6. Experimental Results

In this section, we evaluate how well a greedy search algorithm performs, both in terms of solution quality and in terms of search time, when searching through the space of equivalence classes of structures. We compare these results to the same algorithm applied to the space of individual DAGs, in the context of model selection. For simplicity, we restrict our attention to domains in which all variables are discrete, but note that there exist score-equivalent scoring criteria for continuous variables.

Greedy search is a simple algorithm that, given the current state, always moves to the adjacent state that increases the score the most. If no adjacent state has a higher score, the algorithm terminates. In all of our experiments, we used greedy search to perform model selection; that is, we identify a single equivalence class with a high score.

We used a Bayesian scoring criterion for discrete variables in all of our experiments. The criterion, called the *BDeu* criterion, is derived by Heckerman et al. (1995).[5] The criterion measures the relative posterior probability that the distribution from which the data was generated has the independence constraints given in the DAG. The BDeu criterion uses a parameter prior that has uniform means, and requires both a prior equivalence sample size and a structure prior. For all of our experiments, we used a prior equivalent sample size of ten, and a structure prior of 0.001^f, where f is the number of free parameters in the DAG. Let q_i denote the number of configurations of the parent set Π_{x_i}, and let r_i denote the number of states of variable x_i. Then the version of the BDeu criterion used in our experiments is:

$$S_{BDeu}(\mathcal{G}, D) = \log \prod_{i=1}^{n} 0.001^{(r_i-1)q_i} \prod_{j=1}^{q_i} \frac{\Gamma(\frac{10}{q_i})}{\Gamma(\frac{10}{q_i} + N_{ij})} \cdot \prod_{k=1}^{r_i} \frac{\Gamma(\frac{10}{r_i \cdot q_i} + N_{ijk})}{\Gamma(\frac{10}{r_i \cdot q_i})} \tag{3}$$

where N_{ijk} is the number of records in D for which $x_i = k$ and Π_{x_i} is in the jth configuration, and $N_{ij} = \sum_k N_{ijk}$. $\Gamma(\cdot)$ is the *Gamma* function, which satisfies $\Gamma(y+1) = y\Gamma(y)$ and $\Gamma(1) = 1$. Note from Equation 3 that our scoring criterion is decomposable. Heckerman et al. (1995) show that it is score equivalent.

We performed our experiments using the following six real-world datasets.

1. **Microsoft Web Training Data (MSWeb)**

   This dataset, which is available via anonymous ftp from the UCI Machine Learning Repository, contains 32711 instances of users visiting the www.microsoft.com web site on one day in 1996. For each user, the data contains a variable indicating whether or not that user visited each of the 292 areas (i.e., "vroots") of the site.

2. **Nielsen**

   The Nielsen dataset contains data about television watching behavior during a two-week period in 1995. The data was made available courtesy of Nielsen Media Research. The data records whether or not each user watched five or more minutes of network TV shows aired during the given time period. There were 3314 users in the study, and 402 television shows.

5. The BDeu criterion is a version of the BDe criterion that is "uninformed" in the sense that the parameter prior is completely specified by a single equivalent sample size.

3. **EachMovie**

   The EachMovie dataset consists of viewer ratings on movies. The data was collected during an 18-month period beginning in 1995. We used the ratings of the 300 most popular movies by 61,265 viewers. The rating is a discrete variable that is either missing, or is provided as an integer from one to five.

4. **Media Metrix**

   This dataset contains demographic and internet-use data for 4808 individuals during the month of January 1997. We used only the internet-use variables in our experiments; there are 13 such variables that indicate the category of web site visited.

5. **1984 United States Congressional Voting Records (HouseVotes)**

   This dataset contains the 1984 congressional voting records for 435 representatives voting on 17 issues, and is available via anonymous ftp from the UCI Machine Learning Repository. Votes are all three-valued: yes, no, or unknown. For each representative, the political party is given; this dataset is typically used in a classification setting to predict the political party of the representative based on the voting record.

6. **Mushroom**

   The Mushroom dataset, available via anonymous ftp from the UCI Machine Learning Repository, contains physical characteristics of 8124 mushrooms, as well as whether each mushroom is poisonous or edible. There are 22 physical characteristics for each mushroom, all of which are discrete.

For all datasets, we assume that values are *not* missing at random. In particular, we treat "missing" as a distinct, discrete state.

Given a particular scoring function, the goal of the search algorithm in the context of model selection is to identify the single structure with the highest score. Consequently, the score of the best state found by the search algorithm is the best indicator of the algorithm's performance, and therefore we emphasize the score when we compare algorithm performance in the two spaces. Many researchers have compared scoring functions and search algorithms *together*. In this case, learning accuracy is most often measured in terms of a holdout score. To be consistent with previous work, we have included such a score for all of our experiments; our holdout score is simply the average log probability that the selected model assigns to each case in the holdout set.

Our experiments can be described as follows. Because we include a holdout score in our results, we first split each of the datasets into a training set and a test set, consisting of approximately 70 percent and 30 percent of the total data, respectively. Then, for each dataset, we applied a greedy search algorithm in both B-space and E-space, starting each search in the state for which all variables are mutually independent (i.e., the DAG/PDAG containing no edges). At each step in the search, we recorded the score achieved by the algorithm as well as the time taken so far.

In Figure 10, we give plots showing the score as a function of elapsed time for each operation application in both spaces. For all datasets, the search algorithm moves more quickly to higher-scoring equivalence classes in E-space. The most remarkable instance of

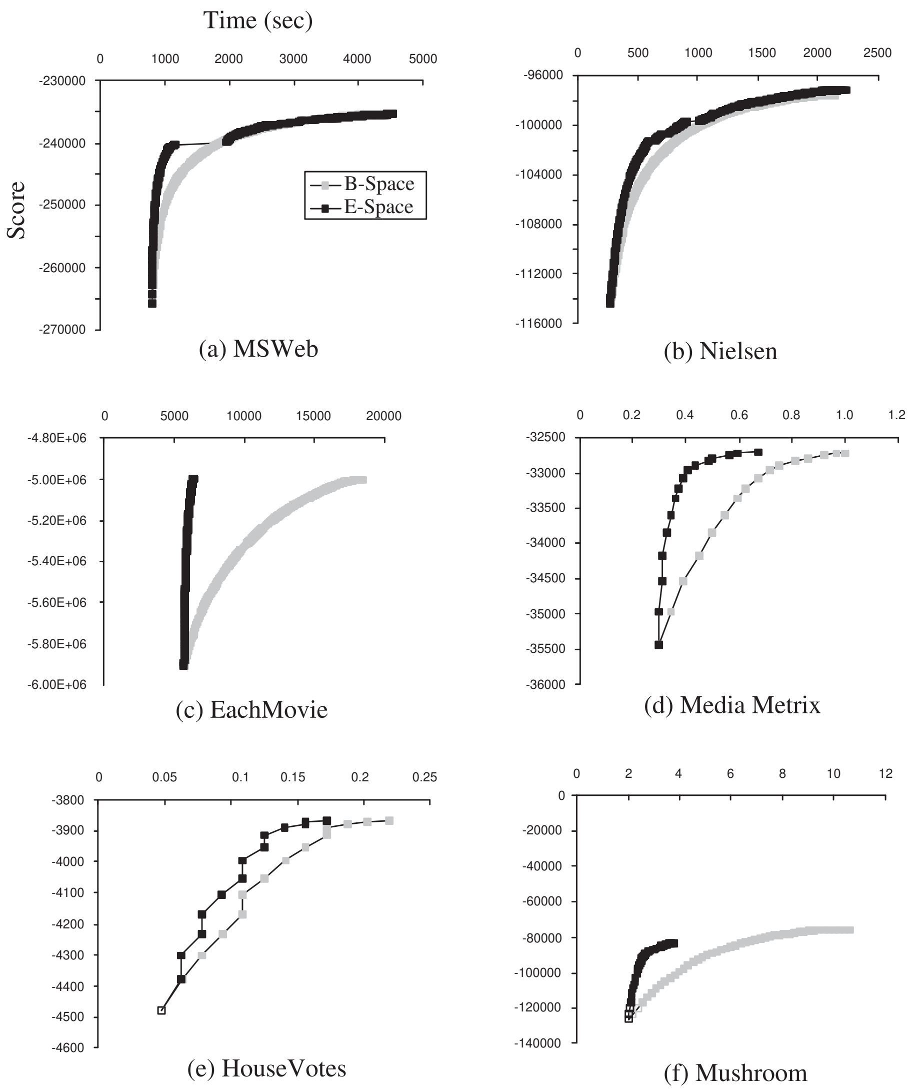

Figure 10: Score as a function of time for the six datasets.

this is in the EachMovie dataset, where the algorithm applied to E-space reaches a local maximum in about a third of the time than it does in B-space.

Of particular interest is the “gap” in the curve of Figure 10(a). The algorithm in E-space initially moves much more quickly than in B-space, but then “stalls” after 153 operator moves and the algorithm applied to B-space catches up. After further investigation, we found that in E-space a MakeV operator was applied on the 153rd operation, and the *majority* of the edges changed from being undirected to directed; consequently, a large number of operators that were previously not valid (i.e., directed edge insertions and directed edge reversals) became valid and thus needed to be scored. This is a problem with the fact that although all of the operators can be scored locally, the result of *applying* an operator

Dataset	B-space Score	E-space Score	Rel. Imp.	B-space Holdout	E-space Holdout	Rel. Imp.	Time Ratio
MSWeb	-10.2857	-10.2801	0.0005	-0.0339	-0.0339	0.0004	0.95
Nielsen	-42.0734	-41.9104	0.0039	-0.0911	-0.0909	0.0017	0.96
EachMovie	-116.6294	-116.4836	0.0013	-0.3745	-0.3763	-0.0047	2.902
Media Metrix	-9.7208	-9.7192	0.0002	-0.7278	-0.7265	0.0017	1.49
HouseVotes	-12.7228	-12.7228	0.0000	-0.6006	-0.6006	0.0000	1.27
Mushroom	-13.3674	-13.8971	-0.0382	-0.4873	-0.5563	-0.1241	2.81

Table 3: Results for the model selected by the greedy algorithm applied to both spaces.

is a non-local change to the equivalence class. As is evident from the figure, such a radical change as the one in Figure 10(a) is not common, but it is certainly a concern. Despite the unlucky large change in the equivalence class, the time to reach a local maximum remains competitive with the search in B-space.

In Table 3 we summarize the results from the local maximum reached by each of the algorithms. In the table we report, for each dataset, (1) the score per training case of the local maximum for each space, (2) the relative improvement of the per-case score in E-space to the per-case score in B-space, (3) the holdout score of the local maximum for each space, (4) the relative improvement of the holdout score in E-space to the holdout score in B-space, (5) the ratio of the time spent in B-space to the time spent in E-space. The relative improvements for the per-case scores and the holdout scores are computed as the score in E-space minus the score in B-space divided by the score in B-space. Relative improvements are reported as opposed to absolute differences so that results may be compared across the different domains.

In all but the Mushroom and HouseVotes datasets, the search in E-space found a better local maximum—that is, the search identified a model with a higher value of the scoring criterion—than did the search in B-space; the two methods identified the same model for the HouseVotes datasets. As shown by the holdout scores, the models with higher values of the particular scoring criterion we used in our experiments tended to have superior out-of-sample prediction accuracy as well. We should note, however, that the goal of the search algorithm is to identify the model with the highest possible score; it is somewhat misleading to compare *search algorithms* based on a predictive score because differences in this score are dependent on the quality of the scoring criterion.

For both the EachMovie dataset and the Mushroom dataset, the search in E-space found a local maximum almost three times faster than the search in B-space. The search in E-space was also significantly faster in both Media Metrics and HouseVotes. For MSWeb and Nielsen, the search in E-space was roughly five percent slower than the search in B-space. The reason the search algorithm can be faster in E-space is that it often has to score fewer operators. In particular, for a state containing an undirected edge in E-space, the only operator that the algorithm needs to consider for that edge is a deletion. When the same state is considered in B-space, however, the algorithm considers both a reversal and a deletion of the edge.

Dataset	Average Lower Bound	Maximum Lower Bound
MSWeb	$301,000$	$10,300,000$
Nielsen	9.7×10^{19}	7.1×10^{21}
EachMovie	$22,300$	$570,000$
Media Metrix	8.5	13
HouseVotes	8.6	15
Mushroom	11.5	22

Table 4: Average and maximum lower bound on the number of DAGs per equivalence class

To investigate the number of DAG representations for the equivalence classes that the greedy search visited in E-space, we counted, at each step of the algorithm, the number of nodes in each undirected component of the completed PDAG. We get a lower bound on the number of DAGs in the equivalence class by multiplying these counts together; every node in an undirected component can be selected as a root node in that component to yield *at least* one unique configuration of the corresponding directed edges in a consistent extension (see the appendix). If the undirected component is not a tree structure, there can be many more such configurations. In Table 4 we show, for each dataset, the average and maximum lower bound on the number of DAGs per equivalence class out of all states visited by the search algorithm applied to E-space.

As we see from the table, the number of DAGs per equivalence class in those states that we encounter during our greedy search can be enormous for some of the datasets, and is significant in all of them. This suggests that despite the fact that the ratio of DAGs to equivalence classes may be 3.7 when considering all possible equivalence classes defined for a set of nodes (see discussion in Section 2), it is important to consider the size of the particular equivalence classes that the search algorithm is likely to encounter.

7. Discussion

In this paper, we have argued that in many situations search algorithms should search over the set of equivalence classes of Bayesian-network structures instead of individual structures. We specified a search space that can be used by any heuristic search algorithm to explicitly search through the space of equivalence classes. We showed how all of our operators can be scored locally, and thus an algorithm searching our search space with a decomposable scoring criterion is able to evaluate adjacent states efficiently. We provided experimental evidence, using six real-world datasets, that suggest that a greedy search in our space can yield slightly better results in less time than the same search applied to the traditional space of DAGs.

The results in Section 6 do not provide overwhelming evidence that E-space should be favored over B-space when using greedy search. This is not too surprising: because a greedy search algorithm is being applied to B-space, the search is necessarily moving to a new equivalence class for each operator. Thus one of the major concerns with using B-space—namely, a loss of efficiency because the search wastes time traversing within an equivalence class—is not an issue. Furthermore, because our experiments were done in the context of model selection, the only real difference between the two spaces is connectivity.

Our experimental results do, however, demonstrate the efficiency of traversing through the space of equivalence classes. This suggests that we can get significant improvements in the running time of non-greedy search algorithms that can waste time in B-space traversing redundant representations of equivalence classes. Examples of such algorithms include Monte-Carlo sampling algorithms (e.g., Madigan et al., 1996), as well as any number of systematic search algorithms such as best-first search or limited discrepancy search. We hope that our contributions will lead to successful implementations of non-greedy search algorithms.

We saw in Section 6 that the (lower bound of the) number of redundant representations of equivalence classes visited by greedy search was *significantly* higher than the hypothesized 3.7 average across *all* equivalence classes for a given set of nodes. This result shows that the degree to which multiple representations will be a problem is highly dependent on the states visited by the search algorithm. We suspect that for most (systematic) search algorithms that spend the majority of time in reasonably sparse graphs, the average number of redundant representations will be on the same order as that encountered by the greedy search. It would be interesting to more rigorously pursue a characterization of those equivalence classes in which the number of DAGs members is small, and try to determine whether or not sequences of such classes are likely to be encountered by heuristic search algorithms.

An interesting extension to this work would be to derive an efficient set of operators for which the neighbor states of each equivalence class correspond to the *inclusion boundary* defined by Kocka, Bouckaert and Studeny (2001). The inclusion boundary of an equivalence class can be explained as follows. Consider the set of all DAGs contained within a particular equivalence class. To each such DAG, we can associate a set of equivalence classes that correspond to the DAGs that result from a single (directed) edge addition or deletion from that DAG. By taking the union, over all DAGs in the equivalence class, of these off-by-one equivalence classes, we get the inclusion boundary. The inclusion boundary was originally used by Meek (1997) in a search algorithm that—under the assumption that the "Meek Conjecture" is true—is optimal for infinite data.

Another interesting extension to this work would be to consider the case when we know the orientation of some of the edges in the "true" DAG model; this scenario can occur when learning causal models for which we know the direction of some of the causal influences. Meek (1995) considers this "prior knowledge" problem and derives rules for extracting consistent extensions of equivalence classes that are consistent with the known edges. Our operators would need to be modified so that only those equivalence classes that are consistent with the prior knowledge are considered during search.

Acknowledgments

I would like to thank Chris Meek, Dan Geiger, David Heckerman and Michael Perlman for useful discussions and suggestions. I would also like to thank the anonymous reviewers for their help with improving this paper.

References

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.

Andersson, S. A., Madigan, D., and Perlman, M. D. (1997). A characterization of Markov equivalence classes for acyclic digraphs. *Annals of Statistics*, 25:505–541.

Blair, J. R. S. and Peyton, B. W. (1993). An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computations*, pages 1–29.

Bouckaert, R. R. (1993). Probabilistic network construction using the minimum description length principle. In *Lecture Notes in Computer Science*, 747:41–48. Springer.

Buntine, W. L. (1991). Theory refinement on Bayesian networks. In D'Ambrosio, B. D., Smets, P., and Bonissone, P. P., editors, *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 52–60. Morgan Kaufmann.

Buntine, W. L. (1996). A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8:195–210.

Chickering, D. M. (1995). A transformational characterization of Bayesian network structures. In Hanks, S. and Besnard, P., editors, *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 87–98. Morgan Kaufmann.

Chickering, D. M. (1996a). Learning Bayesian networks is NP-Complete. In Fisher, D. and Lenz, H., editors, *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag.

Chickering, D. M. (1996b). Learning equivalence classes of Bayesian-network structures. In Horvitz, E. and Jensen, F., editors, *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 150–157. Morgan Kaufmann.

Chickering, D. M., Geiger, D., and Heckerman, D. (1995). Learning Bayesian networks: Search methods and experimental results. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 112–128.

Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467.

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

Dash, D. and Druzdzel, M. J. (1999). A hybrid anytime algorithm for the construction of causal models from sparse data. In Laskey, K. and Prade, H., editors, *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 142–149. Morgan Kaufmann.

Dor, D. and Tarsi, M. (1992). A simple algorithm to construct a consistent extension of a partially oriented graph. Technical Report R-185, Cognitive Systems Laboratory, UCLA Computer Science Department.

Druzdzel, M. and Simon, H. (1993). Causality in Bayesian belief networks. In Heckerman, D. and Mamdani, A., editors, *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 3–11. Morgan Kaufmann.

Gillispie, S. B. and Perlman, M. D. (2001). Enumerating Markov equivalence classes of acyclic digraph models. In Goldszmidt, M., Breese, J., and Koller, D., editors, *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 171–177. Morgan Kaufmann.

Heckerman, D. (1996). A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research.

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

Heckerman, D. and Shachter, R. (1995). Decision-theoretic foundations for causal reasoning. *Journal of Artificial Intelligence Research*, 3:405–430.

Jordan, M., editor (1998). *Learning in Graphical Models*, volume 89. Kluwer, Boston, MA, NATO ASI, Series D: Behavioural and Social Sciences edition.

Kocka, T., Bouckaert, R. R., and Studeny, M. (2001). On characterizing inclusion of Bayesian networks. In Breese, J. and Koller, D., editors, *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 261–268. Morgan Kaufmann.

Madigan, D., Andersson, S. A., Perlman, M. D., and Volinsky, C. T. (1996). Bayesian model averaging and model selection for Markov equivalence classes of acyclic digraphs. *Communications in Statistics - Theory and Methods*, 25:2493–2520.

Meek, C. (1995). Causal inference and causal explanation with background knowledge. In Hanks, S. and Besnard, P., editors, *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 403–410. Morgan Kaufmann.

Meek, C. (1997). *Graphical Models: Selecting causal and statistical models*. PhD thesis, Carnegie Mellon University.

Munteanu, P. and Bendou, M. (2001). The EQ framework for learning equivalence classes of Bayesian networks. In Cercone, N., Lin, T., and Wu, X., editors, *IEEE International Conference on Data Mining*, pages 417–424.

Munteanu, P. and Cau, D. (2000). Efficient score-based learning of equivalence classes of Bayesian networks. In *Lecture Notes in Computer Science*, 1910:96–105. Springer.

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

Pearl, J. and Verma, T. (1991). A theory of inferred causation. In Allen, J., Fikes, R., and Sandewall, E., editors, *Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, pages 441–452. Morgan Kaufmann, New York.

Rissanen, J. (1986). Stochastic complexity and modeling. *The Annals of Statistics*, 14(3):1080–1100.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6:461–464.

Spirtes, P., Glymour, C., and Scheines, R. (1993). *Causation, Prediction, and Search.* Springer-Verlag, New York.

Spirtes, P. and Meek, C. (1995). Learning Bayesian networks with discrete variables from data. In Fayyad, U. and Uthurusamy, R., editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 294–299. AAAI Press.

Suzuki, J. (1993). A construction of Bayesian networks from databases based on an MDL principle. In Heckerman, D. and Mamdani, A., editors, *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 266–273. Morgan Kaufmann.

Tarjan, R. and Yannakakis, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing*, 13:566–579.

Verma, T. and Pearl, J. (1990). Equivalence and synthesis of causal models. In Henrion, M., Shachter, R., Kanal, L., and Lemmer, J., editors, *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 220–227.

Appendix A. Detailed Proofs

In this section, we provide detailed proofs of the main results presented in this paper. In Section A.1, we provide some additional definitions and simple results that will be used frequently throughout the remainder of the proofs to follow. As shown in Figure 9 of Section 5, our approach to proving that each operator can be scored locally is to construct a consistent extension $\mathcal{G}$ of the current completed PDAG such that the operator can be applied directly to $\mathcal{G}$. Proving that such a consistent extension exists is non-trivial: in Section A.2 we prove various general results about consistent extensions that allow us to guarantee that, for each operator, a consistent extension with the desired properties exist.

Recall from Section 4 that we place an additional requirement on the insert operators: we only allow a directed (undirected) edge addition if the edge is compelled (reversible) in the resulting equivalence class. This property is important because it disambiguates the result of the operator for scoring purposes. Detecting whether or not an inserted edge is compelled or not turns out to be trivial: an addition between nodes x and y will be compelled if and only if $\Pi_x \neq \Pi_y$. Proving that this test is correct, however, was not trivial. In Section A.3, we provide the proof that this simple test is correct.

In Section A.4 through Section A.9, we use the results from Section A.2 and Section A.3 to prove, for each operator, that the (necessary and sufficient) validity conditions and increases in score given in Table 2 are correct. Finally, in Section A.10, we prove Theorem 4; that is, we prove that the operators are complete.

A.1 Definitions and Preliminary Results

We use the following definitions throughout the proofs. We say a directed edge $x \to y$ is *covered* if $\Pi_x = \Pi_y \setminus x$. Similarly, we say an undirected edge is covered if $\Pi_x = \Pi_y$. A node x is *higher* than a node y in a DAG or PDAG if there is a directed path from x to y.

The following two results, which apply to PDAGs and therefore also DAGs, provide simple tests for determining that an edge must be compelled, based on other edges.

Proposition 18 *Let $\mathcal{P}$ be any PDAG that admits a consistent extension and contains a compelled edge $x \to y$. If there is an edge, either directed or undirected, between y and some node z such that z and x are not adjacent, then that edge is compelled.*

Proof: Any consistent extension of $\mathcal{P}$ containing $y \leftarrow z$ contains a v-structure that cannot exist in any consistent extension containing $y \to z$, and it follows immediately from Theorem 1 that the edge therefore cannot be reversible. □

Proposition 19 *Let $\mathcal{P}$ be any PDAG that admits a consistent extension such that there is a directed path from x to y consisting of compelled edges. If there is an edge between x and y, it is compelled as $x \to y$.*

Proof: Follows because the directed path from x to y must also exist in any consistent extension, and because consistent extensions must be acyclic. □

The next lemma was proved by Chickering (1995). We use the result so frequently that we find it convenient to refer to the lemma as the *Triangle Lemma.*

Lemma 20 (Chickering, 1995) *Let $\{x, y, z\}$ be any three nodes that form a clique of size three in PDAG $\mathcal{P}$. If any two of the edges in the clique are reversible, then the third edge is reversible as well.*

Some of the operators include the condition that either $N_{x,y}$ or $\Omega_{x,y}$ form a clique. A result of the Triangle Lemma is that for any pair of nodes that both neighbors of some node, if those nodes are adjacent then they must be connected by an undirected edge. Consequently, the condition that these sets form a clique is equivalent to the condition that these sets form a clique consisting entirely of undirected edges.

Lemma 21 *If $x - y$ is an undirected edge in a completed PDAG, then $\Pi_x = \Pi_y$.*

Proof: Suppose not. Without loss of generality, let z be any parent of x that is not a parent of y. The nodes z and y must be adjacent, lest $x - y$ would be compelled (and hence directed) by Proposition 18. If $y \to z$, then $x - y$ would be compelled (and hence directed) by Proposition 19. By the Triangle Lemma (Lemma 20), no undirected edge can exist between z and y. □

Lemma 21 also follows from Condition (iii) in Theorem 4.1 of Andersson et al. (1997).

Lemma 22 *For any directed edge $x \to y$ in a completed PDAG, x is a parent of every node reachable by y via undirected edges.*

Proof: Follows by a repeated application of Lemma 21 along the edges in any undirected path. □

Lemma 23 *If there is a directed path of length one or more from x to y in a completed PDAG $\mathcal{P}^c$, then x and y are not in the same undirected component.*

Proof: Suppose not. Consider the last edge $z \rightarrow y$ in the directed path. By Lemma 22, z is a parent of every node in the undirected component containing y, which means that z is a parent of x. Because there is a directed path from x to z, $\mathcal{P}^c$ contains a cycle, contradicting the fact that it is a completed PDAG. □.

Lemma 23 also follows from the fact that, as proved by Andersson et al. (1997), a completed PDAG is a chain graph.

Lemma 24 *Let $\mathcal{P}^c$ denote a completed PDAG, and let $\mathcal{P}'$ denote the PDAG that results from adding a single edge (directed or undirected) between x and y to $\mathcal{P}^c$. Consider any consistent extension $\mathcal{G}$ of $\mathcal{P}^c$, and the directed graph $\mathcal{G}'$ that results by inserting a directed edge between x and y such that the result is a DAG with the same adjacencies as $\mathcal{P}'$. Any v-structure in $\mathcal{G}'$ but not in $\mathcal{P}'$, or any v-structure in $\mathcal{P}'$ but not in $\mathcal{G}'$, must include the edge between x and y.*

Proof: Let $a \rightarrow b \leftarrow c$ be any v-structure in $\mathcal{G}'$ but not in $\mathcal{P}'$, and assume that neither edge is $x \rightarrow y$ or $y \rightarrow x$. Because a and c are not adjacent in $\mathcal{G}'$, they must also not be adjacent in $\mathcal{G}$ because $\mathcal{G}$ contains a strict subset of the adjacencies of $\mathcal{G}'$. Because $\mathcal{G}$ and $\mathcal{P}^c$ have the same adjacencies, and $\mathcal{G}'$ and $\mathcal{P}'$ have the same adjacencies, a and c are not adjacent in any of the four graphs. Because neither edge in the v-structure is between x and y, this v-structure must exist in $\mathcal{G}$, and consequently the edges are compelled and the v-structure exists in $\mathcal{P}^c$. Because a and c are not adjacent in $\mathcal{P}'$, this v-structure remains after adding the edge between x and y, yielding a contradiction.

Let $a \rightarrow b \leftarrow c$ be any v-structure in $\mathcal{P}'$ but not in $\mathcal{G}'$, and assume that neither edge is the one that was added to $\mathcal{P}^c$. This means that the v-structure existed in $\mathcal{P}^c$, and consequently the edges are compelled and the v-structure exists in $\mathcal{G}$. Because a and c are not adjacent in $\mathcal{P}'$, they cannot be adjacent in $\mathcal{G}'$, and thus $\mathcal{G}'$ must contain the v-structure, yielding a contradiction. □

Lemma 25 *Let $\mathcal{P}^c$ denote a completed PDAG, and let $\mathcal{P}'$ denote the PDAG that results from deleting a single edge (directed or undirected) between x and y from $\mathcal{P}^c$. Consider any consistent extension $\mathcal{G}$ of $\mathcal{P}^c$, and the directed graph $\mathcal{G}'$ that results by deleting the (directed) edge between x and y in $\mathcal{G}$ such that the result is a DAG with the same adjacencies as $\mathcal{P}'$. Any v-structure in $\mathcal{G}'$ but not in $\mathcal{P}'$, or any v-structure in $\mathcal{P}'$ but not in $\mathcal{G}'$, must be of the form $x \rightarrow z \leftarrow y$.*

Proof: Let $a \rightarrow b \leftarrow c$ be any v-structure in $\mathcal{G}'$ but not in $\mathcal{P}'$, and assume that $\{a, c\}$ is not $\{x, y\}$. By definition of a v-structure, a and c are not adjacent in $\mathcal{G}'$, and because the edges in $\mathcal{G}$ are the same as in $\mathcal{G}'$ except for the extra edge between x and y, $a \rightarrow b \leftarrow c$ must exist in $\mathcal{G}$. This implies the edges are compelled and exist in $\mathcal{P}^c$. Because $\mathcal{P}'$ has the same adjacencies as $\mathcal{G}'$, these edges were unaffected by the deletion of the edge between x and y from $\mathcal{P}^c$, and consequently $a \rightarrow b \leftarrow c$ must exist in $\mathcal{P}'$, yielding a contradiction.

Let $a \rightarrow b \leftarrow c$ be any v-structure in $\mathcal{P}'$ but not in $\mathcal{G}'$, and assume that $\{a, c\}$ is not $\{x, y\}$. Because only the edge between x and y was deleted, a and c are not adjacent in $\mathcal{P}^c$. Furthermore, because the edges in $\mathcal{P}'$ are the same as in $\mathcal{P}^c$ except for the absence of the edge between x and y, $a \rightarrow b \leftarrow c$ must exist in $\mathcal{P}^c$. This implies that the v-structure must also exist in $\mathcal{G}$. Because the v-structure exists in $\mathcal{P}'$, neither edge participating in the v-structure was changed by the deletion in either $\mathcal{P}^c$ or $\mathcal{G}$, and thus the v-structure exists in $\mathcal{G}'$, yielding a contradiction. $\square$

A.2 Extracting Consistent Extensions from Completed PDAGs

In this section, we first describe an algorithm that is a simple modification to the well-known *maximum cardinality search* of Tarjan and Yannakakis (1984) for determining whether or not a graph is chordal, and if so, identifying a consistent extension of that graph. We then show how we can apply this algorithm to extract consistent extensions from completed PDAGs, such that these consistent extensions have useful properties for the purpose of proving our main results.

We first need some definitions. A graph is *chordal* if every undirected cycle of length at least four has a chord; that is, the cycle contains a "short cut" undirected edge that connects two non-consecutive nodes in the cycle. A *maximal undirected component* K of a completed PDAG $\mathcal{P}^c$ is a connected subgraph of $\mathcal{P}^c$, where each connecting edge is undirected, and for every node $x \in K$, if there is an undirected edge $y - x$ in $\mathcal{P}^c$, then y is in K. For the remainder of this paper, we use *undirected component* to mean maximal undirected component. We say that y is a *reversible parent* of x in a PDAG or DAG if the edge $y \rightarrow x$ is reversible. Similarly, we say that y is a *compelled parent* of x if $y \rightarrow x$ is compelled.

The following result was proved by both Meek (1995) and Andersson, Madigan and Perlman (1997):

Theorem 26 *The undirected components of a completed PDAG are chordal.*

In the following lemma, we show that we can extract a consistent extension from a completed PDAG by independently extracting consistent extensions from the chordal components of that PDAG.

Lemma 27 *Let $\mathcal{P}^c$ be any completed PDAG and let $\mathbf{K} = \{K_1, ..., K_m\}$ be the set of undirected components of $\mathcal{P}^c$. For any set $\{\mathcal{G}_1, ..., \mathcal{G}_m\}$, where $\mathcal{G}_i$ is any consistent extension of the undirected component K_i, the graph $\mathcal{G}$ that results from replacing each reversible edge in $\mathcal{P}^c$ with the directed edge from the consistent extension of the corresponding undirected component is a consistent extension of $\mathcal{P}^c$.*

Proof: The graph $\mathcal{G}$ clearly has the same skeleton as $\mathcal{P}^c$. Any v-structure in $\mathcal{G}$ that is not in $\mathcal{P}^c$ must contain one reversible edge $x \rightarrow y$ and one compelled edge $y \leftarrow z$ from $\mathcal{P}^c$ because none of the chordal components contain compelled edges. But this is impossible by Proposition 18. Any v-structure in $\mathcal{P}^c$ also exists in $\mathcal{G}$ because $\mathcal{G}$ contains the same compelled edges and adjacencies as $\mathcal{P}^c$.

It remains to be shown that $\mathcal{G}$ is acyclic. Suppose there exists a cycle. Any cycle must include both reversible and compelled edges from $\mathcal{P}^c$ because (1) each $\mathcal{G}_i$ is individually acyclic, (2) any path that contains edges from two separate $\mathcal{G}_i$ and $\mathcal{G}_j$ must pass through at least one directed edge, and (3) there cannot be a directed cycle among the compelled edges in $\mathcal{P}^c$. Consider the *shortest* such cycle, and let $y \to z$ be any edge in the cycle that is reversible in $\mathcal{P}^c$ such that the previous edge $x \to y$ is compelled in $\mathcal{P}^c$. We know that x and z must be adjacent in both $\mathcal{P}^c$ and $\mathcal{G}$, else by Proposition 18, $y \to z$ could not be reversible. We know by the Triangle Lemma (Lemma 20) that the edge between z and x must be compelled. If $x \to z$, then there is a shorter cycle, contradicting the fact that we have considered the shortest. If $x \leftarrow z$, then there is a directed path from z to y in $\mathcal{P}^c$, which implies by Proposition 19 that the edge between y and z must be compelled, yielding a contradiction. $\square$

Tarjan and Yannakakis (1984) provide an algorithm that can be used to extract a consistent extension from any (chordal) undirected component of a completed PDAG. The algorithm proceeds as follows: number the nodes in the component from n to 1 in decreasing order, always selecting next a node adjacent to the largest number of previously numbered vertices, breaking ties arbitrarily. Tarjan and Yannakakis (1984) show that if a component is chordal, which every undirected component in $\mathcal{P}^c$ must be, then for any pair of non-adjacent nodes x and y, every path between x and y contains a node that has a higher label than either x or y (or both).

Given a total ordering τ, we say an undirected edge $x - y$ is *directed to be consistent with* τ if it is directed $x \to y$ if $\tau(x) > \tau(y)$ and is directed $y \to x$ otherwise.

A consequence of the Tarjan and Yannakakis (1984) result is that by directing all edges in an undirected component to be consistent with the total ordering from a maximum cardinality search, where higher ordered nodes always proceed lower ordered nodes, it is impossible to create a v-structure. Furthermore, because the edges are directed to be consistent with a total ordering, there can be no cycle in the resulting component. We emphasize this result with the following lemma. We use *MCS-ordering* to denote a total ordering that can be obtained by a maximum cardinality search.

Lemma 28 *Let $\{\tau_1, ..., \tau_m\}$ denote a set of MCS-orderings over the nodes in the m undirected components $K_1, ..., K_m$ of a completed PDAG $\mathcal{P}^c$. The DAG $\mathcal{G}$ that results from directing each undirected edge within K_i in $\mathcal{P}^c$ to be consistent with the corresponding ordering τ_i is a consistent extension of $\mathcal{P}^c$.*

Proof: Follows immediately from Lemma 27 and from the fact that each τ_i yields a consistent extension of the undirected component K_i. $\square$

Lemma 28 leads to an efficient algorithm—shown in Figure 11—for extracting a consistent extension of a completed PDAG. Note that this algorithm takes as input a *completed* PDAG; it does not work for PDAGs in general, and thus cannot be used in place of algorithm PDAG-TO-DAG from Section 3.

Lemma 29 *Algorithm DAG-MCS returns a valid consistent extension of the completed PDAG.*

Proof: Clearly, the algorithm identifies an MCS-ordering within each of the undirected components: the number of parents from K for each node at step 4 is precisely the number of previously ordered adjacent nodes from the maximum cardinality search algorithm. By

Algorithm DAG-MCS($\mathcal{P}^c$)
Input: completed PDAG $\mathcal{P}^c$
Output: DAG $\mathcal{G}$ that is a consistent extension of $\mathcal{P}^c$

1. Set $\mathcal{G} = \mathcal{P}^c$
2. For each undirected component K in $\mathcal{P}^c$
3. Mark every node in K as "unprocessed"
4. **While** there are unprocessed nodes in K
5. Select the unprocessed node x with the most parents that are in K, breaking ties arbitrarily
6. Direct all undirected edges incident to x away from x in $\mathcal{G}$.
7. Mark x as processed

Figure 11: Algorithm DAG-MCS which extracts a consistent extension from a completed PDAG.

directing the incident undirected edges away from a node whenever that node is selected in step 4, the resulting component edges are consistent with the MCS-ordering. □

As we see below, the "breaking ties arbitrarily" step in the maximum cardinality search algorithm allows us some flexibility in choosing the edge directions in a consistent extension. We use *undirected clique* to denote a clique where every edge is undirected.

Lemma 30 *Let $X = \{x_1, ..., x_n\}$ be the nodes from any undirected clique of size n within some undirected component K of a completed PDAG $\mathcal{P}^c$, and let τ denote any total ordering of the nodes in X. There exists a consistent extension of $\mathcal{P}^c$ for which (1) the edge orientations among the nodes in X are consistent with τ, and (2) any edge between x_i and a node $y \in K$ that is not in X is oriented as $x_i \rightarrow y$.*

Proof: Consider the application of the DAG-MCS algorithm to K. When all nodes from K are unprocessed, we select $\tau(1)$ in step 4. That is, we break the initial zero-parents-from-K tie among all nodes in K by choosing $\tau(1)$. Because the nodes in X form a clique, as long as we continue to process nodes within X at step 4, all unprocessed nodes in X will have the same maximum number of parents from within K; consequently when selecting the *ith* node from K at step 4, we can always break the tie in favor of $\tau(i)$, and thus condition (1) holds. After processing all of the nodes in X, all undirected edges incident to every node $x \in X$ will be directed away from x, and thus condition (2) holds.

Lemma 31 *Let $\mathcal{P}^c$ be a completed PDAG that admits a consistent extension, and let x and y be any pair of nodes that are NOT adjacent. Then $N_{x,y}$ is a clique of undirected edges.*

Proof: Let w and z be any pair of common neighbors in $N_{x,y}$ that are not connected by an undirected edge. From the Triangle Lemma (Lemma 20), we know that w and z cannot be adjacent. But this implies that $x - z - y - w - x$ is a four-cycle in an undirected component of $\mathcal{P}^c$ without a chord, which contradicts the fact that the undirected components of $\mathcal{P}^c$ are chordal. $\Box$

The following lemma uses the results developed in this section to show how to construct a particular consistent extension that will be used later to prove the validity conditions and local score for an edge insertion. Recall that a reversible parent y of a node x is a parent for which the edge $y \rightarrow x$ is reversible.

Lemma 32 *Let $\mathcal{P}^c$ be any completed PDAG, and let x and y be any pair of nodes that are NOT adjacent. Every undirected path between x and y passes through a node in $N_{x,y}$ if and only if there exists a consistent extension in which (1) x has no reversible parents, (2) all nodes in $N_{x,y}$ are parents of y, and (3) y has no other reversible parents.*

Proof (only if): We first show that all three conditions hold if every path between x and y passes through a node in $N_{x,y}$. Conditions (1) and (2) hold for any completed PDAG: from Lemma 31, the set $N_{x,y}$ forms a clique of undirected edges in $\mathcal{P}^c$, and because every node in $N_{x,y}$ is a neighbor of x, the set $\{x\} \cup N_{x,y}$ is a clique as well. Thus by Lemma 30, we can construct a consistent extension where within the component K containing $N_{x,y}$, x is first (a root node in K), all nodes in $N_{x,y}$ are next, and all nodes in $N_{x,y}$ are parents of y. Note that if $N_{x,y}$ is empty, (2) holds trivially. It remains to be shown that there exists such a consistent extension where y has no other reversible parents (condition 3).

Suppose that no such consistent extension exists, and let $\mathcal{G}$ be any consistent extension constructed as described above, using Algorithm DAG-MCS. Let $A \subseteq K$ denote the set of ancestors of y in $\mathcal{G}$ that are contained in the undirected component K but are not contained in $N_{x,y} \cup x$. From Lemma 23, it follows that for every edge e in a directed path from a node in A to y in $\mathcal{G}$, e is reversible in $\mathcal{P}^c$. This means that when component K was being processed by the algorithm, every node in A was chosen in step 4 before y. Consider the step when the *first* node a_f from A was chosen by the algorithm when processing component K. The only processed parents of a_f from K at this point could be (1) the members of $N_{x,y}$ or (2) the node x; otherwise, such a parent belongs to A, contradicting the fact that a_f is the first one chosen. If a_f has x as a processed parent, this means there's an undirected path from x to y that does not pass through a node in $N_{x,y}$, which is a contradiction. Thus the only processed parents of a_f can belong to $N_{x,y}$, and therefore a_f had less than or equal to $|N_{x,y}|$ processed parents from K. If a_f had less than $|N_{x,y}|$ processed parents from K, y would have been chosen instead of a_f by the algorithm. Otherwise, we could have broken the tie between a_f and y in favor of y in step 4 of the algorithm, in which case all incident undirected edges would have been directed away from y.

(if): We now show that if there exists an undirected path between x and y that does not pass through $N_{x,y}$, then at least one of the three conditions does not hold.

Let $x - a_1 - \ldots - a_k - y$ denote the *shortest* undirected path between x and y such that for each a_i, $a_i \notin N_{x,y}$. We now show that in any consistent extension of $\mathcal{P}^c$ for which x has no reversible parents (condition 1) and all members of $N_{x,y}$ are parents of y (condition 2), the last edge $a_k - y$ in the path is directed into y, thus violating condition 3.

Suppose there exists a consistent extension where the last edge is oriented as $a_k \leftarrow y$. Let $a \rightarrow b$ be the edge in the path *closest* to y—that is, fewest edges away from y along the path—that is oriented in the direction of y. We know this edge must exist because x has no reversible parents. Because $a \rightarrow b$ is the closest edge to y in the path with this orientation, we know the next edge is oriented as $b \leftarrow c$. We know a and c must be adjacent, lest there would be the v-structure $a \rightarrow b \leftarrow c$ containing reversible edges. By the Triangle Lemma, the edge connecting a and c must be reversible in $\mathcal{P}^c$. Because x an y are not adjacent, at least one of a or c must be an intermediate node (i.e., not equal to x or y) along the path. But this means there is a shorter path between x and y, yielding a contradiction. $\Box$

Corollary 33 *If there exists a consistent extension satisfying the three conditions of Lemma 32, then there exists such a consistent extension for which (1) the reversible parents of each node in $N_{x,y}$ are contained in $\{x \cup N_{x,y}\}$, and (2) the edges among the nodes in $N_{x,y}$ are oriented to be consistent with any total ordering over those nodes.*

Proof: Both conditions follows from the construction of $\mathcal{G}$ in the "only if" part of Lemma 32. Condition (1) follows immediately. Condition (2) follows from the fact that $\mathcal{G}$ is constructed without specifying the order of the nodes in $N_{x,y}$, and by Lemma 30 which guarantees that the edges can be oriented to be consistent with any total ordering. $\Box$

A.3 Compelled and Reversible Edge Insertions

In this section, we prove the following theorem.

Theorem 34 *Let $\mathcal{P}^c$ be any completed PDAG for which nodes x and y are not adjacent. If the result of adding an edge between x and y is a PDAG that admits a consistent extension, then that edge is compelled if and only if $\Pi_x \neq \Pi_y$.*

This result is important for determining which of the two insertion operators (InsertU or InsertD) is valid for a given completed PDAG. Unfortunately, although the result is simple enough to state and test, we have not found a *simple* proof for the result. Our approach to proving Theorem 34 relies on the results of a particular rule-based method (see Section 3) for extracting completed PDAGs from DAGs.

Meek (1995) derives three rules that can be used to identify the completed PDAG corresponding to a DAG. In particular, all edges of a DAG are first made to be undirected, except for those edges that participate in a v-structure. Then, the algorithm repeatedly chooses one of the three rules, and applies that rule to direct one of the undirected edges. The algorithm terminates when none of the three rules can be applied.

The three rules are as follows:

1. If $w \rightarrow x - y$ with w, y not adjacent, then direct the edge between x and y as $x \rightarrow y$
2. If $x \rightarrow z \rightarrow y$ and $x - y$, then direct the edge between x and y as $x \rightarrow y$.
3. If there is a v-structure $w \rightarrow y \leftarrow z$ and a node x such that x is a neighbor of w, y, and z, then direct the edge between x and y as $x \rightarrow y$.

In the proofs to follow, we refer to the rules above as Rule 1, Rule 2, and Rule 3, respectively. By our Proposition 18, we see that Rule 1 correctly orients undirected edges, as long as all previously oriented edges are compelled. Similarly, Rule 2 follows by an argument similar to our Proposition 19. Rule 3 can be understood by considering all possible orientations of the undirected edges in the configuration; only those containing $x \to y$ are a consistent extension of the configuration subgraph.

Meek (1995) proves that (1) the three rules are sound, and (2) they are complete given that the edges participating in v-structures are directed. The result is that not only are we guaranteed that the procedure described above will identify a completed PDAG, but that it will do so without regard to the order in which compelled edges are identified. This will prove important below; we will consider applying the rules to PDAGs in which edges that are known to be compelled are directed before the rules are applied to create the completed PDAG.

One reason that Theorem 34 is difficult to prove is that after inserting edges into a completed PDAG, edges that were reversible before can become compelled, and edges that were compelled before can become reversible (as an example, see the operator shown in Figure 7). In the following lemmas and corollaries, we demonstrate a method for determining which edges in a completed PDAG will necessarily remain compelled and reversible after an edge addition, based on the topology of the completed PDAG.

Proposition 35 *Let $\mathcal{P}^c$ be a completed PDAG, and let $\mathcal{P}'$ denote the PDAG that results from an edge addition or deletion. If $\mathcal{P}'$ admits a consistent extension, then any compelled edge $x \to y$ in $\mathcal{P}^c$ cannot be compelled in the opposite direction in $\mathcal{P}'$.*

Proof: Follows because any consistent extension of $\mathcal{P}'$ that contains an edge between x and y has that edge oriented as $x \to y$. $\square$

Lemma 36 *Let $\mathcal{P}^c$ be any completed PDAG, and let $\mathcal{P}'$ denote the PDAG that results from adding a new adjacency between a and b. For any edge $x \to y$ in $\mathcal{P}^c$ that is compelled in $\mathcal{P}^c$ but is reversible in $\mathcal{P}'$, there is a directed path of length zero or more from both a and b to y in $\mathcal{P}^c$.*

Proof: Suppose this is not the case. Consider any ordered set of rules applied by the Meek (1995) procedure to orient the edges in $\mathcal{P}^c$, staring with all edges undirected except for those participating in a v-structure. Let $x \to y$ be the *first* edge whose corresponding rule does not apply after the addition, such that y is not a descendant of a and b in $\mathcal{P}^c$.

We now consider each of the three rules that could have been used to direct $x \to y$ in $\mathcal{P}^c$. Suppose it was Rule 1. Because the rule does not match after the addition, either the edge $w \to x$ from the rule is no longer directed, or w and y are now adjacent (or both). If $w \to x$ is no longer directed, we know that x is a descendant of both a and b because $x \to y$ is the first (now undirected) edge for which this property does not hold. But because y is a descendant of x in $\mathcal{P}^c$, it must also be a descendant of both a and b, and we conclude that $w \to x$ must still be directed. The only new adjacency is the edge between a and b, and therefore (without loss of generality) $a = w$ and $b = y$. Because there is a directed path from w to y in $\mathcal{P}^c$, there is a directed path of length zero or more from both a and b to y, yielding a contradiction.

Suppose $x \to y$ was directed by Rule 2. Because the rule no longer applies, either $x \to z$ or $z \to y$ (or both) is no longer directed. In either case, because both z and y are ancestors of y, it follows that y must be a descendant of both a and b, yielding a contradiction. Thus $x \to y$ could not have been directed by Rule 3.

Suppose $x \to y$ was directed by Rule 3. The only way in which this rule can no longer apply to (undirected) $x - y$ is if w and z are adjacent after the addition; it is easy to show that if either of the other two undirected edges in the configuration are previously directed, the remaining edges will be directed by a series of applications of Rule 1 and Rule 2. This implies that (without loss of generality) $a = w$ and $b = z$. But in $\mathcal{P}^c$, both w and z are parents of y, and therefore there is a directed path from both a and b to y. We conclude that $x \to y$ could not have been directed by Rule 3.

Because $x \to y$ was not directed by any of the three rules, it must participate in a v-structure in $\mathcal{P}^c$ that no longer exists as a result of the addition of the edge between a and b. But clearly this implies that a and b are both parents of y. Thus no edge $x \to y$ can exist, and the lemma follows. $\Box$

The following corollary is essentially a re-statement of Lemma 36, except that we use the topology of the *result* of the edge addition.

Corollary 37 *Let $\mathcal{P}^c$ be any completed PDAG, and let $\mathcal{P}'$ denote the PDAG that results from adding a new adjacency between a and b. For any edge $x \to y$ in $\mathcal{P}^c$ that is compelled in $\mathcal{P}^c$ but is reversible in $\mathcal{P}'$, there is a directed path of length zero or more from both a and b to y in $\mathcal{P}'$ that does not pass through the edge between a and b.*

Proof: Follows immediately from Lemma 36 and the fact that every directed edge in $\mathcal{P}^c$ is also directed in $\mathcal{P}'$. $\Box$

Corollary 38 *Let $\mathcal{P}^c$ be any completed PDAG, and let $\mathcal{P}'$ denote the PDAG that results from adding the directed edge $a \to b$. For any compelled edge $x \to y$ in $\mathcal{P}^c$ such that there a directed path of length zero or more from both x and y to b in $\mathcal{P}'$, either $y = b$ or that edge remains compelled as $x \to y$ in $\mathcal{P}'$.*

Proof: Suppose not. By Lemma 36, there is a directed path from both a and b to y in $\mathcal{P}^c$. If $b \neq y$, this implies that $\mathcal{P}'$ is cyclic because there is a directed path from y to b. $\Box$.

Lemma 39 *Let $\mathcal{P}^c$ be any completed PDAG, and let $\mathcal{P}'$ denote the PDAG that results from adding the directed edge $a \to b$. If there exists a compelled edge $x \to b$ in $\mathcal{P}^c$ that is not compelled in $\mathcal{P}'$, then either (1) there exists a v-structure $a \to b \leftarrow z$ in $\mathcal{P}'$, or (2) there exists a length two directed path $a \to z \to b$ in $\mathcal{P}^c$ such that $a \to z$ remains compelled in $\mathcal{P}'$.*

Proof: From Corollary 37, there exists a directed path in $\mathcal{P}'$ from a to b that does not pass through the edge $a \to b$. Let $z \to b$ be the last edge in this path. If z is not adjacent to a, then the v-structure $a \to b \leftarrow z$ exists in $\mathcal{P}'$. If z and a are adjacent, then the edge must be directed as $a \to z$ in both $\mathcal{P}^c$ and $\mathcal{P}'$ because there is a directed path from a to z in $\mathcal{P}^c$. This edge and the edge $z \to b$ together constitute the length two directed path. Because $z \neq b$, it follows from Corollary 38 that $a \to z$ is compelled in $\mathcal{P}'$. $\Box$

The following lemma complements Lemma 21 presented in the previous section.

Lemma 40 *If $x \to y$ is a directed edge in a completed PDAG, then the edge is not covered.*

Proof: If the edge participates in a v-structure, $x \to y \leftarrow z$, then z is a parent of y that is not a parent of x. If the edge does not participate in a v-structure, then it can be directed by applying one of the three rules at some point in the Meek (1995) procedure for constructing a completed PDAG.

If Rule 1 applied, then there is a node z that is a parent of x that is not adjacent to y. If Rule 2 applied, then there exists a parent z of y that is a child of (and hence not a parent of) x. Assume that Rule 3 applied. Then there is a v-structure $w \to y \leftarrow z$ such that x is a neighbor of w, y, and z. Because both edges $x - w$ and $x - z$ are undirected at the time $x \to y$ is directed via Rule 3, the v-structure $w \to x \leftarrow z$ cannot exist in the resulting completed PDAG. Thus at least one of these edges is *not* directed into x. Without loss of generality, assume that w is not a parent of x in the completed PDAG. Then w is a parent of y that is not a parent of x, and consequently $x \to y$ is not covered. □

Lemma 40 also follows from Condition (iv) in Theorem 4.1 of Andersson et al. (1997).

Lemma 41 *Let $x - y$ be any undirected edge in a completed PDAG $\mathcal{P}^c$, and let $\mathcal{P}'$ denote the PDAG that results from inserting $a \to b$ into $\mathcal{P}^c$. If there is a directed path from both x and y to b in $\mathcal{P}'$, then $x - y$ is reversible in $\mathcal{P}'$.*

Proof: Suppose this is not the case. Let $\mathcal{P}^{c\prime}$ be the completed PDAG corresponding to $\mathcal{P}'$, and (without loss of generality) assume that the edge is directed as $x \to y$ in $\mathcal{P}^{c\prime}$.

We now argue that in $\mathcal{P}^c$, there cannot be a directed path of length zero or more from b to any of the nodes in the undirected component $K_{x,y}$ that contains x and y. Suppose there exists such a path. Because there is a directed path from both x and y to b in $\mathcal{P}^c$, we know from Lemma 23 that b cannot itself be contained in $K_{x,y}$, so any such path must be of length at least one. Consider any directed path of length one or more from b to a node in $K_{x,y}$, and let $c \to w$ be the last edge in this path. From Lemma 22, c is a parent of every node in $K_{x,y}$, including both x and y, which yields the contradiction that $\mathcal{P}^c$ is cyclic.

Because there is no directed path from b to any node in $K_{x,y}$ in $\mathcal{P}^c$, it follows from Corollary 38 that every edge $c \to d$ in $\mathcal{P}^c$, where $d \in K_{x,y}$, remains compelled in $\mathcal{P}^{c\prime}$. Because the three rules of Meek (1995) are sound, we can extract $\mathcal{P}^{c\prime}$ by applying them to the PDAG $\mathcal{P}''$ where all edges are initially undirected, except for the edges participating in v-structures and those edges that connect the parents of the nodes in $K_{x,y}$ to the nodes in $K_{x,y}$. For all three rules, an undirected edge is only directed if it is not covered; because all edges in $K_{x,y}$ are initially covered in $\mathcal{P}''$, this means that in order for the *first* edge contained in $K_{x,y}$ to be directed by one of the rules, there must an edge $v \to w$ in $\mathcal{P}^{c\prime}$ such that $w \in K_{x,y}$ and $v \leftarrow w$ is in $\mathcal{P}^c$. But this is impossible by Proposition 35, yielding a contradiction. □

Lemma 42 *Let $\mathcal{P}^c$ be a completed PDAG for which $\Pi_x = \Pi_y$, where x and y are not adjacent. Let $\mathcal{P}'$ denote the PDAG that results from adding $x - y$ to $\mathcal{P}^c$. Any undirected edge $a - b$ such that a and b are reachable by an undirected path from either x or y in $\mathcal{P}^c$ remains reversible in $\mathcal{P}'$.*

Proof: Suppose this is not the case, and let $a - b$ be the *first* undirected edge with these properties that is directed by an application of the rule-based orientation algorithm of Meek (1995).

Suppose $a - b$ is directed by Rule 1. That means that there is a compelled edge $c \rightarrow a$ in $\mathcal{P}'$, and that c and b are not adjacent (our argument is symmetric in a and b). If $c \rightarrow a$ is compelled in $\mathcal{P}^c$, it follows that $a - b$ would be compelled in $\mathcal{P}^c$ by Proposition 18. By Proposition 35, we know that the edge cannot be compelled as $c \leftarrow a$ in $\mathcal{P}^c$, and thus it must be undirected in $\mathcal{P}^c$. But this implies that there is an edge $c - a$ in $\mathcal{P}^c$, where both c and a are reachable by an undirected path from either x or y in $\mathcal{P}^c$, that was directed *before* $a - b$, contradicting the fact that $a - b$ was the first undirected edge with these properties to be directed.

Suppose $a - b$ is directed by Rule 2. That means there is a node c such that $a \rightarrow c \rightarrow b$. Because $a - b$ is the first edge to be chosen, we know that both $a \rightarrow c$ and $c \rightarrow b$ are compelled in $\mathcal{P}^c$. Furthermore, from Proposition 35, both of these edges have the same orientation in $\mathcal{P}^c$. Thus $a - b$ is not covered, and therefore cannot be the same edge as $x - y$. We conclude by Proposition 19 that the edge between a and b is compelled, yielding a contradiction.

Suppose $a - b$ is directed by Rule 3. This means there is a v-structure $c \rightarrow b \leftarrow d$, and undirected edges $a - c$ and $a - d$ at the point the rule is applied. Clearly, the v-structure must also exist in $\mathcal{P}^c$. a and c must be adjacent in $\mathcal{P}^c$, else Rule 1 would apply to the undirected edge $a - b$ because $c \rightarrow b$ is in $\mathcal{P}^c$. By the same argument, a and d must be adjacent. Because $a - b$ is reversible in $\mathcal{P}^c$, we deduce by the Triangle Lemma that both the edge between a and c and the edge between a and d must be compelled. At least one of these edges must be directed away from a, else $\mathcal{P}^c$ would contain the extra v-structure $c \rightarrow a \leftarrow d$ that does not exist in $\mathcal{P}'$. But this implies that $a - b$ would be directed by Proposition 19 in $\mathcal{P}^c$. $\square$

We are now able to prove the main result of this section.

Lemma 43 *Let $\mathcal{P}^c$ be a completed PDAG where nodes x and y are not adjacent. Let $\mathcal{P}'$ denote the PDAG that results from adding the edge $x \rightarrow y$ to $\mathcal{P}^c$. If $\Pi_x \neq \Pi_y$ in $\mathcal{P}^c$, then $x \rightarrow y$ is compelled in $\mathcal{P}'$.*

Proof: Given that $\Pi_x \neq \Pi_y$, either there is a parent of x that is not a parent of y, or a parent of y that is not a parent of x.

Suppose z is a parent of x but not a parent of y in $\mathcal{P}^c$. Because there is a directed path from both z and x to y in $\mathcal{P}'$, and because $x \neq y$, it follows from Corollary 38 that $z \rightarrow x$ remains compelled in $\mathcal{P}'$. If z and y are not adjacent in $\mathcal{P}'$, then $x \rightarrow y$ is compelled by Proposition 18. If there is a reversible edge between z and y in $\mathcal{P}^c$, then because there is a directed path from both z and y to y in $\mathcal{P}'$, the edge remains reversible in $\mathcal{P}'$ by Lemma 41. But this implies that $x \rightarrow y$ is compelled in $\mathcal{P}'$ by the Triangle Lemma. If there is a compelled edge between z and y in $\mathcal{P}^c$, then by our supposition the edge must be directed as $y \rightarrow z$, which implies that the insertion of the edge from x to y created a cycle, contradicting the fact that $\mathcal{P}'$ is a PDAG.

Suppose z is a parent of y but not a parent of x in $\mathcal{P}^c$. If x and z are not adjacent, then $x \rightarrow y$ is compelled because it participates in a v-structure. If z and x are adjacent via an undirected edge, then because there is a directed path from both x and z to y in $\mathcal{P}'$, it follows by Lemma 41 that $x - z$ remains undirected in $\mathcal{P}'$. If $z \rightarrow y$ remains compelled in $\mathcal{P}'$, it follows from the Triangle Lemma that $x \rightarrow y$ is compelled. Otherwise, we know from Lemma 39 that either $x \rightarrow y$ participates in a v-structure in $\mathcal{P}'$, in which case it is compelled in $\mathcal{P}'$, or there is a length two directed path $x \to w \to y$ in $\mathcal{P}^c$ for which $x \to w$ remains compelled in $\mathcal{P}'$. If $w \to y$ also remains compelled in $\mathcal{P}'$, then $x \to y$ is compelled in $\mathcal{P}'$ by Proposition 19. Otherwise $x \to y$ is compelled in $\mathcal{P}'$ by the Triangle Lemma. □

Lemma 44 *Let $\mathcal{P}^c$ be a completed PDAG where nodes x and y are not adjacent. Let $\mathcal{P}'$ denote the PDAG that results from adding the edge $x - y$ to $\mathcal{P}^c$. If $\Pi_x = \Pi_y$ in $\mathcal{P}^c$, then $x - y$ is reversible in $\mathcal{P}'$.*

Proof: Suppose not, and let $\mathcal{P}^{c\prime}$ be the completed PDAG corresponding to $\mathcal{P}'$. From Lemma 40, we know $\Pi_x \neq \Pi_y$ in $\mathcal{P}^{c\prime}$. This means that as a result of the addition, the parent set of at least one of the nodes changed. Without loss of generality, assume that the parent set of y changed.

Suppose that there is a parent z of y in $\mathcal{P}^c$ such that $z \to y$ is not compelled in $\mathcal{P}'$. From Lemma 36, there must be a directed path from x to y in $\mathcal{P}^c$. Let w be the last node in such a path. Because $\Pi_x = \Pi_y$ in $\mathcal{P}^c$, w is a parent of x, which implies that $\mathcal{P}^c$ is cyclic, yielding a contradiction.

Suppose there is a parent z of y in $\mathcal{P}^{c\prime}$ that is not a parent of y in $\mathcal{P}^c$. From Proposition 35, $\mathcal{P}^c$ must contain the undirected edge $z - y$. But it follows immediately from Lemma 42 that $z - y$ remains reversible in $\mathcal{P}'$, yielding a contradiction. □

Theorem 34 *Let $\mathcal{P}^c$ be any completed PDAG for which nodes x and y are not adjacent. If the result of adding an edge between x and y is a PDAG that admits a consistent extension, then that edge is compelled if and only if $\Pi_x \neq \Pi_y$.*

Proof: If $\Pi_x \neq \Pi_y$, it follows from Lemma 43 that if a *directed* edge is inserted between x and y, the resulting edge is compelled in the resulting equivalence class. Suppose an *undirected* edge is inserted between x and y. Without loss of generality, assume the edge is directed as $x \to y$ in the consistent extension. Clearly, the consistent extension is also a consistent extension of the PDAG that results from adding $x \to y$ to $\mathcal{P}^c$ instead of the undirected edge, and thus the edge is compelled.

If $\Pi_x = \Pi_y$, it follows from Lemma 44 that if an *undirected* edge is inserted between x and y, the resulting edge is reversible in the resulting equivalence class. Suppose a *directed* edge is inserted between x and y. Without loss of generality, assume the edge is directed as $x \to y$. We know that if we insert an undirected edge between x and y, the resulting edge is reversible, and thus there exists a consistent extension that contains the edge $x \to y$. Clearly, this consistent extension is also a consistent extension of the PDAG that results from adding $x \to y$ to $\mathcal{P}^c$. □

A.4 The InsertU Operator

Theorem 6 *Let x and y be two nodes that are not adjacent in $\mathcal{P}^c$. The insertion of the undirected edge $x - y$ is valid if and only if (1) every undirected path between x and y contains a node in $N_{x,y}$, and (2) $\Pi_x = \Pi_y$.*

Proof (if): Assume that every path between x and y contains a node in $N_{x,y}$, and that $\Pi_x = \Pi_y$. From Lemma 32, there exists a consistent extension $\mathcal{G}$ for which x has no reversible parents, and the reversible parents of y are precisely those nodes in $N_{x,y}$. We show that by inserting the edge $x \rightarrow y$ into $\mathcal{G}$, the DAG $\mathcal{G}'$ that results is a consistent extension of the PDAG $\mathcal{P}'$ that results from applying the operator to $\mathcal{P}^c$.

By construction of $\mathcal{G}$, $\mathcal{G}'$ cannot contain a cycle given that $\mathcal{G}$ is acyclic. Clearly, $\mathcal{G}'$ has the same skeleton as $\mathcal{P}'$. It remains to be shown that $\mathcal{G}'$ and $\mathcal{P}'$ contain the same v-structures.

Suppose there's a v-structure in $\mathcal{G}'$ that is not in $\mathcal{P}'$. From Lemma 24, this v-structure must include the edge $x \rightarrow y$. By construction, all reversible parents of y are in $N_{x,y}$ and hence adjacent to x, so the other edge in the v-structure must be compelled. But by assumption any compelled parent of y is also a compelled parent of x, so no such v-structure can exist. Suppose there's a v-structure in $\mathcal{P}'$ that is not in $\mathcal{G}'$. From Lemma 24, this v-structure must include the new edge $x - y$, which is impossible because it is undirected.

To complete the proof, we note that the construction of $\mathcal{G}$ is completely symmetric in x and y, and thus the resulting edge is reversible in $\mathcal{P}'$.

(only if): Suppose there is an undirected path from x to y in $\mathcal{P}^c$ that does not include any node in $N_{x,y}$, and consider the shortest such path. The length of the path must be at least three, lest the path would pass through a common neighbor of x and y. The path has no chord because it is shortest. This implies that after adding $x - y$ to $\mathcal{P}^c$, there is an undirected cycle of length four or more without a chord, and consequently there is no consistent extension to $\mathcal{P}'$ and thus the operator is not valid.

Suppose that x and y do not share compelled parents. By Theorem 34 the added undirected edge is compelled in the resulting equivalence class, and therefore the insertion is equivalent to inserting a directed edge, and we conclude that the operator is not valid. □

Corollary 7 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid insertion of an undirected edge $x - y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, N_{x,y}^{+x} \cup \Pi_y) - s(y, N_{x,y} \cup \Pi_y)$$

Proof: Follows immediately by subtracting the score of $\mathcal{G}$ from the score of $\mathcal{G}'$, where these DAGs are defined in the "if" proof of Theorem 6. □

A.5 The DeleteU Operator

Theorem 8 *Let $x - y$ be an undirected edge in completed PDAG $\mathcal{P}^c$. The deletion of $x - y$ is valid if and only if $N_{x,y}$ is a clique of undirected edges.*

Proof (if): Assume that $N_{x,y} = \{n_1, ..., n_k\}$ is a clique of undirected edges. By definition of $N_{x,y}$, and the fact that x and y are adjacent in $\mathcal{P}^c$, the set $\{x \cup y \cup N_{x,y}\}$ is also a clique of undirected edges in $\mathcal{P}^c$. From Lemma 30, there exists a consistent extension where (1) the edges in this clique correspond to the total ordering $x, n_1, ..., n_k, y$, (2) x has no reversible parents and (3) the reversible parents of y are precisely those nodes in $\{x \cup N_{x,y}\}$. Let $\mathcal{G}$ denote any such consistent extension, and let $\mathcal{G}'$ denote the DAG that results from deleting the edge $x \rightarrow y$ from $\mathcal{G}$. We now show that $\mathcal{G}'$ is a consistent extension of the PDAG $\mathcal{P}'$ that results from deleting $x - y$ from $\mathcal{P}^c$.

Clearly, $\mathcal{G}'$ has the same adjacencies as $\mathcal{P}'$. We now show that $\mathcal{G}'$ and $\mathcal{P}'$ have the same set of v-structures. From Lemma 25, any v-structure that is in one but not the other must have the form $x \rightarrow z \leftarrow y$.

Suppose $x \rightarrow z \leftarrow y$ is a v-structure in $\mathcal{G}'$ that is not in $\mathcal{P}'$. At least one of the two edges must be compelled because by construction of $\mathcal{G}$, all common neighbors of x and y are parents of y in $\mathcal{G}'$. If exactly one of the edges were compelled, the 3-clique $\{x, y, z\}$ in $\mathcal{G}$ would contain exactly two reversible edges, which by the Triangle Lemma is impossible. If both edges were compelled, then the v-structure exists in $\mathcal{P}'$.

Suppose $x \rightarrow z \leftarrow y$ is a v-structure in $\mathcal{P}'$ that is not in $\mathcal{G}'$. This v-structure must consist of previously-compelled edges from $\mathcal{P}^c$ because only compelled edges are directed in a completed PDAG. This implies that $\mathcal{G}$ contains both the edge $x \rightarrow z$ and the edge $y \rightarrow z$, and consequently the v-structure must exist in $\mathcal{G}'$ once the edge between x and y is removed, yielding a contradiction.

(only if): Assume $N_{x,y}$ does not form a clique. This means that there exists common neighbors n and m such that when $x - y$ is deleted, the resulting PDAG contains the four cycle $x - n - y - m - x$ without a chord, and thus it does not admit a consistent extension. □

Corollary 9 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid deletion of an undirected edge $x - y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, N_{x,y} \cup \Pi_y) - s(y, N_{x,y}^{+x} \cup \Pi_y)$$

Proof: Follows immediately by subtracting the score of $\mathcal{G}$ from the score of $\mathcal{G}'$, where these DAGs are defined in the "if" proof of Theorem 8. □

A.6 The InsertD Operator

The main proof for the InsertD operator requires the following results about semi-directed paths.

Lemma 45 *Let $\mathcal{P}^c$ be a completed PDAG that contains a semi-directed path from x to y. If there exists a directed edge $z \rightarrow w$ in this path, then there exists a directed path from z to y in $\mathcal{P}^c$.*

Proof: We prove this by induction on the length of the path segment from z to y. For basis, we consider a length one semi-directed path segment where $w = y$, and the lemma holds trivially. Assume the lemma is true for path segments of length k, and consider a length $k+1$ path segment from z to y. If there are no undirected edges in this segment, the segment itself constitutes a directed path from z to y. Otherwise, let $a - b$ be the first undirected edge in the segment that follows $z \rightarrow w$, and let $c \rightarrow a$ be the directed edge that precedes it. By Proposition 18, c and b must be adjacent. By the Triangle Lemma, the edge must be compelled. By Proposition 19 the edge must be directed as $c \rightarrow b$. By replacing the two edges $c \rightarrow a$ and $a - b$ in the semi-directed path with the single edge $c \rightarrow b$, we have a length k semi-directed path segment and the lemma holds by induction. □

Lemma 45 can also be proven using two of the properties of completed PDAGs given by Andersson et al. (1997).

Corollary 46 *Let $\mathcal{P}^c$ be a completed PDAG. If $\mathcal{P}^c$ contains a semi-directed path from x to y consisting of intermediate nodes contained within some set N, then the* shortest *semi-directed path whose intermediate nodes are contained in N consists of exactly two consecutive segments, where the first segment consists entirely of undirected edges and the second segment consists entirely of directed edges.*

Proof: Follows by considering the argument in the induction step of the proof of Lemma 45 applied to the shortest path: because the semi-directed path is made shorter by eliminating the first undirected edge that follows the first directed edge by eliminating a single intermediate node, it follows that the shortest semi-directed path can contain no such undirected edge. $\square$

Corollary 47 *Let $\mathcal{P}^c$ be a completed PDAG. If $\mathcal{P}^c$ contains a semi-directed path from x to y consisting of intermediate nodes contained within some set N, then for any* shortest *semi-directed path whose intermediate nodes are contained in N, there is no edge in $\mathcal{P}^c$ that connects a pair of non-consecutive nodes along the path.*

Proof: Suppose this is not the case, and let z and w be any pair of non-consecutive nodes from $N \cup \{x\} \cup \{y\}$ that are connected by an edge in $\mathcal{P}^c$. Without loss of generality, assume that z precedes w along the path from x to y. Because we are considering the shortest path, the connecting edge must be directed as $z \leftarrow w$. From Corollary 46, we know the path consists of an undirected segment followed by a directed segment. If z follows the last undirected edge along this path, then the edge $z \leftarrow w$ in conjunction with the directed path from z to w constitutes a cycle. Otherwise, there is a path of one or more undirected edges $z - n_1 - ... - n_k$ such that there is a directed path from n_k to z, which is impossible by Lemma 23. $\square$

Recall that $\Omega_{x,y} = \Pi_x \cap N_y$ is the set of parents of node x that are also neighbors of node y.

Theorem 10 *Let x and y be any two nodes that are not adjacent in completed PDAG $\mathcal{P}^c$. The insertion of $x \rightarrow y$ is valid if and only if (1) every semi-directed path from y to x contains at least one node in $\Omega_{x,y}$, (2) $\Omega_{x,y}$ is a clique of undirected edges, and (3) $\Pi_x \neq \Pi_y$.*

Proof (if): Suppose that the three conditions hold. From condition (2) and the fact that y is a neighbor of all members in $\Omega_{x,y}$, it follows that $\{y \cup \Omega_{x,y}\}$ is a clique of undirected edges. Thus by Lemma 30, there exists a consistent extension where the reversible parents of y are precisely those members of $\Omega_{x,y}$. Let $\mathcal{G}$ be any such consistent extension, and let $\mathcal{G}'$ denote the DAG that results when we insert the edge $x \rightarrow y$ into $\mathcal{G}$. We now show that $\mathcal{G}'$ is a consistent extension of the PDAG $\mathcal{P}'$ that results from inserting $x \rightarrow y$ into $\mathcal{P}^c$.

Suppose $\mathcal{G}'$ contains a cycle. This means there's a directed path from y to x in $\mathcal{G}$, and a corresponding semi-directed path from y to x in $\mathcal{P}^c$ that passes through the same nodes. By condition (1), this path must contain some compelled parent z of x that is a neighbor of y, which means there is a directed path from y to z in $\mathcal{G}$. By construction of $\mathcal{G}$, however, z is a parent of y, which implies that $\mathcal{G}$ is cyclic, contradicting the fact that $\mathcal{G}$ is a consistent extension of $\mathcal{P}^c$.

Suppose there is a v-structure in $\mathcal{P}'$ that is not in $\mathcal{G}'$. By Lemma 24, this v-structure is of the form $x \rightarrow y \leftarrow z$. Because $x \rightarrow y$ is the only edge changed by the operation, we know that both (1) $z \rightarrow y$ is in $\mathcal{G}'$, and (2) x and z are not adjacent in $\mathcal{G}'$. But because $x \rightarrow y$ was added to $\mathcal{G}$ to create $\mathcal{G}'$, the v-structure $x \rightarrow y \leftarrow z$ exists in $\mathcal{G}'$.

Suppose there is a v-structure in $\mathcal{G}'$ that is not in $\mathcal{P}'$. By Lemma 24, this v-structure is of the form $x \rightarrow y \leftarrow z$. By construction, we know that $z \rightarrow y$ must be compelled in $\mathcal{G}$ because the only reversible parents of y are parents of x. But this means that $z \rightarrow y$ is directed in $\mathcal{P}^c$, and because the edge did not change as a result of the operation, it must be directed in $\mathcal{P}'$ as well. Because the only adjacency that changed from the operation is the one between x and y, it follows that z and x cannot be adjacent in $\mathcal{P}'$, and consequently the v-structure $x \rightarrow y \leftarrow z$ must exit in $\mathcal{P}$.

To complete the proof of the "if" part, we need to show that $x \rightarrow y$ is compelled in $\mathcal{P}'$. This follows immediately from Theorem 34 given that $\Pi_x \neq \Pi_y$ from condition (3).

(only if): We now show that if any of the three conditions are violated, the insertion is not valid.

(1) Suppose there exists a semi-directed path in $\mathcal{P}^c$ from y to x that does not contain at least one node in $\Omega_{x,y}$, and consider the *shortest* such path.

We first show that this path contains at least three edges. At least one of the edges in the path must be directed, else x and y are in the same undirected component in $\mathcal{P}^c$ and by Lemma 22 we know that $\Pi_x = \Pi_y$, which in turn implies by Theorem 34 that the edge would not be compelled in the resulting equivalence class, and consequently the insertion is not valid. From Corollary 46, we know the shortest semi-directed path from y to x (that does not contain a node from $\Omega_{x,y}$) consists of an undirected segment followed by a directed segment. We know that there is at least one edge in the undirected segment because otherwise there would be a directed path from y to x and the result of the insertion is a cycle and thus the insertion is invalid. We know that there must be at least one additional edge in the path, lest the intermediate node of the path is in $\Omega_{x,y}$.

We now consider two additional properties of this shortest path. First, by Corollary 47, no two nodes can be adjacent along the path. Second, because all directed edges are directed away from y, there can be no pair of edges in the path that constitute a v-structure.

Combining these two properties with the fact that the path must contain at least three edges, we see that after adding the edge $x \rightarrow y$, *all* of the undirected edges in the path must be directed away from y in any directed graph that has the same set of v-structures. But such an orientation is cyclic, so we conclude the insertion is not valid.

(2) Suppose $\Omega_{x,y}$ is *not* a clique of undirected edges. Because $\Omega_{x,y} \subseteq N_y$, we know by the Triangle Lemma that any pair of nodes that are not connected by an undirected edge are not adjacent. Consequently, there must exist a v-structure $z \rightarrow x \leftarrow w$ where both z and w are neighbors of y. Consider the orientation of the edges $z - y$ and $w - y$ in any consistent extension of the PDAG $\mathcal{P}'$ that results after inserting $x \rightarrow y$ in $\mathcal{P}^c$. Because Π_x is non-empty, it follows that $\Pi_x \neq \Pi_y$, and thus from Theorem 34 we conclude that $x \rightarrow y$ must be compelled in $\mathcal{P}'$. Furthermore, $z \rightarrow x$ and $w \rightarrow x$ remain compelled in $\mathcal{P}'$ because they participate in a v-structure. But this means that there is a compelled path both from z to y and from w to y in $\mathcal{P}'$, and consequently both the edges $z - y$ and $w - y$ must be compelled and directed into y in any acyclic consistent extension. Because w and z are not

adjacent, however, such a consistent extension would include the v-structure $w \rightarrow y \leftarrow z$ that is not in $\mathcal{P}'$, which is impossible by definition of a consistent extension.

(3) If $\Pi_x = \Pi_y$, we know by Theorem 34 that the edge would not be compelled in the resulting equivalence class, and therefore the addition is not valid. □

Corollary 11 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid insertion of a directed edge $x \rightarrow y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, \Omega_{x,y} \cup \Pi_y^{+x}) - s(y, \Omega_{x,y} \cup \Pi_y)$$

Proof: Follows immediately by subtracting the score of $\mathcal{G}$ from the score of $\mathcal{G}'$, where these DAGs are defined in the "if" proof of Theorem 10. □

A.7 The DeleteD Operator

Theorem 12 *Let $x \rightarrow y$ be a directed edge in completed PDAG $\mathcal{P}^c$. The deletion of $x \rightarrow y$ is valid if and only if N_y is a clique of undirected edges.*

Proof (if): Assume N_y is a clique of undirected edges. Because all of these nodes are neighbors of y, $N_y \cup y$ is also a clique of undirected edges, and by Lemma 30, there exists a consistent extension $\mathcal{G}$ of $\mathcal{P}^c$ where the reversible parents of y are precisely the elements in N_y. We now show that the DAG $\mathcal{G}'$ that results from deleting $x \rightarrow y$ from $\mathcal{G}$ is a consistent extension of the PDAG $\mathcal{P}'$ that results from deleting $x \rightarrow y$ from $\mathcal{P}^c$.

Clearly, $\mathcal{G}'$ and $\mathcal{P}'$ contain the same adjacencies. We now show that $\mathcal{G}'$ and $\mathcal{P}'$ have the same set of v-structures. From Lemma 25, any v-structure that is in one but not the other must have the form $x \rightarrow z \leftarrow y$.

Suppose the v-structure $x \rightarrow z \leftarrow y$ exists in $\mathcal{G}'$. We know that $y \rightarrow z$ is a compelled edge in $\mathcal{P}^c$ because by construction of $\mathcal{G}$, any undirected edge incident to y in $\mathcal{P}^c$ is directed into y in $\mathcal{G}$. Because $x \rightarrow y$ and $y \rightarrow z$ are compelled in $\mathcal{G}$, we know from Proposition 19 that $x \rightarrow z$ is also compelled in $\mathcal{G}$. Consequently the v-structure must exist in $\mathcal{P}'$.

Suppose the v-structure $x \rightarrow z \leftarrow y$ exists in $\mathcal{P}'$. Both edges $x \rightarrow z$ and $y \rightarrow z$ are directed in $\mathcal{P}^c$ because the only difference between $\mathcal{P}^c$ and $\mathcal{P}'$ is the presence of $x \rightarrow y$. Consequently, these edges have the same orientation in both $\mathcal{G}$ and $\mathcal{G}'$, and thus the v-structure exists in $\mathcal{G}'$.

(only if): Suppose that N_y does not form a clique in $\mathcal{P}^c$. This means there are two nodes z and w such that $z - y - w$ and z and w are not adjacent (we know by the Triangle Lemma that z and w cannot be connected by a *directed* edge). From Lemma 22, x is a parent of both z and w in $\mathcal{P}^c$. We now show that $\mathcal{P}'$ does not admit a consistent extension. Because the v-structure $z \rightarrow y \leftarrow w$ does not exist in $\mathcal{P}'$, any consistent extension must contain either the edge $y \rightarrow w$ or the edge $y \rightarrow z$ (or both). But these edges imply that the consistent extension contains the v-structures $y \rightarrow w \leftarrow x$ and $y \rightarrow z \leftarrow x$, respectively, neither of which are in $\mathcal{P}'$. □

Corollary 13 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid deletion of a directed edge $x \rightarrow y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, N_y \cup \Pi_y^{-x}) - s(y, N_y \cup \Pi_y)$$

Proof: Follows immediately by subtracting the score of $\mathcal{G}$ from the score of $\mathcal{G}'$, where these DAGs are defined in the "if" proof of Theorem 12. □

A.8 The ReverseD Operator

For the reversal operator, we need an additional result about semi-directed paths.

Lemma 48 *Let $x \rightarrow y$ be any edge in a completed PDAG $\mathcal{P}^c$. If there exists a semi-directed path from x to y in $\mathcal{P}^c$ that does not pass through $x \rightarrow y$, such that the path consists of intermediate nodes contained within some set N, then the* shortest *such path either (1) consists of two segments, the first of which consists entirely of undirected edges and the second of which consists entirely of directed edges, or (2) passes through a member of N_y.*

Proof: Let z be the next-to-last node (i.e., adjacent to y) in the shortest such semi-directed path from x to y. The sub-path that ends at z clearly constitutes the shortest semi-directed path from x to z that consists of nodes within N; that is, this path is the shortest without the restriction that it cannot pass through $x \rightarrow y$. From Corollary 46, this path consists of two segments, the first of which consists entirely of undirected edges and the second of which consists entirely of directed edges. Therefore if the edge between z and y is directed as $z \rightarrow y$, then (1) holds. If the edge between z and y is undirected, then z is a neighbor of y and (2) holds. The edge cannot be directed from y to z by definition of semi-directed path. □.

Theorem 14 *Let $x \rightarrow y$ be a directed edge in completed PDAG $\mathcal{P}^c$. The reversal of $x \rightarrow y$ is valid if and only if (1) every semi-directed path from x to y that does not include the edge $x \rightarrow y$ contains at least one node in $\Omega_{y,x} \cup N_y$, and (2) $\Omega_{y,x}$ is a clique of undirected edges.*

Proof (if): Suppose that both conditions hold. From condition (2) and the fact that x is a neighbor of all members in $\Omega_{y,x}$, it follows that $\{x \cup \Omega_{y,x}\}$ is a clique of undirected edges. Thus by Lemma 30, there exists a consistent extension where the reversible parents of x are precisely the members of $\Omega_{y,x}$. Furthermore, because there is a directed edge from x to y, we know by Lemma 23 that x and y are not in the same undirected component; thus by Lemma 27 there exists such a consistent extension where y has no reversible parents. Let $\mathcal{G}$ be any such consistent extension, and let $\mathcal{G}'$ be the graph that results from reversing $x \rightarrow y$ in $\mathcal{G}$. We now show that $\mathcal{G}'$ is a consistent extension of the PDAG $\mathcal{P}'$ that results from reversing $x \rightarrow y$ in $\mathcal{P}^c$.

Suppose $\mathcal{G}'$ contains a cycle. This means there is a directed path from x to y in $\mathcal{G}$ that does not include the edge $x \rightarrow y$, and a corresponding semi-directed path from x to y in $\mathcal{P}^c$ that passes through the same nodes. By construction of $\mathcal{G}$, y is a parent of every element in N_y, and consequently the path cannot pass through any of these elements, lest $\mathcal{G}$ would be cyclic. Thus we conclude by condition (1) that the path must contain some element

$z \in \Omega_{y,x}$, which means there is a directed path from x to z in $\mathcal{G}$. By construction of $\mathcal{G}$, however, z is a parent of x, which implies that $\mathcal{G}$ is cyclic, contradicting the fact that $\mathcal{G}$ is a consistent extension of $\mathcal{P}^c$.

Suppose there is a v-structure in $\mathcal{P}'$ that is not in $\mathcal{G}'$. Because no adjacencies have changed as a result of the reversal, this v-structure is of the form $z \rightarrow x \leftarrow y$. Because the edge between x and y is the only edge changed by the operation, we know that both (1) $z \rightarrow x$ is compelled in $\mathcal{P}^c$, and consequently exists in $\mathcal{G}$, and (2) y and z are not adjacent in $\mathcal{G}$. But because $x \rightarrow y$ was reversed in $\mathcal{G}$ to create $\mathcal{G}'$, the v-structure $z \rightarrow x \leftarrow y$ exists in $\mathcal{G}'$.

Suppose there is a v-structure in $\mathcal{G}'$ that is not in $\mathcal{P}'$. Because no adjacencies have changed as a result of the reversal, this v-structure is of the form $z \rightarrow x \leftarrow y$. By construction of $\mathcal{G}$, we know that $z \rightarrow x$ must be compelled in $\mathcal{G}$ because the only reversible parents of x are parents of y. But this means that $z \rightarrow x$ is directed in $\mathcal{P}^c$, and because the edge did not change as a result of the operation, it must be directed in $\mathcal{P}'$ as well. Because no adjacencies changed from the operation, z and y cannot be adjacent in $\mathcal{P}'$, and consequently the v-structure $z \rightarrow x \leftarrow y$ must exit in $\mathcal{P}'$.

(only if): We now show that if either one of the two conditions is violated, the reversal is not valid.

(1) Suppose there exists a semi-directed path from x to y that does not include the edge $x \rightarrow y$ and does not contain at least one node in $\Omega_{y,x} \cup N_y$, and consider the *shortest* such path. It follows from Lemma 48 that the path must consist of two segments, the first of which consists entirely of undirected edges and the second of which consists entirely of directed edges.

We first show that this path contains at least three edges. There is at least one edge in the directed segment of the path, else by Lemma 22, x would be a parent of itself. There is at least one edge in the undirected segment because otherwise there would be a directed path from x to y and the result of the reversal is a cycle and thus the reversal is invalid. We know that there must be at least one additional edge in the path, lest the middle node of the path is in $\Omega_{y,x}$.

We now consider two additional properties of this shortest path. First, by Corollary 47, no two nodes can be adjacent along the path except for x and y. Second, because all directed edges are directed away from x, there can be no pair of edges in the path that constitute a v-structure.

Combining these two properties with the fact that the path must contain at least three edges, we see that after reversing the edge $x \rightarrow y$, *all* of the undirected edges in the path must be directed away from x in any directed graph that has the same set of v-structures. But such an orientation is cyclic, so we conclude the reversal is not valid.

(2) Suppose $\Omega_{y,x}$ is *not* a clique of undirected edges. Because $\Omega_{y,x} \subseteq N_x$, we know by the Triangle Lemma that any pair of nodes that are not connected by an undirected edge must not be adjacent. Consequently, there must exist a v-structure $z \rightarrow y \leftarrow w$ where both z and w are neighbors of x. Consider the orientation of the edges $z - x$ and $w - x$ in any consistent extension $\mathcal{G}'$ of the PDAG $\mathcal{P}'$ that results after reversing $x \rightarrow y$ in $\mathcal{P}^c$. At least one of these edges must be directed away from x, lest $\mathcal{G}'$ would contain the v-structure $z \rightarrow x \leftarrow w$ that is not in $\mathcal{P}'$. But if either of these edges is directed into x, the resulting graph would contain a cycle because there is a directed path from both z and w to x in $\mathcal{G}'$.

Thus we conclude that no consistent extension exists. □

Corollary 15 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid reversal of a directed edge $x \to y$ to a completed PDAG $\mathcal{P}^c$ is:*

$$s(y, \Pi_y^{-x}) + s(x, \Pi_x^{+y} \cup \Omega_{y,x}) - s(y, \Pi_y) - s(x, \Pi_x \cup \Omega_{y,x})$$

Proof: Follows immediately by subtracting the score of $\mathcal{G}$ from the score of $\mathcal{G}'$, where these DAGs are defined in the "if" proof of Theorem 14. □

A.9 The MakeV Operator

Theorem 16 *Let $x - z - y$ be any length-two undirected path in completed PDAG $\mathcal{P}^c$ such that x and y are not adjacent. Replacing the undirected edges with directed edges to create the v-structure $x \to z \leftarrow y$ is valid if and only if every undirected path between x and y contains a node in $N_{x,y}$.*

Proof (if): From Lemma 32 and Corollary 33, there exists a consistent extension $\mathcal{G}$ where x has no reversible parents, the reversible parents of z are precisely the nodes in $\{x\} \cup N_{x,y} \setminus \{z\}$ (corresponding to the total ordering of $N_{x,y}$ where z is *last*), and the reversible parents of y are precisely the nodes in $N_{x,y}$. We now show that the graph $\mathcal{G}'$ that results from reversing $z \to y$ in $\mathcal{G}$ is a consistent extension of the PDAG $\mathcal{P}'$ that results from applying the MakeV operator to $\mathcal{P}^c$.

Clearly the adjacencies in $\mathcal{G}'$ and $\mathcal{P}'$ are the same. Any v-structure that exists in one of these graphs but not the other must include the edge $y \to z$, and thus is of the form $w \to z \leftarrow y$. Because any compelled edge from $\mathcal{P}^c$ exists as a directed edge (with the same orientation) in both $\mathcal{G}'$ and $\mathcal{P}'$, it follows that $w \to y$ must be reversible in $\mathcal{P}^c$. Because all reversible edges from $\mathcal{P}^c$ except for $x - z$ and $z - y$ are undirected in $\mathcal{P}'$, no such extra v-structure can exist in $\mathcal{P}'$. By construction of $\mathcal{G}$, every reversible parent of z is a member of $\{x\} \cup N_{x,y} \setminus \{z\}$. All of these nodes, except for x, is adjacent to y in $\mathcal{G}$ and thus in $\mathcal{G}'$, and we conclude that the extra v-structure cannot exist in $\mathcal{G}'$ either. It remains to be shown that $\mathcal{G}'$ is acyclic.

Suppose $\mathcal{G}'$ contains a cycle. This cycle must include the edge $y \to z$ because $\mathcal{G}$ is acyclic. This implies there is a directed path from z to y in $\mathcal{G}$. By construction, the only reversible parents of y in $\mathcal{G}$ are nodes in $N_{x,y}$, all of which are parents of z in $\mathcal{G}$; thus, this directed path must enter y via some compelled edge $w \to y$. Because $z \to y$ is reversible in $\mathcal{G}$, we know by Lemma 22 that there must also exist the compelled edge $w \to z$ in $\mathcal{G}$. But because there is a directed path from z to w, this implies $\mathcal{G}$ contains a cycle, which is impossible.

(only if) We now show that if there is an undirected path from x to y in $\mathcal{P}^c$ that does not pass through a node in $N_{x,y}$, then there is no consistent extension of $\mathcal{P}'$.

Suppose there does exist some consistent extension $\mathcal{G}'$. Let $x - a_1 - \ldots - a_k - y$ denote the *shortest* undirected path between x and y such that for each a_i, $a_i \notin N_{x,y}$. Because each a_i is not a common neighbor of x and y, there must be at least two such elements in the path. There can be no edge connecting two non-consecutive nodes in the path because by

Lemma 23, any such edge must be undirected and hence the path would not be shortest. Because there is a *disjoint* path between x and y through the common neighbor z in $\mathcal{G}$, the shortest path participates in an undirected cycle of length four or more. This implies that z must be adjacent to a_1, else the undirected path $z - x - a_1$ in this cycle would not have a chord and thus $\mathcal{G}$ would not be chordal.

First we show that the edge between x and a_1 is directed as $x \rightarrow a_1$ in $\mathcal{G}'$. Suppose this is not the case, and we have $a_1 \rightarrow x$. Because $\mathcal{G}'$ contains the edge $x \rightarrow z$, and z is adjacent to a_1, it follows that the edge between a_1 and z must be directed as $a_1 \rightarrow z$, lest $\mathcal{G}'$ contains a cycle. Because a_1 and y are not adjacent and the edge $y \rightarrow z$ is in $\mathcal{G}'$, this implies there is a v-structure $a_1 \rightarrow z \leftarrow y$ in $\mathcal{G}'$ that does not exist in $\mathcal{P}'$.

Following the same argument as above, we conclude that the edge between y and a_k must be directed as $y \rightarrow a_k$ in $\mathcal{G}'$. But this implies there must be some a_i along the path that has converging directed edges $a_{i-1} \rightarrow a_i \leftarrow a_{i+1}$. Because a_{i-1} and a_{i+1} are not adjacent in $\mathcal{G}'$, this constitutes a v-structure that is not in $\mathcal{P}'$. □

Corollary 17 *For any score-equivalent decomposable scoring criterion, the increase in score that results from a valid application of the MakeV operator is:*

$$s(z, \Pi_z^{+y} \cup N_{x,y}^{-z+x}) + s(y, \Pi_y \cup N_{x,y}^{-z}) - s(z, \Pi_z \cup N_{x,y}^{-z+x}) - s(y, \Pi_y \cup N_{x,y})$$

Proof: Follows immediately by subtracting the score of $\mathcal{G}$ from the score of $\mathcal{G}'$, where these DAGs are defined in the "if" proof of Theorem 16. □

A.10 Completeness of the Operators

In this section, we prove Theorem 4. That is, we prove that given any pair of completed PDAGs $\mathcal{P}_1^c$ and $\mathcal{P}_2^c$, there exists a sequence of valid operators that can be applied to $\mathcal{P}_1^c$ that transforms it into $\mathcal{P}_2^c$. The proof of this result is non-trivial, but the sequence of operators can be described easily as follows: we first transform $\mathcal{P}_1^c$ into a completed PDAG with no edges by deleting edges, either directed or undirected, one at a time until there are none remaining. Next, we construct each of the v-structures that exist in $\mathcal{P}_2^c$ with (one or) two edge additions followed by (if necessary) a MakeV operator. Next, the remaining directed edges from $\mathcal{P}_2^c$ are then inserted one at a time into $\mathcal{P}_1^c$, such that after each addition, $\mathcal{P}_1^c$ contains only directed edges that are also in $\mathcal{P}_2^c$. Finally, all of the undirected edges from $\mathcal{P}_2^c$ are added to $\mathcal{P}_1^c$.

This section is organized as follows. In Section A.10.1 through Section A.10.4, we prove the existence of each "phase" of operators as described above. Finally, in Section A.10.5 we state the proof of Theorem 4.

In the sections to follow, we use DeleteD(x, y) to denote a DeleteD operator that deletes the (directed) edge $x \rightarrow y$. We use analogous notation for the other edge operators. We use MakeV(x, y, z) to denote the MakeV operator that directs the undirected edges $x - y$ and $z - y$ as $x \rightarrow y$ and $z \rightarrow y$, respectively.

A.10.1 Deleting All Edges

From Theorem 26, we know that the undirected components of a completed PDAG are chordal. Following are some results related to chordal components that will prove to be useful in this section.

Lemma 49 (Blair and Peyton, 1993) *Within any chordal component, there exists a node x for which N_x is a clique.*

Given this result, the following corollaries follow easily.

Corollary 50 *For any completed PDAG $\mathcal{P}^c$ containing at least one undirected edge, there exists an undirected edge $x - y$ for which $N_{x,y}$ is a clique.*

Proof: From Theorem 26, any undirected edge in $\mathcal{P}^c$ participates in a chordal component of $\mathcal{P}^c$. This component contains at least one edge, and thus from Lemma 49 there is a node x for which N_x is a clique and where $|N_x| > 0$. Let y be any member of N_x; because $N_{x,y} \subset N_x$, the corollary follows because every subgraph of a clique is itself a clique. □

Corollary 51 *Let $\mathcal{P}^c$ be any completed PDAG containing at least one undirected edge. Then there exists a valid DeleteU operator that can be applied to $\mathcal{P}^c$.*

Proof: Follows immediately from Corollary 50 and Theorem 8. □

Now the main result of this section follows easily.

Lemma 52 *Let $\mathcal{P}^c$ be any completed PDAG containing at least one edge. There exists a sequence of valid DeleteU and DeleteD operators that can be applied to $\mathcal{P}^c$ so that the result of applying all operators in the sequence is a completed PDAG containing no edges.*

Proof: The proof follows by showing that there must exist at least one valid DeleteU or DeleteD operator that can be applied to $\mathcal{P}^c$; because edges are never added as a result of one of these operators, the desired sequence can then be constructed iteratively until the resulting completed PDAG contains no edges. If $\mathcal{P}^c$ contains any undirected edges, we know by Corollary 51 that there is a valid DeleteU operator that we can apply to $\mathcal{P}^c$. If $\mathcal{P}^c$ contains no undirected edges, it follows immediately from Theorem 12 that *any* DeleteD operator is valid. □

A.10.2 Inserting V-Structures

In this section, we show that we can convert a completed PDAG that contains no edges into a completed PDAG that contains precisely the edges that participate in v-structures in some other completed PDAG by a sequence of valid InsertD, InsertU, and MakeV operators. To do so, we need some preliminary results.

Proposition 53 *Let $\mathcal{P}^c$ be any completed PDAG for which every edge is a directed edge that participates in a v-structure. If there is an edge that participates at least two v-structures, then there exists an edge in $\mathcal{P}^c$ that can be deleted such that the resulting PDAG is itself a completed PDAG for which every edge participates in a v-structure.*

Proof: Let $x \rightarrow y$ be any edge that participates in at least two v-structures, and let $Z = \{z_1, ..., z_k\}$, $k \geq 2$ denote the tail nodes of the other corresponding edges. That is, $x \rightarrow y \leftarrow z_i$ is a v-structure in $\mathcal{P}^c$ for each $z_i \in Z$. If there exists a $z_j \in Z$ for which the only v-structure in which $z_j \rightarrow y$ participates is the one with $x \rightarrow y$, then we can delete $z_j \rightarrow y$ from $\mathcal{P}^c$ and the desired property holds. Otherwise, for every $z_j \in Z$, $z_j \rightarrow y$ participates in at least one other v-structure, and thus we can delete $x \rightarrow y$ from $\mathcal{P}^c$ and the desired property holds. □

Proposition 54 *Let $\mathcal{P}^c$ be any completed PDAG for which every edge is a directed edge that participates in a v-structure. If every edge participates in exactly one v-structure, then the PDAG that results from deleting the two edges in any v-structure is itself a completed PDAG for which every edge participates in a v-structure.*

Proof: Suppose that after deleting the two edges $x \rightarrow y$ and $z \rightarrow y$ from a v-structure, there exists some edge that no longer participates in a v-structure. This implies that the edge participated in a v-structure with either $x \rightarrow y$ or $z \rightarrow y$, which contradicts the fact that both of these edges participate in only one v-structure. □

Given these two propositions, we can now prove the main result of this section.

Lemma 55 *Let $\mathcal{P}^{c\prime}$ be any completed PDAG for which every edge is a directed edge that participates in a v-structure. Let $\mathcal{P}^c$ be any the completed PDAG that contains no edges. Then there exists a sequence of valid InsertU, InsertD, and MakeV operators that can be applied in $\mathcal{P}^c$ to convert it into $\mathcal{P}^{c\prime}$.*

Proof: Induction on the number of (directed) edges in $\mathcal{P}^{c\prime}$. For the base case, if $\mathcal{P}^{c\prime}$ contains no edges then $\mathcal{P}^c = \mathcal{P}^{c\prime}$ and thus the lemma holds with a sequence of zero operators. For the induction step, we assume the lemma is true whenever the number of edges is less than k, and consider the case when $\mathcal{P}^{c\prime}$ contains k edges. We note that the number of directed edges in a completed PDAG can never be one.

If there exists an edge in $\mathcal{P}^{c\prime}$ that participates in at least two v-structures, then we know from Proposition 53 that there exists an edge $x \rightarrow y$ in $\mathcal{P}^{c\prime}$ that we can delete such that the resulting PDAG $\mathcal{P}^{c\prime\prime}$ is completed and for which every edge participates in a v-structure. Because $\mathcal{P}^{c\prime\prime}$ contains $k-1$ edges, we know by the induction hypothesis that we can transform $\mathcal{P}^c$ into $\mathcal{P}^{c\prime\prime}$ using a sequence of valid operators. The result of inserting $x \rightarrow y$ into $\mathcal{P}^{c\prime\prime}$ is a PDAG that is identical to $\mathcal{P}^{c\prime}$, and consequently the operator InsertD(x, y) is valid and constitutes the last operator in the desired sequence for the lemma.

To complete the proof, assume that every edge in $\mathcal{P}^{c\prime}$ participates in exactly one v-structure, and let $x \rightarrow y \leftarrow z$ be any v-structure in $\mathcal{P}^{c\prime}$. From Proposition 54, the PDAG $\mathcal{P}^{c\prime\prime}$ that is identical to $\mathcal{P}^{c\prime}$ except that does not contain either $x \rightarrow y$ or $y \leftarrow z$ is itself a completed PDAG for which every edge participates in a v-structure. Because $\mathcal{P}^{c\prime\prime}$ contains $k-2$ edges, we know by the induction hypothesis that we can transform $\mathcal{P}^c$ into $\mathcal{P}^{c\prime\prime}$ using a sequence of valid operators. We now show how to convert $\mathcal{P}^{c\prime\prime}$ into $\mathcal{P}^{c\prime}$ by either two InsertD operators or by two InsertU operators followed by a MakeV operator.

If $\Pi_x \neq \Pi_y$ or $\Pi_z \neq \Pi_y$ in $\mathcal{P}^{c\prime\prime}$ then assume, without loss of generality, that $\Pi_x \neq \Pi_y$. Then InsertD(x, y) is a valid operator in $\mathcal{P}^{c\prime\prime}$ and from Theorem 34 $x \rightarrow y$ is compelled in the resulting completed PDAG $\mathcal{P}^{c\prime\prime\prime}$. Furthermore, it is easy to see that every *other* edge in

$\mathcal{P}^{c'''}$ is compelled because these edges participate in v-structures that are also in $\mathcal{P}^{c'}$ (which has the adjacency between x and y). The result of adding the directed edge $z \rightarrow y$ to $\mathcal{P}^{c'''}$ is a PDAG that is identical to $\mathcal{P}^{c'}$, and thus InsertD(z, y) is a valid operator that completes the desired sequence for the lemma.

The final possibility is that in $\mathcal{P}^{c''}$ we have $\Pi_x = \Pi_y = \Pi_z$. In this case, we can apply InsertU(x, y) to $\mathcal{P}^{c''}$, which is clearly valid because $\mathcal{P}^{c''}$ contains no undirected edges. Because all other edges in $\mathcal{P}^{c''}$ participate in v-structures that also exist in $\mathcal{P}^{c'}$, and because x and y are adjacent in $\mathcal{P}^{c'}$, we know that all of these edges participate in v-structures after adding $x - y$ to $\mathcal{P}^{c''}$, and thus we know from Theorem 34 that the completed PDAG $\mathcal{P}^{c'''}$ that results from InsertU(x, y) is identical to $\mathcal{P}^{c''}$ except that it contains the undirected edge $x - y$. Following a similar set of arguments, we see that we can next apply InsertU(z, y) to $\mathcal{P}^{c'''}$ and the resulting PDAG is a completed PDAG $\mathcal{P}^{c''''}$ that is identical to $\mathcal{P}^{c'''}$ except that is contains the undirected edge $z - y$. Finally, because the result of simultaneously directing the two undirected edges $x - y$ and $z - y$ in $\mathcal{P}^{c''''}$ as $x \rightarrow y$ and $z \rightarrow y$ results in a PDAG that is identical to $\mathcal{P}^{c'}$, we conclude that MakeV(x, y, z) is a valid operator that completes the desired sequence for the lemma. □

A.10.3 Inserting Compelled Edges

In this section, we consider the phase that transforms the PDAG containing only edges that participate in v-structures into a PDAG that only contains compelled edges, using valid InsertD operators. We first prove the following proposition about the "rules approach" to identifying compelled edges in an equivalence class; this approach was detailed in Section A.3.

Proposition 56 *Let $\mathcal{P}^c$ be any completed PDAG containing only compelled edges. Then all of the edges not participating in a v-structure can be identified by undirecting them and then repeatedly applying only Rule 1 and Rule 2 from Section A.3.*

Proof: Suppose this is not the case, and consider the *last* edge $x \rightarrow y$ in $\mathcal{P}^c$ to be oriented using Rule 3, where there is a v-structure $w \rightarrow y \leftarrow z$ and the edges $x - w$ and $x - z$ have yet to be oriented. It is easy to see that the orientation of an edge $a \rightarrow b$ depends only on those edges $c \rightarrow d$ for which there is a directed path (of length 0 or more) from d to b; this property holds for the edges in the rules themselves, and by the transitivity of the directed-path relation, it holds in general. Because there is a directed edge from both x and w to y, there cannot be a directed path from y to either of these nodes, and thus $x - w$ can be oriented without first orienting $x \rightarrow y$. Thus if we do not perform the last application of Rule 3, $x - w$ will still be oriented by either Rule 2 or Rule 1. If the edge gets oriented as $x \rightarrow w$, then we can orient the edge between x and y using Rule 2. Otherwise, ($w \rightarrow x$), the edge between x and z, if not already oriented as $x \rightarrow z$, will be oriented by Rule 1, in which case we can again orient the edge between x and y using Rule 2. Because the last application of Rule 3 is never necessary, none of them are necessary and the proposition follows. □

We can now prove the main result of this section.

Lemma 57 *Let $\mathcal{P}^{c\prime}$ be any completed PDAG containing only compelled edges, and let $\mathcal{P}^c$ be the completed PDAG containing precisely those compelled edges that participate in v-structures in $\mathcal{P}^{c\prime}$. Let c denote the number of edges in $\mathcal{P}^{c\prime}$ that do not participate in a v-structure. Then there exists a sequence of $c+1$ completed PDAGs $\mathcal{P}^c = \mathcal{P}^c{}_1, ..., \mathcal{P}^c{}_{c+1} = \mathcal{P}^{c\prime}$ for which $\mathcal{P}^c{}_{i+1}$ results from a valid InsertD operator applied to $\mathcal{P}^c{}_i$.*

Proof: Induction on the number of edges c that do not participate in v-structures in $\mathcal{P}^{c\prime}$. For the basis, if $c = 0$ then clearly $\mathcal{P}^c = \mathcal{P}^{c\prime}$ and the lemma follows trivially. For the induction step, we assume that the lemma is true for $c < k$ and consider the case when $c = k$.

From Proposition 56, we can orient all of the c compelled edges not participating in v-structures in $\mathcal{P}^{c\prime}$ by first undirecting them and then repeatedly applying either Rule 1 or Rule 2 from Section A.3. Consider the *last* edge $x - y$ that is oriented in $\mathcal{P}^{c\prime}$ using this procedure. Clearly, because only Rule 1 and Rule 2 are applied, none of the previous edge orientations depend on the presence of $x - y$. Thus if we delete the edge between x and y from $\mathcal{P}^{c\prime}$, resulting in $\mathcal{P}^{c\prime\prime}$, all of the edges remain compelled in the same direction; by the induction hypothesis there is the desired sequence of valid InsertD operators that transform $\mathcal{P}^c$ into $\mathcal{P}^{c\prime\prime}$. Without loss of generality, assume the edge between x and y is oriented as $x \to y$ in $\mathcal{P}^{c\prime}$. Because the PDAG that results from adding the edge $x \to y$ to $\mathcal{P}^{c\prime\prime}$ is identical to $\mathcal{P}^{c\prime}$, we conclude that InsertD(x, y) is valid in $\mathcal{P}^c$, and thus we can append this operator to the sequence that produced $\mathcal{P}^{c\prime\prime}$ and the lemma follows. □

A.10.4 Inserting Reversible Edges

In this section, we show how to transform, using a sequence of valid InsertU operators, a completed PDAG that contains only directed edges into a completed PDAG with the same set of directed edges and additional undirected edges. We first need the following result.

Proposition 58 *Let $\mathcal{P}^c$ be any completed PDAG containing the undirected edge $x - y$. The result of deleting $x - y$ from $\mathcal{P}^c$ is a completed PDAG that is identical to $\mathcal{P}^c$ except that it does not contain the edge $x - y$.*

Proof: Let $\mathcal{P}^{c\prime}$ be the completed PDAG that results from deleting $x - y$. It is easy to see that any v-structure contained in $\mathcal{P}^c$ remains in $\mathcal{P}^{c\prime}$. Consider the sequence of rules from Section A.3 applied to $\mathcal{P}^c$ to determine the orientation of the compelled edges that do not participate in v-structures. The only rule that can potentially involve the edge $x - y$ is Rule 3. Thus every *other* rule also applies to $\mathcal{P}^{c\prime}$. But for any Rule 3 application that involves $x - y$ in $\mathcal{P}^c$, the edge that gets oriented in $\mathcal{P}^c$ must participate in a v-structure in $\mathcal{P}^{c\prime}$ and is thus already directed. Thus every directed edge in $\mathcal{P}^c$ is also directed in $\mathcal{P}^{c\prime}$. The proof follows by noting that no *extra* rule can apply after deleting an undirected (i.e., reversible) edge. □

We can now prove the main result of this section.

Lemma 59 *Let $\mathcal{P}^{c\prime}$ be any completed PDAG, and let $\mathcal{P}^c$ be any completed PDAG that contains identical directed edges and no undirected edges. Let c denote the number of undirected edges in $\mathcal{P}^{c\prime}$. Then there exists a sequence of $c+1$ completed PDAGs $\mathcal{P}^c = \mathcal{P}^c{}_1, ..., \mathcal{P}^c{}_{c+1} = \mathcal{P}^{c\prime}$ for which $\mathcal{P}^c{}_{i+1}$ results from a valid InsertU operator applied to $\mathcal{P}^c{}_i$.*

Proof: We first show that there exists a sequence of valid DeleteU operators that can transform $\mathcal{P}^{c\prime}$ into $\mathcal{P}^{c}$, and then show that each such deletion has a corresponding inverse InsertU operator.

From Corollary 51 in Section A.10.1, as long as there is at least one undirected edge in $\mathcal{P}^{c\prime}$, there exists a valid DeleteU operator that we can apply to $\mathcal{P}^{c\prime}$. Furthermore, from Proposition 58, after each such deletion, the resulting completed PDAG only differs from $\mathcal{P}^{c\prime}$ by the presence of the deleted edge. Thus there exists a sequence $\mathcal{P}^{c\prime} = \mathcal{P}^{c\prime}_{1}, \ldots, \mathcal{P}^{c\prime}_{c+1} = \mathcal{P}^{c}$ for which $\mathcal{P}^{c\prime}_{i+1}$ results from a valid DeleteU operator applied to $\mathcal{P}^{c\prime}_{i}$.

We complete the proof by demonstrating that for any pair $\mathcal{P}^{c\prime}_{i}, \mathcal{P}^{c\prime}_{i+1}$, we can transform $\mathcal{P}^{c\prime}_{i+1}$ into $\mathcal{P}^{c\prime}_{i}$ by a valid InsertU operator. Let $x - y$ be the edge that was deleted in the transition from $\mathcal{P}^{c\prime}_{i}$ to $\mathcal{P}^{c\prime}_{i+1}$, and consider the PDAG $\mathcal{P}$ that results from inserting $x - y$ into $\mathcal{P}^{c\prime}_{i+1}$. Because $\mathcal{P}^{c\prime}_{i}$ differs from $\mathcal{P}^{c\prime}_{i+1}$ only by the presence of $x - y$, it follows that $\mathcal{P} = \mathcal{P}^{c\prime}_{i}$. $\square$

A.10.5 Proof of Theorem 4

We now provide the proof of Theorem 4; given the results from the previous sections, the proof follows easily.

Theorem 4 *The operators InsertU, InsertD, DeleteU, DeleteD, and MakeV are complete.*

Proof: Consider any pair of completed PDAGs $\mathcal{P}^{c}_{1}$ and $\mathcal{P}^{c}_{2}$. We can transform $\mathcal{P}^{c}_{1}$ into $\mathcal{P}^{c}_{2}$ by a sequence of the given operators as follows. First, from Lemma 52, we know we can transform $\mathcal{P}^{c}_{1}$ into the completed PDAG containing no edges by a sequence of valid DeleteU and DeleteD operators. Next, from Lemma 55, we can transform $\mathcal{P}^{c}_{1}$ into the completed PDAG that contains precisely those edges in $\mathcal{P}^{c}_{2}$ that participate in v-structures using a sequence of valid InsertU, InsertD and MakeV operators. Next, from Lemma 57, we can transform $\mathcal{P}^{c}_{1}$ into the completed PDAG that contains precisely those edges that are compelled in $\mathcal{P}^{c}_{2}$ using a sequence of InsertD operators. Finally, from Lemma 59, we can transform $\mathcal{P}^{c}_{1}$ into $\mathcal{P}^{c}_{2}$ by a sequence of valid InsertU operators. $\square$