Journal of Artificial Intelligence Research 59 (2017) 771-813 Submitted 1/17; published 8/17

Uniform Random Generation and Dominance Testing for CP-Nets

Thomas E. Allen THOMAS.ALLEN@CENTRE.EDU
Centre College

Judy Goldsmith GOLDSMIT@CS.UKY.EDU
University of Kentucky

Hayden Elizabeth Justice HAYDEN.JUSTICE259@TOPPER.WKU.EDU
Western Kentucky University

Nicholas Mattei N.MATTEI@IBM.COM
IBM T.J. Watson Research Center

Kayla Raines KDSM225@UKY.EDU
University of Kentucky

Abstract

The generation of preferences represented as CP-nets for experiments and empirical testing has typically been done in an ad hoc manner that may have introduced a large statistical bias in previous experimental work. We present novel polynomial-time algorithms for generating CP-nets with n nodes and maximum in-degree c uniformly at random. We extend this result to several statistical cultures commonly used in the social choice and preference reasoning literature. A CP-net is composed of both a graph and underlying cp-statements; our algorithm is the first to provably generate both the graph structure and cp-statements, and hence the underlying preference orders themselves, uniformly at random. We have released this code as a free and open source project. We use the uniform generation algorithm to investigate the maximum and expected flipping lengths, i.e., the maximum length over all outcomes o_1 and o_2, of a minimal proof that o_1 is preferred to o_2. Using our new statistical evidence, we conjecture that, for CP-nets with binary variables and complete conditional preference tables, the expected flipping length is polynomial in the number of preference variables. This has positive implications for the usability of CP-nets as compact preference models.

1. Introduction

Modeling, capturing, and reasoning with preferences is a fundamental topic that spans artificial intelligence, including constraint programming (Rossi, Venable, & Walsh, 2011), social choice (Chevaleyre, Endriss, Lang, & Maudet, 2008; Brandt, Conitzer, Endriss, Lang, & Procaccia, 2016), recommendation systems (Ricci, Rokach, Shapira, & Kantor, 2011), machine learning (Fürnkranz & Hüllermeier, 2010), and multi-agent systems (Goldsmith & Junker, 2009). Preference handling systems require some way to model, learn, reason with, and aggregate preferences. In this work we focus on one of the most commonly studied preference models, *conditional preference networks (CP-nets)* (Boutilier et al., 2004).

Preferences involve at least one user or decision maker and a set of objects $\mathcal{O}$, known as candidates, outcomes, or alternatives, depending on the context. A user prefers o to o', $o \succ o'$, if o is "better" or makes her "happier." If it is difficult or impossible to compare two objects, then they are incomparable, $o \parallel o'$. In this work we are concerned with objects that are combinatorial (factored), i.e., the objects are defined as a subset of the Cartesian product of a number of features or variables.

A common example is choosing a meal where we have an option of appetizer, main course, and dessert, each of which has a domain of values. In the common general additive independence (GAI) setting we assume that there is a utility for each of the features and the value of the object is the sum of these utility values (Bacchus & Grove, 1995; Fishburn, 1999; Gonzales & Perny, 2004). However, our interest is in *ordinal* preferences, i.e., those that specify only the ordering and not specific values, as these types of preferences are generally considered to be easier to elicit from end users.

CP-nets are a factored, compact, and qualitative representation used to model, elicit, and reason about preferences. CP-nets have garnered considerable attention, particularly within the preference handling community (Domshlak, Hüllermeier, Kaci, & Prade, 2011). CP-nets have many potential and important applications—automated negotiation (Aydoğan et al., 2013), interest-matching in social networks (Wicker & Doyle, 2007), cybersecurity (Bistarelli, Fioravanti, & Peretti, 2007), and as aggregation primitives for making group decisions (Lang & Xia, 2009; Mattei, Pini, Rossi, & Venable, 2013; Xia, Conitzer, & Lang, 2011a; Dorn, Krüger, & Scharpfenecker, 2015; Dorn & Krüger, 2015), to name a few. One explanation for the popularity of CP-nets is their seemingly intuitive and visual representation of the language many of us use to describe what we want. There are a variety of other graphical compact preferences representations, such as lexicographic preference models (Yaman, Walsh, Littman, et al., 2010), LP trees (Booth, Chevaleyre, Lang, Mengin, & Sombattheera, 2010; Liu & Truszczynski, 2013), and CP-net variants such as TCP-nets (Brafman, Domshlak, & Shimony, 2006), UCP-nets (Boutilier, Bacchus, & Brafman, 2001), CI-nets (Bouveret, Endriss, & Lang, 2009), mCP-nets (Rossi, Venable, & Walsh, 2004), PCP-nets (Cornelio, 2012; Cornelio, Goldsmith, Mattei, Rossi, & Venable, 2013; Bigot, Fargier, Mengin, & Zanuttini, 2013), and CP-nets with locally partially ordered preferences (Wilson, 2004); extensive surveys can be found in a number of works including those by Allen (2016), Domshlak et al. (2011), Rossi et al. (2011), and Amor, Dubois, Gouider, and Prade (2016).

One avenue for advancing research in preference handling is through the use of experimental studies. Ideally, these would investigate the actual choices that people make in various real-world tasks. However, real-world data are often messy, not openly available, notoriously difficult to collect reliably, hard to interpret, and nonexistent for certain preference formalisms including CP-nets (Allen, Chen, Goldsmith, Mattei, Popova, Regenwetter, Rossi, & Zwilling, 2015; Mattei & Walsh, 2013). In the absence of analytic results and real-world data, researchers typically turn to synthetic data to perform experiments (Cohen, 1995); the goal being to better understand the mathematical properties of the preference formalisms. *However, if one wishes to study preferences that are representable by CP-nets, it is not obvious how to generate synthetic datasets of CP-nets in a principled manner.*

The CP-net formalism encodes a subset of partial orders. If one wants to experiment with CP-nets, i.e., preferences that are representable as CP-nets that may come from human users or be extracted form text or other signals, then one needs to be able to generate random data from this subset of partial orders. Numerous papers involving CP-nets and related preference models use synthetic datasets generated at "random" (Guerin, 2012; Guerin, Allen, & Goldsmith, 2013; Eckhardt & Vojtáš, 2009, 2010; Li, Vo, & Kowalczyk, 2011; Liu, Yao, Xiong, Liu, & Wu, 2013; Liu, Xiong, Wu, Yao, & Liu, 2014; Santhanam, Basu, & Honavar, 2010; Kronegger, Lackner, Pfandler, & Pichler, 2014; Bigot, Mengin, & Zanuttini, 2014; Cornelio et al., 2013). Most of these studies leave the exact details of the generation method unspecified. Those that provide sufficient detail to replicate the procedure, such as our own work (Guerin et al., 2013), use a method that results in large sample bias in their "random" procedures. Fundamentally, this means that they do not sample the space of

CP-nets uniformly at random, but rather some other, unspecified distribution. This bias, as we will discuss in Section 3, is problematic as it may give misleading results by, e.g., under-sampling easy reasoning cases. Hence, with a skewed or unknown sample distribution, it is impossible to say what happens "on average" when testing algorithms for or working with CP-nets.

More generally, methods for generating random data have long been of interest to computer scientists—Alan Turing advocated for a random number generator in the 1951 Ferranti Mark I computer (Knuth, 1997)—and continue to be an active topic of research. Random generation not only of numbers, but of combinatorial objects such as spanning trees and paths in directed graphs have been studied across both mathematics and computer science (Kulkarni, 1990). To our knowledge, methods for generating complex preference models such as CP-nets in a uniform manner have not yet received attention. There is considerable value in being able to generate CP-nets uniformly at random, including: enabling experimental analysis of CP-net reasoning algorithms, unbiased blackbox testing, effective Monte Carlo algorithms, analysis of *all* CP-nets to better understand their properties, and simulations for decision making or social choice experiments. Complementing theoretical results with empirical experiments, whether from real data or from data generated according to a distribution, may provide a window into feasible algorithms that provide good results in practice; biased generation may heavily skew these results.

In social choice and preference handling experimentation, principled methods exist to generate simulated data using *generative cultures* (Berg, 1985; Walsh, 2011; Mattei, Forshee, & Goldsmith, 2012). Such cultures have their drawbacks and limitations (Regenwetter, Grogman, Marley, & Testlin, 2006; Popova, Regenwetter, & Mattei, 2013), but provide a first step in experimentation for fields where data are hard to gather. Generative cultures over strict, linear orders are well defined in social choice. However, there is not currently a computationally efficient way to generate cultures of preferences over more complex structures such as CP-nets. As a first step in generating CP-nets according to any statistical cultures used in social choice, we must be able to generate samples uniformly at random from the space of CP-nets.

In the next section we give an overview of CP-nets and the notations we will use throughout the paper. In Section 3 we discuss the challenges associated with generating CP-nets for computational testing and give the details of our main technical result, generating both the dependency structure and preference tables of CP-nets uniformly at random. We also discuss extensions of our generation procedure to statistical cultures typically defined in the preference and social choice literature (Brandt et al., 2016). In Section 4 we use our method to perform a comprehensive analysis on average flipping sequence length when preferences are generated uniformly at random. Finally in Section 5, based on the results in Section 4, we give a strong motivation for and technical description of defining a depth-limited notion of dominance testing for CP-nets.

1.1 Contributions

In this work we make the following fundamental contributions to the study of CP-nets with binary variable domains.

- We provide novel algorithms and open source code to generate CP-nets uniformly at random in polynomial time with any number of variables and a bound on in-degree.
- We show how to leverage these novel algorithms to generate CP-nets according to statistical cultures used in the study of preferences.

- We perform a comprehensive analysis of CP-nets drawn uniformly at random to argue that it is reasonable to constrain the search for dominance between outcomes.
- We propose two algorithms that incorporate depth limiting into the dominance testing problem for binary-valued CP-nets.

2. Conditional Preference Networks (CP-Nets)

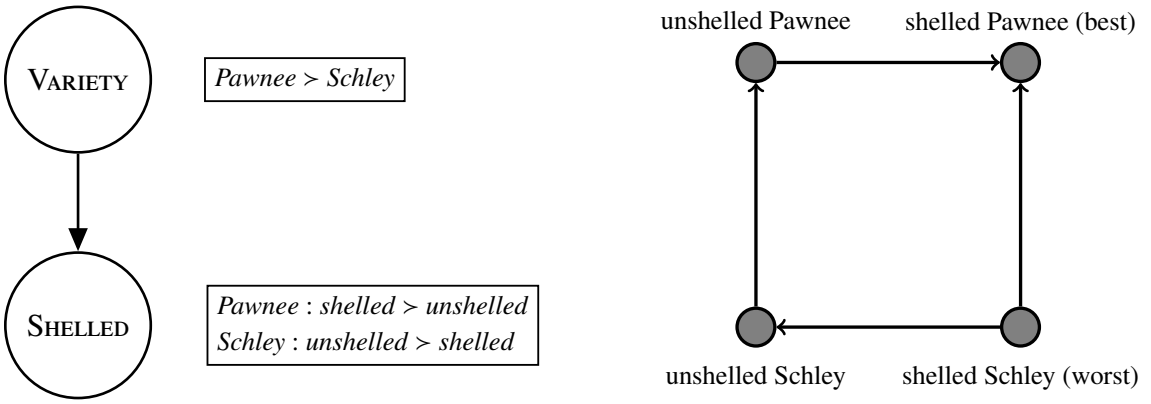
CP-nets were first proposed by Boutilier et al. (2004). They exploit conditional *ceteris paribus* preference rules to enable a compact representation of a preference relation. In their most general form, CP-nets are allowed to have cycles in the dependency graph, with only two constraints on the geometry: self-loops are disallowed, and a small constant bound on the in-degree of nodes in the dependency graph is assumed. The features that characterize the outcome space are discrete, but can be multivalued, and the tables that specify the local preferences can be partially specified (i.e., incomplete) and can express indifference as well as strict preference. While such general models can represent a broader range of problems, reasoning with such models may be intractable, and there is no guarantee that the resulting order on outcomes will be consistent. Consequently, various restrictions are usually applied to the set of possible CP-net models, either as a requirement for algorithms or as an aid in proving theoretical results. For example, it is usually assumed that the dependency graph is free of cycles and that indifference is disallowed. Many researchers limit attention to CP-nets with binary features (Xia et al., 2011a; Dimopoulos, Michael, & Athienitou, 2009; Kronegger et al., 2014; Lang & Mengin, 2008) or restrict the dependency graph to a subclass of directed acyclic graphs, such as polytrees. Note that regardless of the type of graph, a *bound on in-degree* is always assumed to ensure a compact model.

Example 1. *Consider a pastry chef who sometimes purchases pecans. Pecans can be characterized by various features, such as* VARIETY *(e.g.,* Pawnee *or* Schley*) and whether they have already been* SHELLED *(*shelled *or* unshelled*). When objects are factored in this way, they are typically called outcomes, because they are the outcome of how their characteristic features have been instantiated. Figure 1 shows both the CP-net and the induced preference graph that is encoded by the CP-net.*

When outcomes are factored, a subject may hold ceteris paribus *preferences over the features. When the chef says that she prefers Pawnee pecans, she does not necessarily mean that she prefers every Pawnee order to every Schley. Other factors may also affect her happiness with the order, such as price, quality, etc. However, if all other factors are held constant (Latin* ceteris paribus*), she prefers the Pawnee to other varieties.*

Ceteris paribus *preferences can be conditional or unconditional. Suppose the buyer always prefers the Pawnee variety to the Schley. In that case, the preference does not depend on any other factor, so it is said to be unconditional, written Pawnee* $\succ$ *Schley. However, suppose the chef prefers that pecans be shelled prior to shipping if they are Pawnee, but shipped unshelled if they are Schley. In this case the preferences are conditional, written as Pawnee* : *shelled* $\succ$ *unshelled and Schley* : *unshelled* $\succ$ *shelled.*

The nodes in Figure 1a represent the features over which the subject holds preferences. The directed edge from VARIETY *to* SHELLED *indicates that the subject's* ceteris paribus *preference for whether the pecan is shelled or unshelled depends on the variety. We refer to* VARIETY *in this case as the parent of* SHELLED*. In contrast, the preference over* VARIETY *is unconditional, so that node has no*



(a) Conditional Preference Network.

(b) Induced Preference Graph.
Note: Following Boutilier et al. (2004), edges in the induced preference graph are in the direction of improvement, i.e., from the less to the more preferred outcome.

Figure 1: CP-net and Induced Preference Graph

parents. The boxes beside each node are conditional preference tables *(CPTs) specifying the* ceteris paribus rules *over each node given the values of all combinations of values of the parent nodes.*

The CP-net in Figure 1a induces the preference graph shown in Figure 1b. Each rule in the CP-net induces a non-empty set of edges in the preference graph. For example, the rule Schley : *unshelled* $\succ$ *shelled corresponds to the directed edge* $(shelled, Schley) \rightarrow (unshelled, Schley)$ *in the preference graph, and the rule Pawnee* $\succ$ *Schley corresponds to the edges* $(shelled, Schley) \rightarrow (shelled, Pawnee)$ *and* $(unshelled, Schley) \rightarrow (unshelled, Pawnee)$.

Formally, a preference relation $\succ$ is a partial order (an antisymmetric, transitive binary relation) on a set of outcomes $\mathcal{O}$, where $o \succ o'$ means o is preferred to o'. We assume $\mathcal{O}$ is finite and can be factored into variables $\mathcal{V} = \{X_1, \ldots, X_n\}$ with associated domains $\mathrm{Dom}(X_i) = \{x_1^i, \ldots, x_d^i\}$ such that $\mathcal{O} = \mathrm{Dom}(X_1) \times \cdots \times \mathrm{Dom}(X_n)$. For ease of exposition we assume domain sizes are homogeneous; i.e., $|\mathrm{Dom}(X_i)| = d$ for all $X_i \in \mathcal{V}$.

When a variable is constrained to exactly one value of its domain, we say the value has been assigned to it. We designate by $\mathrm{Asst}(U)$ the set of all assignments to $U \subseteq \mathcal{V}$. An assignment to all variables $U = \mathcal{V}$ designates a unique outcome $o \in \mathcal{O}$. We denote by ux_k^i the combination of $u \in \mathrm{Asst}(U)$ and $x_k^i \in \mathrm{Dom}(X_i)$, where $U \cap \{X_i\} = \emptyset$. The symbols $\bar{}$ and $\setminus$ denote set complementation and subtraction; e.g., $\overline{\mathbf{U}} \equiv \mathcal{V} \setminus U$. For d-ary variables the total outcome space is $|\mathcal{O}| = d^n$; i.e., exponential space is required to store $\succ$. However, since $\mathcal{O}$ is factored, a CP-net potentially provides a compact model of $\succ$.

Definition 2. *A* conditional preference table $\mathrm{CPT}(X_i)$ *specifies the preferences over a node* $X_i \in \mathcal{V}$ *given an assignment to its parents. The set of parents of* X_i, $\mathrm{Pa}(X_i)$, *is the set of nodes in* $\mathcal{V} \setminus X_i$ *on which the values of* X_i *depend. Each CPT consists of* ceteris paribus *preference rules (CPRs) of the form* $u : \succ^i$ *specifying a linear order on* X_i *for all* $u \in \mathrm{Asst}(\mathrm{Pa}(X_i))$.

Definition 3. *A* CP-net *is a directed acyclic graph (DAG) in which each node $X_i \in \mathcal{V}$ is labeled with a conditional preference table for its variable. An edge (X_h, X_i) indicates that the preferences over X_i in $\succ$ depend on the value of X_h. We thus call X_h a* parent *of X_i. We denote by* $\mathrm{Pa}(X_i)$ *the set of all such parents.*

We use the term *dependency graph* to refer to the graph of a CP-net apart from its CPTs. The term *DAG* always refers to a *labeled* directed acyclic graph. The term $\mathrm{CPT}(X_i|X_h = x_k^h)$ denotes all rules of $\mathrm{CPT}(X_i)$ of the form $ux_k^h : \succ^i$ where $x_k^h \in \mathrm{Dom}(X_h)$, $X_h \in \mathrm{Pa}(X_i)$, $u \in \mathrm{Asst}(\mathrm{Pa}(X_i) \setminus \{X_h\})$. We assume here that CPTs are complete, i.e., have rules for all d^m assignments to parents, where $m = |\mathrm{Pa}(X_i)|$ is the *in-degree* of X_i. Since the number of rules is exponential in m, we make the customary assumption that in-degree is bounded by a small constant, i.e., $|\mathrm{Pa}(X_i)| \leq c$ for all X_i. We use $o[X_i]$ for the value of variable X_i in o, and $o[\overline{X_i}]$ for the value of all variables other than X_i in o.

As illustrated in Figure 1, a CP-net induces an exponentially larger graph known as the *induced preference graph*.

Definition 4 (Preference graph)**.** *The* induced preference graph *(PG) of a CP-net N is a digraph $H = (\mathcal{O}, \mathcal{C})$ in which $(o', o) \in \mathcal{C}$ if and only if there exists a CPR $u : \succ^i$ in the CPT of a node X_i in N, such that $(o[X_i], o'[X_i]) \in \succ^i$, $o[X_i] \neq o'[X_i]$, $o[\overline{X_i}] = o'[\overline{X_i}]$, $u \in \mathrm{Asst}(\mathrm{Pa}(X_i))$, $u = o[\mathrm{Pa}(X_i)] = o'[\mathrm{Pa}(X_i)]$, $o \in \mathcal{O}$, $o' \in \mathcal{O}$, and $X_i \in \mathcal{V}$. A directed edge from o' to o thus indicates that $o' \prec o$ and that the Hamming Distance between o and o' is 1.*

The *ceteris paribus* rules specify preferences for outcomes that differ in just one feature. The transitive closure of these rules sometimes allows us to compare outcomes that differ in more than one feature. For example, suppose only two items are in stock, (unshelled, Pawnee) and (shelled, Schley). In that case, we anticipate that the pastry chef will prefer the unshelled Pawnee; *Schley* : *unshelled* $\succ$ *shelled* entails that shelled Schley is less preferred than unshelled Schley, and *Pawnee* $\succ$ *Schley* entails that unshelled Schley is less preferred than unshelled Pawnee. Thus, we have:

$$(\mathit{shelled}, \mathit{Schley}) \prec (\mathit{unshelled}, \mathit{Schley}) \prec (\mathit{unshelled}, \mathit{Pawnee}).$$

A ranking over the outcomes induced from the CP-net like the one above is known as an *improving flipping sequence* and is the basis for reasoning about the relationship between arbitrary outcomes with respect to a CP-net. Note that every such improving flipping sequence corresponds to a path along directed edges in the preference graph. In this case, the flipping sequence counts as a proof that the subject prefers unshelled Pawnee to shelled Schley, and we say that the first outcome *dominates* the other. Later, we will also be interested in the length of a shortest such sequence connecting two outcomes. In this case we say that the ordered pair of outcomes (unshelled Pawnee, shelled Schley) has a *flipping length* of 2.

Definition 5 (Flipping sequence)**.** *A* flipping sequence *is a path in the induced preference graph of a CP-net.*

In general, if there exists an improving flipping sequence from o' to o, then we write $N \models o \succ o'$ and say the CP-net *entails the dominance* of o over o' in the induced order. If no path exists in either direction, i.e., if $N \models o \not\succ o'$ and $N \models o' \not\succ o$, then we can reason that the two outcomes are incomparable with respect to the CP-net; i.e., $N \models o \parallel o'$. Note that there may be multiple such paths between o and o'. The search for such a path is known as *dominance testing* (DT).

Definition 6 (DT problem)**.** *A* dominance testing problem *is a decision problem for which the input is a triple* (N, o, o') *consisting of a CP-net* N *on* $\mathcal{V} = \{X_1, \ldots, X_n\}$ *and outcomes* o *and* o'*,* $o \in \mathcal{O}$*,* $o' \in \mathcal{O}$*,* $\mathcal{O} \equiv \mathrm{Dom}(X_1) \times \cdots \times \mathrm{Dom}(X_n)$*. The answer is yes if and only if* $N \models o \succ o'$*.*

Definition 7 (Flipping length)**.** *The* flipping length *is the length of the shortest path between a pair of outcomes in the induced preference graph* H *of a CP-net* N*,*

$$\mathrm{FL}(N, o', o) = \mathrm{minpath}_H(o', o).$$

If no such path (flipping sequence) exists, then the flipping length is undefined, which we write as $\mathrm{FL}(N, o', o) = \infty$*.*

3. Generating CP-Nets Uniformly at Random

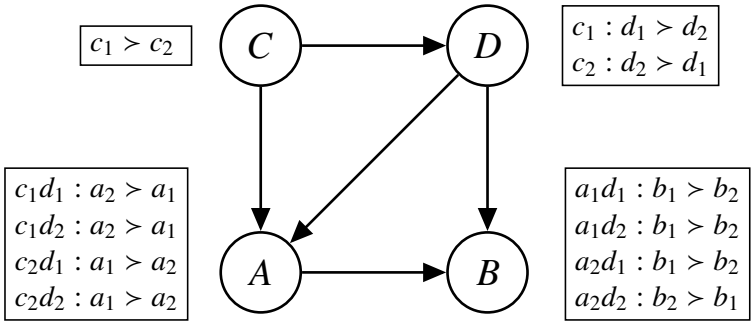
This section introduces a method for generating acyclic CP-nets uniformly at random. A key idea of this method is that the structure of a CP-net is equivalent to a tuple of sets representing the parents of nodes in the network. We will consider how to enumerate all such *dagcodes*, as these tuples are known (Steinsky, 2003), and how to calculate the number of CP-nets — the possible graphs and conditional preference tables (CPTs) — that extend a partial dagcode. The resulting novel recurrence makes it possible to generate the graph and CPTs, node by node, such that all CP-nets with a given domain size and bound on in-degree are equiprobable.

Section 3.1 highlights two problems, bias and degeneracy, that result from commonly used naïve generation methods. Section 3.2 shows how to encode and avoid degeneracy in the CPTs. Section 3.3 explains how to encode and count the dependency graphs. These results are then brought together in Section 3.4 to create an algorithm that samples the space of CP-nets uniformly. We extend our results in Section 3.5 to show how, using the uniform random generator, we can generate preferences according to generative cultures commonly used in social choice. Note that in this section it is assumed that domains are *homogeneous* but possibly multivalued, i.e., $d = d_1 = \cdots = d_n$, where $d_i = |\mathrm{Dom}(X_i)|$, for all $X_i \in \mathcal{V}$, and that the CPTs are *complete*, i.e., have d^m rules, one for every assignment to the m parents, $m = |\mathrm{Pa}(X_i)|$.

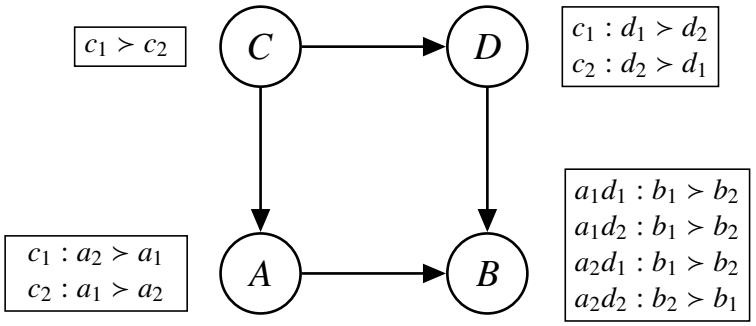
3.1 Naïve Generation, Bias, and Degeneracy

If one wants to generate CP-nets without regard for the resulting distribution, many simple random methods exist. For example, initialize a CP-net with n nodes, no edges, and empty CPTs; choose a random subset of pairs (X_h, X_i), $h < i$, inserting an edge from each X_h to X_i; generate a CPT for each X_i with $d^{|\mathrm{Pa}(X_i)|}$ rules, each a random permutation of the d values of X_i; and randomly permute the n labels. One suspects that something along these lines is meant when a paper states, "We generated 1000 CP-nets at random." However, this naïve approach to generation leads to two problems, *degeneracy* and *bias*. Let us first consider the problem of *degeneracy*, which occurs when one or more dependencies in the graph are not reflected in the conditional preference rules (CPRs).

Example 8. *Consider the CP-net in Figure 2. The edge* (D, A) *in Figure 2a indicates that the preference over the values of A depends on the value of D. However, in examining the CPT of A closely, one can observe that the preference over A does not in fact depend on D. The preferences can thus be represented by the simpler CP-net shown in Figure 2b.*

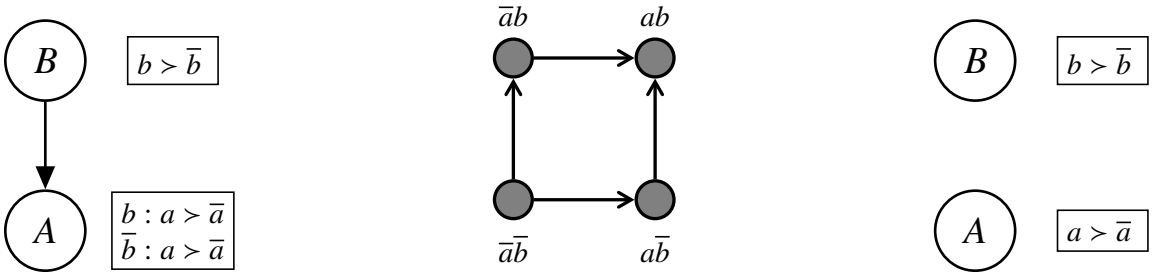


(a) Degenerate CP-Net



(b) Equivalent Non-Degenerate CP-Net

Figure 2: An Example of Degeneracy in CP-Nets



(a) One Edge (b) Induced PG (c) No Edge

Figure 3: Degeneracy Can Violate Basic Assumptions of an Experiment

Degeneracy in synthetic datasets is problematic for two reasons. First, dependencies in the graph, such as edge (D, A) in Example 8, can be *fictional*; that is, the presence of an edge in the graph does not necessarily express any factual information about the induced preference order. Second, if degeneracy can occur, multiple, apparently different CP-net models can map to the same induced preference order.

Example 9. *Suppose a researcher wants to test a new DT algorithm to understand how running time varies as the number of edges in the network increases. As a first step, he generates two sets of*

CP-nets using a naïve approach. The CP-nets in each set have binary domains and two nodes; the first set has just one edge, and the second set has no edge. However, if each CPR is assigned in the manner of a coin flip, an expected 50% of the CP-nets in the first set will be degenerate like the one in Figure 3a. As such, its induced preference graph, shown in Figure 3b, will be identical to that of the no-edge CP-net in Figure 3c. Thus, one of the basic assumptions of the experiment, that the two sets induce different preference orders, is violated.

Naïve generation methods also result in another problem that can violate the basic assumptions of experiments: *statistical bias*. To understand this bias, consider the following dependency graphs and their associated CP-net counts[1] as shown in Figure 4. Both graphs have 5 nodes and 5 edges, but the number of CP-nets associated with each graph differs greatly. Observe that for the chain-shaped graph on the left, when $d = 2$, there are just two ways to choose each of the n CPTs such that they are consistent with the dependency graph. The CPT of root E could be $[e_1 \succ e_2]$ or $[e_2 \succ e_1]$. The other nodes, each of which has only one parent, also have two (non-degenerate) possibilities for their CPT; e.g., CPT(A) could be $[b_1 : a_1 \succ a_2,\ b_2 : a_2 \succ a_1]$ or $[b_1 : a_2 \succ a_1,\ b_2 : a_1 \succ a_2]$. However, in the case of the star-shaped graph on the right, CPT(A') has $d^4 = 16$ rules, each with $d! = 2$ possible orderings. In all, over one million CP-nets have the graph on the right, while only 32 have the graph on the left. Further observe that the ratio of this imbalance very rapidly increases with the domain size d. Thus, if the algorithm above in fact generated the two graphs with equal likelihood, it would grossly oversample CP-nets with the first graph, while correspondingly undersampling those with the second.

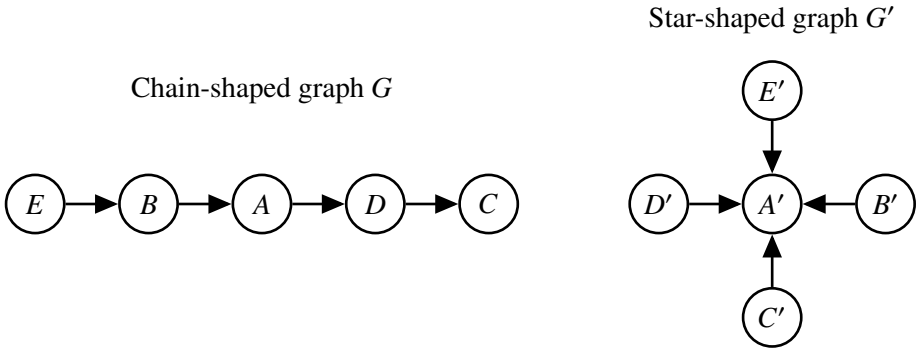
However, the naïve algorithm *does not even generate the two DAGs with equal likelihood*. Because there are $5! = 120$ ways to permute the labels of the first DAG, but only 5 ways to permute those of the second, the star-shaped DAG on the right would be generated 24 times as often as the chain-shaped DAG on the left. Despite this, the CP-nets in the star-shaped case would still be greatly undersampled.

3.2 Counting and Generating the CPTs

The notion of a degenerate CPT introduced in Section 3.1 can be generalized with the help of a *bijection* (a mapping that is *one-to-one* and *onto*) with discrete multivalued functions. One can model each CPT(X_i) as a function $f : \{0, \ldots, d-1\}^m \to \{0, \ldots, d! - 1\}$, where $m = |\mathrm{Pa}(X_i)|$. The inputs correspond to the values of the m parents of X_i. The output corresponds to one of the $d!$ orders of the domain of X_i.

Observe that if variables are binary ($d = 2$), f is a Boolean function. In the Boolean case the values x_1^h and x_2^h of each parent X_h can map to 0 and 1 respectively. The two possible linear orders $x_1^i \succ x_2^i$ and $x_2^i \succ x_1^i$ can correspond to outputs 0 and 1. One can thus model the degenerate CPT of node A from Example 8 with the truth table in Figure 5. If variables are multivalued, the mapping is similar (for details see Allen, 2016). For mapping the outputs, one can use Lehmer codes (1960) (see also the discussion of the factorial number system in Knuth, 1998, Section 3.2.2, Algorithm P). Thus, any CPT can be encoded as an equivalent *function vector* F of length d^m. This mapping helps us formalize the notion of degeneracy introduced in Section 3.1.

1. The CP-net counts for G and G' are $\psi_d(0)(\psi_d(1))^4$ and $(\psi_d(0))^4\psi_d(4)$, respectively, where $\psi_d(m)$ is the number of non-degenerate CPTs with m parents and d-ary domains, as discussed in Section 3.2.



(a) Dependency Graphs that Differ in Maximum In-Degree

d	CP-nets with G	CP-nets with G'
2	32	1.03×10^{6}
3	7,776	1.39×10^{66}
4	7,962,624	7.16×10^{358}
5	24,883,200,000	6.38×10^{1307}

(b) Number d-ary CP-nets for Dependency Graphs Above

Figure 4: How Naïve Generation Can Lead to Bias

CPT(A)
$c_1 d_1 : a_2 \succ a_1$
$c_1 d_2 : a_2 \succ a_1$
$c_2 d_1 : a_1 \succ a_2$
$c_2 d_2 : a_1 \succ a_2$

In_1	In_2	F
0	0	1
0	1	1
1	0	0
1	1	0

Figure 5: CPT and Corresponding Boolean Function

Definition 10 (Degeneracy)**.** *A function f is* vacuous[2] *in variable u_k if and only if its output never depends on u_k; i.e., for all $u \in \{0, \ldots, d-1\}^m$,*

$$\begin{aligned} & f(u_1, \ldots, u_{k-1}, 0, u_{k+1}, \ldots, u_m) \\ & \quad = f(u_1, \ldots, u_{k-1}, 1, u_{k+1}, \ldots, u_m) \\ & \quad = \cdots = f(u_1, \ldots, u_{k-1}, d-1, u_{k+1}, \ldots, u_m). \end{aligned}$$

Function f is degenerate *if it is vacuous in a variable; otherwise, it is* non-degenerate*. By extension, a CPT is degenerate (respectively vacuous in a parent variable) if function f to which it maps is degenerate (respectively vacuous in an input).*

Let us denote by $\phi_d(m)$ the total number of distinct CPTs for a node with m parents. Let us denote by $\chi_d(m)$ the number of those that are degenerate, and by $\psi_d(m)$ the number that are non-

2. Such a variable is sometimes said to be *vacated* or *fictional* (O'Connor, 1997).

degenerate. It follows immediately from Definition 10 that

$$\phi_d(m) = \chi_d(m) + \psi_d(m). \tag{1}$$

First consider binary domains, $d = 2$. Because CPTs and Boolean functions are in one-to-one correspondence, $\phi_2(m)$ is equivalent to the number of Boolean functions of m inputs, and $\psi_2(m)$ is equivalent to the number of non-degenerate Boolean functions. Hu (1968, §2) (also see Harrison, 1965 and O'Connor, 1997) proved that for Boolean functions:

$$\phi_2(m) = 2^{2^m}, \tag{2}$$

$$\psi_2(m) = \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} 2^{2^k}, \tag{3}$$

$$\lim_{m\to\infty} \frac{\chi_2(m)}{\phi_2(m)} = 0, \tag{4}$$

and

$$\lim_{m\to\infty} \frac{\psi_2(m)}{\phi_2(m)} = 1. \tag{5}$$

Let us now generalize these results to homogeneous domains of arbitrary size $d > 0$.

Theorem 11 (Number of CPTs)**.** *For every non-negative integer d, m,*

$$\phi_d(m) = d!^{d^m}. \tag{6}$$

Proof. Each rule of CPT(X_i) specifies one of $d!$ linear orders of Dom(X_i). The number of CPRs is $|\text{Asst}(\text{Pa}(X_i))| = d^m$, where $m = |\text{Pa}(X_i)|$. Because each rule can be assigned independently, $\phi_d(m) = d!^{d^m}$. ❑

Theorem 12 (Number of Non-Degenerate CPTs)**.** *For every non-negative integer d, m,*

$$\psi_d(m) = \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} d!^{d^k}. \tag{7}$$

Theorem 13 (Convergence to Non-Degeneracy)**.** *For every non-negative integer d, m,*

$$\lim_{m\to\infty} \frac{\chi_d(m)}{\phi_d(m)} = 0, \tag{8}$$

$$\lim_{m\to\infty} \frac{\psi_d(m)}{\phi_d(m)} = 1. \tag{9}$$

The proofs of Theorems 12 and 13 very closely follow the rather lengthy proofs of Hu (1968)[§10] for Boolean functions, except that ϕ_d is needed in place of ϕ_2; the primary change is to replace every occurrence of 2^{2^k} with $d!^{d^k}$.

We observe that deciding whether a CPT is degenerate can be computed in time polynomial in the size of the CPT (for details see Allen, 2016). While O'Connor (1997) showed that deciding whether a Boolean function is vacuous in a variable (and hence degenerate) is Co-NP-complete, this assumes the input takes the form of arbitrary Boolean expressions. However, the size of a CPT

IS-FUNCTION-DEGENERATE(F, d, m)

Input: F output vector with d^m rows
d domain size
m number of inputs

Output: returns **true** if F is degenerate, **false** if non-degenerate

1: **for** $h \leftarrow 0$ **to** $m - 1$ **do** ▷ Iterate over the m inputs
2: *vacuous* $\leftarrow$ **true** ▷ Assume vacuous in input h until proved otherwise
3: **for** $k \leftarrow 1$ **to** $d - 1$ **do** ▷ Iterate over all domain values except 0
4: $r \leftarrow 0$ ▷ Indexes entries for which $\text{In}_h = 0$
5: $s \leftarrow d^h k$ ▷ Indexes entries for which $\text{In}_h = k$
6: $t \leftarrow d^h$ ▷ Count down until time to skip "column" In_h
7: **while** $s < d^m$ **do**
8: **if** $F[r + 1] \neq F[s + 1]$ **then** ▷ Note indexing starts at 1
9: *vacuous* $\leftarrow$ **false** ▷ Output depends on input h
10: **else**
11: $r \leftarrow r + 1$ ▷ Continue sequential search through F
12: $s \leftarrow s + 1$
13: $t \leftarrow t - 1$
14: **if** $t = 0$ **then**
15: $r \leftarrow r - d^h + d^{h+1}$ ▷ Skip "column" In_h
16: $s \leftarrow s - d^h + d^{h+1}$
17: $t \leftarrow d^h$
18: **end if**
19: **end if**
20: **end while**
21: **if not** *vacuous* **then**
22: **break** ▷ F depends on In_h, so move on to next input
23: **end if**
24: **end for**
25: **if** *vacuous* **then**
26: **return true** ▷ Since F is vacuous in input h, it is degenerate
27: **end if**
28: **end for**
29: **return false** ▷ F depends on all m inputs; therefore it is non-degenerate

Figure 6: Algorithm Is-Function-Degenerate Decides Whether Function Vector F is Degenerate

is already exponential in m, since it is assumed that CPTs are complete; we have no contradiction here.

Let us now consider how to leverage these results to generate non-degenerate CPTs in an efficient, uniformly random manner. For tiny values of d and m, one can choose uniformly from a modest-sized table of non-degenerate functions (e.g., $\psi_2(4) = 64594$). For larger values, one can use *rejection sampling*, generating a random integer in the range $(0\,..\,d! - 1)$ for each of the d^m elements of vector F and repeating this process in the *unlikely* event (e.g., < 0.0001 for $m > 4$ and very rapidly converging to 0 as m increases) that the result is degenerate. With probability $\psi_d(m)/\phi_d(m)$, asymptotic to 1, a non-degenerate CPT is obtained on a given attempt. Finally, observe that it is possible to generate *all* non-degenerate CPTs by generating all $\phi_d(m)$ vectors F and outputting the corresponding CPT only when Is-Function-Degenerate(F), defined in Figure 6, answers **false**.

3.3 Encoding and Counting Dependency Graphs

This section considers how to model the dependency graph as a *dagcode* (Steinsky, 2003), inspired by Prüfer codes for labeled trees (Kreher & Stinson, 1999). The encoding makes it easier to count the number of ways to complete a partially constructed DAG in order to avoid bias. In this section the dagcode is first treated as an abstraction and then related to the dependency graph.

Definition 14 (Dagcode)**.** *For any positive integer n, a* dagcode $A = \langle A_1, \ldots, A_{n-1} \rangle$ *is a tuple of* $n - 1$ *subsets* $A_j \subset \{1, \ldots, n\}$ *that satisfy the* cardinality constraint

$$\left| \bigcup_{k \leq j} A_k \right| \leq j \tag{10}$$

for all j, $1 \leq j < n$.

Observe from Definition 14 that tuples $\langle \{1\}, \{1, 3\} \rangle$ and $\langle \{3\}, \emptyset \rangle$ are valid dagcodes (in which $n = 3$), but $\langle \{1, 2\}, \emptyset \rangle$ and $\langle \emptyset, \{1, 2, 3\} \rangle$ are not, since each violates the cardinality constraint. Steinsky (2003) proved that dagcodes correspond one-to-one with DAGs and described efficient algorithms for converting dagcodes to DAGs and vice versa. The algorithm shown in Figure 7 maps an encoding A to its corresponding graph G.

Applied to CP-nets, each subset $A_j \subset \{1, \ldots, n\}$ in the dagcode corresponds to the parents of some node X_i in the dependency graph: i.e., $h \in A_j \implies X_h \in \mathrm{Pa}(X_i)$. Note that the root node with the smallest label is implicit; informally, it is helpful to consider every dagcode as having an implicit element $A_0 \equiv \emptyset$. The order in which the remaining $n - 1$ parent sets $\mathrm{Pa}(X_i)$ occur in the dagcode depends on the order of the child node X_i with respect to other nodes in the graph and the relative size of node label i, as follows:

1. If X_h is an ancestor of X_i in the DAG, the encoded parent set $\mathrm{Pa}(X_h)$ is ordered before $\mathrm{Pa}(X_i)$ in the dagcode.
2. If $h < i$ and X_h is neither an ancestor nor a descendant of X_i, then $\mathrm{Pa}(X_h)$ is ordered before $\mathrm{Pa}(X_i)$.

Example 15. *The dagcode* $\langle \{1\}, \{1, 3\} \rangle$ *corresponds to a DAG with* $n = 3$ *nodes depicted below.*

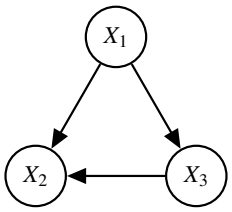
DAGCODE-TO-DAG(A)

Input: dagcode $A = \langle A_1, \ldots, A_{n-1} \rangle$

Output: corresponding DAG G

$n \leftarrow \text{length}(A) + 1$
$Q \leftarrow \{1, \ldots, n\}$
initialize DAG G with n nodes and no edges
for $j \leftarrow n - 1$ **downto** 1 **do** ▷ Iterate over dagcode: A_j is the parent set
  $i \leftarrow \max\left(Q \setminus \bigcup_{k=1}^{j} A_k\right)$ ▷ of X_i, where i is the largest unused label
  for all $h \in A_j$ **do**
    insert edge to X_i from its parent X_h
  end for
  $Q \leftarrow Q \setminus \{i\}$
end for
output DAG G

Figure 7: Algorithm DAGCODE-TO-DAG Generates a DAG from Its Dagcode



The subsets $\{1\}$ *and* $\{1, 3\}$ *indicate that one node has parent* X_1 *and another has parents* X_1 *and* X_3*; the third (implicit) node is a root. The mapping from parent sets to their children can be recovered from* DAGCODE-TO-DAG *(Figure 7) (Steinsky, 2003, adapted) working right to left as follows:* $A_2 = \{1, 3\}$ *corresponds to the parents of* X_2 *since 2 is the largest unassigned label not in* $\{1\} \cup \{1, 3\}$*.* $A_1 = \{1\}$ *corresponds to the parents of* X_3 *since 3 is the largest unassigned label not in* $\{1\}$*. The remaining root node is* X_1*.*

Observe that a DAG has bounded in-degree c if and only if

$$|A_j| \leq c \tag{11}$$

for all A_j in the corresponding dagcode: every node X_i in the DAG corresponds to the parent set of an element A_j in the dagcode, with the exception of a root with in-degree 0.

The generation method in Section 3.4 depends on counting the number of *extensions* to a partially specified graph. Consider a partial encoding $A_{<3} = \langle \{1\}, \{2\}, \_ \rangle$ of a graph with $n = 4$ nodes and bound $c = 1$ on in-degree. Here the $\_$ could be any subset of $\{1, 2, 3, 4\}$ of cardinality 0 or 1 such that the resulting dagcode is valid, viz., $\emptyset, \{1\}, \{2\}, \{3\}$, or $\{4\}$.[3] One can generalize this as follows.

3. Note that while a partial dagcode specifies the parents of some nodes, the mapping from parent sets to their children in general cannot be determined until the dagcode is fully specified.

ALL-DAGs($n, c, j, q, U, A_{<j}$)

Inputs:

	n	number of nodes
	c	bound on in-degree
	j	index of current element A_j
	q	current value of $\lvert U \rvert$
	U	current value of $A_1 \cup \cdots \cup A_{j-1}$
	$A_{<j}$	partial dagcode

1: **if** $j = n$ **then**
2: DAGCODE-TO-DAG($A_{<n}$)
3: **return**
4: **end if**
5: **for all** $s, t \geq 0, s \leq q, s + t \leq c, q + t \leq j$ **do**
6: **for all** $S \subseteq U,\ |S| = s$ **do**
7: **for all** $T \subseteq \overline{U},\ |T| = t$ **do**
8: $A_j \leftarrow S \cup T$; include A_j with $A_{<j}$ to form $A_{\leq j}$
9: ALL-DAGs($n, c, j + 1, q + t, U \cup A_j, A_{\leq j}$)
10: **end for**
11: **end for**
12: **end for**

Figure 8: Algorithm: Generate All DAGs that Extend Dagcode $A_{<j}$

Definition 16 (Partial dagcode)**.** *A* partial dagcode $A_{<j} = \langle A_1, \ldots, A_{j-1}, \_, \ldots, \_ \rangle$ *is a dagcode for which only elements* A_1 *through* A_{j-1} *have been specified, such that*

$$\left| \bigcup_{\ell \leq k} A_\ell \right| \leq k \tag{12}$$

for all k, $1 \leq k < j$, *and all* $A_k \subset V$, $V = \{1, \ldots, n\}$.

A partial dagcode is said to respect a bound c on in-degree *when*

$$|A_k| \leq c \tag{13}$$

for all A_k, *where* c *is an arbitrary non-negative integer.*

The algorithm shown in Figure 8 generates all extensions to $A_{<j}$ by recursively combining $A_{<j}$ with each A_j such that the resulting partial dagcode $A_{<j+1}$ satisfies the constraints on cardinality and in-degree. To generate all DAGs with n nodes and bound c on in-degree, ALL-DAGs($n, c, 1, 0, \emptyset, A_{<1}$) is called.

Theorem 17. ALL-DAGs *generates each DAG exactly once.*

Proof. Because dagcodes are in one-to-one correspondence with DAGs (Steinsky, 2003, Cor. 1), it suffices to show that each dagcode is generated exactly once. For this let us use the *recursion invariant:* Each time Line 1 of ALL-DAGs is reached, $A_{<j}$ is valid; that is, for all k, $1 \leq k < j$,

$\left|\bigcup_{\ell \le k} A_\ell\right| \le k$ and $|A_k| \le c$ to satisfy Equations 12 and 13. The proof will show that under this assumption, Figure 8 generates each A_j such that the invariant holds for $A_{<j+1}$.

Base case: Observe that for $j = 1$ the invariant holds trivially for the empty dagcode $A_{<1} = \langle \_, \ldots, \_ \rangle$, since $|\emptyset| \le 0$.

Inductive hypothesis: Assume the invariant holds for $A_{<j}$, $1 \le j < n$. Let $U = \bigcup_{k<j} A_k$ and $q = |U|$. Observe that the invariant will also hold for $A_{<j+1}$ so long as one chooses $A_j \subset V$ such that $|U \cup A_j| \le j$ and $|A_j| \le c$. One can select each element of A_j either from U or $\overline{U}$. Let $A_j = S \cup T$, where $S \subseteq U$ and $T \subseteq \overline{U}$. Let $s = |S|$ and $t = |T|$; hence, $0 \le s \le q$ and $0 \le t \le n - q$. Observe that $|U \cup A_j| \le j \iff q + t \le j$, and $|A_j| < c \iff s + t \le c$. Line 5 of All-DAGs iterates over all (s, t) that satisfy these conditions. Lines 6–7 of All-DAGs, then, iterate over all $A_j = S \cup T$ such that the invariant holds for $A_{<j+1}$. Thus, each A_j is generated such that $A_{<j+1}$ is valid. Furthermore, since no pair (s, t) is ever repeated in the outer loop and $S \cap T \equiv \emptyset$, no subset $A_j = S \cup T$ is ever repeated.

Termination: Since j increments with each descent, recursion bottoms out at $j = n$, and a DAG corresponding to fully specified dagcode $A = A_{<n}$ is output. After all valid combinations $\langle A_1, \ldots, A_{n-1} \rangle$ are output, All-DAGs terminates. ❑

From All-DAGs it is possible to derive a new recurrence for the number of DAGs that is more easily extended to CP-nets than those of Robinson (1973) and Steinsky (2003). Let us denote by $a_{n,c}$ the *number of DAGs* (respectively dagcodes) with n nodes and bound c on in-degree, and by $a_{n,c}(j, q)$ the number of extensions to a partial dagcode $A_{<j}$, where $q \equiv \left|\bigcup_{k<j} A_k\right|$. That is, $a_{n,c}(j, q)$ is the number of ways to choose the remaining elements $A_j, \ldots, A_{n-1}$ such that the cardinality and in-degree constraints in Equations 12 and 13 are satisfied.

Theorem 18 (Number of DAGs)**.** *For all non-negative integers n and c,*

$$a_{n,c} = a_{n,c}(1, 0). \tag{14}$$

For all j, $0 < j < n$*,*

$$a_{n,c}(j, q) = \sum_{\substack{s \ge 0,\ t \ge 0,\\ s \le q,\ s+t \le c,\\ q+t \le j}} \binom{q}{s} \binom{n-q}{t} a_{n,c}(j+1, q+t). \tag{15}$$

For $j = n$*,*

$$a_{n,c}(j, q) = 1. \tag{16}$$

Proof. (Strong induction.) In the proof of Theorem 17, the proof employed a form of strong induction on j increasing. To show that Equation 15 is correct, let us again use strong induction, this time on j decreasing.

Base case ($j = n$): One DAG is generated at Line 3 of All-DAGs; hence, $a_{n,c}(n, q) = 1$ for all q, as claimed in Equation 16.

Inductive hypothesis: Assume $a_{n,c}(j', q')$ gives the correct count for $j' > j$ and all q'. The proof will show that the resulting count for $a_{n,c}(j, q)$ is also correct. Observe that, whatever the size of set $U \subset V$, the loop at Line 6 of All-DAGs iterates over the $\binom{q}{s}$ ways to choose s elements from U. Similarly, the loop at Line 7 of All-DAGs iterates over the $\binom{n-q}{t}$ ways to choose t elements from

Table 1: Number of DAGs $a_{n,c}$ with n Nodes and Bound c on In-Degree

n	c=0	1	2	3	4	5	6
1	1						
2	1	3					
3	1	16	25				
4	1	125	443	543			
5	1	1,296	13,956	26,566	29,281		
6	1	16,807	695,902	2,556,342	3,605,817	3,781,503	
7	1	262,144	50,741,797	435,055,552	922,125,667	1,112,308,744	1,138,779,265

$\overline{U}$. Note that the number of DAGs generated in the body of the outermost loop depends on s and t, which differ on each iteration. Thus, for all (s, t) as defined in Line 5 of All-DAGs, one can take the sum of the DAGs generated in the loop body, obtaining the result given in Equation 15.

Finally, observe that all dagcodes parameterized by n, c extend the fully *un*specified dagcode $A_{<1} = \langle \_, \ldots, \_ \rangle$, for which $j = 1$ and $q = 0$. Thus, $a_{n,c} = a_{n,c}(1, 0)$, the result given in Equation 14. ❑

One can verify that for DAGs with unbounded in-degree ($c \geq n - 1$), the recurrence yields the sequence 1, 1, 3, 25, 543, 29281, 3781503, 1138779265, …, as expected (Sequence A003024 in Sloane, 2016). Table 1 gives values of $a_{n,c}$ from Equation 14 for $n = 1$ to 7 and $c < n$.

3.4 Generating CP-Nets

The insights of Section 3.2 can be used to extend All-DAGs (Figure 8) to obtain a new algorithm that generates All-CP-nets, presented in Figure 9. CP-nets with the same dependency graph differ if any rule of a CPT differs. To generate all combinations of CPTs, one needs only introduce a new innermost loop iterating over the possibilities, as described at the end of Section 3.2. Note that since the dagcode is partial, there is not yet enough information to construct the CPT: the parents are known, but the label of the child to which they belong and its domain values are not. However, sufficient information is available to iterate over the corresponding function vectors F_j, since the number of parents ($|A_j| = s + t$) and the size (d) of every domain is known, so we do that instead.

Each F_j is included in a tuple $F = \langle F_0, \ldots, F_{n-1} \rangle$ that we call a *cpt-code*. (The expressions $F_{<j}$ and $F_{\leq j}$, analogous to $A_{<j}$ and $A_{\leq j}$, are used here for a *partial cpt-code*.) Since a root node is implicit in the dagcode, F contains an additional element F_0 corresponding to that node's CPT,[4] and All-CP-nets is invoked with $j = 0$ instead of 1:

$$\text{All-CP-nets}(n, c, d, 0, 0, \emptyset, \langle \_, \ldots, \_ \rangle, \langle \_, \ldots, \_ \rangle). \tag{17}$$

When $j = n$, the encoding is complete: A and F fully and uniquely characterize a CP-net N. Build-CP-net is then called, as shown in Figure 10 (analogous to Dagcode-to-DAG in Figure 7) to decode it—the DAG from A, the CPTs from F.

Theorems 17 and 18 can similarly be extended to CP-nets.

Theorem 19. All-CP-nets *generates each CP-net exactly once.*

4. Note that the algorithm also creates an additional element $A_0 \equiv \emptyset$ for the dagcode.

ALL-CP-NETS($n, c, \boxed{d}, j, q, U, A_{<j}, \boxed{F_{<j}}$)

Inputs: n number of nodes
c bound on in-degree
$\boxed{d}$ size of domains
j is the index of current elements A_j, $\boxed{F_j}$
$q = |U|$, where $U = A_1 \cup \cdots \cup A_{j-1}$
$A = \langle A_1, \ldots, A_{n-1} \rangle$ partial dagcode
$\boxed{F = \langle F_0, \ldots, F_{n-1} \rangle}$ partial CPT code

```
1:  if j = n then
2:     BUILD-CP-NET( A_<n, [F_<n] )
3:     return
4:  end if
5:  for all s, t ≥ 0, s ≤ q, s + t ≤ c, q + t ≤ j do
6:     for all S ⊆ U, |S| = s do
7:        for all T ⊆ V \ U, |T| = t do
8:           if j > 0 then
9:              A_j ← S ∪ T
10:             include A_j with A_<j to form A_≤j
11:          end if
12:          for all vectors F_j of length d^|A_j| with elements in the range (0 .. d! − 1) do
13:             if not IS-FUNCTION-DEGENERATE(F_j) then
14:                ALL-CP-NETS(n, c, d, j + 1, q + t, U ∪ A_j, A_≤j, F_≤j)
15:             end if
16:          end for
17:       end for
18:    end for
19: end for
```

Note: The boxes highlight the differences from the algorithm in Figure 8.

Figure 9: Algorithm: Generate All CP-Nets that Extend $A_{<j}$

Proof. Observe that ALL-CP-NETS (Figure 9) is identical to ALL-DAGs (Figure 8) insofar as the graph is concerned. In the proof of Theorem 17, it has already been shown that each DAG is generated just once and that the algorithm terminates. Thus ALL-CP-NETS generates CP-nets for every possible dependency graph with n nodes and bound c on in-degree.

The principal difference from ALL-CP-NETS is the inclusion of a new innermost loop at Line 12 iterating over all possible function vectors F_j, such that F_j is non-degenerate. Note that these correspond to all possible CPTs for the current node via the mapping described in Section 3.2. Further note that each possible CPT for the root node is also generated in the innermost loop, since the algorithm is called with $j = 0$. Thus, if $A_{<j}$ and $F_{<j}$ are valid, $A_{\leq j}$ and $F_{\leq j}$ will also be valid, and each A_j and F_j will be generated exactly once, for all j, such that $0 \leq j < n$. Therefore, every

BUILD-CP-NET(A, $\boxed{F}$)

Input: $A = \langle A_1, \ldots, A_{n-1} \rangle$ dagcode defining graph
$\boxed{F = \langle F_0, \ldots, F_{n-1} \rangle \text{ cpt-code defining CPTs}}$

Output: the corresponding $\boxed{\text{CP-net } N}$

1: $n \leftarrow \text{length}(A) + 1$
2: $Q \leftarrow \{1, \ldots, n\}$
3: initialize $\boxed{\text{CP-net } N}$ with n nodes, no edges, $\boxed{\text{empty CPTs}}$
4: **for** $j \leftarrow n - 1$ **downto** 1 **do** ▷ Iterate over dagcode: A_j is the parent set of
5: $\quad i \leftarrow \max\left(Q \setminus \bigcup_{k=1}^{j} A_k\right)$ ▷ X_i, where i is the largest unused label
6: $\quad$ **for all** $h \in A_j$ **do**
7: $\quad\quad$ insert edge to X_i from its parent X_h
8: $\quad$ **end for**
9: $\quad \boxed{\text{construct CPT}(X_i) \text{ from } A_j, F_j}$
10: $\quad Q \leftarrow Q \setminus \{i\}$
11: **end for**
12: $i \leftarrow$ the only remaining element in Q
13: $\boxed{\text{construct CPT}(X_i) \text{ from } F_0}$
14: output $\boxed{\text{CP-net } N}$

Note: The boxes highlight the differences from the algorithm in Figure 7.

Figure 10: Algorithm: Construct CP-Net from Its Encoding

non-degenerate CP-net with n nodes, bound c on in-degree, and d-ary domains will be generated exactly once. ❑

Let us denote by $a_{n,c,d}$ the *number of CP-nets* with n nodes, bound c on in-degree, and d-ary domains; and by $a_{n,c,d}(j, q)$, where $q \equiv \left|\bigcup_{k<j} A_k\right|$, the number of those that extend $A_{<j}$.

Theorem 20 (Number of CP-Nets)**.** *For all non-negative integers n, c, and d,*

$$a_{n,c,d} = a_{n,c,d}(0, 0). \tag{18}$$

For all j, $0 \le j < n$*,*

$$a_{n,c,d}(j, q) = \sum_{\substack{s \ge 0,\ t \ge 0,\\ s \le q,\ s+t \le c,\\ q+t \le j}} \binom{q}{s} \binom{n-q}{t} \psi_d(s+t)\, a_{n,c,d}(j+1, q+t). \tag{19}$$

For $j = n$*,*

$$a_{n,c,d}(j, q) = 1. \tag{20}$$

Table 2: Number of Binary CP-Nets with Complete CPTs and Unbounded In-Degree

Nodes	Number of CP-nets
1	2
2	12
3	488
4	481776
5	157549032992
6	4059976627283664056256
7	5242534484601779604747295174905035666696576

Table 3: Odds of Generating a Degenerate Function at Random on a Given Attempt

$\frac{\chi_d(m)}{\phi_d(m)}$	$m=0$	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
$d=2$	0	0.500	0.375	0.148	0.014	7.6×10^{-5}
$d=3$	0	0.028	4.2×10^{-5}	3.0×10^{-14}	3.8×10^{-42}	4.3×10^{-126}
$d=4$	0	7.2×10^{-5}	5.5×10^{-17}	1.7×10^{-66}	4.0×10^{-265}	5.0×10^{-1060}
$d=5$	0	4.8×10^{-9}	5.2×10^{-42}	3.6×10^{-208}	1.0×10^{-1039}	5.6×10^{-5198}

Note that the loop at Line 12 executes $\psi_d(s+t)$ times, since the in-degree of the node modeled by A_j is $s+t$. Otherwise, the proof is nearly identical to that of Theorem 18.

Table 2 shows the number of binary CP-nets with unbounded in-degree ($c \geq n-1$) up to 7 nodes (cf. sequence A250110 in Sloane, 2016). From the values, it is evident that generating all CP-nets is feasible only for tiny n, c, and d. To generate larger *random* instances, we propose an efficient method that relies on Equation 19. Algorithm Random-CP-net, as shown in Figure 11, generates a dagcode one A_j at a time, such that all *CP-nets* (as opposed to DAGs) are equally likely. To satisfy the cardinality constraint, the algorithm keeps track of node labels $U = \bigcup_{k<j} A_k$ that already occur in $A_{<j}$, choosing s labels for A_j from U and the other t from $\overline{U}$, subject to constraints on cardinality and in-degree. It also chooses a non-degenerate function F_j for the CPT (see Section 3.2). To avoid bias, (s,t) is chosen such that all extensions to $A_{<j}$ are equally likely, using a table precomputed by Compute-Distribution (Figure 12). Build-CP-net (Figure 10) outputs the result.

Theorem 21. *For all non-negative integers n, c, and d,* Random-CP-Net(n,c,d) *generates each CP-net N with uniform probability* $P(N) = 1/a_{n,c,d}$.

Proof. Line 1 randomly selects one of the $\psi_d(0) = d!$ possibilities for the CPT of the root node implicit in A; thus, $P(F_0) = 1/d!$. Each A_j, F_j, $0 < j < n$, is then generated, conditioned on $U_j = \bigcup_{k<j} A_k$ and $q_j = |U_j|$. Line 5 chooses integers s and t with probability

$$\binom{q_j}{s}\binom{n-q_j}{t}\psi_d(s+t)\,\frac{a_{n,c,d}\left(j+1,q_j+t\right)}{a_{n,c,d}\left(j,q_j\right)}. \tag{21}$$

RANDOM-CP-NET(n, c, d)

Input: n number of nodes
c bound on in-degree
d size of the domains
Output: CP-net N generated uniformly at random

1: $F_0 \leftarrow$ random constant function with $d!$ outputs ▷ For CPT of a root node
2: $U \leftarrow \emptyset$
3: $q \leftarrow 0$
4: **for** $j \leftarrow 1$ to $n-1$ **do** ▷ Iterate over dagcode
5: $s, t \leftarrow$ values in cols. 1–2 of a row of $\texttt{DIST}_{n,c,d}(j, q)$ selected randomly according to the weights in col. 3 ▷ Weighted selection
6: $S \leftarrow$ subset of size s selected randomly from U
7: $T \leftarrow$ subset of size t selected randomly from $\overline{U}$
8: $A_j \leftarrow S \cup T$
9: $U \leftarrow U \cup T$
10: $q \leftarrow q + t$
11: **repeat**
12: $F_j \leftarrow$ random function with $|A_j|$ inputs, $d!$ outputs ▷ CPT for current node
13: **until** F_j is non-degenerate ▷ Note the odds in Table 3
14: **end for**
15: BUILD-CP-NET(A, F) ▷ See Figure 10

Figure 11: Algorithm: Generate a CP-Net Uniformly at Random

Then, given s, t, and U, Lines 6–13 choose S, T, and F_j with probability

$$\frac{1}{\binom{q_j}{s}} \frac{1}{\binom{n-q_j}{t}} \frac{1}{\psi_d(s+t)}. \tag{22}$$

Multiplying Equations 21 and 22 and simplifying gives us the probability of generating A_j and F_j given U_j in Lines 5–13:

$$P(A_j, F_j | U_j) = \frac{a_{n,c,d}\left(j+1, q_j + t\right)}{a_{n,c,d}\left(j, q_j\right)} = \frac{a_{n,c,d}\left(j+1, q_{j+1}\right)}{a_{n,c,d}\left(j, q_j\right)}, \tag{23}$$

since $q_j + t = q_{j+1}$ for $j = 1$ to $n-1$ (Line 10).

Since A and F uniquely characterize a CP-net, $P(N) = P(A, F)$. Altogether, iterating through all values of j in the **for** loop at Line 4, the probability of generating N is:

$$P(N) = P(F_0)P(A_1F_1|U_1)P(A_2F_2|U_2) \cdots P(A_{n-2}F_{n-2}|U_{n-2})P(A_{n-1}F_{n-1}|U_{n-1}) \tag{24}$$

$$= \frac{1}{d!} \frac{a_{n,c,d}(2, q_2)}{a_{n,c,d}(1, q_1)} \frac{a_{n,c,d}(3, q_3)}{a_{n,c,d}(2, q_2)} \cdots \frac{a_{n,c,d}(n-1, q_{n-1})}{a_{n,c,d}(n-2, q_{n-2})} \frac{a_{n,c,d}(n, q_n)}{a_{n,c,d}(n-1, q_{n-1})}. \tag{25}$$

COMPUTE-DISTRIBUTION(n, c, d)

Input: n number of nodes
c bound on in-degree
d size of the domains
Output: $\texttt{DIST}_{n,c,d}$ values of s, t and weights $P(s, t \mid j, q)$

1: **for** $j \leftarrow n-1$ **downto** 1 **do**
2: **for** $q \leftarrow j$ **downto** 0 **do**
3: $\texttt{DIST}_{n,c,d}(j, q) \leftarrow$ table with 0 rows and 3 columns
4: **for all** $s, t \geq 0,\ s \leq q,\ s+t \leq c,\ q+t \leq j$ **do**
5: $weight \leftarrow \binom{q}{s}\binom{n-q}{t}\psi_d(s+t)\dfrac{a_{n,c,d}(j+1, q+t)}{a_{n,c,d}(j, q)}$
6: append row $[s, t, weight]$ to $\texttt{DIST}_{n,c,d}(j, q)$
7: **end for**
8: sort rows on col. 3; assert that col. 3 sums to 1 (optional)
9: **end for**
10: **end for**
11: **return** $\texttt{DIST}_{n,c,d}$

Figure 12: Algorithm: Compute Tables for Uniform CP-Net Generation

One can use Equation 19 to verify that

$$a_{n,c,d}(0, 0) = d!\, a_{n,c,d}(1, 0). \tag{26}$$

Also, $q_1 = \left|\bigcup_{k<1} A_k\right| = 0$. One can thus rewrite the first term of Equation 25 as

$$P(F_0) = \frac{1}{d!} = \frac{a_{n,c,d}(1, q_1)}{a_{n,c,d}(0, 0)}\,. \tag{27}$$

Further observe that the numerator of the last term is $a_{n,c,d}(n, q_n) = 1$. All terms except the first then cancel out, leaving us with

$$P(N) = \frac{1}{a_{n,c,d}(0, 0)} = \frac{1}{a_{n,c,d}}\,, \tag{28}$$

which proves the case. ❑

Theorem 22. COMPUTE-DISTRIBUTION *(Figure 12) runs in time and space polynomial in the number of nodes n.*

Proof. Observe that the nested loops are bounded by n. One can compute $a_{n,c,d}(j, q)$ with the help of a table. This computation need only be performed once for each j and q, and the ranges of j and q are similarly bounded by n. ❑

Algorithm RANDOM-CP-NET is also efficient. Random subset sampling and proportional (i.e., weighted) sampling can be performed efficiently (Bringmann & Panagiotou, 2012; Knuth, 1997, 3.4.2), and with high probability the inner loop will execute just once, as discussed in Section 3.2.

Table 4: CPU Time (in Seconds) to Generate 100 CP-nets Uniformly at Random ($c = 4$)

	$d = 2$	$d = 3$	$d = 4$
$n = 10$	0.23	0.83	2.45
$n = 20$	0.67	2.23	5.78
$n = 30$	1.61	3.85	9.78
$n = 40$	3.28	6.38	14.98
$n = 50$	6.08	9.89	21.77

Table 4 provides a practical example of how quickly one can generate CP-nets using the method described above.[5] The table shows the average time (over 10 trials), in seconds, required to generate 100 CP-nets with $n = 10$ to 50 nodes, domains of size $d = 2$ to 4, and a bound of $c = 4$ on in-degree, on a MacBook Pro computer with a 2.7 GHz Intel Core i5 processor and 8 GB RAM.

3.5 Generalizing CP-Net Generation to Statistical Cultures

In social choice, it is common to generate sets of plausible preferences, called profiles, in order to reason about aggregating the preferences of many agents. This random generation continues because there is a paucity of real preference data in the formats desired (Mattei & Walsh, 2013). Plausibility in this domain generally means that the preferences are generated according to one or more *statistical cultures*. Recent work in computational social choice uses more complicated preference structures, including CP-nets, as inputs to common voting schemes; see, e.g., the work of Dorn and Krüger (2015), Dorn et al. (2015), Grandi, Luo, Maudet, and Rossi (2014), Li, Vo, and Kowalczyk (2015), Mattei et al. (2013), Cornelio, Grandi, Goldsmith, Mattei, Rossi, and Venable (2015), Xia and Conitzer (2010), Xia, Conitzer, and Lang (2011b), Lang, Mengin, and Xia (2012), and Conitzer, Lang, and Xia (2011). Hence, an interesting direction for future work is to complement these theoretical studies with empirical studies where the preference profiles of CP-nets are drawn from the commonly used statistical cultures used in social choice experiments.

Over the years multiple statistical models have been proposed to generate election pseudo-data to analyze (for examples, see Regenwetter et al., 2006 and Tideman and Plassmann, 2012). For instance, Gehrlein (2002) and Tsetlin, Regenwetter, and Grofman (2003) provide analyses of the probability of occurrence of Condorcet's Paradox in a variety of election cultures. Gehrlein exactly quantifies these probabilities and concludes that Condorcet's Paradox will only occur with very small electorates with any real likelihood. Gehrlein states that some of the statistical cultures used to generate election pseudo-data, specifically the Impartial Culture, may actually represent a worst-case scenario when analyzing voting rules for the likelihood of observing Condorcet's Paradox and analyzing whether or not profiles are single-peaked (Gehrlein, 2002). Tideman and Plassmann provide a more complete discussion of the variety of statistical cultures in the literature (Tideman & Plassmann, 2012) and a good summary of the effects of these cultures and their effect on the presence of voting anomalies (Mattei, 2011, 2012; Mattei et al., 2012).

In voting, three of the main distributions used are the Impartial Culture, the Impartial Anonymous Culture, and the Urn Model. Each of these cultures can be thought of as a distribution over the set of possible preferences which leads to a generative process for adding more preferences to the set of preferences — the profile.

5. The implementation is available at https://github.com/nmattei/GenCPnet.

Impartial Culture (IC): Each possible preference should appear in the final preference profile with equal likelihood; i.e., we draw a preference uniformly at random from the set of possible preferences. For instance, if the set of preferences is expressed as a strict linear order over m elements, then there are $m!$ possible preferences; each preference is added to the profile with probability $1/m!$. We can use the methods developed in the previous sections both to find the distribution over the set of CP-nets and to draw (add to the profile) as many CP-nets as we want.

Impartial Anonymous Culture (IAC): Every distribution over preferences has an equal likelihood of being the distribution that has been used to create the preference profile. For example, if the set of all preferences that can be expressed is the set of all strict linear orders then IAC says that any distribution over this set is equally likely to be the generative process. Observe that in IC there is exactly one distribution that is allowed, i.e., the vector $1/m!$. For IAC, any vector over the $m!$ orders is allowed assuming that it sums to 1.0, i.e., it is a probability distribution. We will explain how to generalize this to CP-nets below.

Urn Model: The Polya Eggenberger urn model is a method designed to introduce some correlation between preferences and does not assume a uniform random distribution (Berg, 1985). We describe a similar setup to the one described by Walsh (2010). Imagine again that the space of all preferences is defined as the set of all strict linear orders over m candidates. Given as input some $\alpha \in \mathbb{Z}^+$, we start with a jar containing one of each possible strict linear order. We draw an order at random and place it back into the jar with α copies of this order. We repeat this procedure until we have created a sufficient number of votes. Each time we draw an order from the jar we add it to the profile that we are building until we have as many preferences as we desire in our profile. We will explain how to generalize this to CP-nets below.

We can leverage our results on uniform random generation from the previous sections to now define a polynomial time procedure to produce preferences over CP-nets using either the Impartial Culture, Impartial Anonymous Culture, or Urn Models. Observe that the Urn Model is a strict generalization of the other two models. If we set $\alpha = 0$ for the Urn Model then we have IC, if we have $\alpha = 1$ for the Urn model then we have IAC (Walsh, 2011). As the other two are special cases of Urn Models, we give a description of how to build Urn Models where α is given as a parameter to the algorithm. Hence, the below methodology can, with the appropriate selection of a, generate votes according to any of these cultures.

Given α, the number of copies of each preference to add when drawn; n, the number of nodes in the CP-nets; c the bound on the in-degree of each node; and $|P|$, the number of preferences desired for our profile, we wish to build a preference profile of CP-nets of size $|P|$. From Theorem 20 we know the size (number of CP-nets) that start off in the "uniform random" bucket without having to enumerate them all; let j be the number of CP-nets given n and c. We generate CP-nets uniformly at random using the Random-CP-net algorithm; we will keep track of each of the CP-nets we draw in order. We build our profile, P, of CP-net preferences iteratively. At each successive iteration $i \in [0, \ldots, |P| - 1]$, we generate a random number k such that $0 \leq k < j + i \cdot \alpha$. If we have $0 \leq k < j$, then we call Random-CP-net and add α copies of this CP-net to $|P|$. If we have $j \leq k < j + i \cdot \alpha$, we choose the $i = \left\lfloor \frac{(k-j)}{\alpha} \right\rfloor$ CP-net in our current P, and add α more copies of that CP-net to P.

4. Flipping Sequence Length

Despite their advantages and conceptual beauty, one of the chief objections to CP-nets is that the problem of *dominance* — deciding whether one outcome is better than another with respect to

the network — is computationally hard: deciding dominance in CP-nets is known to be PSPACE-complete in general (Goldsmith, Lang, Truszczyński, & Wilson, 2008); in certain instances it is hard even to verify a solution (Boutilier et al., 2004). Except for special cases (e.g., tree-structured CP-nets), dominance testing in CP-nets necessitates a search, with possible backtracking, for a path in the exponentially larger induced preference graph of the CP-net.

This section argues that it is reasonable to limit the depth of this search. The experiments described here show that most of the time the *flipping length* is not much longer than the Hamming distance between the outcomes; a solution, if it exists, is likely to be found at relatively shallow depth in the search tree with respect to the number of variables. Using parameters such as Hamming distance (HD) and average path length (APL), both of which are easy to compute, one can estimate *a priori* (via statistical experiments) a depth to which the search tree must be traversed to find a solution with high confidence. One can then adapt existing DT algorithms to limit the depth of searches to this learned depth.

This technique of improving the efficiency of algorithms through learning a set of parameters for which the algorithm is well-behaved is widely used in practical applications and is known as algorithm configuration (Hutter, Hoos, Leyton-Brown, & Stützle, 2009). Using algorithm configuration to specify search strategies based on easily computable properties of input instances has led to many practical refinements to heuristics for known hard problems such as SAT (Xu, Hutter, Hoos, & Leyton-Brown, 2008) and planning (Rintanen & Gretton, 2013).

For completeness we recall a few definitions we need in order to explore the interplay between flipping length and other parameters.

Definition 23 (Diameter)**.** *The* diameter *of a digraph is the length of the longest shortest path between any pair of nodes,*

$$\mathrm{Diam}(G) = \max_{s,t \in V} \left|\mathrm{minpath}_G(s,t)\right|.$$

Definition 24 (APL)**.** *The* average path length *(Costa, Rodrigues, Travieso, & Villas Boas, 2007) of a digraph with n nodes is*

$$\mathrm{APL}(G) = \frac{1}{n(n-1)} \sum_{s \neq t} d(s,t),$$

where $n = |V|$ and each $d(s,t) = |\mathrm{minpath}_G(s,t)|$, the shortest path between the pair of nodes s and t, provided such a path exists; otherwise, $d(s,t) = 0$.

Definition 25 (Hamming Distance)**.** *The* Hamming distance *of a pair of outcomes* $\mathrm{HD}(o,o')$*, $o \in \mathcal{O}$, $o' \in \mathcal{O}$, is the number of variables in the outcomes for which the values differ, i.e.,*

$$\mathrm{HD}(o,o') = |\{\, X_i : o[X_i] \neq o'[X_i] \,\}|.$$

We denote by $\mathrm{DT}(\mathcal{N},\mathcal{O})$ the set of all DT problem instances (N,o,o'), such that $N \in \mathcal{N}$, $o \in \mathcal{O}$, and $o' \in \mathcal{O}$, and by $\mathrm{DT}(\mathcal{N},\mathcal{O} \mid \theta)$ the set of instances satisfying one or more conditions θ. For example, $\mathrm{DT}(\mathcal{N},\mathcal{O} \mid \mathrm{HD}(o,o') = h)$ denotes the set of DT problem instances (N,o,o') in which the Hamming distance between o and o' is h.

To understand the DT problem as fully as possible for small n, we first studied *all* DT instances up to $n = 4$ binary variables — 123,334,656 instances.[6] To accomplish this, we used All-CP-nets (Figure 9) to generate the sets $\mathcal{N}_2$ to $\mathcal{N}_4$ consisting respectively of all binary acyclic CP-nets

6. $\sum_{n=1}^{4} |\mathcal{N}_n||\mathcal{O}_n|^2 = (2)(2)^2 + (12)(4)^2 + (488)(8)^2 + (481776)(16)^2 = 123334656$.

$n = 2$

Flipping Length	Hamming distance $h = 1$	$h = 2$
$\ell = 2$		20
$\ell = 1$	48	

$n = 3$

Flipping Length	Hamming distance $h = 1$	$h = 2$	$h = 3$
$\ell = 4$		144	
$\ell = 3$			1,496
$\ell = 2$		4,656	
$\ell = 1$	5,856		

$n = 4$

Flipping Length	Hamming distance $h = 1$	$h = 2$	$h = 3$	$h = 4$
$\ell = 6$		39,360		
$\ell = 5$			539,328	
$\ell = 4$		891,648		2,856,080
$\ell = 3$			11,184,768	
$\ell = 2$		17,906,304		
$\ell = 1$	15,416,832			

Figure 13: Number of DT Solutions Given Hamming Distance and Flipping Length

with 2 to 4 nodes. We *uncompacted* each CP-net $N \in \mathcal{N}_n$ to obtain its induced preference graph H and applied the Floyd–Warshall all-pairs-shortest-path algorithm (Floyd, 1962; Warshall, 1962; Cormen, Stein, Rivest, & Leiserson, 2001) to H to determine the flipping length $\mathrm{FL}(N, o', o)$ for all pairs of outcomes $\mathcal{O}_n^2$. We then aggregated solutions according to the Hamming distance between outcomes and other prospective parameters.[7]

Figure 13 summarizes the results for DT problem instances that entail dominance. The rows of each table correspond to the flipping length ℓ, and the columns correspond to the Hamming distance h. Each entry at position (ℓ, h) corresponds to the number of DT problems with that flipping length and Hamming distance, i.e.,

$$\left|\mathrm{DT}(\mathcal{N}_n, \mathcal{O}_n \mid \mathrm{HD}(o, o') = h \wedge \mathrm{FL}(N, o, o') = \ell)\right|,$$

$o \in \mathcal{O}_n$, $o' \in \mathcal{O}_n$, $N \in \mathcal{N}_n$, $\ell \in \mathbb{Z}^+$. The entries that are left blank correspond to a count of 0. Since no DT instance had a flipping length ℓ greater than 2, 4, and 6 for $n = 2$, 3, and 4, respectively,

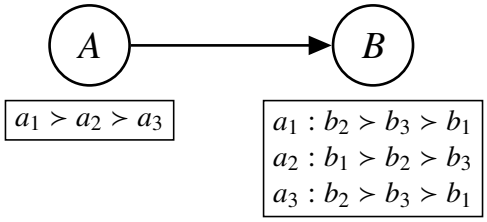
7. We performed these experiments using MATLAB and the MatlabBGL Boost Graphics Library package running under Windows 10 on a Dell Vostro 470 computer with an Intel i5-3450 processor and 8GB RAM. Generating all DT problem instances up to four binary nodes took us about 90 minutes.

Hamming distance h	1	2	3	4
Mean Flipping Length $\hat{\ell}$	1.0	2.1	3.1	4.0

Figure 14: Mean Flipping Length $\hat{\ell} = \text{MFL}(\mathcal{N}_4, \mathcal{O}_4 \mid \text{HD}(o, o') = h)$

the blank rows for higher values of ℓ are not shown. Also, note that DT instances that did not entail dominance are excluded from the table, since in that case the flipping length ∞ is undefined.

These data provide important insights into the space of DT problems that we can generalize to CP-nets for all n. First, notice that if the Hamming distance is 1, then the flipping length, if defined, must also be 1. Also observe that the flipping length is always *at least* the Hamming distance. Further observe that if the Hamming distance is even (respectively odd), the flipping length, if a path exists from o' to o in H exists, will also be even (respectively odd). We define the *parity* of a pair of outcomes o and o' as the parity of their Hamming distance. One can verify that for every binary-valued DT instance that entails dominance, the parity of the flipping length is the parity of the outcomes. We will see later (Figure 19) that for CP-nets with binary features, this simple observation can halve the number of iterations in the iterative approach to dominance testing that we present in Section 5. However, this result does not generalize to multivalued variables, since for $d \geq 3$, a flip does not necessarily change the Hamming distance from o' to the current outcome. The proof is a simple counter-example. Consider the three-valued CP-net below.



Observe that the shortest flipping sequence from a_3b_3 to a_1b_1 is $a_3b_3 \succ a_2b_3 \succ a_2b_1 \succ a_1b_1$. Thus, while the (a_3b_3, a_1b_1) has even parity (the values of both variables differ, hence Hamming distance is 2), the flipping length is odd, and no shorter path is available since one cannot flip B from b_3 to b_1 without first flipping A to a_2.

From the values in each column h of the tables in Figure 13, one can compute the mean flipping length $\hat{\ell}$ given Hamming distance h, as shown in Figure 14 for the case of $n = 4$. From the data one can observe that the mean flipping length of the set $\text{DT}(\mathcal{N}_n, \mathcal{O}_n \mid \text{HD}(o, o') = h \wedge o \succ o')$ is close to the Hamming distance. Furthermore, by calculating the ratio of each count to the total counts in each column and computing the cumulative sum on ℓ ascending, we obtain the empirical distribution shown in Figure 15. Since the data for $n \leq 4$ reflect *all* DT instances, for these we have the cumulative density function (c.d.f.) of the flipping length ℓ itself, conditioned on the value of h.

This suggests a useful notion in searching for a flipping sequence. Consider a setting in which we are performing DT on an instance (N, o, o') and have already searched the induced preference graph to a depth of k. With the help of a table such as the one in Figure 15, we could then compute the probability that a path from o' to o of length $\ell \geq k + 2$ exists. We could then either continue the search to depth $k + 2$ or, if the cost of computational resources were too high, halt the search and report that it was *very likely* the case, but not guaranteed, that $o \not\succ o'$.

ℓ	$h = 1$	$h = 2$	$h = 3$	$h = 4$
6	1.000	1.000	1.000	1.000
5	1.000	0.998	1.000	1.000
4	1.000	0.998	0.954	1.000
3	1.000	0.951	0.954	
2	1.000	0.951		
1	1.000			

Figure 15: Cumulative Density Function (c.d.f.) Resulting from Figure 13

In addition to Hamming distance, observe that the *average path length* of the dependency graph APL(G) also serves as an important parameter for estimating flipping length. In graphs for which the value of APL(G) is relatively low (such as DAGs of unbounded in-degree), the values of flipping length also tend to be lower (closer to h). Conversely, in graphs for which the value of APL(G) is relatively high, one can observe that flipping sequences tend to be longer. The three-dimensional bar chart in Figure 16 illustrates the mean flipping length $\hat{\ell}$ given average path length $L = \text{APL}(G)$ of the dependency graph and the Hamming distance $h = \text{HD}(o, o')$ of the pair of outcomes.

Note that the values of h and L are not distributed uniformly across problem instances; Hamming distances h, for example, are distributed binomially. Figures 17a and 17b illustrate the distribution of h with respect to $\mathcal{O}_4^2$, and the distribution for APL(G) with respect to $\mathcal{N}_4$.

Finally, let us consider the *maximum* value that $\text{FL}(N, o, o')$ can take. In the analysis of $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_3$, and $\mathcal{N}_4$ (see Figure 13 for $\mathcal{N}_4$), the longest flipping lengths for any DT instance were 1, 2, 4, and 6, respectively. In each case, a *chain* CP-net induced a sequence of that length.[8] One can also observe that chains have the maximum *average path length* of any graph (Gulyás, Horváth, Cséri, & Kampis, 2011). Thus, because of their relatively long flipping lengths, chain CP-nets are of interest to us despite DT being solvable in polynomial time on all trees (Boutilier et al., 2004).

We generated binary chain CP-nets with $n = 1$ to 12 nodes, uncompacting each[9] into its corresponding preference graph and applying the Floyd–Warshall algorithm to compute the diameter of H. For $n = 1$ to 12, we found that binary chain CP-nets had maximum flipping lengths of 1, 2, 4, 6, 9, 12, 16, 20, 25, 30, 36, and 42, consistent with the formula $\lfloor (n+1)^2/4 \rfloor$, as described in the work of Sloane (2016), sequence A002620. In all of our experiments involving complete binary[10] acyclic CP-nets, including those described in the sections that follow, we have not yet encountered a flipping length that exceeds these values for a given n. Hence, the following can be stated as an interesting open problem.

Conjecture 26. *Let (N, o, o') be an arbitrary DT instance, where N is a complete CP-net on n binary variables. Then, for all n, N, o, o', the longest flipping length is*

$$\max(\text{FL}(N, o, o')) = \left\lfloor \frac{1}{4}(n+1)^2 \right\rfloor. \tag{29}$$

8. Note that in some cases other graphs had maximum flipping lengths as long as those of chain CP-nets.
9. Because every chain CP-net on n binary nodes with complete CPTs is the same up to symmetry, it is only necessary to generate one instance for each integer n.
10. The bound does not hold for multivalued domains in general.

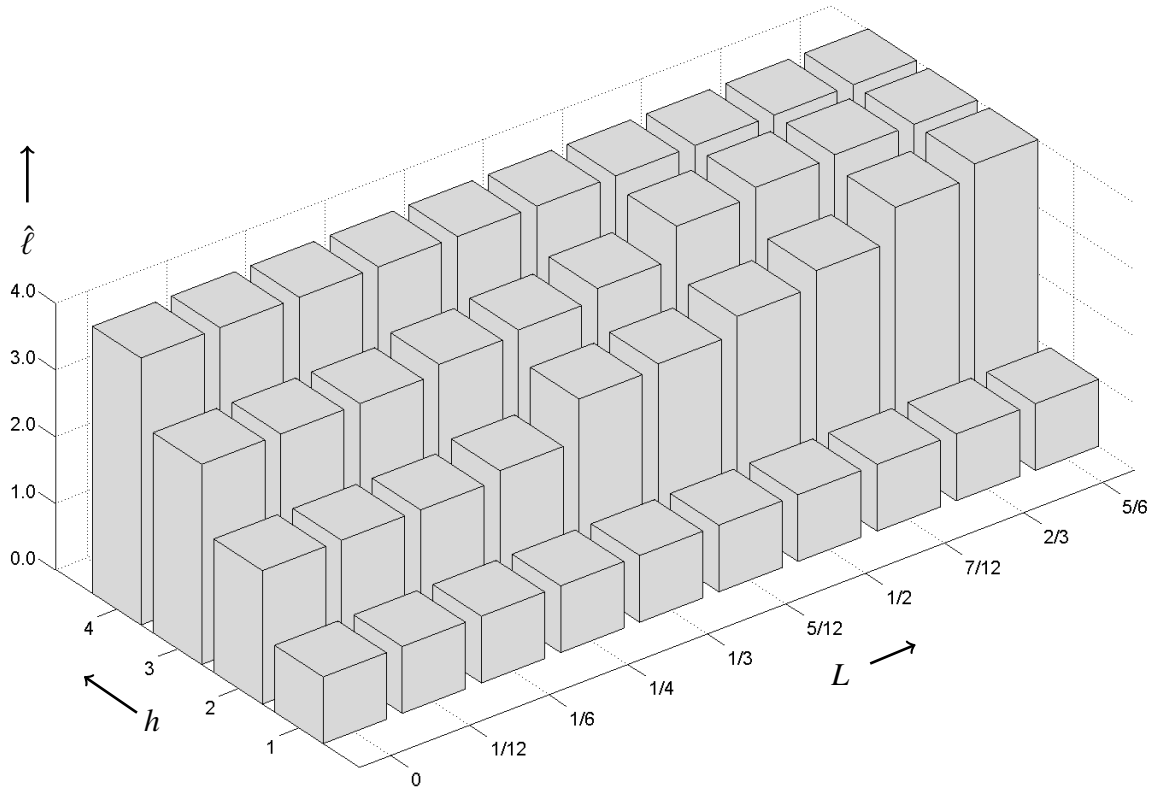
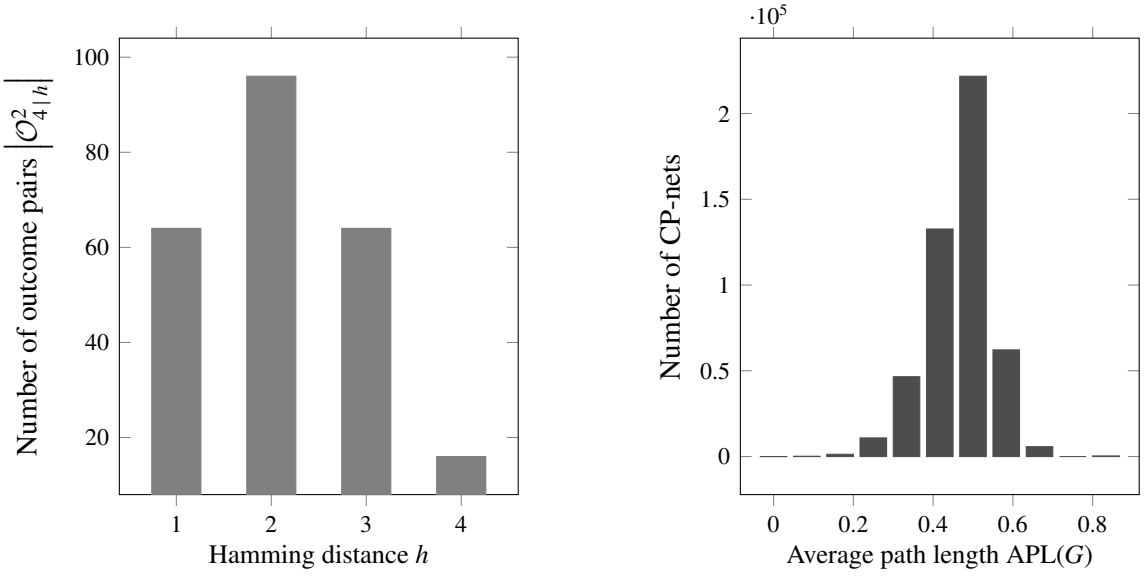


Figure 16: Mean Flipping Length $\hat{\ell}$ as a Function of HD h and APL L



(a) Distribution of HD

(b) Distribution of APL

Figure 17: Distribution of Parameter Values over DT Problem Instances ($n = 4$)

Note that this is consistent with the $\Omega(n^2)$ asymptotic *lower* bound shown by Boutilier et al. (2004). If the conjecture holds then we would have $\Theta(n^2)$ as a tight upper and lower bound. An immediate result would be that dominance testing in such CP-nets would be NP-complete.

The exhaustive analysis of tiny cases suggests that a flipping sequence, if it exists, is probably not much longer than the Hamming distance between the two outcomes. Does this result extend to larger values of n and d? Exhaustive analysis is infeasible for $n > 4$, since $|\mathcal{N}_5| = a_{5,4,3} = 157549032992$, and for multivalued domains $d > 2$ even for 3 nodes, since $|\mathcal{N}_{3,3}| = a_{3,2,3} = 77274933336$. Thus, we use our generation algorithm to produce an unbiased random sample of CP-nets.

For our next experiment, we generated 100 binary-valued CP-nets for n from 5 to 9, obtained the corresponding preference graph of each, and applied the Floyd–Warshall algorithm to compute the maximin path length.[11] Because we assume that in-degree is bounded by a small constant, we varied this bound from $c = 1$ to $c = 4$, such that $c < n$. Our CP-net generation method does not let us specify a bound on APL; however, it can be shown that APL decreases as the bound c on in-degree increases. Thus, by varying c, one can indirectly observe the effect of APL L. Once again, the flipping lengths were relatively close to the Hamming distance. As c increased and thus the APL decreased, the flipping lengths tended to become closer to the Hamming distance.

For larger n, it is no longer practical to work directly with the preference graph or to use the Floyd–Warshall algorithm. For insight into the flipping lengths for larger cases, one thus has to sample from the outcome space as well as the space of CP-nets. In our last experiment, we generated 1000 binary-valued DT problem instances with n ranging from 10 to 15, bound c on in-degree ranging from 1 to 4, and Hamming distances of 2, $\lceil n/2 \rceil$, and n, a total of 72000 DT problems. In previous experiments, flipping lengths depended significantly on Hamming distance. Furthermore, values of h are not distributed evenly across the space of outcome pairs $\mathcal{O}^2_{n,d}$. Instead, the peak of the binomial distribution seen in Figure 17a for $n = 4$ nodes becomes much sharper as n increases, as seen in Figure 18 for $n = 15$. Thus, we choose values of h at both extremes of the distribution as well as in the middle.

Since the Floyd–Warshall algorithm is impractical for problems of this size, we used the iterative deepening Depth-Limited-DT* algorithm to obtain the flipping lengths. Depth-Limited-DT*, adapted from the DT* algorithm of Li et al. (2011), is discussed in the next section. Note that for this experiment the depth limit k is set to ∞ and search continues indefinitely until a solution is found or the algorithm reports **false** indicating that no flipping sequence exists at any depth. We performed a similar experiment for multivalued domains of size $d = 3$, with n ranging from 5 to 10 nodes.[12] We found similar results in all experiments: the average flipping sequence length was only slightly larger than the Hamming distance for that pair. The flipping sequence length went up slightly and not monotonically with the number of preference variables, while the flipping sequence length went down, albeit also not always monotonically, as the in-degree increased and the dependency graph's average path length decreased (for complete tables of results see Allen, 2016).

11. While it took only a fraction of a second to generate the CP-nets (see Table 4), obtaining the preference graph and running the Floyd–Warshall algorithm required considerably more time as n increased. It took only half a second to do this for the set of 100 CP-nets with 5 nodes, but those with 9 nodes required half an hour, and 10 nodes turned out to be impractical on our system. For this experiment we again used MATLAB and MatlabBGL on a Windows 10 Dell Vostro 470 with an Intel i5-3450 processor and 8GB RAM.

12. We reimplemented the iterative DLDT* in C++, allowing us to run the experiment for the 72000 binary problems in 50 minutes on a MacBook Pro computer with a 2.7 GHz Intel Core i5 processor and 8 GB RAM, with the multivalued problems requiring about 100 minutes.

5. Depth-Limited Dominance Testing

To this point, we have assumed that preferences are transitive. That is, if a subject prefers $o \succ o'$ and $o' \succ o''$, then we reason that the subject also prefers $o \succ o''$. This assumption is the basis for constructing flipping sequences via the *ceteris paribus* rules of the CP-net. We have seen that flipping sequences much longer than the Hamming distance are rare. However, sequences of length $O(n^2)$ occur in binary CP-nets with complete tables, and in certain cases flipping lengths can be exponential in n (Boutilier et al., 2004).

For arbitrary, possibly cyclic CP-nets, the dominance problem (DT) is known to be PSPACE-complete (Goldsmith et al., 2008), and in certain cases (in particular, chain CP-nets) flipping lengths can be $\Omega(2^{n/2})$, i.e., exponential in the number of nodes n, provided tables are incomplete and domains are multivalued (Boutilier et al., 2004). However, Boutilier et al. (2004) showed that, in the most general case, dominance testing can be formulated as a STRIPS-type planning problem. More recently, Kronegger et al. (2014) have established several fixed parameter tractability (FPT) results for dominance testing in a generalized class of CP-nets (GCP-nets) (similar to those studied by Goldsmith et al., 2008). Many of their FPT results also apply to CP-nets.

Several tractable subclasses for DT are known. Boutilier et al. (2004) showed that DT can be conducted in $\Theta(n^2)$ time for binary-valued tree CP-nets with their TreeDT algorithm, which also returns a flipping sequence if one exists. Bigot et al. (2013) subsequently described an algorithm they claim can answer dominance in $O(n)$ time for the same class of CP-nets (except that CPTs must also be complete), an unexpected result, since the flipping length is $O(n^2)$ for such CP-nets. Thus, while the decision problem can be answered in linear time, computing the flipping sequence itself requires quadratic time.

In general, DT involves a search for a flipping sequence that connects the two outcomes. Any of the familiar search methods in AI, e.g., iteratively deepening depth-first search, can be employed. Boutilier et al. (2004) introduced two methods of pruning the search tree, *suffix fixing* and *forward pruning*, that work in all cases, as well as a heuristic method, *least-variable flipping*, that is incomplete except for binary-valued tree-structured CP-nets. In addition to the reduction to STRIPS-type planning (Boutilier et al., 2004), DT problems can also be reduced to model checking (Santhanam

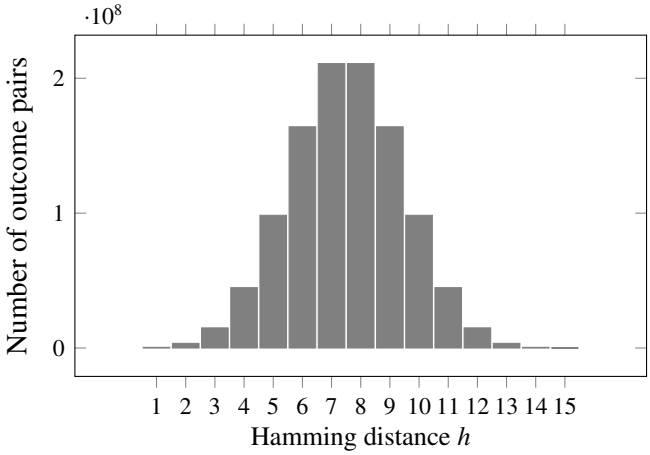


Figure 18: Distribution of HD for $n = 15$

Table 5: Noise Model for Maximum Reliable Flipping Lengths

Probability of Noise ϵ	Max Flipping Length ℓ
0.10%	692
0.50%	138
1.00%	68
5.00%	13

et al., 2010; Santhanam, Basu, & Honavar, 2015). Finally, Li et al. (2011) have proposed a heuristic approach to DT in acyclic, multivalued CP-nets that they call DT*. While the algorithm is inspired by A*, it does not guarantee optimality; i.e., it does not always return a shortest flipping sequence.

Recall that each flip in the sequence is entailed by a particular CPR. CP-nets are considered to be a static, deterministic formalism. That is, it is assumed that for each outcome pair that differs in just one feature, the subject always makes the same choice. One should keep in mind, however, that this determinism is a *modeling* decision, not an intrinsic property of the subject's underlying preferences. Whether N is constructed by the subject, elicited through queries, or learned from data, it is reasonable to allow for the possibility that the model reflects some margin of error, or *noise*. If such errors are small, they can be safely ignored. However, the presence of noise turns out to be problematic for long flipping sequences,

Consider that in a flipping sequence, each flip (o_t, o_{t+1}) is permitted by a particular CPR $u : \succ^i$, where $o_t[X_i] \neq o_{t+1}[X_i]$, $o_t[\overline{X_i}] = o_{t+1}[\overline{X_i}]$, $u = o[\mathrm{Pa}(X_i)]$, $1 \leq i \leq n$, $0 \leq t < \ell$. Let $\epsilon \in (0, 1)$ be the probability that each CPR in the CP-net is noisy. The probability p that a particular flipping sequence of length ℓ entails dominance despite noise is then

$$p = (1 - \epsilon)^{\ell}, \tag{30}$$

which converges to 0 as ℓ tends to infinity.[13] Assuming $p \geq 0.5$ and solving for ℓ gives us

$$\ell \leq \left\lfloor \frac{-1}{\log_2(1 - \epsilon)} \right\rfloor. \tag{31}$$

Table 5 shows the longest reliable values of ℓ for varying noise levels ϵ.

We conjecture that a suitable value for k will turn out to be domain specific and as such is a model parameter that is best learned from data. Finally, we note that the analysis here is closely related to the concept of *Noisy AND* (Díez & Druzdzel, 2007) and also to PCP-nets (Cornelio, 2012; Cornelio et al., 2013; Bigot et al., 2013).

The problem of noise is of course not limited to CP-nets. *Any* predictive model, given data that is too noisy, becomes unreliable. The tendency of very long transitive sequences to accumulate noise, however, is a further reason to limit search depth. As the probability of a flipping sequence corresponding to the "true" preference decreases geometrically with the length of the sequence, it thus seems reasonable in many settings to limit search depth, as we now discuss.

Definition 27 (Depth-limited dominance)**.** *Let (N, o, o') be an arbitrary DT instance. For any positive integer k, N entails the* depth-k dominance *of o over o',*

$$N \models o \succ^k o' \tag{32}$$

13. Here we consider only a single flipping sequence from o' to o. Multiple paths from o' to o of length ℓ are possible, but this complicates the analysis.

if and only if $\mathrm{FL}(N, o', o) \leq k$. *If a flipping sequence of length* $\ell \leq k$ *exists, then we also say the first outcome* k-dominates *the second, and write* $o \succ^k o'$.

We call the search for such a sequence *depth-limited dominance testing* (DLDT). A general algorithm for conducting DLDT is described in Figure 19. The algorithm takes as its input a DT instance, specified depth limit k, and a reference to a blackbox subroutine *DTSolver* that returns **true** if a path of length $\ell \leq k$ exists and *may* return **unknown** if the cutoff is reached before the search tree is fully explored. The algorithm returns the length $\ell \leq k$ of a flipping sequence from o' to o, if such a sequence exists, or ∞ if $o \not\succ^k o'$.

We propose the use of two algorithms that can serve as the *DTSolver* subroutine. The first is an adaptation of the DT* algorithm proposed by Li et al. (2011). The second is SAT-DT, a novel approach that solves DT instances of specified flipping length via reduction to Boolean satisfiability (SAT). The algorithms can easily be adapted to return the flipping sequence as well, if desired.

DLDT(N, o, o', k, *DTSolver*)

Input:	N	CP-net
	o	Goal outcome
	o'	Start outcome
	k	Depth limit
	DTSolver	Subroutine to solve depth-limited DT instances
Output:	ℓ	Flipping length if sequence found, otherwise ∞

```
1:  s ← 2 if N is binary-valued and 1 otherwise
2:  for ℓ ← HD(o, o′) to k step s do                ▷ Iterative search depth
3:      answer ← DTSolver(N, o, o′, ℓ)              ▷ Invoke subroutine
4:      if answer = true then
5:          return ℓ                                ▷ Return flipping length ℓ indicating N ⊨ o ≻^ℓ o′
6:      else if answer = false then
7:          return ∞                                ▷ No solution exists at any depth, so halt search
8:      end if
9:  end for
10: return ∞                                        ▷ No solution with flipping length ℓ ≤ k
```

Figure 19: Generic Algorithm: Depth-Limited Dominance Testing

5.1 Depth-Limited DT*

DT* employs a heuristic approach to dominance testing. The algorithm uses a priority queue and a heuristic function HF detailed by Li et al. (2011) to guide the search. The heuristic function can be computed for any outcome o_t and uses the Hamming distance between the current and goal outcomes, a penalty function, and weights defined on the nodes in the dependency graph. Nodes on the fringe of the search tree with the lowest positive values of $\mathrm{HF}(o_t)$ are searched first; negative values of $\mathrm{HF}(o)$, however, rule out any possibility of finding a solution. Pseudocode, adapted from Li et al. (2011), to which the reader is referred for details on how to compute the heuristic function,

is provided in Figure 20. The boxes emphasize changes that are central to the iterative approach employed here for DLDT. In fact, we use a depth-limited version of DT*: if the depth limit ℓ is reached, no successor nodes in the search tree will be added to the priority queue. If this occurs, the algorithm returns **unknown**. If all solutions can be ruled out, it returns **false**.

5.2 DT-SAT

The section concludes with a reduction of DLDT to SAT, as shown in Figure 21. For this reduction outcomes are modeled as *states* and flips as *actions* that transition between states, employing the *satisfiability as planning* (SATPlan) method of Kautz and Selman (1992). The variable ω denotes a Boolean formula in Conjunctive Normal Form (CNF), i.e., a conjunction of *clauses*, each a disjunction of *literals*. Initially ω is *empty*, with an assumed truth value of **true**, which we denote with the assignment $\omega \leftarrow \top$. To *write* a clause ξ means to conjoin it to ω to form a new formula, $\omega \leftarrow \omega \wedge \xi$. When all clauses have been written thus, a *SAT solver* is called. It is assumed that the solver returns **true** if ω is satisfiable and **false** otherwise.

5.2.1 States

Let $o_0, o_1, \ldots, o_{\ell-1}, o_\ell$ be the outcomes in a flipping sequence, such that $o'{=}o_0$, $o{=}o_\ell$, $\mathrm{HD}(o_t, o_{t+1}){=}1$, $o_t[X_i] \neq o_{t+1}[X_i]$, $o_t[\overline{X_i}] = o_{t+1}[\overline{X_i}]$, $0 \leq t < \ell$, and $1 \leq i \leq n$. We denote by $j = o_t[X_i]$ the value of X_i in the outcome at time t. The outcomes $z_{t,i,j}$ are modeled as Boolean state variables

$$z_{t,i,j} \iff o_t[i] = x_j^i, \tag{33}$$

for all (t, i, j), $0 \leq t \leq \ell$, $1 \leq i \leq n$, $1 \leq j \leq d$. Clauses are written to assert that the initial and final states, o' and o, occur at times 0 and ℓ respectively. It is also asserted that a variable X_i can have *just one* state at time t. That is, the variable has *at least* one state (value) and *at most* one state (value). For this it is helpful to define the operator

$$\operatorname*{\mathfrak{JustOne}}_{1\leq j\leq d} z_{t,i,j} = \begin{array}{l} (z_{t,i,1} \vee z_{t,i,2} \vee \cdots \vee z_{t,i,d}) \\ \wedge (\neg z_{t,i,1} \vee \neg z_{t,i,2}) \\ \wedge (\neg z_{t,i,1} \vee \neg z_{t,i,3}) \\ \quad \vdots \\ \wedge (\neg z_{t,i,1} \vee \neg z_{t,i,d}) \\ \wedge (\neg z_{t,i,2} \vee \neg z_{t,i,3}) \\ \quad \vdots \\ \wedge (\neg z_{t,i,d-1} \vee \neg z_{t,i,d}) \end{array}$$

which is used here in a manner analogous to the summation and product operators, $\sum$ and $\prod$.

5.2.2 Actions

Boolean variables $\alpha_{t,i,j,k}$ are also defined for each possible *action*. In a flipping sequence the value of just one variable changes at each time t. Thus, for all $t < \ell$, $i \leq n$, and distinct $x_j^i, x_k^i \in \mathrm{Dom}(X_i)$, there is a possible action corresponding to a flip from x_j^i to x_k^i. These are expressed in terms of their implications

$$\alpha_{t,i,j,k} \implies z_{t,i,j} \wedge z_{t+1,i,k}, \tag{34}$$

and it is specified that just one action occurs at every timestep $t < \ell$. *Framing rules* are also written to specify that if an action causes a variable to change, the other $n-1$ variables maintain their values

DEPTH-LIMITED DT*(N, o, o', $\boxed{\ell}$)

Input: N CP-net
o Goal outcome
o' Start outcome
ℓ Depth limit
Output: result **true** if $N \models o \succ^{\ell} o'$, **false** if $N \models o' \not\succ o$; $\boxed{\text{else } \textbf{unknown}}$

if HF(o') < 0 **then** ▷ Heuristic function from Li et al. (2011).
  return false
end if
$\boxed{cutoff \leftarrow \textbf{false}}$
insert $(o', 0)$ into priority-queue with priority HF(o')
while priority-queue $\neq \emptyset$ **do**
  $(o', \boxed{L})$ ← remove-first(priority-queue)
  if $o' = o$ **then** ▷ Goal test
    return true
  end if
  for all $X_i \in \mathcal{V}$ **do**
    if improvable(o', X_i) ∧ $X_i \notin$ any-matching-suffix(o', o) **then**
      $o'' \leftarrow$ single-flip(o', X_i)
      if not-repeated(o'') ∧ HF(o'') ≥ 0 **then**
        if $\boxed{L < \ell}$ **then** ▷ Can we flip and not exceed depth limit?
          insert $(o'', L + 1)$ into priority-queue with priority HF(o'')
        else
          $\boxed{cutoff \leftarrow \textbf{true}}$ ▷ Thus will not fully search the graph
        end if
      end if
    end if
  end for
end while
if *cutoff* = **true then**
  $\boxed{\textbf{return unknown}}$
else
  return false
end if

Figure 20: Solver Algorithm: Depth-Limited DT*

DT-SAT(N, o, o', ℓ)

Input: N CP-net
o Goal outcome
o' Start outcome
ℓ Predetermined search depth
Output: Boolean **true** if $N \models o \succ^{\ell} o'$, otherwise **false**

$\omega \leftarrow \top$
for $i \leftarrow 1$ **to** n **do** ▷ Assert initial and final states
  $\omega \leftarrow \omega \wedge z_{0,i,o'[i]}$
  $\omega \leftarrow \omega \wedge z_{\ell,i,o[i]}$
end for
for $t \leftarrow 1$ **to** ℓ **do** ▷ Just one state occurs for all t, i
  for $i \leftarrow 1$ **to** n **do**
    $\omega \leftarrow \omega \wedge \mathfrak{JustOne}_{1 \le j \le d} z_{t,i,j}$
  end for
end for
for $t \leftarrow 1$ **to** $\ell - 1$ **do**
  for $i \leftarrow 1$ **to** n **do**
    for distinct $j, k \le d$ **do**
      $\omega \leftarrow \omega \wedge \alpha_{t,i,j,k} \Rightarrow z_{t,i,j} \wedge z_{t+1,i,k}$ ▷ Implications of actions
      for $h \leftarrow 1$ **to** n s.t. $h \neq i$ **do** ▷ Framing rules
        for $q \leftarrow 1$ **to** d **do**
          $\omega \leftarrow \omega \wedge \alpha_{t,i,j,k} \wedge z_{t,h,q} \Rightarrow z_{t+1,h,q}$
          $\omega \leftarrow \omega \wedge \alpha_{t,i,j,k} \wedge \neg z_{t,h,q} \Rightarrow \neg z_{t+1,h,q}$
        end for
      end for
    end for
  end for
  $\omega \leftarrow \omega \wedge \mathfrak{JustOne}_{1 \le i \le n, 1 \le j,k \le d} \alpha_{t,i,j,k}$
  for each CPR in N of the form $u : x_j^i \succ x_k^i$ **do**
    $\omega \leftarrow \omega \wedge z_{t,p_1,u_1} \wedge \cdots \wedge z_{t,p_m,u_m} \Rightarrow \neg \alpha_{t,i,j,k}$ ▷ Disallow flip unless CPR permits
  end for
end for
answer $\leftarrow$ SAT-solver(ω)
if *answer* = **true then**
  return true
else
  return unknown
end if

Figure 21: Solver Algorithm: DT-SAT

from time t to $t+1$.

$$\alpha_{t,i,j,k} \wedge z_{t,h,q} \implies z_{t+1,h,q} \tag{35}$$

$$\alpha_{t,i,j,k} \wedge \neg z_{t,h,q} \implies \neg z_{t+1,h,q} \tag{36}$$

5.2.3 Modeling the CPRs

For SATPlan it is more natural to express what action did *not* occur. The pairwise relationships between domain values in the CPR is thus expressed as a conjunction of actions that did not occur in a valid flipping sequence of length ℓ.

A rule of the form $u : x_j^i \succ x_k^i$, where $u = u_1 u_2 \cdots u_m \in \text{Asst}(\text{Pa}(X_i))$, means a flip cannot occur from x_j^i to x_k^i in X_i when that node's parents are assigned the values in u; hence action $\alpha_{t,i,j,k}$ cannot occur under such circumstances. Let $X_{p_1}, X_{p_2}, \ldots, X_{p_m}$ denote the parents of X_i, such that $X_{p_1} = x_{u_1}^{p_1}$, $X_{p_2} = x_{u_2}^{p_2}$, etc. Observe that at time t in the flipping sequence, these assignments correspond to the state variables z_{t,p_1,u_1}, z_{t,p_2,u_2}, etc. Thus, the algorithm outputs rules of the form

$$z_{t,p_1,u_1} \wedge z_{t,p_2,u_2} \wedge \cdots \wedge z_{t,p_m,u_m} \Rightarrow \neg \alpha_{t,i,j,k}. \tag{37}$$

6. Conclusions

We have presented a highly tunable algorithm for generating CP-nets uniformly at random for a given number of variables and maximum in-degree. We have shown how to alter the algorithm to produce samples according to statistical cultures popular in voting, including the Impartial Anonymous Culture and Urn Models. We have argued that such generation can be used to provide statistical evidence for conjectures about CP-nets, such as our conjecture about maximum flipping lengths.

We have observed that the biggest impediment to the use of CP-nets is the computational complexity of the dominance testing problem, which is, in the general case, PSPACE-complete. However, our statistical evidence supports the hypothesis that, at least for CP-nets with binary-valued variables and complete CPTs, the dominance problem is in NP. If so, we can leverage heuristics for our favorite NP-complete problems to solve most instances of dominance testing quickly. We also presented a dominance testing algorithm that leverages standard depth-limited search techniques to provide an anytime algorithm. Extending the ideas in this paper to more general classes of CP-nets, such as those with heterogeneous domains, incomplete tables, and cycles in the dependency graph, provides interesting directions for future research. Another interesting direction is to continue to explore the relationship between various flavors of temporal logic and CP-nets to extend our quest for efficient DT implementations.

7. Acknowledgments

This paper is a revised and expanded version of our MPREF 2014 workshop paper Allen, Goldsmith, and Mattei (2014) and our AAAI 2016 Allen, Goldsmith, Justice, Mattei, and Raines (2016) paper. It includes details on all algorithms and proofs as well as an empirical testing section. All code and data is available on GitHub at https://github.com/nmattei/GenCPnet.

We wish to thank Mirek Truszczyński, Cory Siler, John Fike, and the anonymous reviewers for their valuable feedback and helpful suggestions at various stages of this project.

This material is based upon work supported by the National Science Foundation under Grant Nos. CCF-1215985 and IIS-1649152. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Some of this work was complete while Nicholas Mattei was supported by Data61, CSIRO (formerly NICTA) and UNSW, Australia. Data61, CSIRO (formerly NICTA) is funded by the Australian Government through the Department of Communications and the Australian Research Council (ARC) through the ICT Centre of Excellence Program.

References

Allen, T. E., Chen, M., Goldsmith, J., Mattei, N., Popova, A., Regenwetter, M., Rossi, F., & Zwilling, C. (2015). Beyond theory and data in preference modeling: Bringing humans into the loop. In *Proceedings of the Fourth International Conference on Algorithmic Decision Theory (ADT)*.

Allen, T. E., Goldsmith, J., & Mattei, N. (2014). Counting, ranking, and randomly generating CP-nets. In *MPREF 2014 (AAAI-14 Workshop)*.

Allen, T. E. (2016). *CP-nets: From Theory to Practice*. Ph.D. thesis, University of Kentucky.

Allen, T. E., Goldsmith, J., Justice, H. E., Mattei, N., & Raines, K. (2016). Generating CP-nets uniformly at random. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*.

Amor, N. B., Dubois, D., Gouider, H., & Prade, H. (2016). Graphical models for preference representation: An overview. In *Proceedings of the 10th International Scalable Uncertainty Management (SUM 2016)*, pp. 96–111.

Aydoğan, R., Baarslag, T., Hindriks, K. V., Jonker, C. M., & Yolum, P. (2013). Heuristic-based approaches for CP-nets in negotiation. In *Complex Automated Negotiations: Theories, Models, and Software Competitions*, pp. 113–123. Springer.

Bacchus, F., & Grove, A. (1995). Graphical models for preference and utility. In *Proceedings of the Eleventh conference on Uncertainty in Artificial Intelligence*, pp. 3–10. Morgan Kaufmann Publishers Inc.

Berg, S. (1985). Paradox of voting under an urn model: The effect of homogeneity. *Public Choice*, *47*(2), 377–387.

Bigot, D., Fargier, H., Mengin, J., & Zanuttini, B. (2013). Probabilistic conditional preference networks. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Bigot, D., Mengin, J., & Zanuttini, B. (2014). Learning probabilistic CP-nets from observations of optimal items. In *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS)*, pp. 81–90.

Bistarelli, S., Fioravanti, F., & Peretti, P. (2007). Using CP-nets as a guide for countermeasure selection. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, pp. 300–304. ACM.

Booth, R., Chevaleyre, Y., Lang, J., Mengin, J., & Sombattheera, C. (2010). Learning conditionally lexicographic preference relations. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pp. 269–274.

Boutilier, C., Brafman, R., Domshlak, C., Hoos, H., & Poole, D. (2004). CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, *21*, 135–191.

Boutilier, C., Bacchus, F., & Brafman, R. I. (2001). UCP-networks: A directed graphical representation of conditional utilities. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pp. 56–64. Morgan Kaufmann Publishers Inc.

Bouveret, S., Endriss, U., & Lang, J. (2009). Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*.

Brafman, R. I., Domshlak, C., & Shimony, S. E. (2006). On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research*, *25*, 389–424.

Brandt, F., Conitzer, V., Endriss, U., Lang, J., & Procaccia, A. D. (Eds.). (2016). *Handbook of Computational Social Choice*. Cambridge University Press.

Bringmann, K., & Panagiotou, K. (2012). Efficient sampling methods for discrete distributions. In *Automata, Languages, and Programming*, pp. 133–144. Springer.

Chevaleyre, Y., Endriss, U., Lang, J., & Maudet, N. (2008). Preference handling in combinatorial domains: From AI to social choice. *AI Magazine*, *29*(4), 37–46.

Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press.

Conitzer, V., Lang, J., & Xia, L. (2011). Hypercubewise preference aggregation in multi-issue domains. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 158–163.

Cormen, T. H., Stein, C., Rivest, R. L., & Leiserson, C. E. (2001). *Introduction to Algorithms* (2nd edition). McGraw-Hill Higher Education.

Cornelio, C., Goldsmith, J., Mattei, N., Rossi, F., & Venable, K. B. (2013). Updates and uncertainty in CP-nets. In *26th Australasian Joint Conference on Artificial Intelligence*.

Cornelio, C., Grandi, U., Goldsmith, J., Mattei, N., Rossi, F., & Venable, K. (2015). Reasoning with PCP-nets in a multi-agent context. In *Proceedings of the 14th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Cornelio, C. (2012). Dynamic and probabilistic CP-nets. Master's thesis, University of Padua.

Costa, L. d. F., Rodrigues, F. A., Travieso, G., & Villas Boas, P. R. (2007). Characterization of complex networks: A survey of measurements. *Advances in Physics*, *56*(1), 167–242.

Díez, F. J., & Druzdzel, M. J. (2007). Canonical probabilistic models for knowledge engineering. Tech. rep., Universidad Nacional de Educación a Distancia, Madrid, Spain.

Dimopoulos, Y., Michael, L., & Athienitou, F. (2009). Ceteris paribus preference elicitation with predictive guarantees. In *Proceedings of the 21st International Joint conference on Artificial Intelligence (IJCAI-09)*, IJCAI'09, pp. 1890–1895, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Domshlak, C., Hüllermeier, E., Kaci, S., & Prade, H. (2011). Preferences in AI: An overview. *Artificial Intelligence*, *175*(7), 1037–1052.

Dorn, B., & Krüger, D. (2015). On the hardness of bribery variants in voting with CP-nets. *Annals of Mathematics and Artificial Intelligence*, *77*(3-4), 251–279.

Dorn, B., Krüger, D., & Scharpfenecker, P. (2015). Often harder than in the constructive case: destructive bribery in CP-nets. In *International Conference on Web and Internet Economics (WINE 2015)*, pp. 314–327.

Eckhardt, A., & Vojtáš, P. (2009). How to learn fuzzy user preferences with variable objectives. In *Proceedings of the Joint 2009 International Fuzzy Systems Association World Congress and 2009 European Society of Fuzzy Logic and Technology Conference (IFSA/EUSFLAT)*, pp. 938–943.

Eckhardt, A., & Vojtáš, P. (2010). Learning user preferences for 2CP-regression for a recommender system. In *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM-10)*, pp. 346–357.

Fishburn, P. (1999). Preference structures and their numerical representations. *Theoretical Computer Science*, *217*(2), 359–383.

Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communations of the ACM*, *5*(6), 345.

Fürnkranz, J., & Hüllermeier, E. (2010). *Preference Learning: An Introduction*. Springer.

Gehrlein, W. V. (2002). Condorcet's paradox and the likelihood of its occurrence: Different perspectives on balanced preferences. *Theory and Decisions*, *52*(2), 171 – 199.

Goldsmith, J., & Junker, U. (2009). Preference handling for artificial intelligence. *AI Magazine*, *29*(4), 9–12.

Goldsmith, J., Lang, J., Truszczyński, M., & Wilson, N. (2008). The computational complexity of dominance and consistency in CP-nets. *Journal of Artificial Intelligence Research*, *33*(1), 403–432.

Gonzales, C., & Perny, P. (2004). GAI networks for utility elicitation. In *Procedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR-2004)*, pp. 224–233. AAAI Press.

Grandi, U., Luo, H., Maudet, N., & Rossi, F. (2014). Aggregating CP-nets with unfeasible outcomes. In *International Conference on Principles and Practice of Constraint Programming (CP 2014)*, pp. 366–381.

Guerin, J. T., Allen, T. E., & Goldsmith, J. (2013). Learning CP-net preferences online from user queries. In *Proceedings of the Third International Conference on Algorithmic Decision Theory (ADT)*, pp. 208–220. Springer.

Guerin, J. T. (2012). *Graphical Models for Decision Support in Academic Advising*. Ph.D. thesis, University of Kentucky.

Gulyás, L., Horváth, G., Cséri, T., & Kampis, G. (2011). An estimation of the shortest and largest average path length in graphs of given density. *arXiv preprint*, *1101.2549*.

Harrison, M. A. (1965). *Introduction to Switching and Automata Theory*, Vol. 65. McGraw-Hill.

Hu, S.-T. (1968). *Mathematical Theory of Switching Circuits and Automata*. University of California Press.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*(1), 267–306.

Kautz, H., & Selman, B. (1992). Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, pp. 359–363. Wiley.

Knuth, D. E. (1997). *The Art of Computer Programming: Seminumerical Algorithms* (3rd edition)., Vol. 2. Addison-Wesley Longman Publishing Co., Redwood City, CA, USA.

Knuth, D. E. (1998). *The Art of Computer Programming: Sorting and Searching* (2nd edition)., Vol. 3. Addison-Wesley Longman Publishing Co., Redwood City, CA, USA.

Kreher, D. L., & Stinson, D. (1999). *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press.

Kronegger, M., Lackner, M., Pfandler, A., & Pichler, R. (2014). A parameterized complexity analysis of generalized CP-nets. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1091–1097.

Kulkarni, V. G. (1990). Generating random combinatorial objects. *Journal of Algorithms*, *11*(2), 185–207.

Lang, J., & Xia, L. (2009). Sequential composition of voting rules in multi-issue domains. *Mathematical Social Sciences*, *57*(3), 304–324.

Lang, J., & Mengin, J. (2008). Learning preference relations over combinatorial domains. In *Twelfth International Workshop on Non-Monotonic Reasoning (NMR-08)*.

Lang, J., Mengin, J., & Xia, L. (2012). *Aggregating Conditionally Lexicographic Preferences on Multi-issue Domains*, pp. 973–987.

Lehmer, D. H. (1960). Teaching combinatorial tricks to a computer. In *Combinatorial Analysis*, Vol. 10 of *Procedings of Symposia in Applied Mathematics*, pp. 179–193.

Li, M., Vo, Q. B., & Kowalczyk, R. (2011). Efficient heuristic approach to dominance testing in CP-nets. In *Proceedings AAMAS*, pp. 353–360.

Li, M., Vo, Q. B., & Kowalczyk, R. (2015). Aggregating multi-valued CP-nets: a CSP-based approach. *Journal of Heuristics*, *21*(1), 107–140.

Liu, J., Xiong, Y., Wu, C., Yao, Z., & Liu, W. (2014). Learning conditional preference networks from inconsistent examples. *Knowledge and Data Engineering, IEEE Transactions on*, *26*(2), 376–390.

Liu, J., Yao, Z., Xiong, Y., Liu, W., & Wu, C. (2013). Learning conditional preference network from noisy samples using hypothesis testing. *Knowledge-Based Systems*, *40*, 7–16.

Liu, X., & Truszczynski, M. (2013). Aggregating conditionally lexicographic preferences using answer set programming solvers. In *International Conference on Algorithmic Decision Theory*, pp. 244–258. Springer.

Mattei, N. (2011). Empirical evaluation of voting rules with strictly ordered preference data. In *Proceedings of the Second International Conference on Algorithmic Decision Theory (ADT)*, pp. 165–177. Springer.

Mattei, N., Forshee, J., & Goldsmith, J. (2012). An empirical study of voting rules and manipulation with large datasets. In *Proceedings of the 4th International Workshop on Computational Social Choice (COMSOC)*. Springer.

Mattei, N., Pini, M., Rossi, F., & Venable, K. (2013). Bribery in voting with CP-nets. *Annals of Mathematics and Artificial Intelligence*, *68*(1-3), 135–160.

Mattei, N. (2012). *Decision Making Under Uncertainty: Theoretical and Empirical Results on Social Choice, Manipulation, and Bribery*. Ph.D. thesis, University of Kentucky.

Mattei, N., & Walsh, T. (2013). PrefLib: A library of preference data. In *Proceedings of the Third International Conference on Algorithmic Decision Theory (ADT)*. WWW.PREFLIB.ORG.

O'Connor, L. (1997). Nondegenerate functions and permutations. *Discrete Applied Mathematics*, *73*(1), 41–57.

Popova, A., Regenwetter, M., & Mattei, N. (2013). A behavioral perspective on social choice. *Annals of Mathematics and Artificial Intelligence*, *68*(1–3), 1–26.

Regenwetter, M., Grogman, B., Marley, A. A. J., & Testlin, I. M. (2006). *Behavioral Social Choice: Probabilistic Models, Statistical Inference, and Applications*. Cambridge University Press.

Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B. (Eds.). (2011). *Recommender Systems Handbook*. Springer.

Rintanen, J., & Gretton, C. O. (2013). Computing upper bounds on lengths of transition sequences. In *Proceedings of the 25th International Joint conference on Artificial Intelligence (IJCAI-13)*, pp. 2365–2372.

Robinson, R. W. (1973). Counting labeled acyclic digraphs. In Harary, F. (Ed.), *New directions in the theory of graphs: proceedings*, pp. 239–273. Academic Press.

Rossi, F., Venable, K. B., & Walsh, T. (2004). *m*CP nets: Representing and reasoning with preferences of multiple agents. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pp. 729–734. AAAI Press.

Rossi, F., Venable, K., & Walsh, T. (2011). *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*. Morgan & Claypool Publishers.

Santhanam, G. R., Basu, S., & Honavar, V. (2010). Dominance testing via model checking. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*.

Santhanam, G. R., Basu, S., & Honavar, V. (2015). Crisner: A practically efficient reasoner for qualitative preferences. *arXiv preprint*, *1507.08559*.

Sloane, N. (2016). The On-Line Encyclopedia of Integer Sequences. http://oeis.org. Accessed: 2016-03-20.

Steinsky, B. (2003). Efficient coding of labeled directed acyclic graphs. *Soft Computing*, *7*(5), 350–356.

Tideman, N., & Plassmann, F. (2012). Modeling the outcomes of vote-casting in actual elections. In Felsenthal, D., & Machover, M. (Eds.), *Electoral Systems: Paradoxes, Assumptions, and Procedures*. Springer.

Tsetlin, I., Regenwetter, M., & Grofman, B. (2003). The impartial culture maximizes the probability of majority cycles. *Social Choice and Welfare*, *21*(3), 387–398.

Walsh, T. (2010). An empirical study of the manipulability of single transferable voting. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*.

Walsh, T. (2011). Where are the hard manipulation problems?. *Journal of Artificial Intelligence Research*, *42*, 1–39.

Warshall, S. (1962). A theorem on boolean matrices. *Journal of the ACM*, *9*(1), 11–12.

Wicker, A. W., & Doyle, J. (2007). Interest-matching comparisons using CP-nets. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*.

Wilson, N. (2004). Extending CP-nets with stronger conditional preference statements. In *Proceedings of the 19th AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 4, pp. 735–741.

Xia, L., & Conitzer, V. (2010). Strategy-proof voting rules over multi-issue domains with restricted preferences. In *Proceedings of the 6th International Workshop on Internet and Network Economics (WINE 2010)*.

Xia, L., Conitzer, V., & Lang, J. (2011a). Hypercubewise preference aggregation in multi-issue domains. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-10)*.

Xia, L., Conitzer, V., & Lang, J. (2011b). Strategic sequential voting in multi-issue domains and multiple-election paradoxes. In *Proceedings of the 12th ACM Conference on Electronic Commerce (EC 2011)*.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2008). Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, *32*, 565–606.

Yaman, F., Walsh, T. J., Littman, M. L., et al. (2010). Learning lexicographic preference models. In *Preference learning*, pp. 251–272. Springer.