Generating CP-nets Uniformly at Random

Thomas E. Allen
University of Kentucky
Lexington, Kentucky, USA
teal223@g.uky.edu

Judy Goldsmith
University of Kentucky
Lexington, Kentucky, USA
goldsmit@cs.uky.edu

Hayden E. Justice
The Gatton Academy, WKU
Bowling Green, Kentucky, USA
hayden.justice259@topper.wku.edu

Nicholas Mattei
Data61 and UNSW
Sydney, Australia
nicholas.mattei@nicta.com.au

Kayla Raines
University of Kentucky
Lexington, Kentucky, USA
kayla.raines@live.com

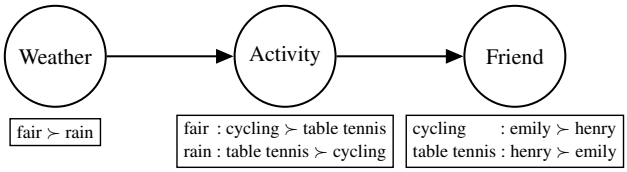
Abstract

Conditional preference networks (CP-nets) are a commonly studied compact formalism for modeling preferences. To study the properties of CP-nets or the performance of CP-net algorithms *on average*, one needs to generate CP-nets in an equiprobable manner. We discuss common problems with naïve generation, including sampling bias, which invalidates the base assumptions of many statistical tests and can undermine the results of an experimental study. We provide a novel algorithm for provably generating acyclic CP-nets uniformly at random. Our method is computationally efficient and allows for multi-valued domains and arbitrary bounds on the indegree in the dependency graph.

1 Introduction

Modeling, capturing, and reasoning with preferences is a fundamental topic that spans artificial intelligence, including constraint programming (Rossi, Venable, and Walsh 2011), social choice (Chevaleyre et al. 2008), recommendation systems (Ricci et al. 2011), machine learning (Fürnkranz and Hüllermeier 2010), multi-agent systems (Goldsmith and Junker 2009), and other fundamental areas. One of the most commonly studied preference models is the *conditional preference network (CP-net)* (Boutilier et al. 2004).

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 et al. 2011). CP-nets have many potential and important applications—automated negotiation (Aydoğan et al. 2013), interest-matching in social networks (Wicker and Doyle 2007), cybersecurity (Bistarelli, Fioravanti, and Peretti 2007), and as aggregation primitives for making group decisions (Lang and Xia 2009; Mattei et al. 2013; Xia, Conitzer, and Lang 2011), 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. For example, the CP-net below displays the statement, "If the weather is fair, I prefer to go cycling, but if it is raining, I'd rather play table tennis."



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 experiment, 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.

Experimental research in preference handling requires the use of real-world or simulated data. Real-world data are often messy, not openly available, notoriously difficult to collect reliably, hard to interpret, and nonexistent for CP-nets (Allen et al. 2015; Mattei and Walsh 2013). Principled methods exist to generate simulated data in social choice and preference handling using *generative cultures* (Berg 1985; Walsh 2011; Mattei, Forshee, and Goldsmith 2012). Such cultures have their drawbacks and limitations (Regenwetter et al. 2006; Popova, Regenwetter, and Mattei 2013), but provide a first step in experimentation for fields where data are hard to gather. While generative cultures over strict, linear

Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

orders are well defined in social choice, there is not an analog for preferences over more complex structures such as CP-nets. To generalize any statistical cultures used in social choice, we need to be able to generate samples uniformly at random from a specified set of CP-nets.

A key idea of our 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 show how to enumerate all such *dagcodes*, as these tuples are known, 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 allows us to generate the graph and CPTs, node by node, such that all acyclic CP-nets with a given domain size and bound on indegree are equiprobable.

We first formalize definitions that we need to discuss CP-nets and random generation, highlighting two problems—bias and degeneracy—that result from commonly used naïve generation methods. We show how to encode and count the dependency graphs. We next show how to avoid degeneracy in the CPTs and extend the recurrence to count all combinations of CPTs that a partially specified CP-net can have. Finally, we bring these results together to create an algorithm that samples the space of CP-nets uniformly.

2 Preliminaries

A preference relation $\succ$ is a partial order (a reflexive, 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 $\mathbf{V} = \{X_1, \ldots, X_n\}$ with associated domains $\mathrm{Dom}(X_i) = \{x_1^i, \ldots, x_d^i\}$ such that $\mathcal{O} = \mathrm{Dom}(X_1) \times \cdots \times \mathrm{Dom}(X_n)$.[1] We assume domain sizes are homogeneous; i.e., $|\mathrm{Dom}(X_i)| = d$ for all $X_i \in \mathbf{V}$. When a variable is constrained to exactly one value of its domain, we say the value has been assigned to it. We designate by $\mathrm{Asst}(\mathbf{U})$ the set of all assignments to $\mathbf{U} \subseteq \mathbf{V}$. An assignment to all variables $\mathbf{U} = \mathbf{V}$ designates a unique outcome $o \in \mathcal{O}$. We denote by $\mathbf{u}x_k^i$ the combination of $\mathbf{u} \in \mathrm{Asst}(\mathbf{U})$ and $x_k^i \in \mathrm{Dom}(X_i)$, where $\mathbf{U} \cap \{X_i\} = \emptyset$. The symbols $\overline{\ }$ and $\setminus$ denote set complementation and subtraction; e.g., $\overline{\mathbf{U}} \equiv \mathbf{V} \setminus \mathbf{U}$. 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 *conditional preference network* potentially provides a compact model of $\succ$.

Definition 1. *A* CP-net *is a directed acyclic graph (DAG) in which each node $X_i \in \mathbf{V}$ 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.*

Definition 2. *A* conditional preference table $\mathrm{CPT}(X_i)$ *specifies the preferences over node X_i given an assignment to its parents. Each CPT consists of* rules *of the form $\mathbf{u} : \succ^i$ specifying a linear order on X_i for all $\mathbf{u} \in \mathrm{Asst}(\mathrm{Pa}(X_i))$.*

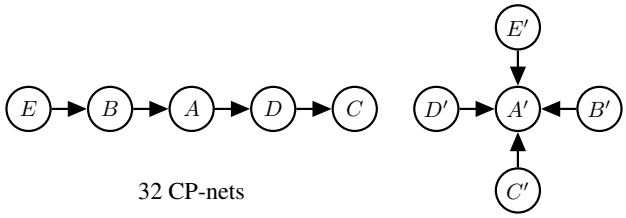
We use the term *dependency graph* to refer to the graph of a CP-net apart from its CPTs. The term *DAG* always refers to a *labeled* directed acyclic graph. $\mathrm{CPT}(X_i|X_h = x_k^h)$ denotes all rules of $\mathrm{CPT}(X_i)$ of the form $\mathbf{u}x_k^h : \succ^i$ where $x_k^h \in \mathrm{Dom}(X_h)$, $X_h \in \mathrm{Pa}(X_i)$, $\mathbf{u} \in \mathrm{Asst}(\mathrm{Pa}(X_i) \setminus \{X_h\})$. 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 *indegree* of X_i. Since the number of rules is exponential in m, we make the customary assumption that indegree is bounded by a small constant, i.e., $|\mathrm{Pa}(X_i)| \leq c$ for all X_i.

[1] The notation follows that of Boutilier et al. (2004).

3 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. We suspect that something along these lines is meant when we read in papers, "We generated 1000 CP-nets at random." However, the resulting distribution is *biased* statistically, and this bias calls into question the validity of experiments and the ensuing analysis of algorithms and methods.

To understand this bias, consider the following dependency graphs and their associated CP-net count (for $d = 2$).

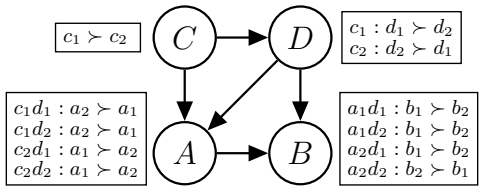


Observe that for the chain-shaped graph on the left, there are just two ways to choose each of the n CPTs such that they are consistent with the dependency graph. The CPT of root E could be $[e_1 \succ e_2]$ or $[e_2 \succ e_1]$. The other nodes, each of which has only one parent, also have two possibilities for their CPT; e.g., $\mathrm{CPT}(A)$ 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, $\mathrm{CPT}(A')$ 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 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.

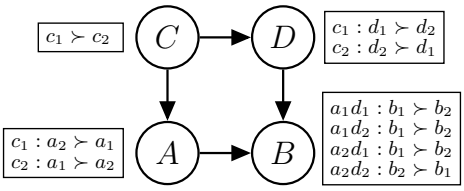
However, the naïve algorithm *does not even generate the two DAGs with equal likelihood*. Since there are $5! = 120$ ways to permute the labels of the first DAG, but only 5 ways to permute those of the second, the star-shaped DAG on the right would be generated 24 times as often as the chain-shaped DAG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

A separate problem, *degeneracy*, can arise in assigning the CPTs, regardless of how the DAGs are generated.

Example 3. *Consider the following CP-net.*



The edge from D to A 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. It can thus be represented by the following, simpler CP-net.



Understanding when a CPT is degenerate is crucial to generating CP-nets uniformly at random.

4 Encoding and Counting Graphs

To facilitate unbiased generation, we model the dependency graphs of CP-nets as *dagcodes* (Steinsky 2003), inspired by Prüfer codes for labeled trees (Kreher and Stinson 1999). We first treat the dagcode as an abstraction and then show how it relates to the dependency graph.

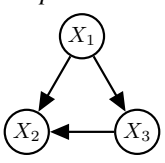
Definition 4. *A* dagcode $A = \langle A_1, \ldots, A_{n-1} \rangle$ *is a tuple of* $n-1$ *subsets* $A_j \subset \{1, \ldots, n\}$ *that satisfy the* cardinality constraint $|\bigcup_{k \le j} A_k| \le j$ *for all* j, $1 \le j < n$.

Observe from Def. 4 that tuples $\langle \{1\}, \{1,3\} \rangle$ and $\langle \{3\}, \emptyset \rangle$ are valid dagcodes ($n = 3$), 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 (see Alg. 1) and vice versa.

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



The sets $\{1\}$ *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 *(Alg. 1) (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 indegree c iff $|A_j| \le c$ 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 indegree 0.

Our generation method 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 indegree. 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 $\{4\}$.

Generalizing, let $A_{<j} = \langle A_1, \ldots, A_{j-1}, \_, \ldots, \_ \rangle$ be a *partial dagcode* for which only elements A_1 through A_{j-1} have been specified, such that for all k, $1 \le k < j$, $A_k \subset V$, $V = \{1, \ldots, n\}$, $|\bigcup_{\ell \le k} A_\ell| \le k$, and $|A_k| \le c$.

Algorithm 2 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 indegree. To generate all DAGs with n nodes and bound c on indegree, we call ALL-DAGS$(n, c, 1, 0, \emptyset, A_{<1})$.

Theorem 6. ALL-DAGS *generates each DAG exactly once.*

Proof. (Sketch.) Since 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 we will use the *recursion invariant:* Each time Line 1 is reached, $A_{<j}$ is valid; that is, for all k, $1 \le k < j$, $|\bigcup_{\ell \le k} A_\ell| \le k$ and $|A_k| \le c$. We will show that under this assumption, Alg. 2 generates each A_j such that the invariant holds for $A_{<j+1}$. *Base case:* Observe that the invariant holds trivially for the empty dagcode $A_{<1} = \langle \_, \ldots, \_ \rangle$.

Inductive hypothesis: Assume the invariant holds for $\mathrm{A}_{<j}$, $1 \le j < n$. Let $U = \bigcup_{k<j} A_k$ and $q = |U|$. Observe that the invariant will also hold for $A_{<j+1}$ so long as we choose $A_j \subset V$ such that $|U \cup A_j| \le j$ and $|A_j| \le c$. We 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 3 iterates over all (s,t) that satisfy these conditions. Lines 4–5, 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, Alg. 2 terminates. □

From ALL-DAGS we derive a new recurrence for the number of DAGs that is more easily extended to CP-nets

DAGCODE-TO-DAG(A)

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

Output: corresponding DAG G

1: $n \leftarrow \text{length}(A) + 1$
2: $Q \leftarrow \{1, \ldots, n\}$
3: initialize DAG G with n nodes and no edges
4: **for** $j \leftarrow n-1$ downto 1 **do**
5: $\quad i \leftarrow \max\left(Q \setminus \bigcup_{k=1}^{j} A_k\right)$
6: $\quad$ **for all** $h \in A_j$ **do**
7: $\quad\quad$ insert edge to X_i from its parent X_h
8: $\quad Q \leftarrow Q \setminus \{i\}$
9: output DAG G

Algorithm 1: Generate a DAG from its dagcode

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

Inputs:

n	number of nodes
c	bound on indegree
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: $\quad$ DAGCODE-TO-DAG($A_{<n}$); **return**
3: **for all** $s, t \geq 0, s \leq q, s+t \leq c, q+t \leq j$ **do**
4: $\quad$ **for all** $S \subseteq U,\ |S| = s$ **do**
5: $\quad\quad$ **for all** $T \subseteq \overline{U},\ |T| = t$ **do**
6: $\quad\quad\quad A_j \leftarrow S \cup T$; include A_j with $A_{<j}$ to form $A_{\leq j}$
7: $\quad\quad\quad$ ALL-DAGS($n, c, j+1, q+t, U \cup A_j, A_{\leq j}$)

Algorithm 2: Generate all DAGs that extend dagcode $A_{<j}$

than those of Robinson (1973) and Steinsky (2003).

Let $a_{n,c}$ denote *the number of DAGs* (resp. dagcodes) with n nodes and bound c on indegree. Let $a_{n,c}(j,q)$, where $q = |\bigcup_{k<j} A_k|$, denote the number of extensions to a partial dagcode $A_{<j}$. 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 indegree constraints are satisfied.

Theorem 7. $a_{n,c} = a_{n,c}(1,0)$. *For* $j = n$, $a_{n,c}(j,q) = 1$; *for all* j, $0 < j < n$, $a_{n,c}(j,q) =$

$$\sum_{\substack{s\geq 0,\ t\geq 0,\\ s\leq q,\ s+t\leq c,\\ q+t\leq j}} \binom{q}{s}\binom{n-q}{t} a_{n,c}(j+1, q+t). \qquad (1)$$

Proof. (Sketch.) *Base case* ($j = n$): One DAG is generated at Line 2; hence, $a_{n,c}(n,q) = 1$ for all q. *Inductive hypothesis:* Assume $a_{n,c}(j',q')$ gives the correct count for $j' > j$ and all q'. We 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 4 iterates over the $\binom{q}{s}$ ways to choose s elements from U. Similarly, the loop at Line 5 iterates over the $\binom{n-q}{t}$ ways to choose t elements from $\overline{U}$. 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 3, we take the sum of the DAGs generated in the loop body, obtaining the result given in Eq. 1. 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, $a_{n,c} = a_{n,c}(1,0)$. □

5 Counting and Generating the CPTs

We can generalize the notion of a degenerate CPT introduced in Section 3 with the help of a bijection with discrete multi-valued functions. We model each $\text{CPT}(X_i)$ as a function $F_j : \{0, \ldots, d-1\}^m \to \{0, \ldots, d!-1\}$. The inputs correspond to the values of the m parents of X_i. The output corresponds to one of the $d!$ orderings on the domain of X_i.

Observe that if variables are binary ($d = 2$), F_j is a Boolean function. In that 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 the outputs 1 and 0. For example, we can model the degenerate CPT of node A from Example 3 with the following truth table.

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

In_1	In_2	Out
0	0	0
0	1	0
1	0	1
1	1	1

Formally, we say $F_j(\mathbf{u})$ is *vacuous* in variable u_k iff its output never depends on u_k; i.e., for all $\mathbf{u} \in \{0, \ldots, d-1\}^m$,

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

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

Let $\phi_d(m)$ be the total number of possible CPTs for a node with m parents, and let $\psi_d(m)$ be the number of those that are non-degenerate. First consider binary domains ($d = 2$). Since 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, §10) (cf. Harrison 1965, O'Connor 1997) proved that for Boolean functions $\phi_2(m) = 2^{2^m}$, $\psi_2(m) = \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} 2^{2^k}$, and $\lim_{m\to\infty} {}^{\psi_2(m)}/_{\phi_2(m)} = 1$.

We now generalize these results to domains of size $d > 1$.

Theorem 8. $\phi_d(m) = d!^{d^m}$

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

BUILD-CP-NET(A, F)

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

Output: the corresponding CP-net N

1: $n \leftarrow \text{length}(A) + 1$; $Q \leftarrow \{1, \ldots, n\}$
2: initialize CP-net N with n nodes, no edges, empty CPTs
3: **for** $j \leftarrow n-1$ downto 1 **do**
4: $i \leftarrow \max(Q \setminus \bigcup_{k=1}^{j} A_k)$
5: **for all** $h \in A_j$ **do**
6: insert edge to X_i from its parent X_h
7: construct $\text{CPT}(X_i)$ from A_j, F_j
8: $Q \leftarrow Q \setminus \{i\}$
9: $i \leftarrow$ the only remaining element in Q
10: construct $\text{CPT}(X_i)$ from F_0
11: output CP-net N

Algorithm 3: Construct CP-net from its encoding

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

Inputs: n number of nodes
c bound on indegree
d size of domains
j is the index of current elements A_j, F_j
$q = |U|$, where $U = A_1 \cup \cdots \cup A_{j-1}$
$A_{<j}$, $F_{<j}$ are the partial dagcode and cpt-code

1: **if** $j = n$ **then**
2: BUILD-CP-NET($A_{<n}, F_{<n}$); **return**
3: **for all** $s, t \geq 0, s \leq q, s + t \leq c, q + t \leq j$ **do**
4: **for all** $S \subseteq U,\ |S| = s$ **do**
5: **for all** $T \subseteq V \setminus U,\ |T| = t$ **do**
6: **if** $j > 0$ **then**
7: $A_j \leftarrow S \cup T$; include A_j with $A_{<j}$ to form $A_{\leq j}$
8: **for all** $F_j : \{0, \ldots, d-1\}^{|A_j|} \to \{0, \ldots, d!-1\}$ **do**
9: **if** F_j is non-degenerate **then**
10: ALL-CP-NETS($n, c, d, j+1, q+t, U \cup A_j, A_{\leq j}, F_{\leq j}$)

Algorithm 4: Generate all CP-nets that extend $A_{<j}$

Theorem 9. $\psi_d(m) = \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} d!^{d^k}$.

Theorem 10. $\lim_{m\to\infty} \psi_d(m)/\phi_d(m) = 1$.

(The proofs of Thms. 9 and 10, omitted here, follow those of Hu (1968) [§2, §10] for Boolean functions.)

Theorem 11. *Determining whether a CPT (resp. its corresponding function F_j) is degenerate can be conducted in time polynomial in the size of the CPT (resp. domain of F_j).*

Proof. (Sketch.) Recall that a CPT of $X_i \in \mathbf{V}$ is degenerate if it is vacuous in any parent X_h. It is vacuous in X_h if $\text{CPT}(X_i | X_h = x_k^h) = \text{CPT}(X_i | X_h = x_\ell^h)$ for all x_k^h, $x_\ell^h \in \text{Dom}(X_h)$. Note that we can check the CPT of X_i for degeneracy using an algorithm with 4 nested loops: 1. Iterate over each $X_h \in \text{Pa}(X_i)$ to check whether the CPT is vacuous in X_h. 2. For each X_h, iterate over the d values of $x_k^h \in \text{Dom}(X_h)$. 3. For each x_k^h, $1 < k \leq d$, iterate over the $|\text{Asst}(\text{Pa}(X_i))| = d^{|\text{Pa}(X_i)|}$ rules to determine whether $\text{CPT}(X_i | X_h = x_k^h) = \text{CPT}(X_i | X_h = x_1^h)$ in each case. 4. For each rule $\mathbf{u}x_k^h : \succ^i$, $\mathbf{u} \in \text{Asst}(\text{Pa}(X_i) \setminus \{X_h\})$, iterate over the d values that specify the linear order $\succ^i$ on X_i. Let m denote the number of parents of X_i. Note that the input is d^{m+1}, since the CPT has d^m rules of length d. The nested loops require $O(md^{m+2})$ time. Thus, we can check for degeneracy in time polynomial in the size of the input.

Moreover, since CPTs of d-ary nodes with m parents correspond one-to-one with functions $F_j : \{0, \ldots, d-1\}^m \to \{0, \ldots, d!-1\}$, the proof applies also to the latter. □

We can leverage these results to generate non-degenerate CPTs efficiently and uniformly. For tiny values of d and m, we can choose uniformly from a modest-sized table of non-degenerate functions (e.g., $\psi_2(4) = 64594$). For larger values, we use *rejection sampling*, generating a random permutation $\succ^i$ for each assignment to parents 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, we will obtain a non-degenerate CPT on a given attempt.

6 Generating CP-nets

We can now extend ALL-DAGS (Alg. 2) to generate ALL-CP-NETS (Alg. 4). CP-nets with the same dependency graph differ if any rule of a CPT differs. To generate all combinations of CPTs, we need only introduce a new innermost loop iterating over the possibilities. Since the dagcode is partial, we do not yet have all of the information we need to construct the CPT: we know the parents, but not the child to which they belong. However, we do have enough information to iterate over the corresponding functions F_j, since we know the number of parents ($|A_j| = s + t$) and the size (d) of every domain, 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*. (We use $F_{<j}$ and $F_{\leq j}$, analogous to $A_{<j}$ and $A_{\leq j}$, 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, and we invoke Alg. 2 with $j = 0$ instead of 1. When $j = n$, the encoding is complete: A and F fully and uniquely characterize a CP-net N. We call Alg. 3 (cf. Alg. 1) to decode it—the DAG from A, the CPTs from F.

Theorems 6 and 7 can similarly be extended to CP-nets. Let $a_{n,c,d}$ denote *the number of CP-nets* with n nodes, bound c on indegree, and domains of size d. Let $a_{n,c,d}(j, q)$, $q = |\bigcup_{k<j} A_k|$, be the number of those that extend $A_{<j}$.

Theorem 12. *Algorithm 4 generates, exactly once, each CP-net N that extends partial dagcode $A_{<j}$.*

Theorem 13. $a_{n,c,d} = a_{n,c,d}(0, 0)$. *For j=n, $a_{n,c,d}(j, q) = 1$; for all j, $0 \leq j < n$, $a_{n,c,d}(j, q) =$*

$$\sum_{\substack{s\geq 0,\ t\geq 0,\\ s\leq q,\ s+t\leq c,\\ q+t\leq j}} \binom{q}{s}\binom{n-q}{t} \psi_d(s+t) a_{n,c,d}(j+1, q+t). \quad (2)$$

The loop at Line 8 executes $\psi_d(s+t)$ times. The proofs of Thms. 12–13 are otherwise congruent to those of Thms. 6–7.

Generating *all* CP-nets is feasible only for small n, c, and d.[2] To generate larger *random* instances, we propose an

[2] For example, $a_{6,5,2} = 4059976627283664056256$.

COMPUTE-DISTRIBUTION(n, c, d)

Input: n number of nodes
c bound on indegree
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: $weight \leftarrow \binom{q}{s}\binom{n-q}{t}\psi_d(s+t)\frac{a_{n,c,d}(j+1, q+t)}{a_{n,c,d}(j, q)}$
6: append row $[s, t, weight]$ to $\texttt{DIST}_{n,c,d}(j, q)$
7: sort rows on col. 3; assert that col. 3 sums to 1 (optional)
8: **return** $\texttt{DIST}_{n,c,d}$

Algorithm 5: Compute tables for uniform CP-net generation

efficient method that relies on Eq. 2. Algorithm 6 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, we keep 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 indegree. We also choose a non-degenerate function F_j for the CPT (see Sec. 5). To avoid bias, we choose (s, t) such that all extensions to $A_{<j}$ are equally likely, using a table precomputed by Alg. 5. Finally, we call Alg. 3 to output N.

Theorem 14. *Algorithm 6 generates each CP-net N with uniform probability $P(N) = 1/a_{n,c,d}$.*

Proof. (Sketch.) 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 4 chooses integers s and t with probability

$$\binom{q_j}{s}\binom{n-q_j}{t}\psi_d(s{+}t)\frac{a_{n,c,d}\,(j{+}1, q_j{+}t)}{a_{n,c,d}\,(j, q_j)}. \tag{3}$$

Then, given s, t, and U, Lines 5–10 choose S, T, and F_j with probability

$$\frac{1}{\binom{q_j}{s}}\frac{1}{\binom{n-q_j}{t}}\frac{1}{\psi_d(s+t)}. \tag{4}$$

Multiplying Eq. 3 and 4 and simplifying gives us the probability of generating A_j and F_j given U_j in Lines 4–10:

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

since $q_j + t = q_{j+1}$ for $j = 1$ to $n-1$ (Line 7).

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 3, the probability of generating N is: $P(N)$

$$= P(F_0)P(A_1F_1|U_1)P(A_2F_2|U_2)\cdots P(A_{n-1}F_{n-1}|U_{n-1})$$
$$= \frac{1}{d!}\frac{a_{n,c,d}\,(2, q_2)}{a_{n,c,d}\,(1, q_1)}\frac{a_{n,c,d}\,(3, q_3)}{a_{n,c,d}\,(2, q_2)}\cdots\frac{a_{n,c,d}\,(n, q_n)}{a_{n,c,d}\,(n{-}1, q_{n-1})}.$$

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

Input: n number of nodes
c bound on indegree
d size of the domains
Output: CP-net N generated i.i.d.

1: $F_0 \leftarrow$ random constant function with $d!$ outputs
2: $U \leftarrow \emptyset;\ \ q \leftarrow 0$
3: **for** $j \leftarrow 1$ to $n-1$ **do**
4: $s, t \leftarrow$ 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
5: $S \leftarrow$ subset of size s selected randomly from U
6: $T \leftarrow$ subset of size t selected randomly from $\overline{U}$
7: $A_j \leftarrow S \cup T;\ \ U \leftarrow U \cup T;\ \ q \leftarrow q + t$
8: **repeat**
9: $F_j \leftarrow$ random function with $|A_j|$ inputs, $d!$ outputs
10: **until** F_j is non-degenerate
11: BUILD-CP-NET(A, F)

Algorithm 6: Generate a CP-net uniformly at random

One can use Eq. 2 to verify that $a_{n,c,d}(0, 0) = d!a_{n,c,d}(1, 0)$; also, $q_1{=}|\bigcup_{k<1} A_k|{=}0$. We can thus rewrite the first term as $P(F_0) = 1/d! = a_{n,c,d}(1, q_1)/a_{n,c,d}(0, 0)$. 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

$$P(N) = \frac{1}{a_{n,c,d}(0, 0)} = \frac{1}{a_{n,c,d}} \tag{5}$$

which proves our case. □

Theorem 15. *Algorithm 5 runs in time and space polynomial in the number of nodes n.*

Proof. (Sketch.) Observe that the nested loops are bounded by n. We compute $a_{n,c,d}(j, q)$ with the help of a table. We need only perform this computation once for each j and q, and the ranges of j and q are similarly bounded by n. □

Algorithm 6 is also efficient. Random subset sampling and proportional (i.e., weighted) sampling can be performed efficiently (Bringmann and Panagiotou 2012; Knuth 1997, 3.4.2). The efficiency of rejection sampling (the inner loop) is discussed in Section 5.

7 Conclusion

We have presented an efficient and provably uniformly random method for generating CP-nets. The method allows for bounds on indegree and multi-valued domains. The recurrence of Theorem 13 can also be adapted to generate CP-nets from other distributions. For example, to generate the DAGs without weighting these by the number of CPT combinations, one can simply remove the $\psi_d(s+t)$ factor. Similarly, it is possible to generate tree-shaped CP-nets by changing the condition $s + t \leq c$ in Line 4 of Alg. 5 to $s + t = 1$.

We have implemented our method in C++ using the GnuMP library (Granlund et al. 2014), allowing generation of thousands of CP-nets per second. Our code is available at `http://cs.uky.edu/~goldsmit/papers/GeneratingCPnetCode.html`.

Acknowledgements

We thank Dr. Miroslaw Truszczyński for his suggestions and also the anonymous reviewers for their feedback.

Data61 (formerly known as NICTA) is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

Allen, T. E.; Chen, M.; Goldsmith, J.; Mattei, N.; Popova, A.; Regenwetter, M.; Rossi, F.; and 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)*.

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

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

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

Boutilier, C.; Brafman, R.; Domshlak, C.; Hoos, H.; and 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.

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

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

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

Fürnkranz, J., and Hüllermeier, E. 2010. *Preference Learning: An Introduction*. Springer.

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

Granlund, T., and the GMP development team. 2014. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.0.0 edition. `http://gmplib.org`.

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

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

Knuth, D. E. 1997. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co.

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

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

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

Mattei, N., and Walsh, T. 2013. PrefLib: A library of preference data. In *Proceedings of the Third International Conference on Algorithmic Decision Theory (ADT)*. `http://www.preflib.org`.

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

Mattei, N.; Forshee, J.; and 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.

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

Popova, A.; Regenwetter, M.; and Mattei, N. 2013. A behavioral perspective on social choice. *Annals of Mathematics and Artificial Intelligence* 68(1–3):135–160.

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

Ricci, F.; Rokach, L.; Shapira, B.; and Kantor, P. B., eds. 2011. *Recommender Systems Handbook*. Springer.

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

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

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

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

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

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