Multidisciplinary Workshop on Advances in Preference Handling: Papers from the AAAI-14 Workshop

Counting, Ranking, and Randomly Generating CP-Nets*

Thomas E. Allen
University of Kentucky
Lexington, Kentucky, USA
thomas.allen@uky.edu

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

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

Abstract

We introduce a method for generating CP-nets uniformly at random. As CP-nets encode a subset of partial orders, ensuring that we generate samples uniformly at random is not a trivial task. We present algorithms for counting CP-nets, ranking and computing the rank of an arbitrary CP-net for a given number of nodes, and generating a CP-net given its rank. We also show how to generate all CP-nets with a given number of nodes.

Introduction

Research in learning and manipulating complex preference structures from data is undertaken in a variety of communities including preference handling, machine learning, and operations research (Fürnkranz and Hüllermeier 2010). The ability to learn or elicit preferences quickly and efficiently has impacts in domains ranging from product recommendation (Ricci et al. 2011) to robot management (Goldsmith and Junker 2009).

Theoretical studies of learning CP-nets typically focus on specific query types and/or CP-nets restricted to acyclic networks or networks that have a tree structure (Koriche and Zanuttini 2010; Guerin, Allen, and Goldsmith 2013). A PAC algorithm for learning CP-nets is given by Dimopoulos, Michael, and Athienitou (2009); however, these results rely on strong guarantees about the underlying structure of the CP-nets. Results complemented by 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*.

Real-world data is often messy, notoriously difficult to collect reliably, and hard to interpret due to violations of the input model, such as intransitivity of preference under repeated experiment (Regenwetter, Dana, and Davis-Stober 2011). For this and other reasons, high quality empirical datasets relating to CP-nets are in short supply. Often empirical preference data take forms that are easier to elicit, such as partial orders, linear orders (Mattei and Walsh 2013), or pairwise comparisons built from two-alternative forced choice comparisons (that frequently violate transitivity (Tversky 1969)).

Using more complex models, such as CP-nets, to model individual preferences as primitives for aggregation is an emerging topic in computational social choice (Xia, Conitzer, and Lang 2008; Maudet et al. 2012). Often preferences are more complex than linear orders, and we may wish to aggregate these individual preference models in a meaningful way (Lang and Xia 2009; Mattei et al. 2013). Experimental research in social choice often uses distributions over preferences or *generative cultures* in addition to real data (Berg 1985; Walsh 2011; Mattei, Forshee, and Goldsmith 2012). While these cultures have their drawbacks and limitations (Regenwetter et al. 2006), they provide a first step in experimentation for fields where data is hard to gather.

While these cultures are well defined in the social choice community, there does not seem to be an analog for preferences over more complex structures such as CP-nets. In order to generalize any of the statistical cultures used in social choice, we need to be able to sample, uniformly at random, from the set of all CP-nets. However, since CP-nets can encode a subset of partial orders, this problem may be computationally hard, as counting the number of partial orders of a finite set is a standing open problem in mathematics and is only known for finite sets of up to about 18 outcomes (Brinkmann and McKay 2002; Sloane 2014). The counting problem for posets is conjectured to be hard, meaning that sampling uniformly at random from the set is likely hard as well (Jerrum, Valiant, and Vazirani 1986). Though generating posets uniformly at random has received some attention, most have considered how to generate sets that have particular properties, such as constant density of relations between any two given elements (Gehrlein 1986). It is unclear if the complexity of randomly sampling posets extends to the number of CP-nets.

The main contribution of this paper is a method to randomly generate CP-nets of a given number of nodes, generalizing results for counting the number of labeled directed acyclic graphs (LDAGs) (Steinsky 2003). Our method relies on algorithms for counting and ranking CP-nets and for

*This work is partially supported by the National Science Foundation, under grant CCF-1215985. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

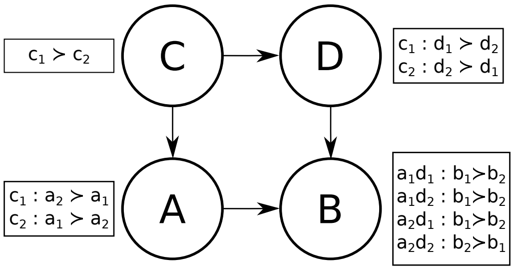

Figure 1: Binary CP-net with complete CPTs

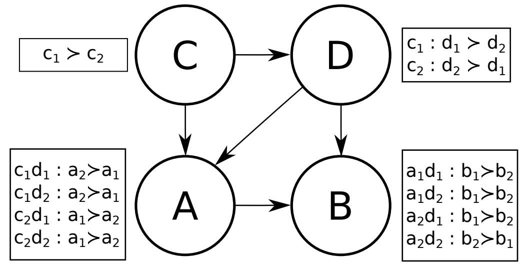

Figure 2: A degenerate CP-net

generating CP-nets given their ranks. While for brevity we limit discussion to binary CP-nets with complete tables, our method can be extended to more general cases.

In the following section we define CP-nets and their component parts and discuss the relationship between conditional preference tables and nondegenerate Boolean functions. We then discuss related work on counting and generating LDAGs, partially ordered sets (posets), and nondegenerate Boolean functions. Finally, we present our method for generating CP-nets at random. These methods, adapted from the work in LDAG generation, are able to count, rank, and unrank CP-nets, as well as generate all the CP-nets of a given number of nodes.

Preliminaries

A *partial order* $\lhd$ on a set S is a reflexive, antisymmetric, and transitive binary relation. That is, for all distinct $y, z \in S$, one of three cases must hold: (i.) y *precedes* z, (ii.) y *succeeds* z, or (iii.) y and z are *incomparable* w.r.t. $\lhd$. Such a set is called a *partially ordered set* or *poset*. A *total order* is a partial order in which no pair is incomparable. That is, for any distinct pair of elements in the ordered set, one element must precede the other. If S is a finite, totally ordered set, we can speak of the respective *rank* of its elements. Informally, the rank of an element is its ordinal position in the set.

Definition 1. *Let $\lhd$ be a total order on a finite set S. The* rank *of an element y in S is given by*

$$\mathrm{rank}_{S,\lhd}(y) = \#\{z \mid z \lhd y,\ y \in S,\ z \in S\}.$$

Thus, the first element in an ordered set has rank 0, the second has rank 1, and the last has rank $\#S - 1$, where $\#S$ is the number of elements in the set. A *ranking algorithm* is one that computes the rank of an element w.r.t. a totally ordered set. An *unranking algorithm* is one that outputs the element that has some given rank (Kreher and Stinson 1999).

Definition 2. *A* preference relation $\succ_v$ *is a partial order on a set of* outcomes $\mathcal{O}$ *by a* voter *(or subject) v.*

When no confusion would arise, we drop the subscript and write $o \succ o'$ to indicate that the voter regards o as better than o', $o \prec o'$ to indicate that o is worse than o', and $o \bowtie o'$ to indicate incomparability. Here we assume $\mathcal{O}$ is finite and can be factored into *variables* (or *features*) $\mathcal{V} = \{X_1, \ldots, X_n\}$ with associated binary *domains*, such that $\mathcal{O} = X_1 \times \cdots \times X_n$, where $X_i = \{x_1^i, x_2^i\}$. When a variable is constrained to just one of its values, we say that the value has been *assigned* to the variable and write $X_i = x_1^i$ or $X_i = x_2^i$. We designate by $\mathrm{Asst}(U)$ the set of all assignments to $U \subseteq \mathcal{V}$. An assignment to all variables $U = \mathcal{V}$ (an *instantiation*) designates a unique outcome $o \in \mathcal{O}$.

For binary variables, the total number of outcomes on n features is $\#\mathcal{O} = 2^n$. Thus, if each outcome is regarded as an atom, exponential space is required to store a preference relation $\succ_v$. However, since $\mathcal{O}$ is factored, a *conditional preference network* (CP-net) N_v (Boutilier et al. 2004) can potentially provide a compact model of $\succ_v$. While CP-nets can model multivalued variables and may in general contain cycles, here we restrict attention to binary CP-nets without cycles.

Definition 3. *A (binary, acyclic)* CP-net N_v *is a directed acyclic graph in which each node is labeled with a conditional preference table of a variable $X_i \in \mathcal{V}$, where $X_i = \{x_1^i, x_2^i\}$. An edge (X_h, X_i) indicates that the preferences over X_i in $\succ_v$ depend on the value of X_h, which we call a* parent *of X_i.*

Definition 4. *A* conditional preference table $\mathrm{CPT}(X_i)$ *specifies the voter's preference over X_i given an assignment to the node's parents, $\mathrm{Pa}(X_i) \subseteq \mathcal{V} \setminus \{X_i\}$. A CPT consists of a totally ordered set of* conditional preference rules *(CPRs) of the form $u : x_1^i \succ x_2^i$ or $u : x_2^i \succ x_1^i$, where $u \in \mathrm{Asst}(\mathrm{Pa}(X_i))$.*

Here we assume a lexicographic order on the CPRs by (i.) variable X_i and (ii.) value $x_j^i \in X_i$.

If $\mathrm{CPT}(X_i)$ contains rules for all 2^m assignments to the parents of X_i, where $m = \#\,\mathrm{Pa}(X_i)$ is the *indegree* of X_i, we say the CPT is *complete*; otherwise it is *incomplete*. We define $\mathrm{size}(\mathrm{CPT}(X_i))$ as the number of CPRs in the CPT and the size of a CP-net as the sum of the sizes of its CPTs. Observe that with no bound on in-degree, the size of the largest CP-net with n nodes and complete CPTs is

$$\max(\mathrm{size}(\mathcal{N})) = \sum_{k=0}^{n-1} 2^k = 2^n - 1. \qquad (1)$$

The semantics of a CP-net depend on the notion of a *flipping sequence* between a pair of comparable outcomes.

Definition 5. *Let $\mathbf{r} = \langle r_1, \ldots, r_\ell \rangle$ be a ranking over a subset of the outcomes in $\mathcal{O}$ s.t. for $j, k \in \{1, \ldots, \ell\}, j < k$ implies $r_j \prec r_k$. We call such a ranking an* improving flipping sequence *if r_j and r_{j+1} differ in the value of just one variable $X_i \in \mathcal{V}$ for all $j < \ell$.*

u_1	u_2	$h_1(u)$	$h_2(u)$	$h_3(u)$
0	0	0	0	0
0	1	0	0	0
1	0	0	1	0
1	1	1	1	0

Table 1: Truth Tables for Boolean Functions h_1, h_2, and h_3

A simple CP-net is shown in Fig. 1. Observe that the preference over feature C is unconditional: any outcome with $C = c_1$ is preferred to any with $C = c_2$. The preference over B, however, depends on C. If $C = c_1$, any outcome with $D = d_1$ is preferred to any with $D = d_2$. On the other hand, if $C = c_2$, outcomes with $D = d_2$ are preferred to any with $D = d_1$. Using this CP-net, we can observe, for example, that $a_2b_1c_1d_1 \succ a_1b_1c_1d_1$ without having to list all $240 \sim O(2^{2n})$ pairs of elements in the $\succ$ relation.

We require that the CPTs of a CP-net should agree with its dependency graph. For example, consider the CP-net in Fig. 2. The graph specifies that the preference over A depends on D, and the complete CPT of A contains an entry for all assignments to $C \times D$ as expected. However, on closer inspection we can observe from the CPT of A that in each case where $C = c_1$, $a_2 \succ a_1$ and where $C = c_2$, $a_1 \succ a_2$. Thus A only depends on C, not D. The preference relation entailed by the CP-net in Fig. 2 is thus identical to the simpler CP-net in Fig. 1. We refer to CP-nets like that of Fig. 2 as *degenerate*, since they are not in simplest form. In contrast, the CP-net of Fig. 1 is *nondegenerate* since its graph agrees with its CPTs. We can formalize this notion of *degeneracy* with the help of Boolean functions.

We denote by $F_2 = \{0, 1\}$ the binary field, closed under multiplication and modulo-2 addition, and by $F_2^k = \{0, 1\}^k$ the k-dimensional vector space over F_2.

Definition 6. *A* Boolean function *is a function of the form $f : F_2^k \to F_2$, where k is the number of* inputs.

We denote by $u = \langle u_1, \ldots, u_k \rangle$ the k-bit vector of inputs to $f(u)$ and by $u_{-j} = \langle u_1, \ldots, u_{j-1}, u_{j+1}, \ldots, u_k \rangle$ the values of u exclusive of u_j. We denote by $\mathrm{TT}(f)$ the 2^k-bit vector of 0s and 1s that compose the *truth table* of f and often use this to characterize a particular function. For example, if $h_1 \in F_2^2$, shown in Table 1, is the binary AND operation, we may write $h_1(u_1, u_2) = [0001]$ or equivalently $h_1 = [0001]$ when it is clear from context what the input sequences are. We will often assume that the input sequences to the function count up in the normal way for binary numbers, as illustrated by the sequence of u_1 and u_2 in Table 1. We say a function $f(u)$ is *vacuous in a variable* u_j if the output of f does not depend on u_j; that is, $f(u) = f(u_{-j})$ for all $u \in F_2^k$. We refer to such variables *vacated* or *fictional* (O'Connor 1997).

Definition 7. *A* degenerate Boolean function $f\colon F_2^k \to F_2$ *is one that is vacuous in at least one of its variables. If a function is not* degenerate*, it is* nondegenerate*.*

For example, the function $h_2\colon F_2^2 \to F_2$, where $h_2(u_1, u_2) = [0011]$, is degenerate since it is vacuous in u_2. The constant Boolean function $h_3\colon F_2^3 \to F_2$, where $h_3(u_1, u_2) = [0000]$, is similarly degenerate.

With this is mind, we can observe that each $\mathrm{CPT}(X_i)$ of a CP-net with n binary nodes corresponds to a Boolean function $f_i : F_2^k \to F_2$, where $k < n$ is the indegree (number of parents) of X_i. For this we adopt the convention that 1 in $\mathrm{TT}(f_i)$ maps to a rule in $\mathrm{CPT}(X_i)$ of the form $u : x_1^i \succ x_2^i$ and that 0 maps to $u : x_2^i \succ x_1^i$. For example, the CPTs of A, B, C, and D in Fig. 1 correspond to the Boolean functions $f_A(C) = [01]$, $f_B(A, D) = [1110]$, $f_C(\emptyset) = [1]$, and $f_D(C) = [10]$. We can further observe that the CP-net in Fig. 2 is degenerate since the CPT of A corresponds to the degenerate Boolean function $f_A(C, D) = [0011]$, which is vacuous in D.

Related Work

Kreher and Stinson (1999) provide a general discussion of ranking, unranking, and generating combinatorial objects. Robinson (1973; 1977) studied the problem of counting DAGs, both labeled and unlabeled, deriving the recurrence in our Eq. (2). Steinsky (2003) derived additional recurrences, along with a method of encoding LDAGs similar to Prüffer codes for labeled trees, as well as algorithms for ranking and unranking these so-called DAG codes. We build upon these results for the special case of CP-nets. The number of *transitively closed* LDAGs, or posets, is a long-standing open problem in mathematics. Erné and Stege (1991) include an extensive bibliography on the topic, as well as an exponential-time algorithm for counting labeled posets, extending the work of Culberson and Rawlins (1990) for the unlabeled case. Harrison (1965) and Hu (1968) studied degeneracy in Boolean functions, including the result shown in Eq. (3). The latter proved that the ratio of degenerate to nondegenerate Boolean functions converges to 0 as $n \to \infty$. Recent studies of degeneracy in Boolean functions have taken an interest in their cryptographic properties. O'Connor (1997), for example, proved that deciding whether a Boolean function f in DNF is vacuous in a variable (and hence degenerate) is NP-complete, but on average can be answered in linear time in the number of Boolean variables. In the case of CP-nets, however, we are interested in the size of the description, which, as shown in Eq. (1), is already exponential in n. Guerin, Allen, and Goldsmith (2013) randomly generated a set of CP-nets for a learning experiment; however, the algorithm requires specifying the number of edges in the LDAG in advance and cannot be used to sample randomly from a set of CP-nets in which the number of edges is allowed to vary.

Counting CP-nets

Let Δ_n be the set of all labeled directed acyclic graphs (LDAGs) with n nodes. The number of such graphs can be obtained via the recurrence $\#\Delta_n = a_n$, where $a_0 = 1$ and

$$a_n = \sum_{k=1}^{n} (-1)^{k+1} \binom{n}{k} 2^{k(n-k)} a_{n-k} \qquad (2)$$

Nodes	Number of CP-nets
0	1
1	2
2	12
3	488
4	481,776
5	157,549,032,992

Table 2: The Number of CP-nets with Complete CPTs

for $n > 0$ (Robinson 1977), or from one of Steinsky's recurrences (2003), yielding the sequence (Sloane [A003024])

$$1, 1, 3, 25, 543, 29281, 3781503, 1138779265, \ldots.$$

Next, let $\mathcal{F}_m$ be the set of all Boolean functions $f\colon F_2^m \to F_2$, with $\mathcal{D}_m \subset \mathcal{F}_m$ those that are degenerate and $\mathcal{G}_m = \mathcal{F}_m \setminus \mathcal{D}_m$ those that are nondegenerate. The cardinality of $\#\mathcal{D}_m$ (Harrison 1965) is given by

$$\#\mathcal{D}_m = \sum_{j=1}^{m} (-1)^{j+1} \binom{m}{j} 2^{2^{m-j}}, \tag{3}$$

derived via the inclusion–exclusion principle, which for $m = 1$ to 6 yields the sequence (Sloane [A005530])

$$2, 6, 38, 942, 325262, 25768825638, \ldots.$$

Moreover, since $\mathcal{D}_m$ and $\mathcal{G}_m$ form a partition of $\mathcal{F}_m$,

$$\#\mathcal{G}_m = 2^{2^m} - \#\mathcal{D}_m, \tag{4}$$

where we also designate this number of nondegenerate functions by $\gamma(m)$.

Finally, let $\mathcal{N}_n$ be the set of binary, acyclic CP-nets of n nodes with complete CPTs. The number of such CP-nets is then the number of possible CPTs for all possible dependency graphs; that is,

$$\#\mathcal{N}_n = \sum_{D \in \Delta_n} \prod_{X_j \in D} \gamma(\mathrm{pa}(X_j)), \tag{5}$$

where $\mathrm{pa}(X_j) = \#\,\mathrm{Pa}(X_j)$ is the number of parents of the node with label $j \le n$ in graph D. The computed number of CP-nets for small n is shown in Table 2.

Theorem 8. *Eq. 5 gives the correct cardinality for $\mathcal{N}_n$.*

Proof. (Sketch.) Observe that two CP-nets are distinct if their graphs differ and that CP-nets with the same graph are distinct if any entry of a CPT differs. Recall that the allowable CPTs for a node are in one-to-one correspondence with the nondegenerate Boolean functions of m variables, where m is the node's indegree. Let $\mathcal{N}_{n,D}$ denote the set of CP-nets that have the same DAG D with n labeled nodes. Observe that the CP-nets in $\mathcal{N}_{n,D}$ can be characterized by the tuples $(c_1, \ldots, c_n)$, where $0 \le c_j < \gamma(\mathrm{pa}(X_j))$, and the number of nondegenerate Boolean functions $\gamma(m)$ is obtained from Eqs. 3 and 4. The number of such n-tuples is $\#\mathcal{N}_{n,D} = \prod_{X_j \in D} \gamma(\mathrm{pa}(X_j))$. Finally, $\#\mathcal{N}_n = \sum_{D \in \Delta_n} \#\mathcal{N}_{n,D}$. ❑

Ranking CP-nets

To generate CP-nets randomly from some desired distribution (e.g., uniform), it is helpful to have some way of ranking them. Random generation is then reduced to the simple task of selecting an integer and generating the CP-net of corresponding rank. We first define a lexicographic order $>$ over $\mathcal{N}_n$ the set of CP-nets with n nodes. Next, we present an algorithm to compute the rank of a CP-net N w.r.t. $\mathcal{N}_n$.

We order two CP-nets $N, N' \in \mathcal{N}_n$ as follows: (1) We first compare dependency graphs. For this we use the ranking method described by Steinsky (2003): we convert the labeled DAG of each CP-net to its corresponding DAG code using DAGTODAGCODE and then find the rank of each DAG code using DAGCODERANK. (Due to space limitations, we do not reproduce the two algorithms here.) Let r and r' be the ranks of the DAG codes of N and N'. If $r > r'$, we say that $N > N'$. (2) If N and N' have the same graph, we next compare the CPTs of their nodes in order. Recall that the CPT of a node with k parents corresponds to a Boolean function f of k variables and that we can describe such functions succinctly via their 2^k-bit truth tables. We order truth tables of the same length by comparing the values of their respective bits in order as for binary numerals. Let f_1 and f_1' be the Boolean functions of node X_1 in N and N'. If N and N' have the same graph and $\mathrm{TT}(f_1) > \mathrm{TT}(f_1')$, we say that $N > N'$. If $f_1 = f_1'$, we next compare the Boolean functions for X_2, X_3, etc.

Algorithm 1 computes the rank $r < \#\mathcal{N}_n$ of a CP-net N. The `while` loop determines the relative rank of N w.r.t. CP-nets with the same graph. We next find the graph's rank w.r.t. Δ_n, counting $\#\{D_i : D_i \in \Delta_n, i < d\}$ with the help of Eq. 5. Alg. 2 performs the inverse unranking operation: that is, for valid r and N, CP-NET-UNRANK(CP-NET-RANK$(N, n), n) = N$ and CP-NET-RANK(CP-NET-UNRANK$(r, n), n) = r$. Lines 1–10 in Alg. 2 invert the operations of lines 7–9 in Alg. 1, and lines 11–20 in Alg. 2 invert the `while` loop of Alg. 1. The algorithms for ranking and unranking CP-nets depend respectively on algorithms for ranking and unranking the nondegenerate Boolean functions that correspond to CPTs provided in Algs. 3 and 4. Alg. 5 determines whether a given Boolean function is degenerate. All of the algorithms assume the availability of the set of labeled DAGs Δ_n, generated just-in-time using the method described by Steinsky (2003) or stored in a database for repeated use.

Generating CP-nets

With this framework in place, we can generate a CP-net uniformly at random by selecting a random integer $r < \#\mathcal{N}_n$ and calling CP-NET-UNRANK(r, n). Where this proves infeasible, we provide the following sketch of a heuristic random generation method: Generate a sequence of random integers $\langle d_0, \ldots, d_{\ell-1} \rangle$, one for each $i < \ell$ where $d_i < \#\Delta_n$. Each integer d_i corresponds to the rank of a labeled DAG D_i randomly sampled from Δ_n, the full set of LDAGs on n nodes. Compute the number of CP-nets c_i for each graph of rank d_i, as well as the sum $s = \sum_{i=0}^{\ell-1} c_i$ of all CP-nets of all graphs in the sample. Finally, generate an integer $r < s$ and

Input: CP-net N with n nodes
Output: rank r of the CP-net w.r.t. $\mathcal{N}_n$

$r \leftarrow$ CPT-RANK$(\mathrm{CPT}(X_1), \mathrm{pa}(X_1))$
$j \leftarrow 2$
while $j \leq n$ **do**
  $r \leftarrow r \cdot \gamma(\mathrm{pa}(X_j))$
  $r \leftarrow r +$ CPT-RANK$(\mathrm{CPT}(X_j), \mathrm{pa}(X_j))$
$G \leftarrow$ graph of N
$d \leftarrow$ DAGCODERANK(DAGTODAGCODE(G, n))
$r \leftarrow r + \sum_{D_i \in \Delta_n} \prod_{X_j \in D_i} \gamma(\mathrm{pa}(X_j))$

Algorithm 1: CP-NET-RANK$(N, n) \rightarrow r$

Input: rank r of a CP-net w.r.t. $\mathcal{N}_n$
Output: the corresponding CP-net N

$i \leftarrow 0$
$r' \leftarrow 0$
while $r' \leq r$ **do**
  $r'' \leftarrow r'$
  $r' \leftarrow r' + \prod_{X_j \in D_i} \gamma(\mathrm{pa}(X_j))$
  $i \leftarrow i + 1$
$i \leftarrow i - 1$
$r' \leftarrow r - r''$
$N \leftarrow$ initial CP-net with n nodes and graph D_i
$r_n \leftarrow 1$
for $j \leftarrow n$ **down to** 1 **do**
  $m_j \leftarrow \gamma(\mathrm{pa}(N.X_j))$
  $r_{j-1} \leftarrow r_j \cdot m_j$
for $j \leftarrow 1$ **to** n **do**
  $a_j \leftarrow r'$ div r_j
  $r' \leftarrow r' \mod r_j$
  $\mathrm{CPT}(N.X_j) \leftarrow$ CPT-UNRANK(a_j, X_j, N, n)

Algorithm 2: CP-NET-UNRANK$(r, n) \rightarrow N$

Input: CPT C with indegree m
Output: rank r of the corresponding Boolean function of m variables w.r.t. $\mathcal{G}_n$

$f \leftarrow$ Boolean function corresponding to CPT C
$b \leftarrow$ convert $\mathrm{TT}(f)$ from binary numeral to integer
$r \leftarrow 0$
for $k \leftarrow 0$ **to** $b - 1$ **do**
  if not DEGENERATE(k, m) **then**
    $r \leftarrow r + 1$

Algorithm 3: CPT-RANK$(C, m) \rightarrow r$

Input: rank r of a function in $\mathcal{G}_{\mathrm{pa}(X_j)}$
node X_j of CP-net $N \in \mathcal{N}_n$
Output: CPT C to be assigned to X_j in N

$k \leftarrow 0$; $r' \leftarrow 0$; $m = \mathrm{pa}(X_j)$
while $r' \leq r$ **do**
  if not DEGENERATE(k, m) **then**
    $r' \leftarrow r' + 1$
  $k \leftarrow k + 1$
$B \leftarrow$ convert $k - 1$ from an integer to a binary vector
$C \leftarrow$ from B construct CPT for X_j in CP-net N

Algorithm 4: CPT-UNRANK$(r, X_j, N, n) \rightarrow C$

Input: integer k s.t. $0 \leq k < 2^n$ corresponding to a Boolean function of n variables
Output: *true* iff Boolean function is degenerate

$L \leftarrow 2^n \times n$ matrix s.t. each row$_i$ is the n-bit binary numeral representation of i, for $0 \leq i < 2^n$
$B \leftarrow$ convert k to an 2^n-bit binary numeral vector
for $j \leftarrow 1$ **to** n **do**
  $I_0 \leftarrow \{i : L(i, j) = 0\}$
  $I_1 \leftarrow \{i : L(i, j) = 1\}$
  if $B[I_0] = B[I_1]$ **then**
    return true
return false

Algorithm 5: DEGENERATE$(k, n) \rightarrow$ Boolean

$N \leftarrow$ initial CP-net with n nodes and empty CPTs
for $D \in \Delta_n$ **do**
  $N.G \leftarrow D$
  $a_0 \leftarrow 0$; $m_0 \leftarrow 2$
  for $i \leftarrow 1$ **to** n **do**
    $a_i \leftarrow 0$
    $m_i \leftarrow \gamma(\mathrm{pa}(N.X_i))$
    $\mathrm{CPT}(N.X_i) \leftarrow \mathrm{NDBF}(\mathrm{pa}(N.X_i), 0)$
  repeat
    PROCEDURE-USING(N)
    $i \leftarrow n$
    while $a_i = m_i - 1$ **do**
      $a_i \leftarrow 0$
      $\mathrm{CPT}(N.X_i) \leftarrow \mathrm{NDBF}(\mathrm{pa}(N.X_i), 0)$
      $i \leftarrow i - 1$
    if $i > 0$ **then**
      $a_i \leftarrow a_i + 1$
      $\mathrm{CPT}(N.X_i) \leftarrow \mathrm{NDBF}(\mathrm{pa}(N.X_i), a_i)$
  until $i = 0$

Algorithm 6: GENERATE-ALL-CP-NETS(n)

unrank the corresponding CP-net using an algorithm adapted from CP-NET-UNRANK. We observe that some graphs have more CP-nets than others; our method of sampling reflects this distribution. However, in drawing i.i.d. from all CP-nets from a *subset* of graphs rather than from *all* those in Δ_n, this method necessarily has a statistical bias that depends on the sample size.

Sometimes it is necessary to generate all CP-nets to test their properties or perform an experiment. Algorithm 6 provides a blisteringly efficient way to visit, for example, all 157,549,032,992 CP-nets with 5 nodes. The outer loop iterates over the set of all dependency graphs (LDAGs of n nodes), while the inner `repeat` loop iterates over the possible assignments to the CPTs of the n nodes. For this we use the generation method described by Steinsky (2003). The inner loop is modeled after the mixed-radix generation method of Knuth (2011) for iterating over n-tuples. We assume the availability of a function or database entry $\mathrm{NDBF}(q, a_i)$ that returns the nondegenerate Boolean function of q variables that has the index a_i, where $0 \leq j < \gamma(m)$. Here a_i indexes the CPTs, while $m_i = \gamma(\mathrm{pa}(X_i))$ is the number of possible CPTs for each node. For each CP-net that is generated, a call is made to PROCEDURE-USING(N) that, for example, performs an experiment or writes the CP-net's description to a database for later use.

Conclusion

Generating sets of preferences represented as partial orders at random is problematic, especially when the set of outcomes is factored. In principle we could generate such a relation directly by generating a random poset. However, unlike linear orders, which are easy to generate, the number of posets is not known for finite sets larger than about 18 elements, and we are not aware of provably approximate methods for uniformly randomly generating posets of larger size (Gehrlein 1986). If the outcomes are factored, generating a preference relation directly as a poset would limit us to preferences over only 3 or 4 binary variables using known direct enumeration methods. To create such a data set for 5 variables, it would be necessary to generate posets of 32 outcomes—far beyond what is currently possible. By generating CP-nets uniformly at random using our exact method, we can easily exceed these limitations, despite the method's exponential complexity. Moreover, with our heuristic method we can randomly generate CP-nets with many nodes representing thousands of outcomes. In future work we plan to extend our algorithms to CP-nets with multivalued variables and incomplete tables, as well as special cases such as tree-shaped CP-nets and those with bounded indegree, and to make our source code available online.

References

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

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.

Brinkmann, G., and McKay, B. D. 2002. Posets on up to 16 points. *Order* 19(2):147–179.

Culberson, J. C., and Rawlins, G. J. 1990. New results from an algorithm for counting posets. *Order* 7(4):361–374.

Dimopoulos, Y.; Michael, L.; and Athienitou, F. 2009. Ceteris paribus preference elicitation with predictive guarantees. In *Proc. IJCAI* .

Erné, M., and Stege, K. 1991. Counting finite posets and topologies. *Order* 8(3):247–265.

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

Gehrlein, W. V. 1986. On methods for generating random partial orders. *Operations research letters* 5(6):285–291.

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

Guerin, J. T.; Allen, T. E.; and Goldsmith, J. 2013. Learning CP-net preferences online from user queries. In *Proc. ADT*. Springer. 208–220.

Harrison, M. A. 1965. *Introduction to switching and automata theory*, volume 65. McGraw-Hill New York.

Hu, S.-T. 1968. *Mathematical theory of switching circuits and automata*. Univ of California Press.

Jerrum, M. R.; Valiant, L. G.; and Vazirani, V. V. 1986. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science* 43:169–188.

Knuth, D. E. 2011. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms Part 1*. Addison-Wesley.

Koriche, F., and Zanuttini, B. 2010. Learning conditional preference networks. *Artificial Intelligence* 174(11):685–703.

Kreher, D. L., and Stinson, D. 1999. *Combinatorial algorithms: generation, enumeration, and search.* CRC Press.

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 *Proc. ADT*.

Mattei, N.; Pini, M. S.; Rossi, F.; and Venable, K. B. 2013. Bribery in voting with CP-nets. *Annals of Mathematics and Artificial Intelligence*.

Mattei, N.; Forshee, J.; and Goldsmith, J. 2012. An empirical study of voting rules and manipulation with large datasets. In *Proc. ComSoc*. Springer.

Maudet, N.; Pini, M. S.; Venable, K. B.; and Rossi, F. 2012. Influence and aggregation of preferences over combinatorial domains. In *Proc. AAMAS* , 1313–1314.

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

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

Regenwetter, M.; Dana, J.; and Davis-Stober, C. P. 2011. Transitivity of preferences. *Psychological Review* 118(1).

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.

Robinson, R. W. 1977. Counting unlabeled acyclic digraphs. In *Combinatorial mathematics V*. Springer. 28–43.

Sloane, N. J. A. 2014. The On-Line Encyclopedia of Integer Sequences. http://oeis.org. Accessed: 2014-04-15.

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

Tversky, A. 1969. Intransitivity of preferences. *Psychological review* 76(1):31.

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

Xia, L.; Conitzer, V.; and Lang, J. 2008. Voting on multiattribute domains with cyclic preferential dependencies. In *Proc. AAAI*, 202–207.