Learning Tree-Structured CP-nets with Local Search

Thomas E. Allen
Centre College
Danville, Kentucky, USA

Cory Siler and **Judy Goldsmith**
University of Kentucky
Lexington, Kentucky, USA

Abstract

Conditional preference networks (CP-nets) are an intuitive and expressive representation for qualitative preferences. Such models must somehow be acquired. Psychologists argue that direct elicitation is suspect. On the other hand, learning general CP-nets from pairwise comparisons is NP-hard, and — for some notions of learning — this extends even to the simplest forms of CP-nets. We introduce a novel, concise encoding of binary-valued, tree-structured CP-nets that supports the first local-search-based CP-net learning algorithms. While exact learning of binary-valued, tree-structured CP-nets — for a strict, entailment-based notion of learning — is already in P, our algorithm is the first space-efficient learning algorithm that gracefully handles noisy (i.e., realistic) comparison sets.

1 Introduction

Imagine a sandwich vendor that offers a subscription service to busy professionals who often can't spare the time to pick up or place an order for food. Because of the high demand around lunchtime, the vendor prepares the sandwiches ahead of time, offering only a limited selection each day. The delivery robot decides which kind of sandwich to bring to each subscriber, anticipating their preferences based on choices they have made when buying from the vendor in the past.

For a system to personalize itself to a user like this, it must be able to construct and reason with some form of *preference representation*. If the alternatives result from combining a large number of features (e.g., meats, cheeses, vegetables, and condiments), then the user cannot possibly rank them all explicitly. In addition, the user's preferences could be conditional; e.g., mustard may be preferred with cheese, but mayonnaise with meat.

In this paper, we discuss the problem of acquiring conditional preference networks (CP-nets) (Boutilier et al. 2004a) from comparisons representing prior choices. A CP-net consists of a node for each feature in the model — e.g., a binary variable indicating inclusion or exclusion of an item (e.g., sandwich ingredient), or a multi-valued variable representing a category (e.g., beverage) or state of the world (e.g., the day of the week). A directed edge from one node to another indicates that the preference over the latter feature depends on the value of the former. Such preferences take the form of rules such as, "If my sandwich has lettuce, I prefer cheese to no cheese, all else being equal." CP-nets have garnered significant interest within the computational preferences community and have been proposed for applications including cybersecurity (Bistarelli, Fioravanti, and Peretti 2007), negotiation (Aydoğan et al. 2013), and interest-matching in social networks (Wicker and Doyle 2007).

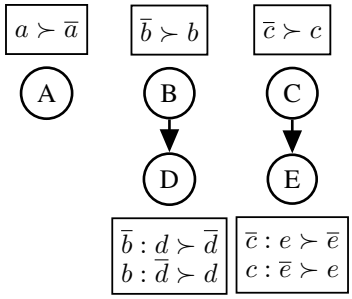

Figure 1: A binary-valued, tree-structured CP-net

Many problems involving CP-nets and their variants, including learning a CP-net that is consistent with comparisons and using that CP-net to determine which of two arbitrary outcomes is preferred, are known to be computationally hard in the worst case (Goldsmith et al. 2008; Lang and Mengin 2009). However, the second of these, the *dominance* problem, is known to be easy for CP-nets for which the dependency structure is a directed tree or forest (henceforth, *tree-structured* CP-nets) and the variables are binary-valued (Boutilier et al. 2004a; Bigot et al. 2013). Here we restrict our attention to this subclass in order to take advantage of efficient dominance testing. Tree-structured CP-nets are also the foci of other recent work by Alanazi, Mouhoub, and Zilles (2016) and Koriche and Zanuttini (2010).

Methods for acquiring a preference model generally fall into one of two categories. The first, *elicitation*, asks the user directly about their preference structure. The second, *learning*, relies instead on observing the person's choices. With regard to the former, computer scientists often assume that eliciting CP-nets is straightforward: We explain CP-nets to the user, and the user writes down the CP-net

that corresponds to their preferences (Boutilier et al. 2004b; Domshlak and Brafman 2002). Psychologists, however, question our ability to introspect in this way. They point out that people's reported preferences are often inconsistent with their choices and, unexpectedly, that introspection about preferences often *decreases* the chooser's satisfaction with their choice (Shafir, Simonson, and Tversky 1993; Wilson and Schooler 1991).

On the other hand, when presented with alternatives, people often seem to know *what* they want, even if they cannot explain the underlying reasoning process in a controlled experiment. Based on this assumption that we can use our preferences without fully understanding their underlying form, several recent CP-net learning algorithms depend on *comparison sets* — sets of binary choices from some domain. Some algorithms assume that the choices have been made prior to run time (Chevaleyre et al. 2010; Lang and Mengin 2008), a process known as *passive learning*. Others adaptively offer alternatives in an effort to decrease the number of queries needed. Such *active learning* paradigms include querying users about their preferences directly (Chevaleyre et al. 2010; Guerin, Allen, and Goldsmith 2013; Labernia et al. 2016) or Angluin-style learning queries (Alanazi, Mouhoub, and Zilles 2016; Koriche and Zanuttini 2010). In a smart-house setting where the AI controls the choices on offer, we could use a query-based learning algorithm. If we are stuck with whatever the deli's choices were (i.e., retrospective choices), we will use the sort of passive learning algorithms proposed here.

Lang and Mengin (2009) show that passively learning even a separable CP-net (with no dependencies) is NP-hard. The active learning algorithms (Koriche and Zanuttini 2010; Alanazi, Mouhoub, and Zilles 2016; Labernia et al. 2016) are efficient, both because the "membership queries" are restricted to computationally easy dominance queries (outcomes that differ on one variable in the general case, or any pair of outcomes for tree-structured CP-nets) (Koriche and Zanuttini 2010), and because they are in the active learning paradigm, where the learner specifies the next dominance test/swap query. Alanazi, Mouhoub, and Zilles (2016) also address PAC learning complexity of acyclic and tree-structured CP-nets.

Both in psychological experiments and in many preference data sets, such as those in the PrefLib repository (Mattei and Walsh 2013), we find examples of noisy and inconsistent preferences. For example, Kamishima and Akaho (2010) point out that when consumers were asked to rank ten sushi items and then later to assign rating scores to the same items, in 68% of the cases the ordering implied by the ratings did *not* agree with the ranking elicited only minutes before. There are many explanations for such phenomena, including a lack of computation power to consistently compute optima, and a very human desire for variety which drives apparent inconsistencies (McAlister and Pessemier 1982).

To our knowledge only Liu et al. (2013) have addressed the problem of passively learning general CP-nets from noisy data. (They also look at the sushi data set.) However, their algorithm explicitly builds the induced preference graph, which is exponentially larger than the CP-net. While we focus on a more specific class of models, our approach implicitly represents the preference graph by constructing only the CP-net, and thus is far more efficient with respect to space. Furthermore, we can control the time that our local-search-based algorithms take.

In Section 2 we introduce basic definitions. In Section 3 we discuss the meaning, and computational complexity, of learning a CP-net from comparison sets. In Section 4 we introduce a novel encoding of tree-structured CP-nets. In Section 5 we show how to exploit this encoding to facilitate local search, introducing WALK-CP-NET as an example. In Section 6 we describe experiments run with our algorithm. In Section 7 we summarize our contributions.

2 Preliminaries

A preference relation $\succ$ is a strict partial order (a reflexive, antisymmetric, transitive relation) on a set of outcomes $\mathcal{O}$. We indicate by $o \succ o'$ that outcome o is preferred to o'. If neither outcome is preferred to the other, we say that they are incomparable and write $o \parallel o'$. We assume $\mathcal{O}$ is finite and can be factored into n variables $V = \{X_1, \ldots, X_n\}$ with associated domains, s.t. $\mathcal{O} = X_1 \times \cdots \times X_n$. When domains are binary, we write $X_i = \{x_i, \overline{x_i}\}$. When a variable is constrained to just one value, we say the value has been assigned to the variable and write $X_i = x_i$ or $X_i = \overline{x_i}$. We designate by $\mathrm{Asst}(U)$ the set of all assignments to $U \subseteq V$. An assignment to all variables designates a unique outcome.

Definition 1. *A CP-net is a directed acyclic graph $G = (V, E)$ in which an edge (X_h, X_i) indicates that the preferences over X_i in general depend on the value of X_h. Each node is labeled with a conditional preference table (CP-table) specifying the preference over the value of that variable given all assignments to its parent variables* $\mathrm{Pa}(X_i)$.

A CP-table consists of rules $u : \succ$, where $u \in \mathrm{Asst}(\mathrm{Pa}(X_i))$ and $\succ$ is a linear order on the domain of X_i. If X_i is binary, the rules are of the form $u : x \succ \overline{x}$ or $u : \overline{x} \succ x$. We assume that CP-tables are complete, i.e., contain a rule for every combination of parent values.

If two outcomes differ in exactly one variable, then any CP-net (with complete CP-tables) will specify which outcome is preferred. We can think of a CP-net as a compact representation of a directed graph called the *induced preference graph* on the space of outcomes, with a edge between any pair of outcomes that differs on exactly one variable, directed toward the more preferred outcome; such an edge represents an *improving flip*. A directed path through this graph is an *improving flipping sequence*. Given a CP-net N and an arbitrary pair of outcomes (o, o'), if there is an improving flipping sequence from o' to o, then N entails that o is preferred to o'; we say o *dominates* o' and write $N \models o \succ o'$. The problem of deciding, for a given CP-net and pair of outcomes (o, o'), if there is an improving flipping sequence from o' to o, is called *dominance testing.* The computational complexity of dominance testing depends on the structure of the dependency graph G. A *tree-structured* CP-net is one in which each connected component of G takes the shape of a rooted tree with edges in the direction of the traversal (Fig. 1). While dominance testing is known to be

exceptionally hard (i.e., PSPACE-complete) for CP-nets in general (Goldsmith et al. 2008), dominance can be determined in linear time for tree-structured CP-nets (Bigot et al. 2013). Their computational properties and the relative ease with which trees can be represented (see the next section), make them attractive for applications that require real-time decision making and recommendations.

3 The Learning Problem and its Complexity

Let $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ be a set of comparisons $e_j = [\![o_j \succ o'_j]\!]$, typically obtained by observing the choices of the user. In general we do not assume $\mathcal{E}$ is consistent (i.e., it could be intransitive). Such inconsistency could arise due to noise (the user accidentally selects an unintended item from a list), aggregation (they select a meal that a guest will also enjoy), or changing preferences (they no longer enjoy a particular item as much now as when the data were collected). If $\mathcal{E}$ is intransitive, no preference relation $\succ$ can model $\mathcal{E}$ perfectly. Nevertheless, in practice we may still need to find a model that fits the data reasonably well.

As a starting point for talking about the complexity of learning tree-structured CP-nets, we refer to the algorithm of Dimopoulos, Michael, and Athienitou (2009). When it succeeds, this algorithm returns a CP-net that entails all comparisons in a set. However, in general it may fail even if an entailing CP-net exists; the authors prove the algorithm's completeness only for comparison sets with CP-nets that satisfy a specific subtype of entailment (*transparent* entailment). Additionally, we can prove (proof omitted for brevity) that the algorithm is complete for comparison sets that are entailed (even nontransparently) by a binary-valued, tree-structured CP-net. This, combined with the fact that the algorithm runs in polynomial time for a fixed indegree, gives us:

Theorem 1. *The problem of deciding whether there exists a binary-valued, tree-structured CP-net that entails a given set of comparisons is in P.*

However, when we relax the idea of "learning" to be broader than simply entailing the data set, the problem quickly becomes intractable. Lang and Mengin (2009) explore three notions of compatibility between a CP-net and a data set: In addition to *implicative* compatibility, wherein the CP-net simply entails all comparisons as described before, a comparison set may be *strongly* compatible with the CP-net — there is a completion of the CP-net's preference ordering that entails all comparisons — or *weakly* compatible with the CP-net — each individual comparison is consistent with (not necessarily entailed by) the CP-net. Lang and Mengin show that learning binary-valued *separable* CP-nets (*i.e.,* CP-nets with no edges in the dependency graph) that are strongly or weakly compatible with a comparison set is NP-hard; we observe that separable CP-nets are tree-structured, so learning a weakly or strongly compatible tree-structured CP-net from comparisons is hard as well.

Because comparison sets may be noisy, and because learning a tree-structured CP-net N consistent with data $\mathcal{E}$ may be hard, we instead try to optimize the CP-net's adherence to the comparisons. Michael and Papageorgiou (2013) also consider an optimization approach; they suggest using a MAX-SAT solver in conjunction with Dimopoulos, Michael, and Athienitou's algorithm to choose CP-tables that maximize the number of comparisons transparently entailed by the CP-net. For our approach, we optimize for the value of a fitness function $f(N, \mathcal{E})$.

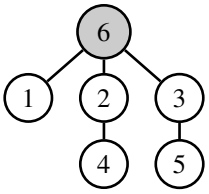

Figure 2: Undirected, labeled tree for the dependency graph of Figure 1

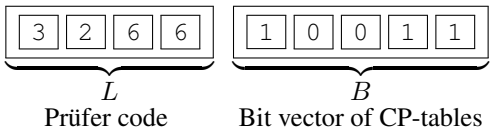

Figure 3: Treecode for the CP-net in Figure 1

We use the following simple fitness function: For each comparison $e_j = [\![o_j \succ o'_j]\!]$, if the model entails the same ordering ($N \models o_j \succ o'_j$), we award one point: $\alpha_j = 1$. If the model entails the opposite ordering ($N \models o'_j \succ o_j$), we subtract a point: $\alpha_j = -1$. If $o_j \parallel o'_j$ with respect to N, $\alpha_j = 0$. The fitness is the mean of the points awarded,

$$f(N, \mathcal{E}) = \frac{1}{m} \sum_{j=1}^{m} \alpha_j, \tag{1}$$

and the learning problem is to find a model N^* with maximum fitness, $N^* = \arg\max_N f(N, \mathcal{E})$.

4 Encoding Tree-Structured CP-nets

We can represent any tree-structured CP-net over n binary variables uniquely as a *treecode* (L, B), consisting of a *Prüfer code* L with $n-1$ integers in the range $\{1, \ldots, n+1\}$ for the dependency graph and a *bit vector* B with n integers in the range $\{0, 1\}$ for the CP-tables (Allen 2016). Figures 4 and 5 show how to convert a binary-valued, tree-structured CP-net to its treecode and from a treecode to its CP-net respectively, a mapping that we explain below.

As a basis for representing the graph, we refer to Prüfer (1918), who showed how to encode any undirected tree with k labeled nodes in a sequence of $k-2$ integers that has come to be known as a Prüfer code: $L = \langle L_1, \ldots, L_{k-2} \rangle$, where $L_j \in \{1, \ldots, k\}$, $1 \leq j \leq k-2$. In order to make use of this encoding, we must convert the *directed* and possibly *disconnected* dependency graph into a connected, undirected tree as follows.

Note that the dependency graph $G = (V, E)$ of a tree-structured CP-net is a forest G_F of undirected, labeled, rooted trees. To obtain G from G_F, in particular the directions of each edge $(X_h, X_i) \in E$, one need only traverse

TREE-STRUCTURED-CP-NET-TO-TREECODE(N)

Input: N Binary-valued, tree-structured CP-net
Output: L Prüfer code encoding the digraph
B Bit vector encoding the CPTs

1: **for** $i \leftarrow 1$ to n **do**
2: $B[i] \leftarrow 0$ or 1 based on mappings in Equation 2
3: relabel nodes V in N with the integers 1 to n
4: $R \leftarrow$ nodes of digraph G of N that are roots
5: $V \leftarrow V \cup \{n+1\}$
6: $E \leftarrow E \cup \{(n+1, r)\}$ for all $r \in R$
7: make edges E undirected
8: $L \leftarrow$ the Prüfer code for $(E, n+1)$
9: **return** (L, B)

Figure 4: Algorithm: Tree-Structured CP-net to Treecode

each tree in G_F from its root. Furthermore, any forest with n nodes can be mapped to an equivalent tree with $n+1$ nodes by introducing a new node, r, inserting edges from r to the root of each tree in forest G_F, and distinguishing r as the root of the newly formed tree with $n+1$ nodes. Conversely, to recover the forest from the tree, one can simply remove the root and its edges. Observe that the tree in Figure 2 corresponds to the dependency graph of the CP-net in Figure 1. After obtaining a tree, we can derive a corresponding Prüfer code (and vice versa) with algorithms described in the works of Kreher and Stinson (1999) and Allen (2016).

To construct the bit vector B, observe that there are only two possible CP-tables for a node with no parents or one parent (as shown in Equation 2). We can thus represent each of the n CP-tables of a tree-structured CP-net as a bit $B[i] =$ 0 or 1 according to the following mappings, for nodes with 0 and 1 parent, respectively.

$$\begin{array}{|c|} \hline \overline{x_i} \succ x_i \\ \hline \end{array} \leftrightarrow B[i]{=}0 \quad \begin{array}{|l|} \hline \overline{x_h} : \overline{x_i} \succ x_i \\ x_h : x_i \succ \overline{x_i} \\ \hline \end{array} \leftrightarrow B[i]{=}0$$
$$\begin{array}{|c|} \hline x_i \succ \overline{x_i} \\ \hline \end{array} \leftrightarrow B[i]{=}1 \quad \begin{array}{|l|} \hline \overline{x_h} : x_i \succ \overline{x_i} \\ x_h : \overline{x_i} \succ x_i \\ \hline \end{array} \leftrightarrow B[i]{=}1 \qquad (2)$$

The treecode representation facilitates local search by providing a straightforward way to define "neighboring" CP-nets: Two tree-structured CP-nets are *neighbors* if their treecodes differ in exactly one element. For a treecode over n variables, there are n^2 neighbors: there are n choices of CP-tables to change, and $n^2 - n$ potential edges to add/subtract (corresponding to changes to the Prüfer code).

5 Learning via Local Search

To show how our encoding facilitates learning tree-structured CP-nets with local search, we present a learning algorithm, WALK-CP-NET (Allen 2016), in Figure 6. Inspired by the GSAT (Selman, Levesque, and Mitchell 1992) and WALKSAT (Selman, Kautz, and Cohen 1994) algorithms for Boolean satisfiability, WALK-CP-NET combines hill-climbing with random perturbations.

TREECODE-TO-TREE-STRUCTURED-CP-NET(L, B)

Input: L Prüfer code
B Bit vector encoding the CPTs
Output: N Binary-valued, tree-structured CP-net

1: $n \leftarrow$ length of B
2: assert that L has length $n-1$
3: $V \leftarrow \{1, \ldots, n+1\}$
4: $E \leftarrow$ the tree for L
5: $G = (V, E)$
6: traverse G from node $n+1$ assigning directions to edges in order of traversal
7: $V \leftarrow V \setminus \{n+1\}$
8: $E \leftarrow E \setminus \{(n+1, k)\}$ for all $(n+1, k) \in E$
9: initialize CP-net N with digraph G
10: obtain CPTs of N from B using mappings in (2)
11: **return** N

Figure 5: Algorithm: Treecode to Tree-Structured CP-net

The algorithm maintains a current model, N, and a best model seen so far, N^*. It starts by initializing N to a random treecode. With probability $\pi \in [0, 1]$ (a user-selected parameter), the algorithm randomly walks (Line 12); otherwise, it attempts hill-climbing (Line 15). In the random walk, the algorithm replaces N with a randomly selected neighbor.

In hill-climbing, the algorithm uses dominance testing on $\mathcal{E}$ to evaluate the fitness (Equation 1) of each neighbor, and chooses a neighbor with the highest score. Note that if N is locally optimal, the fitness level may stay the same, or even decrease. If so, the search may have reached a plateau. To avoid getting stuck there, the algorithm increments a *strikes* counter. However, if the new fitness is strictly better, and the new model is the best yet encountered, the counter is reset to 0 (Line 8).

When the counter eventually reaches *max-strikes* (a user-selected parameter), the algorithm does a *random restart*, choosing a potentially distant model uniformly at random, and increments the restart counter, up to *max-restarts* (a user-controlled parameter). When *max-restarts* is reached, the algorithm terminates, returning N^*, the best model encountered. The algorithm can also terminate (Line 10) if it encounters a model that entails all comparison examples in $\mathcal{E}$, so N^* is provably a global optimum.

6 Experiments

The WALK-CP-NET configuration used in these experiments uses a *max-strikes* count of 3, a *max-restarts* count of 10, and a random walk probability of 30%. We chose these parameters empirically, based on trials on small CP-nets.

For each trial, to generate a comparison set given number of variables n and noise parameter p, we generated a random size-n tree-structured CP-net and chose random out-

WALK-CP-NET($\mathcal{E}$, n, π, max-strikes, max-restarts)

Input: $\mathcal{E}$ comparison set
n # of binary variables
π Prob. of random walk
max-strikes Counter to detect plateaux
max-restarts Max # of restarts
Output: N^* Fittest CP-net encountered

$N^* \leftarrow$ null model with fitness $-\infty$
for restarts $\leftarrow 1$ to max-restarts **do**
  $N \leftarrow$ random treecode over n variables
  strikes $\leftarrow 0$
  repeat
    if $f(N, \mathcal{E}) > f(N^*, \mathcal{E})$ **then**
      $N^* \leftarrow N$
      strikes $\leftarrow 0$
      if $f(N^*, \mathcal{E}) = 1$ **then**
        return N^*
    if random value in $[0, 1] \leq \pi$ **then**
      $N \leftarrow$ random neighbor of N
    else
      previous-fitness $\leftarrow f(N, \mathcal{E})$
      $N \leftarrow$ fittest neighbor of N
      if $f(N, \mathcal{E}) \leq$ previous-fitness **then**
        strikes $\leftarrow$ strikes $+ 1$
  until strikes $=$ max-strikes
return N^*

Figure 6: Algorithm: WALK-CP-NET

come pairs (o, o'), performing dominance testing on them with respect to the CP-net. If the CP-net entailed $o \succ o'$ or $o' \succ o$, we added the correct comparison to the comparison set with probability $1 - p$; with probability p, we added the opposite comparison instead. If the CP-net entailed neither comparison, with probability $1-p$ we did not add the pair to the comparison set; with probability p, we randomly chose between $o \succ o'$ and $o' \succ o$ to add to the comparison set. We repeated this process with distinct outcome pairs until 100 comparisons had been added to the set.

We ran 10 trials of WALK-CP-NET on each combination of variable count for the CP-nets (out of 10, 15, and 20) and noise parameter p for the data generation (out of .04, .08, and .16). Figures 7, 8, and 9 show the average percentage of the comparisons in our comparison sets that were consistent with, or entailed by, the learned CP-nets. Ideally, we would compare the quality of the learned CP-nets to the optimal tree-structured CP-nets, but since this is impractical at all but the smallest variable counts, we instead take as our baseline the original CP-nets used for generating the comparison sets.

The proportions of the comparison sets *entailed* by the learned CP-nets were modest, though they averaged at least 75% of the proportions entailed by the CP-nets used for generation for each set of trials. However, our algorithm did remarkably well at learning CP-nets *consistent* with the comparison sets, usually outperforming the generator CP-nets; it is worth investigating whether a fitness function specifically tuned for entailment or consistency can improve performance in terms of one of those metrics.

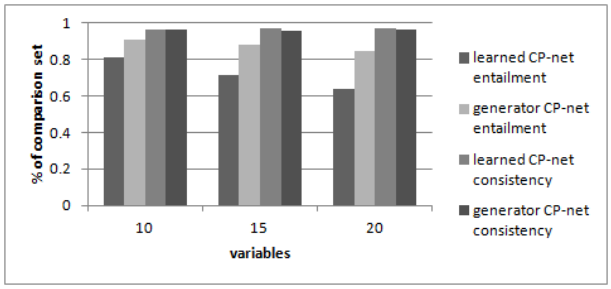

Figure 7: Averages with noise parameter $p = .04$

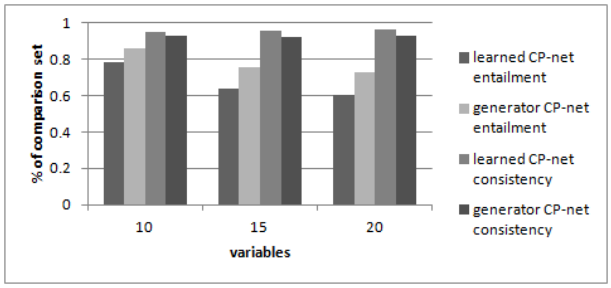

Figure 8: Averages with noise parameter $p = .08$

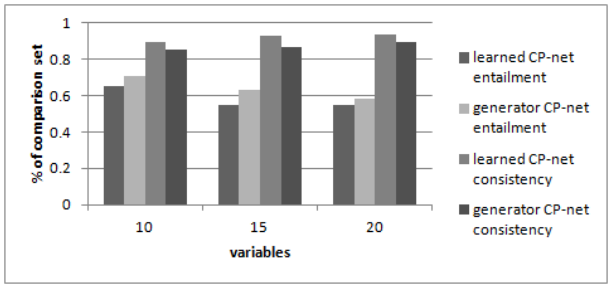

Figure 9: Averages with noise parameter $p = .16$

7 Conclusions

In this paper, we have introduced an elegant encoding of binary-valued, tree-structured CP-nets, whose computational properties make them a particularly useful subclass. We showed that our encoding can be leveraged to neatly code local search algorithms for learning these CP-nets. While algorithms for learning tree-structured CP-nets have been studied before (Alanazi, Mouhoub, and Zilles 2016; Koriche and Zanuttini 2010), ours is the first to robustly handle noisy comparison sets. Since there seems to be a consensus among psychological studies of preferences that human choices only noisily reflect underlying preferences, it is crucial to be able to learn preference models from noisy data. Local search, as exemplified by WALK-CP-NET, offers a flexible learning tool that lets us customize the tradeoff between efficiency and accuracy to suit the application.

Future work includes investigating local search for broader classes, such as acyclic CP-nets in general. While these do not support polynomial-time dominance testing, a fitness function based on an efficiently-testable metric like

transparent entailment (Michael and Papageorgiou 2013) could be substituted.

Acknowledgments

We thank Nicholas Mattei for helpful conversations. This work was partially supported by NSF grant 1215985.

References

Alanazi, E.; Mouhoub, M.; and Zilles, S. 2016. The complexity of learning acyclic CP-nets. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 1361–1367.

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

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.

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

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. 2004a. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research* 21:135–191.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004b. Preference-based constrained optimization with CP-nets. *Computational Intelligence* 20(2):137–157.

Chevaleyre, Y.; Koriche, F.; Lang, J.; Mengin, J.; and Zanuttini, B. 2010. Learning ordinal preferences on multiattribute domains: The case of CP-nets. In *Preference Learning*. Springer. 273–296.

Dimopoulos, Y.; Michael, L.; and Athienitou, F. 2009. Ceteris paribus preference elicitation with predictive guarantees. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, volume 9, 1890–1895.

Domshlak, C., and Brafman, R. I. 2002. CP-nets-reasoning and consistency testing. In *In Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning*. Citeseer.

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

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

Kamishima, T., and Akaho, S. 2010. Nantonac collaborative filtering: A model-based approach. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, 273–276. New York, NY, USA: ACM.

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.

Labernia, F.; Yger, F.; Mayag, B.; and Atif, J. 2016. Query-based learning of acyclic conditional preference networks from noisy data. In *DA2PL 2016: From Multicriteria Decision Aid to Preference Learning*.

Lang, J., and Mengin, J. 2008. Learning preference relations over combinatorial domains. *Proceedings of the Twelth International Workshop on Non-Monotonic Reasoning* 207–214.

Lang, J., and Mengin, J. 2009. The complexity of learning separable ceteris paribus preferences. In *IJCAI-09*, 848–853. San Francisco, CA, USA: Morgan Kaufmann.

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

Mattei, N., and Walsh, T. 2013. PrefLib: A library of preference data. In *Proceedings of Third International Conference on Algorithmic Decision Theory*, 259–270. Springer.

McAlister, L., and Pessemier, E. 1982. Variety seeking behavior: An interdisciplinary review. *Journal of Consumer Research* 311–322.

Michael, L., and Papageorgiou, E. 2013. An empirical investigation of ceteris paribus learnability. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 1537–1543. AAAI Press.

Prüfer, H. 1918. Neuer Beweis eines Satzes über Permutationen. *Archiv der Mathematik und Physik* 27(1918):742–744.

Selman, B.; Kautz, H. A.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 337–343. MIT Press.

Selman, B.; Levesque, H. J.; and Mitchell, D. G. 1992. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 440–446. AAAI Press.

Shafir, E.; Simonson, I.; and Tversky, A. 1993. Reason-based choice. *Cognition* 49(1):11–36.

Wicker, A. W., and Doyle, J. 2007. Interest-matching comparisons using CP-nets. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, 1914–1915. AAAI Press.

Wilson, T. D., and Schooler, J. W. 1991. Thinking too much: introspection can reduce the quality of preferences and decisions. *Journal of Personality and Social Psychology* 60(2):181.