CP-nets: From Theory to Practice

Thomas E. Allen
thomas.allen@uky.edu

University of Kentucky
Lexington, Kentucky

1 Introduction

Consider the problem of buying an automobile. The vehicle that a customer prefers may depend on many factors including the customer's life commitments, hobbies, income level, concern about the environment, and so on. For example, a customer may prefer a minivan to a sports car if he has young children. Another customer may prefer a pick-up truck with a towing package to one that lacks such a feature if she enjoys hiking and kayaking. Numerous other factors, such as whether the paint shows pollen or whether the spare tire is accessible could also prove important.

Many decision domains, like this one, have a combinatorial structure. If I am asked to choose from a very small set of alternatives $O = \{O_1, \ldots, O_4\}$, I can usually provide at least a partial ordering. I may be unsure how to compare alternatives O_1 and O_3, but know that I like O_4 better than both, and O_2 less than O_1 and O_3. However, if the preference involves a large number of features (e.g., make, model, dealership, warranty, color, etc.), then this approach is no longer practical, or even possible, given time constraints. Indeed, the best alternative may not yet exist in the physical world. Perhaps I could describe my most preferred alternative in terms of its features, but it does not exist unless I have it built to my custom specifications.

Formally, a preference relation $\succsim$ is a partial preorder on a set of alternatives (or outcomes) O. The expression $o \succ o'$ means that o is preferred to o'. If neither outcome is preferred to the other, they are said to be incomparable, written $o \bowtie o'$. In this case, $O_4 \succ O_1$, $O_4 \succ O_3$, $O_1 \succ O_2$, and $O_3 \succ O_2$. Since the relationship is assumed to be transitive, one can further reason that $O_4 \succ O_2$. However, since the relationship between O_1 and O_3 is left unspecified, we say that these alternatives are incomparable ($O_1 \bowtie O_3$). In the discussion that follows, let us assume that O is finite and can be factored into features (variables) $V = \{X_1, \ldots, X_n\}$ with associated domains, s.t. $O = X_1 \times \cdots \times X_n$. For example, the binary valued feature BRAKES = {*antilock*, *conventional*} would be one of many features that collectively model the set of all conceivable alternatives in the set **Automobiles**.

Conditional preference networks (CP-nets) have been proposed for such problems [4]. Rather than comparing alternatives as atoms, the decision maker considers the interplay of the features—how the preference over one feature depends on the values of others in the decision domain. For example, if I am purchasing a vintage sports car, I may be willing to forgo modern safety features such as anti-lock brakes, but if I am purchasing a new minivan, then I definitely prefer anti-lock brakes. One can formalize

this as

$$minivan \wedge new : antilock \succ conventional.$$

Such statements are known as *ceteris paribus rules*, from the Latin "as long as everything else stays the same." That is, if I am comparing two new minivans, one with anti-lock brakes and one with conventional brakes, I will always prefer the alternative with antilock brakes, provided the values of other features (price, color, make, model, etc.) are the same for both alternatives. The relationship among the features is further specified through a *dependency graph*, a directed graph in which the nodes represent the relevant features and a directed edge from one feature to another indicates that the latter (known as the child) depends on the former (the parent). The rules themselves are stored in *conditional preference tables* of the feature to which they apply.

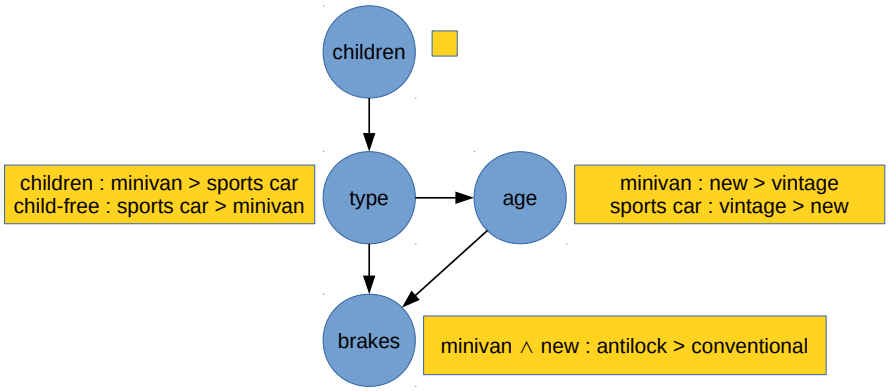

Fig. 1. A simple CP-net

Definition 1. *A* CP-net $\mathcal{N}$ *is a directed graph. Each node represents a variable* $X_i \in V$ *and is annotated with a* conditional preference table (CPT) *describing the subject's preferences over the domain of* X_i *given its dependencies. An edge* (X_h, X_i) *indicates that the preferences over* X_i *depend directly on the value of* X_h.

Alternatives that differ in just one feature (e.g., two **Automobile**s that are identical except for their AGE) can be compared directly, provided CPT(AGE) contains an applicable rule. Alternatives that differ in more than one feature can be compared only if a transitive sequence of rules exists between the two alternatives. For example, $\langle$ *children*, *sports car*, *vintage*, *conventional* $\rangle \prec \langle$ *children*, *minivan*, *vintage*, *conventional* $\rangle \prec \langle$ *children*, *minivan*, *new*, *conventional* $\rangle \prec \langle$ *children*, *minivan*, *new*, *antilock* $\rangle$. Such sequences are known as (improving) flipping sequences, since each rule "flips" the value of just one variable.

There is much to like about CP-nets. They let us model preferences over factored domains with exponentially many conceivable alternatives. They capture visually the if-then rules that many of us think we employ when we reason about such alternatives. They are qualitative; that is, they only ask us to specify whether one thing is better than another, without assigning a numeric weight as to precisely *how much* we prefer it. Finally, the problem of determining the optimal (most preferred) outcome with respect to a CP-net can be answered in linear time in the number of features if CPTs are complete.

On the other hand, while many academic papers discuss CP-nets (at last count, the seminal journal paper had over 750 citations, according to Google Scholar) and many applications have been proposed, we are not yet aware of their use in real-world applications. There are several reasons for this. First, determining dominance—whether one arbitrary outcome is better than another with respect to a CP-net—is known to be very hard (PSPACE-complete) in the general case [5]. An application for automobile shoppers, to continue our example, is not particularly useful if it requires several days in the worst case to determine whether one vehicle is better than some other vehicle that I am considering! Additionally, much of the research on CP-nets makes strong, often unrealistic assumptions, such as that the features must be binary, CPTs must be complete (contain a rule for every combination of parent features), that indifference is not allowed, or that the graph must conform to a particular structure (e.g., rooted tree, acyclic digraph, etc.). My research involves addressing these issues to make CP-nets more useful for complex engineering applications.

In the sections that follow, I briefly discuss three areas of ongoing research: how to generate CP-nets i.i.d., practical approaches to the problem of dominance testing (DT), and new methods for learning CP-nets.

2 Generating CP-nets

Often, one needs to generate CP-nets in an i.i.d. manner. For example, consider that we wish to compare the *expected* running time of two algorithms that perform dominance testing. Both algorithms are designed for the same sort of input—e.g., binary valued features and complete, acyclic CPTs with an assumed bound on in-degree (number of parents). To compare the two algorithms, we wish to generate a set of dominance testing problems that are representative of the DT problems in this set. We are not aware of real-world datasets of DT problems; moreover, even if such datasets were readily available, one would like to compare the algorithms' expected performance on *any* allowable input. Generating the outcome pairs for such an experiment is easy since one can just assign the value of each feature as an independent coin-flip. However, it is not so clear how to generate the CP-nets such that each instance is valid and equally likely with respect to the set of all CP-nets under consideration. Often researchers simply permute the nodes and insert edges at random so as to avoid cycles and then randomly assign CPT entries. However, it is easy to show that this approach leads to statistical bias, with high-indegree CP-nets grossly undersampled at the expense of low-indegree CP-nets.

Other motivations for generating CP-nets i.i.d. include Monte Carlo algorithms (e.g., for learning, reasoning with, or aggregating CP-nets), statistical analyses of the properties of CP-nets (such as how expected flipping sequence length varies as the

number of edges in the dependency graph increases), voting profiles for social choice experiments, black-box testing of algorithms for correctness, and finding hard problem instances. In each case, an unbiased generation algorithm is needed.

Many asymptotic results are known for problems that involve learning or reasoning with CP-nets. However, the combinatoric properties of CP-nets are less understood. Since these properties are central to understanding how to generate unbiased random instances, I began by studying how to count the number of CP-nets for various types of graphs (directed trees, polytrees, directed acyclic graphs with bounded indegree, etc.) [2]. For this I built upon results such as Prüfer codes, which can be used to represent and generate labeled trees [8], and the so-called DAG codes for directed acyclic graphs [11], introducing novel recurrences for counting and generating CP-nets efficiently. As part of this process, I studied the problem of assuring that a CPT generated at random was consistent with the graph, showing that this is equivalent to the set of *nondegenerate functions* of k Boolean inputs where k is the node's indegree. I have also developed and implemented (in C++ and x86 assembly language) a novel method for generating random binary, complete CP-nets i.i.d. for various graphs, including directed trees, polytrees, DAGs with bounded indegree, and unbounded DAGs. My implementation can generate thousands of CP-nets with up to 100 features uniformly at random in just a few seconds. I am presently extending this method to accommodate more general classes of CP-nets, such as those with multi-valued variables, incomplete CPTs, CP-nets that can model indifference as well as strict preferences, and CP-nets that are compatible with a partially specified (incomplete) CP-net.

3 Reasoning with CP-nets

While the problem of finding the most preferred outcome can be conducted in linear time in the number of nodes, the problem of dominance testing is known to be hard in general, requiring worst-case exponential time in practice. A significant aspect of this complexity is the possibility of exponentially long flipping sequences [4]. In earlier work, I observed that long sequences appeared to be rare [1]. I also reasoned that very long transitive sequences were unlikely to provide useful information about human preferences if the possibility of noise was taken into account. That is, if even some small percentage of the rules could be represented incorrectly—for example, due to an entry error during the elicitation process, then the probability that a given sequence correctly represents the decision maker's preferences diminishes to what would be expected from chance as the length of the transitive sequence increases. However, my earlier experiments relied on generation methods that did not provide i.i.d. guarantees. More recently, I have been performing additional experiments using the method of i.i.d. generation discussed above. These show that in most cases the expected flipping length is relatively close to the Hamming distance, the number of features in which the two alternatives differ. I have also shown that the average path length of the dependency graph is a good predictor of flipping sequence length.

I have also shown how to reduce general dominance testing problems to one of Boolean satisfiability so that the heuristic methods employed by modern SAT solver can be leveraged. Others have elsewhere shown how to reduce the problem to one of

planning [4], model checking [10], or (indirectly) constraint satisfaction [9]. In more recent work, I have suggested the possibility of limiting search depth based on the expected length of flipping sequence given parameters that are easy to compute.

In future work, I hope to show the effect of indifference and incomparability on flipping sequence lengths. I am also comparing the performance of different algorithms that have been proposed and considering how to combine such algorithms into a portfolio for dominance testing. Finally, I am interested in discovering better heuristic methods for dominance testing.

4 Eliciting and Learning CP-nets

CP-nets can be elicited directly or indirectly from a user or learned from observational data. Direct elicitation assumes human subjects can introspect on the cause-and-effect processes that underlie their preferences. This is a particularly strong assumption that is unlikely to be realistic in complex domains [3].

Indirect elicitation relies on the weaker assumption that subjects can answer whether one alternative is preferred to another ("I prefer the blue minivan to the red one") without providing explanations for such preferences. At the ADT-2013 conference [7], I presented a heuristic algorithm, earlier proposed by Guerin [6], for indirectly eliciting CP-nets through user queries. In that paper we assumed strict, complete preferences over binary variables and that the user could answer queries consistently.

In later work [1], I considered the problem of learning CP-nets from choice data. There I relaxed some of the customary modeling assumptions in favor of models with multi-valued variables over which the subject may be indifferent and/or inconsistent. I presented the case that such CP-nets are necessary for many simple, real-world problems. I then showed how to leverage the power of SAT solvers to learn such CP-nets from choice data. More recently, I have proposed a novel encoding for tree-shaped CP-nets that enables local search. This method allows searching for a suitable tree-shaped CP-net even if the comparison set is inconsistent due to noise. Moreover, since it is a learning- rather than an elicitation-based method, it does not depend on the subject's capacity to introspect or to respond or choose consistently. In future work, I hope to improve on this method and extend it to a richer class of CP-nets.

References

1. T. E. Allen. CP-nets with indifference. In *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on*, pages 1488–1495. IEEE, 2013.
2. T. E. Allen, J. Goldsmith, and N. Mattei. Counting, ranking, and randomly generating CP-nets. In *MPREF 2014 (AAAI-14 Workshop)*, 2014.
3. T. E. Allen, M. Chen, J. Goldsmith, N. Mattei, A. Popova, M. Regenwetter, F. Rossi, and C. Zwilling. Beyond theory and data in preference modeling: Bringing humans into the loop. In *Proc. ADT*, LNAI. Springer, 2015.

4. C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.
5. J. Goldsmith, J. Lang, M. Truszczynski, and N. Wilson. The computational complexity of dominance and consistency in CP-nets. *Journal of Artificial Intelligence Research*, 33(1):403–432, 2008.
6. J. T. Guerin. *Graphical Models for Decision Support in Academic Advising*. PhD thesis, University of Kentucky, 2012.
7. J. T. Guerin, T. E. Allen, and J. Goldsmith. Learning CP-net preferences online from user queries. In *Proc. ADT*, LNAI. Springer, 2013.
8. D. L. Kreher and D. Stinson. *Combinatorial algorithms: generation, enumeration, and search*. CRC Press, 1999. ISBN 9780849339882.
9. M. Li, Q. B. Vo, and R. Kowalczyk. Efficient heuristic approach to dominance testing in CP-nets. In *Proc. AAMAS*, pages 353–360, 2011.
10. G. R. Santhanam, S. Basu, and V. Honavar. Dominance testing via model checking. In *AAAI*, 2010.
11. B. Steinsky. Efficient coding of labeled directed acyclic graphs. *Soft Computing*, 7(5):350–356, 2003. ISSN 1432-7643. doi: 10.1007/s00500-002-0223-5. URL http://dx.doi.org/10.1007/s00500-002-0223-5.