2016

CP-nets: From Theory to Practice

Thomas E. Allen
University of Kentucky, tomallen4@gmail.com
Digital Object Identifier: http://dx.doi.org/10.13023/ETD.2016.131

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Thomas E. Allen, Student

Dr. Judy Goldsmith, Major Professor

Dr. Mirosław Truszczyński, Director of Graduate Studies

CP-NETS: FROM THEORY TO PRACTICE

DISSERTATION

A dissertation submitted in partial
fulfillment of the requirements for the
degree of Doctor of Philosophy in the
College of Engineering at the
University of Kentucky

By

Thomas E. Allen

Lexington, Kentucky

Director: Dr. Judy Goldsmith, Professor of Computer Science

Lexington, Kentucky

2016

ABSTRACT OF DISSERTATION

CP-NETS: FROM THEORY TO PRACTICE

Conditional preference networks (CP-nets) exploit the power of *ceteris paribus* rules to represent preferences over combinatorial decision domains compactly. CP-nets have much appeal. However, their study has not yet advanced sufficiently for their widespread use in real-world applications. Known algorithms for deciding dominance—whether one outcome is better than another with respect to a CP-net—require exponential time. Data for CP-nets are difficult to obtain: human subjects data over combinatorial domains are not readily available, and earlier work on random generation is also problematic. Also, much of the research on CP-nets makes strong, often unrealistic assumptions, such as that decision variables must be binary or that only strict preferences are permitted. In this thesis, I address such limitations to make CP-nets more useful. I show how: to generate CP-nets uniformly randomly; to limit search depth in dominance testing given expectations about sets of CP-nets; and to use local search for learning restricted classes of CP-nets from choice data.

KEYWORDS: artificial intelligence, combinatorial preferences, decision making, applications of local search, conditional preference networks

Author's signature: Thomas E. Allen

Date: April 29, 2016

CP-NETS: FROM THEORY TO PRACTICE

By

Thomas E. Allen

Director of Dissertation: Judy Goldsmith

Director of Graduate Studies: Mirosław Truszczyński

Date: April 29, 2016

To Susan, Luke, and Seth

ACKNOWLEDGMENTS

Five years ago, after a number of years serving in a different profession, I returned to graduate school to pursue a Ph.D. in computer science. I recognize how fortunate I am to have had such an opportunity. Many people helped make this possible—more than I can possibly list here—and I am grateful for their encouragement and support along the way. In particular, I am grateful for my advisor, Dr. Judy Goldsmith. Shortly after I began planning to pursue graduate studies, I came across some of her journal articles on computational decision making. We met for lunch at the Mellow Mushroom restaurant in Lexington. I rambled on about my "crazy idea" of pursing a career in academia, and she encouraged me to take the GRE and apply to Ph.D. programs. This was the start of a mentoring process that has continued up to the present time. I am grateful for her extraordinary support and friendship throughout this process. I am thankful also to Dr. Raphael Finkel, the Director of Graduate Studies for the Department of Computer Science at the time. I had submitted all the pieces of my application for admission except the essay explaining my goals in pursuing a Ph.D., which at that point were somewhat less than clear. He took the unusual step of contacting me by telephone and admitting me to the program after a lengthy conversation. I am grateful to the members of my Doctoral Advisory Committee. In addition to Dr. Goldsmith, who served as the director, that committee consisted of Drs. Mirek Truszczyński, Victor Marek, Clyde Holsapple, and Paul Eakin. Dr. Truszczyński, in addition to serving as our current Director of Graduate Studies, suggested the journal in which we found the article with the encoding for directed acyclic graphs that proved foundational to the method of generating CP-nets that I describe in Chapter 4; I also had the privilege of taking Dr. Truszczyński's special topics course in social networks. From Dr. Marek, I took two courses, one on databases and another on Constraint Satisfaction Programming. The latter introduced me to the use of SAT solvers, which ultimately led to the DT-SAT

reduction that I describe in Section 5.6.2. From Dr. Holsapple, I took a course on Decision Support Systems that helped motivate some of my interest in practical applications for CP-nets. Dr. Holsapple also provided constructive feedback on an earlier draft of this dissertation that resulted in many improvements. I am grateful also to Dr. Eakin, the external member of the committee, for his time and effort near the end of the process.

I am especially grateful to my coauthors. They perused my sometimes bewildering early drafts, asked hard questions and helped me to expand, clarify, and in some cases correct my ideas. Some of them wrote software or revised problematic passages. Dr. Nick Mattei of Data61 and UNSW Australia collaborated with Dr. Goldsmith and me on the workshop paper [2] that eventually led to Chapter 4. Kayla Raines and Hayden Elizabeth Justice, both of whom were undergraduate students at the time, assisted with a later version of that work that was accepted for publication at the prestigious AAAI-16 conference [5]. Cory Siler, another undergraduate student, collaborated with Dr. Goldsmith and me on an earlier draft of what is now Chapter 6. Cory also contributed much of the programming for the experiment summarized in 6.2, as well as the implementation of Lehmer codes mentioned in Section 4.2. Dr. Joshua T. Guerin of the University of Tennessee Martin kindly allowed me to assist in rewriting a chapter of his dissertation for a conference publication that was ultimately accepted for presentation at ADT 2013 [47]. That was my first academic publication and also my introduction to conditional preference networks. I am grateful to Dr. Francesca Rossi, whom we visited at the University of Padua in Italy in November 2013, and to her students, in particular Cristina Cornelio, who introduced us to PCP-nets; they provided invaluable feedback on our work. In addition, Drs. Rossi, Goldsmith, and I later collaborated on a CP-nets tutorial at AAAI-16. I am grateful to Dr. Mike Regenwetter of the University of Illinois Urbana-Champaign, and to his students, in particular Dr. Anna Popova, Muye Chen, and Chistopher Zwilling. Drs. Regenwetter, Goldsmith, and Rossi and their labs have collaborated on a human subjects study involving CP-nets [3] that has been nearing completion for the past several years. Mike also introduced me to his col-

league, Olgica Milenkovic, leading to the invitation to present my paper on CP-nets at the Allerton Conference in 2013 [1], parts of which later found there way into Chapter 5. Five of my papers—three conference papers [3, 5, 47] and two workshop papers [2, 4]—also went through a referee process, occasionally more than once. Along with my coauthors I am grateful to the anonymous reviewers for their constructive critiques.

My studies at the University of Kentucky were made possible by teaching and research assistantships and fellowships, including a grant from the National Science Foundation (CCF-121598), Graduate School Academic Year Fellowships in 2013–2014 and 2015–2016, a Verizon Fellowship in Fall 2014, and the Thaddeus B. Curtz Memorial Scholarship Award in 2012. I am also grateful to the citizens of the Commonwealth of Kentucky, where I have lived since 2004, for their support of higher education.

I am deeply appreciative also of my family and friends for their support in this endeavor. I still remember the considerable satisfaction of my father, Dr. T. Eugene Allen III, when he completed his Ph.D. in political science when I was a small child. He and my mother, Ann Lyn Allen, who was also my high school mathematics teacher, met in a statistics class in graduate school. They instilled in me a high regard for the value of education. My siblings, Dr. Martha Allen-Dietrich of Georgia College and State University, and Dr. Timothy E. Allen of the University of Virginia, have also offered guidance and encouragement during my transition into academic life. Finally, and most especially, I am grateful to Susan, my wife, and to my sons, Luke and Seth, for letting me devote so many years to such a challenging and transformative process. My appreciation for them is beyond words, and to them this work is lovingly dedicated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 Introduction

This dissertation involves models of computational preferences in the field of artificial intelligence—in particular, a class of models known as conditional preference networks (CP-nets). Increasingly, through advances in artificial intelligence, we entrust the details of our lives to machines. In the near future it seems likely that autonomous vehicles will deliver our packages and chauffeur us. Smart homes and other assistive technologies will provide for the elderly, the disabled, and the young, as well as many of us who are usually quite capable of caring for ourselves, but who nevertheless prefer not to do our own cooking, cleaning, and maintenance. Already, various mobile applications are helping plan our schedules, suggesting activities, and influencing our decisions and social interactions on a wide scale through such innovations as recommender systems and AI-based decision support systems.

Of course, there is more than one way that such a future could play out. Dystopian science fiction novels and films continually remind us that this could all be a Faustian bargain. Already, many of us wonder if our smart phones, fitness bands, and various other clanging, buzzing devices have in fact changed our lives for the better. On the other hand, one can envision a more promising future, in which disabled people can participate in society more equitably and in which all of us can focus our energies on the pursuits and relationships that we value most, leaving the frustrating drudgery to machines.

For this second, brighter future to be possible, the machines must have some way of knowing what we want. Suppose I step into my autonomous vehicle in a few years. What type of route do I prefer, the scenic route or one that is more direct? Would my answer always be the same? Presumably, it would sometimes differ depending on various conditions. If my schedule that day were busy, I may prefer the more direct route. On the other hand, if the car were taking care of the driving, perhaps a more peaceful route would

allow me complete my presentation, or at least arrive at my destination less harried, more prepared to meet the challenges at hand.

Computational models such as CP-nets provide a way to represent such preferences. A machine can then use such a representation—a mathematical model—to reason about what I would prefer under various circumstances. The machine could then recommend alternatives to me, or even act as my proxy, making decisions on my behalf. *I'm sorry Tom. I'm afraid I can't do that. Yes, the scenic route would be nice, but—don't forget—you have a class to teach!* As we will see, CP-nets have much to offer. On the other hand, matters involving computational complexity have thus far stymied the integration of CP-net models and algorithms into actual applications. This thesis addresses some of those problems so that machines of the near future can better understand our preferences and adapt to us.

Section 1.1 offers a brief introduction to how one can go about modeling preferences mathematically. Section 1.2 discusses the major computational problems that go along with such models. Section 1.3 introduces our primary topic in this dissertation, conditional preference networks. Section 1.4 discusses some of the challenges to the adoption of CP-nets. An overview of subsequent chapters follows.

1.1 Preferences

Modeling, capturing, and reasoning with preferences is a fundamental topic that spans artificial intelligence, including constraint programming [90], social choice [22], recommendation systems [85], machine learning [38], and multi-agent systems [41]. Preferences have also been studied in philosophy, economics, psychology, and other disciplines. Let us consider two example applications that motivate our later discussions.

Example 1. *Consider that a foodservice distributor has created a decision support system for its salespersons. The system advises representatives on when to contact food service operators, along with questions to ask, issues to address, and products to recommend. Among*

the many features of the system is one that tracks the preferences of each customer. What items does the customer buy? When do they buy each item? Which items are purchased together? If a particular item is not available, what else does that affect in the order? Such data are mined to construct a profile that can support the activity of the sales representative and increase the customer's satisfaction. For example, the system may have learned that when a certain pastry chef buys pecans, she also increases her order of brown sugar. It may also have learned that she prefers a particular variety of pecans (Pawnee) to others (e.g., Schley). Thus, if more brown sugar has been ordered than usual, the representative may be prompted to ask about pecans and specifically to mention the Pawnee variety if these are in stock.

Example 2. *Next consider that a team of engineers has designed a home automation system to provide support to persons of advanced age who wish to continue living independently. The system enforces a set of (hard) constraints, such as that indoor temperature must be within a range commensurate with human health and that rooms must be sufficiently well-lit when the subject is moving through the house. Aside from these, however, the system allows the subject maximal determination over his environment. That is, subject to constraints, control of the home is governed by the subject's preferences. We expect that such preferences will sometimes be conditional; for example, the subject may prefer to converse by video with a friend on a particular night of the week, but play a favorite video game on some other night.*

Systems such as these require some way to model, learn, reason with, and perhaps aggregate preferences. Preferences involve at least one *subject*, sometimes known as the *user* or *decision maker*, and a set of *objects* $\mathcal{O}$, known as *candidates*, *outcomes*, or *alternatives*, depending on the context. Formal definitions will follow in Chapter 2, but for now, we can say informally that a subject *prefers* the first object (o) to the second (o') if the first is "better" or makes her "happier" or "more satisfied" than the other in a given setting. Symbolically, one can write this as $o \succ o'$. Such expressions of course also have a dual

form, because we can just as easily say that the second object is "worse" or makes her less happy, which one can write as $o' \prec o$.

1.1.1 Utility, Strict Preference, and Indifference

A common assumption in economics is that the subject associates with each object a real-valued *utility* that depends on the value or happiness that the object provides. Under this assumption, $o \succ o'$ suggests that the utility of the first object exceeds that of the second; that is, $u(o) > u(o')$. A natural idea, then, would be to model this utility function $u : \mathcal{O} \to \mathbb{R}$ explicitly—either the value of each object to the subject or, perhaps even better, what the subject's utilities *should* be if she had perfect information. In many settings, however, it is difficult to assign numerical values even when the preference is apparent. Suppose a subject would like to play the board game Monopoly with a friend. All things being equal, she prefers to play this game with Sarah rather than Tamara. However, she may be unable to *quantify* just how much she prefers Sarah to Tamara in this context. She may find it difficult or discomfiting to assign values to her two friends. As observed in Section 1.2.2, it is in fact possible to use utility functions to model preferences. Throughout most of this work, however, preferences are modeled *qualitatively*, leaving the underlying utilities implicit, something for economists to ponder.

Sometimes a subject may be equally happy with two objects. In that case, we say that the subject is *indifferent* as to the two and write $o \sim o'$. Note that this does not necessarily imply that the two objects are the same (in which case one could write $o = o'$) or that he is unable to distinguish the two. When a friend tells us that he is equally happy with fair weather and snow, we do not generally assume that he is unable to distinguish the two. On the other hand, if two objects really *are* the same, we will assume that the subject is indifferent; i.e., $o = o' \implies o \sim o'$.

The possibility of indifference allows us to speak of *weak preferences*, in contrast to those that are *strict* (or *strong*). For example, a subject may say that one object is "at

least as good as" the second, which one can write, $o \gtrsim o'$. In this case the subject may *either* strictly prefer the first object ($o \succ o'$) *or* may be indifferent ($o \sim o'$). Additional information would be needed to determine which of these is the case. If we were to later learn that the subject regards the second object as at least as good as the first ($o' \gtrsim o$), we could then reason that the subject is indifferent as to the two objects, assuming the subject's preferences are expressed in a consistent way.

1.1.2 Incomparability, Incompleteness, and Missing Information

In some cases, a subject may find it impossible to compare two objects. When this occurs, we refer to the two objects as *incomparable* and write $o \parallel o'$. Incomparability can occur when two objects are vividly different or when some multicriteria decision is involved. For example, if each of two candidates in a political election has one quality that a voter admires and one quality she despises, she may find it difficult to compare the two. Incomparability can also occur when the *subject* lacks information *about the objects*. For example, a diner perusing a menu in some foreign language that he hardly understands may be unable to compare various items on the menu. This does not mean, however, that he is equally happy with all of the items. In other words, incomparability is *not* the same as indifference; it simply means that the subject is unable to state a preference.

It is important, however, to distinguish a lack of information by the *subject* from the lack of information of an *observer*, for example, an artificial *agent*[1] that is assisting the subject through recommendations [80]. Incomparability and missing information are thus related, but not identical. If we, as an outside observer, *know* that the subject is unable to compare two objects because the *subject* has a lack of information, then we may say that the subject finds the two objects incomparable. However, *if we simply do not know* the subject's preferences, then we cannot say with any certainty that the subject finds the two objects incomparable. He may prefer one to the other or be indifferent. This can occur

[1] In this work, except where otherwise noted, *agent* refers to an artificial intelligence application.

when we have not yet asked the subject about his preferences, when he has declined to reveal them to us because of a lack of trust, or when he simply has never thought about his preference over this pair of objects. In such cases one may write $o \; ? \; o'$.

A related situation arises when the subject does in fact have a preference, but the model, while consistent with that preference, does not include it in the representation. For example, consider that a subject prefers $o \succ o'$, $o \succ o''$, and $o' \succ o''$. Suppose further than we model this preferences $o \succ o' \wedge o \succ o''$, but omit the relationship between o' and o'' from the model—perhaps out of a desire for a more succinct representation. Observe that, mathematically, the model is a *partially ordered set* (poset) of objects, while the subject's true preferences are a *linear extension* of that poset (see Figure 1.1). In this case, then, incomparability arises not from the subject herself, nor from the observer's knowledge of the subject, but from how the preferences are modeled.

(a) True preference (b) Preference model

Figure 1.1: A Subject's Preference Order and a Model Consistent with that Order

Thus, in discussions about preferences, *incompleteness* has different meanings. However, it also has an unambiguous mathematical meaning: We say that a preference relation is *incomplete* if any two objects in the set are incomparable; otherwise, the relation is *complete*. Most often in this work, we will reserve the words *complete* and *incomplete* for situations where we have in mind the mathematical concept.

1.1.3 Transitivity and Inconsistent Preferences

Ordinarily we regard preferences as *transitive*. That is, if there are three objects, and the subject prefers the first to the second and the second to the third, then we can infer that the subject also prefers the first to the third, even if we have not asked her explicitly to compare

these two objects *directly*. That is, we say that $o \succ o' \wedge o' \succ o'' \implies o \succ o''$, and may write $o \succ o' \succ o''$ to emphasize this. Moreover, if transitivity is violated—if a subject tells us that he strictly prefers the first object to the second, the second to the third, and the third to the first—we say that the subject's preferences contain a *cycle* (see Figure 1.2). Through the transitive property, we can see that each object is strictly preferred to itself. We call such preferences *inconsistent* and may reason that the one who holds them is *irrational*.

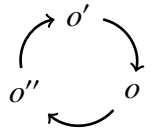


Figure 1.2: A Preference Cycle

Customarily, we also regard indifference as transitive. That is, $o \sim o' \wedge o' \sim o'' \implies o \sim o''$; hence we may write $o \sim o' \sim o''$, chaining the $\sim$ operator as we do for $\succ$. However, in human preferences this assumption does not always hold. Consider the example, cited by Peter C. Fishburn [36], of a person's preferences over the amount of sugar in coffee (see Figure 1.3). Suppose that the subject is accustomed to coffee with no sugar. Nonetheless, if asked in a taste test to choose between a cup of coffee without sugar and one with only 1 grain of sugar, we expect that she would be indifferent. Similarly, she would be indifferent to a choice between a cup with 1 grain and one with 2 grains; the difference would again be imperceptible. However, given a choice between a cup of coffee with no sugar and a cup with *ten spoonfuls of sugar*, it is unlikely that she would still be indifferent! Formally, $0 \sim 1 \sim 2 \sim \cdots \sim 1000 \not\Rightarrow 0 \sim 1000$. In most of the discussion that follows, however, we will assume that transitivity holds for indifference (and hence also weak preferences) as well as strict preferences.

On the other hand, incomparability, in the sense that it is used in this work, is *not* transitive. Consider a café owner who strictly prefers *oranges* to *bananas*, but has never heard of *durians*, a fruit native to southeast Asia. Because he has never before encountered this particular fruit, he cannot compare it to either *oranges* or *bananas*. Thus, *oranges* ||

Figure 1.3: Intransitive Indifference

durians and *durians* || *bananas*, but these facts *do not imply* that *oranges* || *bananas*. In fact, in this case we have already established that the subject has a strict preference: *oranges* $\succ$ *bananas*. Thus, one should not write expressions of the form $o \parallel o' \parallel o''$ because this incorrectly suggests transitivity, which in general does not hold for incomparability.

1.1.4 Factored Outcomes and *Ceteris Paribus* Preferences

Throughout this work, the objects over which a subject holds preferences will be characterized by *features* (or *attributes*). Example 1 mentions a pastry chef who sometimes purchases pecans. Pecans can be characterized by various features, such as VARIETY (e.g., *Pawnee* or *Schley*) and whether they have already been SHELLED (*shelled* or *unshelled*). When objects are *factored* in this way, they are typically called *outcomes*, because they are the outcome of how their characteristic features have been instantiated. For clarity, such features may be written in SMALLCAPS, with their associated values in *italics*.

When outcomes are factored, a subject may hold *ceteris paribus* preferences *over the features*. When the chef says that she prefers *Pawnee* pecans, she does not necessarily mean that she prefers *every* Pawnee order to every Schley. Other factors may also affect her happiness with the order, such as price, quality, expected date of delivery, and so on. However, *if all other factors are held constant*, she prefers the Pawnee to other varieties. All else being equal (Latin *ceteris paribus*), she prefers one variety to another.

Ceteris paribus preferences can be *conditional* or *unconditional*. Suppose the buyer does in fact *always* prefer the Pawnee variety to the Schley. In that case, the preference does not depend on any other factor, so it is said to be *unconditional*. One can write this preference as *Pawnee* $\succ$ *Schley*. Note that the notation here is identical to that used for

preferences over outcomes. Context, however, makes it clear that the preference is of the *ceteris paribus* type, because *Pawnee* and *Schley* describe one *attribute* of a pecan rather than fully instantiated outcomes.

Often a preference *does* depend on the value of additional other features. Suppose the chef prefers that pecans be *shelled* prior to shipping if they are *Pawnee*, but shipped *unshelled* if they are *Schley*. In this case the *ceteris paribus* preferences are *conditional*. One can write this as *Pawnee* : *shelled* $\succ$ *unshelled* and *Schley* : *unshelled* $\succ$ *shelled*, where the : operator indicates the dependency.

1.2 Preferences over Combinatorial Domains

In Sections 1.1.1–1.1.3 we considered how to model a subject's preferences over a set of objects as a mathematical relation.[2] In Section 1.1.4 we considered that the objects could be multi-featured outcomes. Features complicate things. Consider an assistive robot that must visit a deli and purchase lunch for a client. The client has a busy schedule and cannot be contacted during the process. The deli offers a *combinatorial* number of alternatives. That is, customers have a choice of breads, meats, cheeses, vegetables, condiments, and so on. Of all possible sandwiches that can be assembled, which does the busy client prefer most? Suppose the client has asked for a tuna salad sandwich with only cucumber and tomato, but no tuna is available that day. What is her next best alternative? If turkey is selected instead of the tuna, will she still prefer cucumber and tomato, or some different set of toppings? Suppose further that the robot does not have the luxury of stopping by a deli, but must choose instead from a vendor who offers a set of preassembled, wrapped sandwiches. Further suppose that the offering of sandwiches varies from day to day. Given today's choices, which one will the client prefer most? Note that the client may also have specific constraints—e.g., religious requirements, or a minimum or maximum number of calories, or some maximum amount that she is willing to spend on a sandwich.

[2] Henceforth, when we speak of preferences, we will assume that they are those of a particular subject.

A variety of methods have been proposed for modeling preferences over *combinatorial domains*. We first discuss two *non-compact* representations, the *preference graph* and the *partial order graph*, because these inform our later discussions (Section 1.2.1). We then consider some examples of *compact models* such as GAI value functions and soft constraints (Section 1.2.2). In Section 1.2.3 we introduce some of the more common problems in preference handling: learning preferences, finding most preferred outcomes, reasoning with preference models, and aggregating preferences.

1.2.1 Non-compact Representations

If the objects over which a subject holds preferences are conceived as *factored outcomes* (see Section 1.1.4) then one can observe that the number of these outcomes will be exponential in the number of features. Consider some foodservice product from Example 1 that can be fully described by 10 binary features (e.g., type, color, brand, gluten-free, etc.). The total number of outcomes is then $2^{10} = 1024$. If the number of features is increased to 20, the number of outcomes is 2^{20}, greater than one million. This does not mean, of course, that each such outcome is available for purchase or even that it presently exists in the physical world. Nonetheless, it is possible to *conceive* of each outcome and thus plausible that a subject may hold preferences over it. Indeed, if customers prefer an object that does not yet exist, this is valuable information.

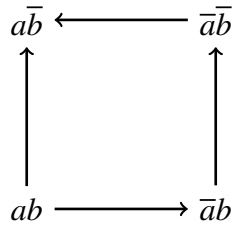
We can conceive of a matrix or graph that represents a subject's preferences over *every* pair of outcomes (see Figure 1.4a). In the example, the entry 1 in the cell in row $a\overline{b}$ and column ab indicates that $a\overline{b} \succ ab$; the entry 0 in row ab and column $a\overline{b}$ indicates that $ab \not\succ a\overline{b}$. If outcomes are labeled with n binary features, there would be 2^n outcomes in all, requiring a matrix with 2^{2n} entries. If we limit ourselves to strict preferences, then we reduce the number of such entries by about half: because $o \succ o'$ implies $o' \not\succ o$ and $o \not\succ o$ for all i, we can limit our attention to the cells of the upper triangular representing distinct unordered pairs of outcomes (as in Figure 1.4b).

$\succ$	ab	$a\overline{b}$	$\overline{a}b$	$\overline{a}\overline{b}$
ab	0	0	0	0
$a\overline{b}$	1	0	1	1
$\overline{a}b$	1	0	0	0
$\overline{a}\overline{b}$	1	0	1	0

(a) Full Preference Relation

$\succ$	ab	$a\overline{b}$	$\overline{a}b$	$\overline{a}\overline{b}$
ab	-	0	0	0
$a\overline{b}$	-	-	1	1
$\overline{a}b$	-	-	-	0
$\overline{a}\overline{b}$	-	-	-	-

(b) Full Strict Relation



(c) Preference Graph

Figure 1.4: Non-compact Representations of the Same Strict Preference Relation

If the preferences can be modeled consistently by *ceteris paribus* rules, as introduced in Section 1.1.4, we can reduce the number of entries even further. In that case, it is sufficient to represent only the relationship between pairs of outcomes that differ in the value of *just one* variable. By exploiting transitivity in this way, we can represent a set of preferences as a graph in which each *vertex* represents a conceivable outcome. A directed edge from one vertex to another means that the second vertex is strictly better than the first (or weakly better if indifference is allowed). Furthermore, such an edge can exist only if the two vertices differ in the value of just one feature. To compare outcomes that differ in more than one feature, we check for a path connecting the two along directed edges. For example, in Figure 1.4c, the path from $\overline{a}b$ to $a\overline{b}$ indicates that $a\overline{b} \succ \overline{a}b$. Such a graph is known as a *preference graph* [16] or sometimes as the *outcome graph*. Note that the preference graph takes the geometric shape of a *hypercube* if all of the features are binary.

Another noncompact representation is a type of polytope known as the *partial order graph* [83]. In this representation, vertices represent all (or some subset of) *strict partial orders over objects* rather than the objects themselves. Such objects are not necessarily factored outcomes. Moreover, it is possible also to represent non-strict and even inconsistent orders, such as $o \succ o' \succ o'' \succ o$ (see Figure 1.2), as vertices.

1.2.2 Compact Preference Formalisms

The representations discussed in Section 1.2.1 require polynomial space in the number of outcomes, which is exponential in the number of features n. If n is large, such a rep-

resentation is infeasible. We are thus interested in *compact models* for which the space requirement is polynomially bounded in the number of features. Various *compact preference formalisms* have thus been designed that leverage closed-form functions or logical rules to enable models that scale as the number of features increases. Compact preference formalisms are not without their drawbacks. Observe that there are more explicit representation instances than there are compact ones for an order over n objects, so not all explicit representations can be represented compactly. Moreover, many graph problems that can be solved in polynomial time in an exponentially large graph turn out to be PSPACE-complete in a graph that compactly represents the exponentially larger original graph [9, 42, 61].

General additive independence (GAI) value functions [8, 35, 44] assign numerical value to the *utility* of each outcome to the subject as expected by the agent. In general such functions are *not* compact, but GAI value functions leverage a property known as *additive independence* to enable the utility of an outcome to be computed efficiently by summing a series of separate utility functions, one for each feature. If the total added utility of the first outcome is greater than that of the second, one can infer from the model that $o \succ o'$. However, not all utility functions are additive. Moreover, as discussed in Section 1.1.1, it is not always clear how to assign numerical values to human preferences.

Preferences can also be modeled as a constraint satisfaction program (CSP) through methods that employ *soft constraints*. In Example 2 it was observed that a home automation system would likely enforce hard constraints, such as that indoor temperature must not get so low as to let pipes burst. Aside from such hard constraints, the system also takes into account the preferences of the resident. Perhaps he prefers 70 °F rather than 65 °F indoors. Such a preference can also be modeled as a constraint. In contrast to the requirement that the temperature *must* not be allowed to drop below a certain threshold, however, the desire for a comfortable room would be modeled as a *soft* constraint—something to be optimized with a *constraint solver* along with other features that affect the resident's comfort, but not something the system must achieve under all circumstances or report failure. A number of

soft constraint formalisms exist, such as fuzzy, weighted, and probabilistic constraints. A more general method known as *semiring-based soft constraints* encompasses most of these other approaches and has specifically been applied to model and work with preferences [13, 14, 90]. However, this method requires finding a particular semiring value for each variable assignment in each constraint, again introducing the problem of *quantifying* preferences that often can be expressed more naturally in a *qualitative* way.

1.2.3 Common Problems in Preference Handling

Certain problems are inherent in an application such as those described in our opening examples. First, the system needs some way to obtain a model of the subject's preferences. The possibilities for **learning** this include:

- **Direct construction.** The subject or a human expert working closely with the subject explicitly constructs a preference model based on the subject's introspection. This approach is problematic for several reasons. It requires a significant amount of knowledge about the mathematical model. Also, the subject may not have time for this, and if an outside consultant is hired, the cost would likely be prohibitive except for high-value domains. Moreover, repeated psychological studies have shown that human beings cannot reliably introspect on our own preferences [3, 77, 78, 103].

- **Active elicitation.** An *agent* (computer system) poses a series of *queries* to the subject. A model is then inferred from the subject's replies. An advantage of this approach is that a model of the subject's preferences is available to the system from the outset. A disadvantage is that the process of answering repeated queries can be tedious. Moreover, if this process is shortened or terminated early (e.g., by a frustrated user), the resulting model may not adequately reflect the subject's preferences.

- **Passive learning.** A third approach is to observe what a subject does over time. This is the approach envisioned in Example 1, where we consider that a model of a chef's

preferences over foodservice items could be learned from customer order data. An advantage of this approach is that it does not demand anything of the subject at the outset; the user can begin using the system immediately. The disadvantage is that weeks or months may be required before the system can learn a suitable model.

Hybrid approaches to learning are also possible. For example, a system could initialize the preference model by posing only a few queries to the subject and then refine this model over the succeeding months using the passive learning approach. In some settings, it may be advisable to consider that a subject's preferences may change over time. Such a model would need some way of continually learning new information about the subject's preferences, while simultaneously "forgetting" outdated information, particularly if it were found to conflict with more recently observed preferences.

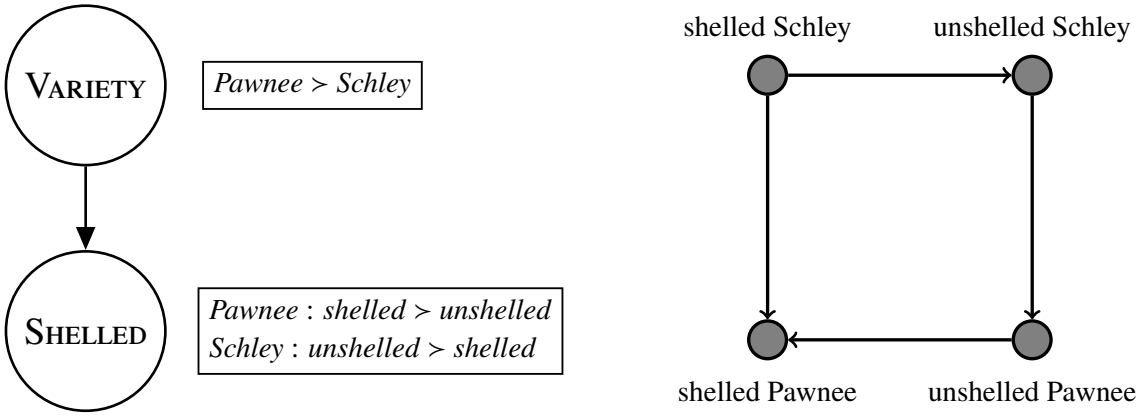
Once a model is available, algorithms are required to draw inferences from the model about the subject's preferences. Common problems of this sort include:

- **Optimization.** What is the (a) *most* (or *least*) preferred outcome?[3] Similarly, what are the *k-best* (*worst*) outcomes?

- **Reasoning.** Given some pair of outcomes that the subject may not have considered previously, which (if either) is the subject likely to prefer?

- **Aggregation.** If preference models are available for more than one subject, one becomes interested in whether the preferences can be aggregated to produce outcomes for the group that meet some criteria of optimality (e.g., Pareto efficiency). As such, preference aggregation is closely related to *social choice* topics such as *voting*.

[3]This assumes, of course, that optimal outcomes are defined with respect to the model, which turns out to be the case with CP-nets.



(a) Conditional preference network (b) Preference graph

Figure 1.5: CP-net and Induced Preference Graph

1.3 Conditional Preference Networks

We now turn to the preference formalism that will be our focus in the discussions that follow, the **conditional preference network** (CP-net). First proposed by Boutilier et al. [16], CP-nets exploit conditional *ceteris paribus* preference rules (1.1.4) to enable a compact representation of the preference graph (1.2.1). We define CP-nets formally in Chapter 2, but at this point it is helpful to introduce them with examples corresponding to the applications in Examples 1 and 2 at the beginning of the chapter.

The CP-net in Figure 1.5a models the preferences described in Section 1.1.4. Recall that the chef prefers the Pawnee variety of pecans to the Schley. If the pecan does happen to be a Pawnee, she prefers that it be delivered shelled; otherwise, she prefers the pecans unshelled. The nodes in Figure 1.5a represent the features over which the subject holds preferences. The directed edge from VARIETY to SHELLED indicates that the subject's *ceteris paribus* preference for whether the pecan is shelled or unshelled depends on the variety. We refer to VARIETY in this case as the *parent* of SHELLED. In contrast, the preference over VARIETY is unconditional, so that node has no parents. The boxes beside each node are *conditional preference tables* (CPTs) specifying the *ceteris paribus rules* over each node given the values of all combinations of values of the parent nodes.

The CP-net in Figure 1.5a *induces* the preference graph shown in Figure 1.5b. Each rule in the CP-net induces a non-empty set of edges in the preference graph. For example, the rule

$$\textit{Schley} : \textit{unshelled} \succ \textit{shelled} \tag{1.1}$$

corresponds to the directed edge (*shelled Schley*, *unshelled Schley*) in the preference graph, and the rule

$$\textit{Pawnee} \succ \textit{Schley} \tag{1.2}$$

corresponds to the edges (*shelled Schley*, *shelled Pawnee*) and (*unshelled Schley*, *unshelled Pawnee*).

The *ceteris paribus* rules, by their nature, specify preferences for outcomes that differ in just one feature. The transitive closure of these rules sometimes allows us to compare outcomes that differ in more than one feature. For example, suppose only two items are in stock, *unshelled Pawnee* and *shelled Schley*. In that case, we anticipate that the pastry chef will prefer the unshelled Pawnee: Equation 1.1 entails that shelled Schley is less preferred than unshelled Schley, and Equation 1.2 entails that unshelled Schley is less preferred than unshelled Pawnee. Thus, we have:

$$\textit{shelled Schley} \prec \textit{unshelled Schley} \prec \textit{unshelled Pawnee}. \tag{1.3}$$

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

Now consider a slightly more complex CP-net corresponding to Example 2. Suppose our automated home has windows with blinds that the system can open and close automatically, as well as sensors that report weather conditions in realtime. Suppose also that the subject prefers that the blinds be open during the day and closed at night, with certain exceptions. For example, he prefers the blinds always be open when it is snowing. He prefers both snow and fair weather to rain. The subject's preferences as described can be modeled with the CP-net in Figure 1.6. In terms of this simple model, three features contribute to

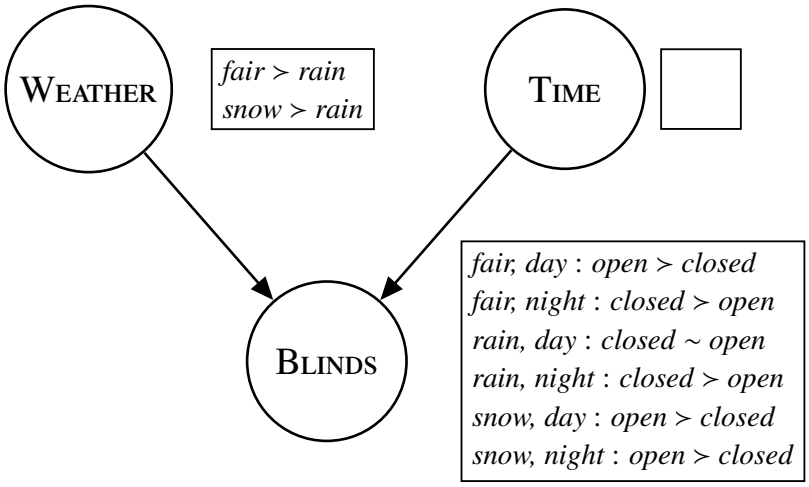


Figure 1.6: A CP-net with Indifference over Multivalued Domains

the subject's happiness—weather, time of day, and the state of the blinds—with 12 possible outcomes in all (fair day with blinds closed, rainy night with blinds closed, etc.). Such features are modeled as variables with discrete domains. For example, the variable WEATHER has a multivalued domain consisting of *fair*, *rain* and *snow*, while the other two variables are binary. The edges from WEATHER and TIME to BLINDS indicate that the preference over BLINDS depends on these other two features. Note that the subject's preference over WEATHER is *unconditional* because it does not depend on any other factor. Moreover, the CPT for TIME is empty, because we have no information on the subject's preferences for day versus night. In this particular example, we model lack of information as *incomparability*. Finally, observe that on a rainy day, the resident of the smart home is equally happy with the blinds open or closed, expressed here in the form of a rule specifying indifference.

Again, care should be taken to distinguish incomparability from *indifference*. While we are told that the subject is equally satisfied with open or closed blinds on a rainy day, we should not assume, in the absence of additional information, that he is also equally happy with day and night. He may well prefer one to the other. Moreover, indifference here is transitive, while incomparability is not. Finally, it is worth noting that the preferences here are modeled *deterministically* rather than as probability distributions.

1.4 CP-nets: Challenges to Adoption

Each method of modeling preferences—CP-nets, GAI functions, soft constraints, etc.—has its strengths, weaknesses, and unique characteristics. The choice of a preference modeling language for an application may be compared to an engineer's choice of a programming language or software application; various factors such as the specifics of the project and the practitioner's familiarity with the available tools may influence this decision.

There is much to like about CP-nets. They let us concisely 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 efficiently (in time linear in the number of features) if the graph is free of cycles and CPTs are complete.

On the other hand, while many academic papers discuss CP-nets (over 800 to date, according to Google Scholar) and many interesting applications have been proposed—automated negotiation [7], interest-matching in social networks [102], cybersecurity [15], and as aggregation primitives for making group decisions [63, 74, 104], among others—we are not yet aware of their use in real-world applications. There are several reasons for this. First, as we discuss in Section 3.4, determining dominance—whether one arbitrary

outcome is better than another with respect to a CP-net—is known to be computationally hard in many cases. This is significant, since one doubts that, say, a decision support system implementation would be particularly useful if it sometimes required several days to determine whether a customer preferred one item that was in stock over another! Second, the study of how to learn CP-nets is still in its relative infancy. Some algorithms have been proposed. However, as we discuss in Section 3.5, the proposals to date either make unrealistic assumptions or rely on methods that do not scale to networks of realistic size. Third, while academic papers do sometimes evaluate their proposed learning or reasoning algorithms experimentally, the methods for these evaluations turn out to be problematic. As we discuss in Section 3.6, human subjects data for CP-nets at present is non-existent, and the situation with preference data over multi-feature domains is hardly any better. Experiments using synthetic datasets have been equally problematic, because they have relied on naïve eneration methods that suffer from statistical bias.

This thesis addresses several of these limitations in an effort to make CP-nets more useful and to further their adoption in engineering applications. Chapter 2 consists of formal definitions. In Chapter 3, I discuss the work of previous researchers to provide an overview of the state of the art for CP-nets research. In Chapter 4, I show how to encode, count, and generate CP-nets uniformly at random. Because the computational time for determining dominance depends on flipping length, in Chapter 5 I use the generation method to study the expected flipping length of a dominance testing problem given certain parameters that are easy to compute and show how to use this expectation to limit search depth in certain cases. In Chapter 6, I show how to use local search to learn tree-shaped CP-nets from choice data. A concluding chapter summarizes contributions and some interesting possibilities for future research.

Chapter 2 Definitions

Preferences were introduced informally in Chapter 1. The present chapter is a more formal introduction of the terms, notation, and concepts used in the research that is discussed in subsequent chapters. Section 2.1 reviews ordered sets. Section 2.2 reviews graph theoretic concepts and classes of digraphs commonly encountered in working with CP-nets. Section 2.3 introduces notation for outcomes over multi-featured domains. Section 2.4 formalizes preferences on factored outcomes, CP-nets, and related concepts such as dominance testing and flipping lengths. Section 2.5 concludes with tables of commonly used acronyms and notation.

2.1 Ordered Sets

The reader is presumed to be familiar with elementary set, order, and graph theoretic concepts. Those less familiar with such topics are referred to a textbook, such as that of Rosen [88]. Nonetheless, since terminology and notation tend to differ among communities,[1] a brief review is in order.

Preferences as considered in this work involve ordered finite sets. A *linear order* here refers to a strict total order on a set, i.e., an irreflexive, antisymmetric, transitive, total binary relation. Thus, if $\blacktriangleright$ is a linear order on S, then, for all $a \in S$, $b \in S$, just one of the following is true: $a \blacktriangleright b$, $b \blacktriangleright a$, or $a = b$. The expression $a \blacktriangleright b$ is read, "a is ordered before b," and in this case $a = b$ is read, "a is the same as b." Note that the $=$ operator here indicates that a and b refer to the same element; it should not be confused with $\sim$, discussed below. A set thus ordered is known as a *ranking*. We denote the set of all such rankings (the permutations or *symmetric group*) of a finite set S as $\mathfrak{S}(S)$.

[1] "You keep using that word. I do not think it means what you think it means." –Inigo Montoya

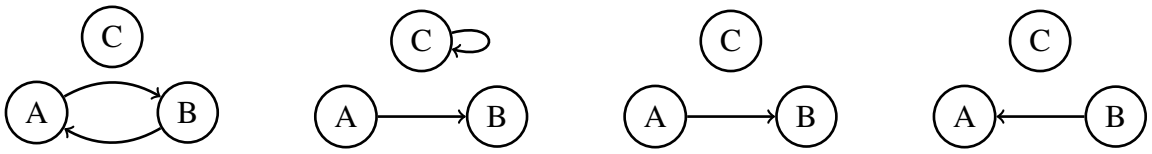
A *partial order* differs from a linear order in that the relation is not total; that is, some elements in the set may be *incomparable*. Thus, if $\triangleright$ is a partial order on S, then, for all $a, b \in S$, just one of the following is true: $a \blacktriangleright b$, $b \blacktriangleright a$, $a = b$, or $a \parallel b$. The expression $a \parallel b$ is read, "a cannot be compared to b." (Other authors sometimes use $\bowtie$ or $\times$ to denote incomparability.) A set thus ordered is a *partially ordered set* or *poset*. Note that every ranking is also a poset. If a poset is not a ranking, the ordering is said to be *incomplete*.

A linear order $\blacktriangleright$ on S is said to be a *linear extension* or *linearization* of a partial order $\triangleright$ on S if and only if $(a, b) \in \triangleright \implies (a, b) \in \blacktriangleright$ for all $a \in S$, $b \in S$. That is, if a is ordered before b in the poset, then a must also be ordered before b in the linear extension. However, if $a \parallel b$ in the poset, then it must either be the case that $a \blacktriangleright b$ or $b \blacktriangleright a$ in the linear extension. For example, let $S = \{a, b, c\}$ and let $\triangleright = \{(a, c), (b, c)\}$ be a partial order on S; i.e., $a \blacktriangleright c$, $b \blacktriangleright c$, and $a \parallel b$. Then (the only) two linear extensions of $\triangleright$ are $a \blacktriangleright b \blacktriangleright c$ and $b \blacktriangleright a \blacktriangleright c$. A ranking that extends a poset in this manner is also said to be *compatible* with that poset. Note that an infix ordering operator such as $\blacktriangleright$ can be chained, e.g., $a \blacktriangleright b \blacktriangleright c$, only for rankings; $\blacktriangleright$ is not chained for an order that may be incomplete.

A *preorder* is a reflexive, transitive binary relation. Informally, a preordered set differs from a poset in that "ties" are allowed between pairs of distinct elements. That is, if $\trianglerighteq$ is a preorder on S, then, for all $a, b \in S$, just one of the following is true: $a \triangleright b$, $b \triangleright a$, $a \sim b$, or $a \parallel b$, where $\sim$ is read, "a is ordered equally with b." For a preorder, $a = b \implies a \sim b$, but the converse does not hold. Note that every poset is also a preordered set, and every preordered set is also *consistent*, i.e., closed under transitivity. If a binary relation on a set is intransitive, it is said to be *inconsistent*.

2.2 Graphs

A *directed graph* (digraph) is a pair $G = (V, E)$ in which V is a set of *nodes* (also known as vertices) and E is a set of *directed edges* (or arcs). When no confusion can result, we may drop the qualifier and refer to a directed edge simply as an *edge*. Each edge $(u, v) \in E$



(a) Cycle (b) Loop (c) Edge A to B (d) Edge B to A

Figure 2.1: Labeled Digraphs

consists of an ordered pair of nodes, $u \in V$, $v \in V$. While digraphs may have cycles, such as that in Figure 2.1a, the graphs in this work are free of *loops* such as that shown in Figure 2.1b; thus, we assume $u \neq v$ for all $(u, v) \in E$. Throughout this work, we also assume all graphs are *labeled*; that is, the digraph of Figure 2.1c is distinguished from that of Figure 2.1d. An *undirected* graph differs from a digraph in that E is composed of *unordered* pairs of nodes $\{u, v\}$. With the exception of Chapter 6 or as otherwise noted, the graphs in this work are assumed to be directed.

Let $G = (V, E)$ be a digraph and $v \in V$ an arbitrary node in G. If there exists a node u such that $(u, v) \in E$, then u is said to be a *parent* of v. Formally, the parents of v are defined as: $\mathrm{Pa}(v) = \{u : (u, v) \in E\}$. If there exists a node u such that $(v, u) \in E$, then u is said to be a *child* of v. Formally, the children of v are defined as: $\mathrm{Ch}(v) = \{u : (v, u) \in E\}$. The *indegree* of a node v is its number of parents $|\mathrm{Pa}(v)|$ and the *outdegree* is its number of children $|\mathrm{Ch}(v)|$. A set of digraphs $\mathcal{G}$ on V is said to have *bounded indegree* c if no node in any digraph in the set has more than c parents; i.e., $|\mathrm{Pa}(v)| \leq c$ for all $v \in V$ for all $G \in \mathcal{G}$.

A *path* from s to t is a sequence of edges, $\langle (u_0, u_1), (u_1, u_2), \ldots, (u_{\ell-1}, u_\ell) \rangle$, where $u_0 = s$ and $u_\ell = t$, such that $(u_k, u_{k+1}) \in E$ and $0 \leq k < \ell$.[2] The *length* ℓ of a path is the number of its edges. If p is a path, $|p|$ denotes path length. We denote by Path_G the set of all paths in G and by $\mathrm{Path}_G(s, t)$ those from s to t. The expressions $s \rightsquigarrow t$ ("there exists a path from s to t") and $t \leftsquigarrow s$ ("t is reachable from s") are true if and only if there exists a path from s to t. If no path exists, we may write $\ell = \infty$. A digraph is said to contain a *cycle* if and only if there exists $u \in V$ such that $u \rightsquigarrow u$.

[2]Note that, for technical reasons, paths of length 0 are excluded from this definition.

In general, more than one path may connect a pair of nodes. Indeed, if s and t participate in a cycle ($s \rightsquigarrow t \ \wedge \ t \rightsquigarrow s$), then there are infinitely many such paths. Thus, one is often interested in the *shortest path*, defined as:

$$\text{minpath}_G(s,t) = \arg\min_{|p|} \ \{p : p \in \text{Path}_G(s,t)\}.$$

Definition 3 (Diameter)**.** *The* diameter *of a digraph is the length of the longest shortest path between any pair of nodes,*

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

Definition 4 (APL)**.** *The* average path length *[27] of a digraph with n nodes is*

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

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

In this work the *density* of a graph G is always defined with respect to a particular set of graphs $\mathcal{G}$. Specifically, density(G) is the ratio of the number of edges in graph G to the maximum number of edges of any graph in the set $\mathcal{G}$. The resulting value is thus a rational number between 0 and 1 inclusive.

Definition 5 (Maximally and almost maximally dense graphs)**.** *If* density(G) = 1*, then G is said to be* maximally dense *with respect to the set. A graph G' obtained by removing just one edge from a maximally dense graph is said to be* almost maximally dense*.*

A *directed acyclic graph* (DAG) is a digraph that does not contain a cycle. That is, $\text{Path}_G(v,v) = \emptyset$ for all $v \in V$. In Figure 2.1, note that only the digraphs shown in 2.1c and 2.1d are DAGs. The digraphs shown in Figures 2.1a and 2.1b are not DAGs since they contain respective paths $\langle A, B, A\rangle$ and $\langle C, C\rangle$.

A *directed tree* (or *arborescence* [45]) is a digraph such that, for just one node s, called the *root*, and every other node t, $s, t \in V$, $s \neq t$, there exists just one path from s to t. Note

that every directed tree is also a DAG, but the converse does not hold. It can be shown that every node of a directed tree has just one parent, with the exception of the root. A *directed forest* (or *tree-shaped* graph) is a digraph in which each node has at most one parent; thus, a directed tree is also a directed forest. A *chain* is a directed tree with just one leaf; i.e., the edges impose a ranking on the nodes.

A *polytree* is a digraph such that the underlying *undirected* graph does not contain a cycle [16, 28, 82]. Formally, consider that for each directed graph $G = (V, E)$ one can construct an undirected graph $G' = (V, E')$, such that $(u, v) \in E \implies \{u, v\} \in E'$. Then, if G' is acyclic, G is a polytree. Note that every DAG is a polytree, but the converse does not hold. Further note that the graph need not be connected, and that a node may have more than 1 parent.

A *directed path singly connected graph* (DPSCG) is a digraph with "at most one directed path between any pair of nodes" [16]. That is, $|\text{Paths}(s, t)| \leq 1$ for all $s, t \in V$. Note that while every polytree is a DPSC graph, the converse does not hold. A *max-δ-connected graph* has at most δ directed paths between any pair of nodes. That is, $|\text{Paths}(s, t)| \leq \delta$ for all $s, t \in V$. Note that a DPSC graph is also max-δ-connected (with $\delta = 1$), and every max-δ-connected graph is also a DAG.

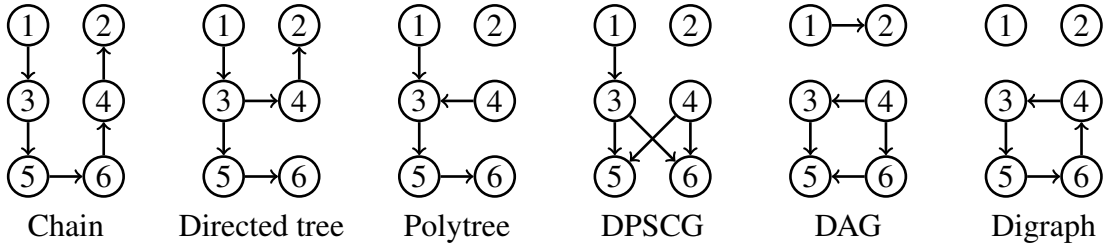


Figure 2.2: Common Classes of Digraphs ($n = 6$)

Finally, a digraph is an *antichain* if it has no directed edges ($E = \emptyset$). Figure 2.2 illustrates some of the common classes of graphs encountered in discussing CP-nets. The relationship between these classes of graphs can be summarized thus:

$$\text{chains} \subseteq \text{directed trees} \subseteq \text{polytrees} \subseteq \text{DPSCGs} \subseteq \text{DAGs} \subseteq \text{digraphs}.$$

2.3 Outcomes

Recall from Section 1.1.4 that in this work we are interested in *multi-featured* preferences. Indeed, in many settings, such as the smart home of Example 2, the *object* actually results from specifying a value for each feature (e.g., whether the window blinds are open or closed). When an object is factored into features, it is known as an *outcome*.

Let $\mathcal{O}$ be a finite set of outcomes characterized by the values of several features represented by variables $\mathcal{V} = \{X_1, \ldots, X_n\}$ with associated domains $\mathrm{Dom}(X_i) = \{x_1^i, \ldots, x_{d_i}^i\}$, $d_i = |\mathrm{Dom}(X_i)|$, such that $\mathcal{O} \equiv \mathrm{Dom}(X_1) \times \cdots \times \mathrm{Dom}(X_n)$. The domain of a *binary* variable has just two values; if $d_i > 2$, then is X_i is *multivalued*. The variables are *homogeneous* if all domains are of the same size $d = d_1 = \cdots = d_n$; otherwise they are *heterogeneous*. For simplicity, variables may also be denoted with different uppercase letters, with their respective values in lowercase (e.g., $A = \{a_1, a_2\}$, $B = \{b_1, b_2, b_3\}$), or in specific examples with variables in SMALLCAPS and values in *italics*: e.g., FRUIT = {*apple*, *banana*, *tomato*}. Moreover, if a variable X_i is binary, its values may be denoted $\mathrm{Dom}(X_i) = \{x_i, \overline{x_i}\}$.

The values that a variable can take may be *constrained* to a proper subset of its domain. When X_i is constrained to just one value x_j^i, then it is said to be *assigned* that value, in which case one may write, $X_i = x_j^i$. A set of variables $U \subseteq \mathcal{V}$ can similarly be constrained and assigned. An assignment to all variables $U = \mathcal{V}$ (a *full instantiation*) designates a single outcome $o \in \mathcal{O}$. The set of assignments to $U \subseteq \mathcal{V}$ is denoted by $\mathrm{Asst}(U)$, where $\mathrm{Asst}(U) = \mathrm{Dom}(X_{h_1}) \times \cdots \times \mathrm{Dom}(X_{h_m})$, $X_{h_k} \in U$, $m = |U|$, $1 \leq k \leq m$.

The expression $o[X_i]$ denotes the *projection* of an outcome $o \in \mathcal{O}$ onto a variable X_i. We also generalize the use of the postfix $[\cdot]$ operator so that if Q is an outcome or set of outcomes and W is a variable or set of variables, then $Q[W]$ denotes the projection of outcomes Q onto variables W. Similarly, the expression $o[-X_i]$ is the projection of o onto $\mathcal{V} \setminus \{X_i\}$, and in general $Q[-W]$ is the projection of Q onto $\mathcal{V} \setminus Q$. Moreover, when each variable is indexed with a natural number i, i.e., when $V = \{X_1, \ldots, X_n\}$, then $o[i]$, $o[-i]$, $Q[i]$ and $Q[-i]$ are understood to mean $o[X_i]$, $o[-X_i]$, $Q[X_i]$, and $Q[-X_i]$ respectively.

The expression ux^i_k denotes the *combination* of $u \in \text{Asst}(U)$ and $x^i_k \in \text{Dom}(X_i)$, where $X_i \notin U$. In general, if w and u are assignments to disjoint sets W and U, $w \in \text{Asst}(W)$, $u \in \text{Asst}(U)$, $W \cap U = \emptyset$, then wu denotes the combination of w and u.

Example 6. *Let* $\mathcal{V} = \{A, B, C\}$, $\text{Dom}(A) = \{a_1, a_2, a_3\}$, $\text{Dom}(B) = \{b_1, b_2\}$, *and* $\text{Dom}(C) = \{c_1, c_2\}$. *Then* $\text{Asst}(\{B, C\}) = \{b_1c_1, b_1c_2, b_2c_1, b_2c_2\}$, $a_3b_1c_1[A] = a_3$, $a_3b_1c_1[\{B, C\}] = b_1c_1$, *and* $a_3b_1c_1[-C] = a_3b_1$.

Definition 7 (Hamming distance)**.** *The* Hamming distance *of a pair of outcomes* $\text{HD}(o, o')$, $o \in \mathcal{O}$, $o' \in \mathcal{O}$, *is the number of variables in the outcomes for which the values differ, i.e.,*

$$\text{HD}(o, o') = |\{\, X_i : o[X_i] \neq o'[X_i] \,\}|\,. \tag{2.1}$$

For example, $\text{HD}(a_3b_1c_1, a_1b_1c_1) = 1$ and $\text{HD}(a_1b_1c_1, a_2b_2c_2) = 3$. One can observe that $\text{HD}(o, o') = \text{HD}(o', o)$ for all $o, o' \in \mathcal{O}$ and that $\text{HD}(o, o') = 0$ if and only if $o = o'$. Finally, $0 \leq \text{HD}(o, o') \leq n$, where $n = |V|$.

We denote by $\mathcal{O}_n$ the set of all outcomes on n binary features and by $\mathcal{O}_{n,d}$ all outcomes on n d-ary features. We denote by $\mathcal{O}^2_n$ and $\mathcal{O}^2_{n,d}$ all *ordered pairs* of outcomes on n binary and d-ary features. $\mathcal{O}^2_{n\,|\,h}$ and $\mathcal{O}^2_{n,d\,|\,h}$ are the same sets restricted to pairs with Hamming distance h, $0 \leq h \leq n$; for example, $\mathcal{O}_{4\,|\,2} = \{(o, o') : \text{HD}(o, o') = 2,\ o \in \mathcal{O}_4,\ o' \in \mathcal{O}_4\}$.

2.4 Preferences and CP-nets

A preference relation takes the form of a *preorder* if it can model outcomes over which a subject may be indifferent. In this work, however, *strict* (though not necessarily total) preferences are assumed. We thus model preferences as a *partial order*.

Definition 8 (Preference)**.** *A* strict preference relation $\succ_{\mathcal{S}}$ *is a partial order on a set of outcomes* $\mathcal{O}$ *by a subject* $\mathcal{S}$.

When no confusion would result, the subscript will be omitted and $\succ$ used as an infix operator, where $o \succ o'$ indicates that the subject strictly prefers o to o'. Equivalently one

may write $o' \prec o$, because $o \succ o' \iff o' \prec o$. In a preference relation, as for partial orders in general, the infix $\|$ operator denotes incomparability. Recall from Section 2.1 that since preferences are posets, just one of the following is true: $o \succ o'$, $o \prec o'$, $o = o'$, or $o \| o'$. We assume $\mathcal{O}$ is finite and can be factored as described in Section 2.3. Note that for d-ary variables $|\mathcal{O}| = d^n$; that is, exponential space is required to store $\succ$ (see 1.2.1). However, because $\mathcal{O}$ is factored, a *conditional preference network* (CP-net) [16] can compactly model $\succ$.

Definition 9 (CP-net)**.** *A* conditional preference network *is a digraph on* $\mathcal{V} = \{X_1, \ldots, X_n\}$ *in which each node is labeled with a* conditional preference table*. An edge* (X_h, X_i) *indicates that the preferred value of* X_i *in* $\succ$ *depends on the value of its parent variable* X_h*.*

Definition 10 (CPT)**.** *A* conditional preference table $\text{CPT}(X_i)$ *consists of* conditional *ceteris paribus* preference rules (CPRs) $u : \succ^i$ *specifying a linear order on* $\text{Dom}(X_i)$ *for assignments to the parents of* X_i *in the* dependency graph*,* $u \in \text{Asst}(\text{Pa}(X_i))$.

Formally, $\text{CPT}(X_i)$ implements a function $f : \text{Asst}(\text{Pa}(X_i)) \to \mathfrak{S}(\text{Dom}(X_i))$. If $f(u)$ is defined for all $u \in \text{Asst}(\text{Pa}(X_i))$, i.e., a preference over X_i is specified for *every* assignment, the CPT is said to be *complete*; otherwise it is *incomplete*. Unless otherwise specified, we assume CPTs are complete. The expression $\text{CPT}(X_i|X_h = x_k^h)$ denotes all rules of $\text{CPT}(X_i)$ of the form $ux_k^h : \succ^i$ where $x_k^h \in \text{Dom}(X_h)$, $X_h \in \text{Pa}(X_i)$, $u \in \text{Asst}(\text{Pa}(X_i) \setminus \{X_h\})$. Figure 2.3 illustrates a simple chain-shaped, binary-valued CP-net.

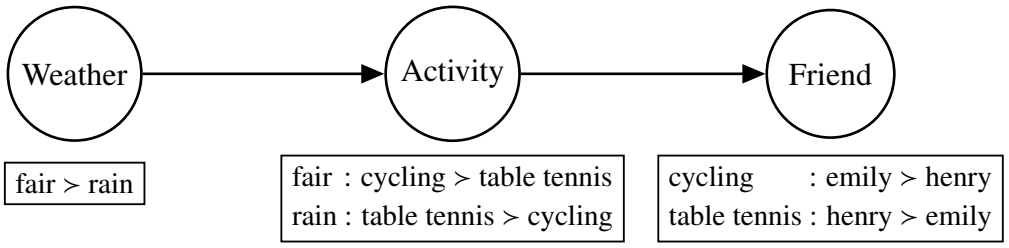


Figure 2.3: A Simple CP-net N

The size of a CPT is defined as the number of rules it contains, and the size of the CP-net is the sum of the sizes of its CPTs. If domains are d-ary and CPTs are complete, a node with m parents has d^m CPRs. Thus, the size of the description is exponential in the maximum indegree of G. To provide a compact model of $\succ$ (see 1.2.2), indegree is assumed to be bounded by a small constant [16], i.e., $|\mathrm{Pa}(X_i)| \leq c$ for all X_i. We assume that domain size is similarly bounded.

The term *dependency graph* (or *graph*) denotes the digraph G of a CP-net apart from its CPTs (*tables*). A *chain*, *tree-shaped*, or *polytree* CP-net is one for which the graph takes the respective shape of a chain, directed *forest*, or polytree (Section 2.2). It is often assumed that the graph of a CP-net is acyclic (i.e., a DAG); if the CP-net may have a cycle, this is usually qualified, e.g., *generally cyclic CP-nets*.

As discussed in Section 1.2.1, a CP-net induces an exponentially larger graph known as the *preference graph* (or *outcome graph*), which we now define formally:

Definition 11 (Preference graph)**.** *The* induced preference graph *(PG) of a CP-net N is a digraph $H = (\mathcal{O}, \mathcal{C})$ in which $(o', o) \in \mathcal{C}$ if and only if there exists a CPR $u : \succ^i$ in the CPT of a node X_i in N, such that $(o[X_i], o'[X_i]) \in \succ^i$, $o[X_i] \neq o'[X_i]$, $o[-X_i] = o'[-X_i]$, $u \in \mathrm{Asst}(\mathrm{Pa}(X_i))$, $u = o[\mathrm{Pa}(X_i)] = o'[\mathrm{Pa}(X_i)]$, $o \in \mathcal{O}$, $o' \in \mathcal{O}$, and $X_i \in \mathcal{V}$. A directed edge from o' to o thus indicates that $\mathrm{HD}(o, o') = 1$ and that $o' \prec o$.*

If variables are binary, $|\mathcal{O}| = 2^n$ and H takes the geometric shape of a directed hypercube, sometimes known as a *Hamming cube* [31]. Figure 2.4 depicts the induced preference graph for the CP-net of Figure 2.3. (Note that FCH, for example, denotes the outcome ⟨*fair*, *cycling*, *henry*⟩.) The rule *fair* ≻ *rain* in the CPT of Weather in N induces the edges along "dimension" Weather in H: (RCH, FCH), (RTH, FTH), (RCE, FCE), and (RTE, FTE). This must be the case, since the subject always prefers *fair* weather to *rain*, regardless of the activity or companion. Similarly, the rule *table tennis* : *henry* ≻ *emily* in CPT(*friend*) induces the directed edges, (FTE, FTH) and (RTE, RTH). Whatever the Weather, the subject prefers *henry* to *emily* as the Friend for Activity *table tennis*.

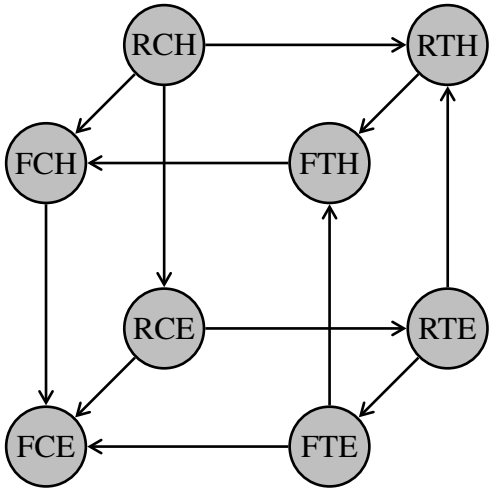


Figure 2.4: Induced Preference Graph H of CP-net N (see Figure 2.3)

The CPRs allow direct comparison between outcomes that differ in the value of just one variable. Comparing outcomes that differ in more than one variable requires finding a path in the preference graph from the less to the more preferred outcome. For example, the outcomes RCE and FCH differ in *two* variables; thus no single rule specifies whether the subject would prefer a rainy day cycling with Emily or a fair day cycling with Henry. However, note that a path connects the two outcomes:

$$\text{RCE} \prec \text{RTE} \prec \text{RTH} \prec \text{FTH} \prec \text{FCH}. \tag{2.2}$$

Such a path is a known as an improving *flipping sequence* (FS), since traversing an edge of the preference graph *flips* the value of a variable such that the subject is more satisfied with the resulting outcome.[3] Paths such as RCE $\rightsquigarrow$ FCH result from the transitive closure of the *ceteris paribus* rules of the CP-net. The existence of such a path counts as a proof that FCH $\succ$ RCE. Note that multiple paths may connect a pair of outcomes. For example, FCE $\succ$ RCE since RCE $\prec$ RTE $\prec$ FTE $\prec$ FCE; however, there is also a shorter, more direct path, RCE $\prec$ FCE.

[3]While the term *flip* is perhaps better suited to binary domains (e.g., *flip a coin*), it is also used of multivalued variables both in everyday speech (e.g., *flip a die*) and in the preference handling literature. Note that the semantics of the rule $u :\succ^i$ allow us to flip directly to any more preferred value in $\text{Dom}(X_i)$ given u. In particular, if $\text{CPT}(B)$ contains the rule $a_1 : b_1 \succ b_2 \succ b_3$, one can flip directly from $o'[AB] = a_1b_3$ to $o[AB] = a_1b_1$ without first having to visit $o''[AB] = a_1b_2$.

Definition 12 (Flipping sequence)**.** *A* flipping sequence *is a path in the induced preference graph of a CP-net.*

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

Definition 13 (DT problem)**.** *A* dominance testing problem *is a decision problem for which the input is a triple* (N, o, o') *consisting of a CP-net* N *on* $\mathcal{V} = \{X_1, \ldots, X_n\}$ *and outcomes* o *and* o'*,* $o \in \mathcal{O}$*,* $o' \in \mathcal{O}$*,* $\mathcal{O} \equiv \mathrm{Dom}(X_1) \times \cdots \times \mathrm{Dom}(X_n)$*. The answer is in the affirmative if and only if* $N \models o \succ o'$*.*

We denote by $\mathrm{DT}(\mathcal{N}, \mathcal{O})$ the set of all DT problem instances (N, o, o'), such that $N \in \mathcal{N}$, $o \in \mathcal{O}$, and $o' \in \mathcal{O}$, and by $\mathrm{DT}(\mathcal{N}, \mathcal{O} \mid \theta)$ the set of instances satisfying one or more conditions θ. For example, $\mathrm{DT}(\mathcal{N}, \mathcal{O}\ \mathrm{HD}(o, o) = h,\ \mathrm{APL}(G) = 0.5)$ denotes the set of DT problem instances (N, o, o'), such that $N \in \mathcal{N}$, $o \in \mathcal{O}$, $o' \in \mathcal{O}$, for which the Hamming distance between the outcomes is h and the average path length of the dependency graph is 0.5.

Definition 14 (Flipping length)**.** *The* flipping length *is the length of the shortest path between a pair of outcomes in the induced preference graph* H *of a CP-net* N*,*

$$\mathrm{FL}(N, o', o) = \mathrm{minpath}_H(o', o). \tag{2.3}$$

If no such path (flipping sequence) exists, then the flipping length is undefined.

When the flipping sequence is undefined, we may write $\mathrm{FL}(N, o', o) = \infty$.

In Chapter 5 we will be interested in the longest flipping length, which we call the *diameter* of the preference graph $\mathrm{Diam}(H)$ (see Definition 3).[4] In the example shown in Figure 2.4, the longest flipping sequence is the one given in Equation 2.2; consequently, $\mathrm{Diam}(H) = 4$. Note that the outcomes (RCE, FCH) that produce this diameter have Hamming distance 2. We will also be interested in the longest flipping length that connects any pair of outcomes with a given Hamming distance h.

Definition 15 (h-Diameter)**.** *The* h-diameter *of the induced preference graph* H *is*

$$\mathrm{Diam}_h(H) = \max_{o,o'} \left|\mathrm{minpath}_H(o, o')\right|, \tag{2.4}$$

such that $\mathrm{HD}(o, o') = h$, $o \in \mathcal{O}$, $o' \in \mathcal{O}$.

One can confirm from Figure 2.4 that while $\mathrm{Diam}(H) = \mathrm{Diam}_2(H) = 4$, the shortest path between any two outcomes that differ in all three variables is $\mathrm{Diam}_3(H) = 3$.

Definition 16 (Mean flipping length)**.** *If* $\mathcal{I} \subseteq \mathrm{DT}(\mathcal{N}, \mathcal{O})$ *is a non-empty set of DT problem instances, then* $\mathrm{MFL}(\mathcal{I})$ *is the* mean flipping length *for* $\mathcal{I}$*,*

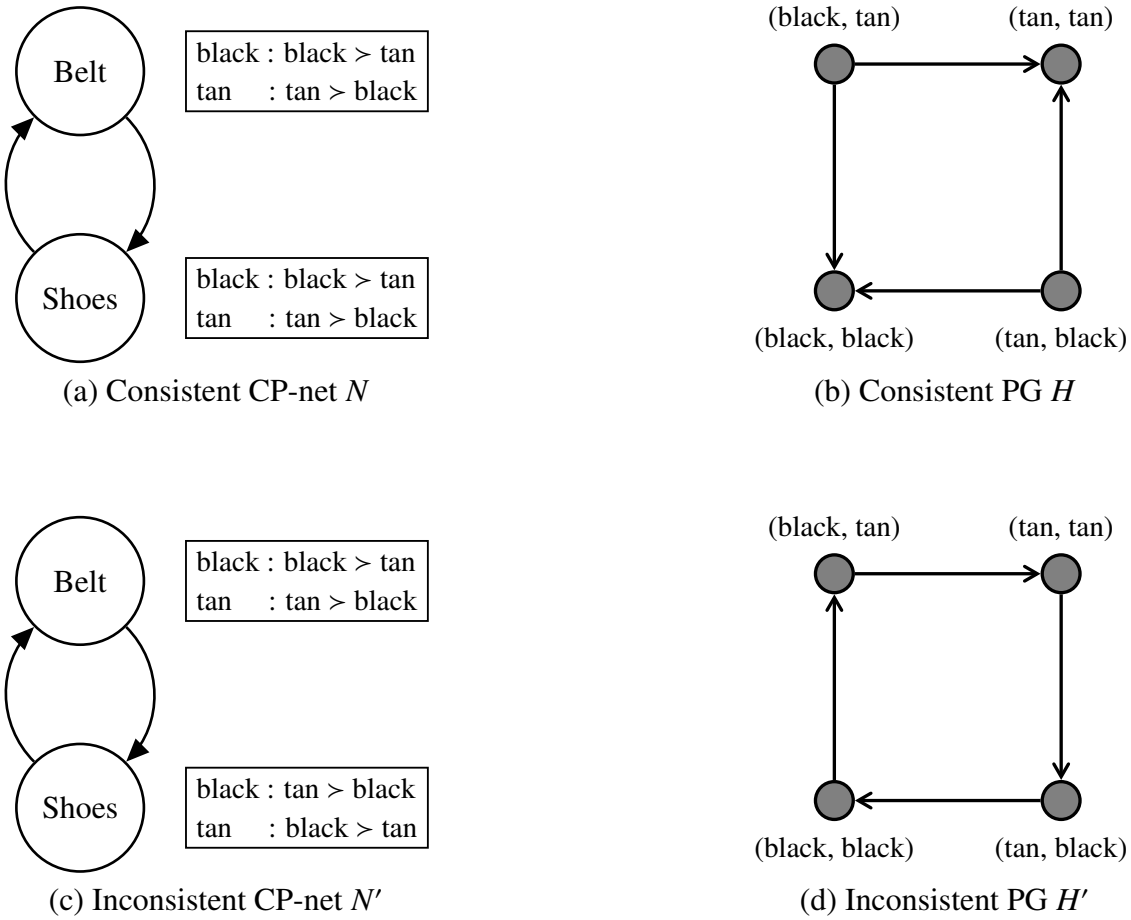
$$\mathrm{MFL}(\mathcal{I}) = \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \mathrm{FL}(N, o, o') \tag{2.5}$$

such that $o \neq o'$ *and* $\mathrm{FL}(o, o')$ *is defined, where* N*,* o *and* o' *are the three elements of tuple* $I = (N, o, o')$*.*

Similarly, $\mathrm{MFL}(\mathcal{I} \mid \theta)$ denotes the mean flipping length for instances $\mathcal{I}$ that satisfy a set of conditions θ. For example, $\mathrm{MFL}(\mathcal{I} \mid \mathrm{HD}(o, o') = 2)$ denotes the mean flipping length for all $I \in \mathcal{I}$ with Hamming distance 2.

We denote by $\mathcal{N}_n$ the set of all CP-nets on n binary features, and by $\mathcal{N}_{n,d}$ the set of CP-nets on n d-ary features. $\mathcal{N}_{n\,|\,c}$ and $\mathcal{N}_{n,d\,|\,c}$ indicate the same sets restricted to dependency graphs with indegree at most c.

[4] Confusingly, this longest flipping length is sometimes called the diameter *of the CP-net* [61]. However, that usage lends itself to confusion between the diameter of the *induced preference graph* and the diameter of the *dependency graph*, $\mathrm{Diam}(G)$, which in this case of course is 2.



(a) Consistent CP-net N (b) Consistent PG H

(c) Inconsistent CP-net N' (d) Inconsistent PG H'

Figure 2.5: Cyclic CP-nets and Induced Preference Graphs

Finally, one can observe that if H (not to be confused with G) contains a cycle, the induced order on the outcomes is inconsistent. Consider the CP-nets in Figure 2.5. CP-net N in Figure 2.5a expresses a preference for coordinating the color of leathers. (The customary rule of fashion is that a black belt should be matched with black shoes, tan with tan, and so on.) The induced preference graph expresses this convention as one would expect. If the subject discovers that he is wearing a black belt and tan shoes, he can improve by switching either shoes or belts. Having made the switch, the outcome is then one of the two incomparable optima. A CP-net could just as easily enforce anti-coordination by inverting the order in all four conditional preference rules. However, in CP-net N' in Figure 2.5c, observe that only the CPT for *shoes* is inverted. The unfortunate result, depicted in Figure 2.5d, is an

Table 2.1: Commonly Used Acronyms

Acronym	Meaning	Section
APL	average path length	2.2
CP-net	conditional preference network	2.4
CPR	conditional preference rule	2.4
CPT	conditional preference table	2.4
DAG	directed acyclic graph (labeled)	2.2
DPSCG	directed path singly connected graph	2.2
DT	dominance testing	2.4
FS	flipping sequence (improving)	2.4
HD	Hamming distance	2.3
PG	Preference graph	2.4

irrational subject who is forever changing shoes and belts in endless hope of improvement. Unfortunately, if the graph G of a CP-net has cycles, finding optima and proving consistency are known to be PSPACE-complete [42]. On the other hand, when G is acyclic, CPTs are complete, and indifference is disallowed—assumptions that we will adopt throughout most of this work—then the optimum outcome is unique, computationally easy to find, and the resulting order on $\mathcal{O}$ is guaranteed to be consistent [16].

2.5 Commonly Used Notation and Abbreviations

The chapter concludes with references for the reader. Table 2.1 provides the meaning of common acronyms. Table 2.2 summarizes notation commonly used throughout this work. The *Section* column specifies the chapter, section, or subsection where the operator or term is defined or discussed, if applicable.

Table 2.2: Commonly Used Notation

Symbol	Most Likely Semantics	Section(s)
$\succsim$	is at least as good as; weak preference relation	1.1; 2.4
$\succ$	is better than, preferred to; strict preference relation	1.1; 2.4
$\succ^i$	linear order on the domain of X_i	2.4
$\|\|$	is incomparable to; incomparability	1.1.1; 2.1
$\sim$	is ordered equally with; indifference	1.1.1; 2.1
$\overline{\ }$	set complement (e.g., $\overline{U} \equiv V \setminus U$); Boolean negation ($x \vee \overline{x}$)	
$\setminus$	set difference	
$\times$	set multiplication, i.e., Cartesian product	
$\models$	models; preferentially entails	2.4
$\alpha_{t,i,j,k}$	Boolean action variable	5.6.2
ϵ	probability of noise for every CPR; other small quantity	5.5
θ	set of conditions or constraints	2.4
ξ	Boolean clause	5.6.2
$\phi_d(m)$	number of d-ary functions F_j with m inputs	4.2
$\chi_d(m)$	number of degenerate d-ary functions F_j with m inputs	4.2
$\psi_d(m)$	number of non-degenerate d-ary functions F_j with m inputs	4.2
ω	Boolean CNF formula	5.6.2
A	dagcode, a tuple that encodes a DAG	4.3
A_j	an element of dagcode	4.3
$A_{<j}$	partial dagcode A	4.3
$a_{n,c}$	number of DAGs parameterized by n and c	4.3
$a_{n,c,d}$	number of CP-nets parameterized by n, c, and d	4.4
Asst$(\cdot)$	set of all assignments to variables	2.3

Continued on the following page

Table 2.2: Commonly Used Notation (continued)

Symbol	Most Likely Semantics	Section(s)
$\mathcal{B}$	all vectors consisting of n bits	6.2
B	vector of n bits $\{0, 1\}$ encoding the CPTs of CP-net in ${}^{T}\mathcal{N}_n$	6.2
$\mathcal{C}$	edges of PG corresponding to *ceteris paribus* rules	2.4
c	bound on the indegree of any node in a graph	2.2
d	domain size (when homogeneous)	2.3
d_i	size of domain of variable X_i; a member of Dom(D)	2.3
Diam($\cdot$)	diameter of graph, i.e., longest path between any two nodes	2.3
Diam$_h(H)$	h-diameter of a preference graph H	2.4
Dom($\cdot$)	domain of a variable	2.3
DT($\mathcal{N}, \mathcal{O}$)	set of all DT problems (N, o, o'), $N \in \mathcal{N}$, $o, o' \in \mathcal{O}$	2.4; 5.1
DT($\mathcal{N}, \mathcal{O} \mid \theta$)	set of all DT problems (N, o, o') satisfying conditions θ	2.4; 5.1
$\mathcal{E}$	set of pairwise outcome comparison data	6.1
E	set of edges in a graph	2.2
E_t	comparison (o_t, o'_t) in $\mathcal{E}$ such that $o_t \succ o'_t$	6.1
F	cpt-code; tuple of function vectors F_j	4.2
F_j	function vector corresponding to a CPT	4.4
$F_{<j}$	partial cpt-code	4.3
FL(N, o, o')	flipping length; i.e., length of a shortest flipping sequence	2.4
G	directed graph, esp. dependency graph of a CP-net	2.2
H	preference graph (a Hamming cube if binary)	1.2.1; 2.4
h	index, esp. of a parent node; Hamming distance	2.4
HD($\cdot, \cdot$)	Hamming distance; i.e., number of variables that differ	2.3
$\mathcal{I}$	set of DT instances	2.4

Continued on the following page

Table 2.2: Commonly Used Notation (continued)

Symbol	Most Likely Semantics	Section(s)
I	DT instance (N, o, o')	2.4
i	index, esp. of a variable or node representing a feature	2.3
j	index, e.g., of a child node, encoding, etc.	2.3; 4.3
k	bound on flipping length for DLDT; index	5.5
$\mathcal{L}$	square matrix consisting of all flipping lengths	5.2
$\mathcal{L}_n$	all Prüfer codes with $n-1$ integers ranging from 1 to $n+1$	6.2
L	average path length; other length; Prüfer code	5.2; 6.2
ℓ	index; length of a sequence, data set, or path	2.2; 6.1
M	adjacency matrix of a graph	5.2
m	number of parents of a node; number of inputs to a function	2.4
$\mathrm{MFL}(\mathcal{I})$	mean flipping length of a set of DT instances	2.4; 5.1
$\mathrm{MFL}(\mathcal{I} \mid \theta)$	mean flipping length of DT instances satisfying conditions θ	2.4; 5.1
$\mathcal{N}$	a set of CP-nets	2.4
N	a particular CP-net	2.4
n	number of features, variables, nodes	2.3
$\mathcal{N}_n$	binary acyclic CP-nets with n nodes	5.1
$\mathcal{N}_{n,d}$	d-ary acyclic CP-nets with n nodes	5.1
$\mathcal{N}_{n \mid c}$	binary acyclic CP-nets with n nodes and indegree bound c	5.1
$\mathcal{N}_{n,d \mid c}$	d-ary acyclic CP-nets with n nodes and indegree bound c	5.1
${}^{T}\mathcal{N}_n$	tree-shaped CP-nets on n binary nodes	6.2
$\mathcal{O}$	set of conceivable outcomes	2.3
$\mathcal{O}_n$	set of outcomes over n binary features	2.3
$\mathcal{O}_{n,d}$	set of outcomes over n d-ary features	2.3

Continued on the following page

Table 2.2: Commonly Used Notation (continued)

Symbol	Most Likely Semantics	Section(s)
$\mathcal{O}_n^2$	all pairs of outcomes over n binary features	2.3
$\mathcal{O}_{n \mid h}^2$	all pairs of outcomes over n binary features with HD h	2.3
$\mathcal{O}_{n,d \mid h}^2$	all pairs of outcomes over n d-ary features with HD h	2.3
$\mathrm{Pa}(\cdot)$	parents of a node	2.2
$\mathfrak{S}(\cdot)$	symmetric group; set of all rankings, permutations	2.2
S, T, U	various sets	
$\mathcal{S}$	the subject, preference holder	1.1; 2.4
$\mathcal{T}_n$	set of all treecodes $\mathcal{L}_n \times \mathcal{B}_n$	6.2
t	index of items in a sequence; timestep in SATplan	5.6.2; 6.1
u	node in a graph; utility function; assignment to parents	1.1.1; 2.2
$\mathcal{V}$	set of features, variables, nodes	2.2; 2.3
v	node in a graph	2.2
X_i	a particular feature, variable, node	2.3
$z_{t,i,j}$	Boolean state variable	5.6.2

Chapter 3 Related Work

This chapter provides an overview of the present state of the research involving CP-nets. Because most of that research applies various *restrictions* to the formalism, Section 3.1 summarizes restrictions that can be applied to models, such as limiting the shape of the network, the size of domains, etc. Section 3.2 discusses the problem of finding *optimal outcomes* given a CP-net and also that of finding the k-best outcomes. Section 3.3 considers the problem of checking whether a CP-net is *consistent*. Section 3.4 discusses problems related to *reasoning* with CP-nets, such as dominance testing, ordering queries, and heuristic methods, as well as the complexity of the reasoning problems. Section 3.5 considers the problem of *learning* CP-nets from outcome comparison data or elicited queries and what is known about the complexity of the respective problems. Section 3.6 discusses *experiments* with CP-nets, including efforts to generate CP-nets randomly, experimental validation of algorithms, and available datasets. Section 3.7 concludes the chapter with a discussion of some proposed *extensions* to the CP-net formalism.

3.1 General and Restricted CP-net Models

CP-nets have already been discussed along with examples in Sections 1.3 and 2.4. The CP-net formalism was originally proposed by Craig Boutilier along with coauthors Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole, initially in conference proceedings (1999) [17] and later extended for publication in the *Journal of Artificial Intelligence Research* (2004) [16]. In their most general form, without extensions such as those discussed in Section 3.7, CP-nets are allowed to have cycles in the dependency graph, with only two constraints on the geometry: loops are disallowed, and a small constant bound on indegree is assumed. Moreover, the features that characterize the outcome space, so long as they are discrete, can be multivalued, CPTs can be partially specified (i.e., incomplete),

and CPRs can express a *weak* total order over the domain of the local variable. While such general models can represent a broader range of problems, reasoning with such models may be intractable and there is no guarantee that the resulting order on outcomes will be consistent.

Almost always, however, one or more *restrictions* are applied to the set of possible CP-net models, either as a requirement for algorithms or as an aid in proving theoretical results. Restrictions can be applied to the dependency graph, domains, CPTs, or CPRs. Recall from Sections 2.2 and 2.4 that the dependency graph of a CP-net can be restricted to a subclass of digraphs, such as DAGs, polytrees, etc. Table 3.1 lists some of the more common restrictions to the structure of the dependency graph. By far the most common of these is to exclude cycles; in fact, unless qualified (e.g., *generally cyclic CP-nets*), one can usually assume that the dependency graph is a DAG. Recall from Section 2.4 that, regardless of the type of graph, a *bound on indegree* is always assumed for CP-nets to ensure a compact model.

Similar restrictions can be applied to the variable domains, such as a bound on the cardinality of the largest domain or on the number of different cardinalities when domains are heterogeneous. A rather common restriction is *that all variables must be binary* [29, 61, 65, 104]. Another common restriction [16] is *that CPTs must be complete*, i.e., each CPT is fully specified with a linear order over the local variable for every assignment to parents. Algorithms for learning are an interesting exception in which it is common to output a CP-net that is likely to be *in*complete [29, 48, 58]. Finally, almost all researchers disallow indifference, restricting attention to CP-nets that can model only strict preferences.

3.2 Finding Most Preferred Outcomes

A common problem in working with preferences is to find the (or in some cases, such as Figure 2.5b, *an*) outcome that the subject most prefers. For other compact formalisms (see Section 1.2.2), optimization problems of this type are known to be hard [32, 71]. However,

Table 3.1: CP-nets Characterized by Dependency Graph

Dependency Graph	Resulting Class of CP-net	References
Chain	Chain CP-net	[16]
Antichain	Separable CP-net (SCP-net)	[66]
Directed forest	Tree (or Tree-structured) CP-net	[11, 16, 59]
Polytree	Polytree CP-net	[16]
DPSCG	Directed path singly connected CP-net	[16]
Max-δ-connected	Max-δ-connected CP-net	[16]
DAG	Acyclic CP-net	[16]
Digraph	CP-net (or generally cyclic CP-net)	[16, 42, 61]

Boutilier et al. [16] proved that the problem of *outcome optimization* is easy in acyclic, complete CP-nets. In such cases the most preferred outcome is unique and can be found in linear time in the number of nodes using a *forward sweep* algorithm that they describe.

Brafman et al. [20] proved that the related problem of finding the k-best outcomes in an acyclic CP-net (presumably with complete CPTs) can be computed in polynomial time in the number of variables, assuming the solutions can be linearized in a particular way, which they call a *contextual lexicographical linearization*. If CPTs are *incomplete*, however, the complexity of finding the most preferred and k-best outcomes is believed to be an open problem [16].

3.3 Checking for Consistency

As noted in Section 2.4, CP-nets can induce an intransitive order on outcomes in certain cases. In particular, as noted in Section 2.4, inconsistency can sometimes arise if the dependency graph contains a cycle (see Figure 2.5). Domshlak and Brafman [30] showed that consistency checking could be conducted efficiently for a wide class of cyclic, binary CP-nets. As later shown by Goldsmith et al. [42], however, the general problem is PSPACE-complete. More recently, Santhanam et al. [93, 94] have reduced the problem to one of model checking in the form of the CRISNER CP-nets reasoning tool.

3.4 Reasoning with CP-nets

Given a pair of distinct outcomes, the *reasoning problem* involves determining which outcome, if either, is preferred. Section 3.4.1 discusses *dominance testing*, the strongest method of reasoning. Section 3.4.2 then considers the weaker method or *ordering queries*. Finally, heuristic methods are discussed in Section 3.4.3.

3.4.1 Dominance Testing

Recall from Section 2.4 that, given a CP-net N with strict preferences and a pair of outcomes o and o', *dominance testing* (DT) determines whether there exists an improving *flipping sequence* from o' to o. If so, the CP-net said to *entail* that the first outcome *dominates* the second, written $N \models o \succ o'$.

For arbitrary, possibly cyclic CP-nets, DT is known to be PSPACE-complete [42], and in certain cases (in particular, chain CP-nets) flipping lengths can be $\Omega(2^{n/2})$, i.e., exponential in the number of nodes n, provided tables are incomplete and domains are multivalued, even for chain CP-nets [16]. However, Boutilier et al. showed that, in the most general case, dominance testing can be formulated as a STRIPS-type planning problem. More recently, Kronegger et al. [61] have established several fixed parameter tractability (FPT) results for dominance testing in a generalized class of CP-nets (GCP-nets) (similar to those studied by [42]). Many of their FPT results also apply to CP-nets.

Several tractable subclasses for DT are known. Boutilier et al. [16] showed that DT can be conducted in $\Theta(n^2)$ time for binary valued tree CP-nets with their DT-Tree algorithm, which also returns a flipping sequence if one exists. They claim that the algorithm remains complete, with the same time complexity, when CPTs are incomplete. Bigot et al. [11] subsequently described an algorithm they claim can answer dominance in $O(n)$ time for the same class of CP-nets (except that CPTs must also be complete), an unexpected result, since the flipping length is $O(n^2)$ for such CP-nets. Thus, while the decision problem can be answered in linear time, computing the flipping sequence itself requires quadratic time.

Table 3.2: Computational Difficulty of Dominance Testing

Graph	Domains	CPTs	DT Complexity	Running Time	References
Directed Forest	Binary	Complete	P	$O(n)$	[11]
Directed Forest	Binary	*	P	$O(n^2)$	[16]
Directed Forest	*	*	NP-hard	?	[16]
Polytree	Binary	*	P	$O(2^{2c}n^{2c+3})$	[16]
DPSCG	Binary	*	NP-complete	?	[16]
Max-δ-connected	Binary	*	NP-complete	?	[16]
*	*	*	PSPACE-complete	?	[42]

*No restriction

In the case of polytree CP-nets, Boutilier et al. [16] showed that DT could be answered in polynomial time via their reduction to planning. Specifically, the algorithm requires time $O(2^{2c}n^{2c+3})$ where c is the assumed bound on indegree in the dependency graph. However, even when variables are binary, DT for directed-path singly connected CP-nets is NP-complete if CPTs are complete (and NP-hard otherwise). The problem remains NP-complete when the number of paths between any two nodes in the graph is polynomially bounded (i.e., max-δ-connected graphs). Table 3.2 summarizes the computational complexity and running times of DT algorithms for various classes of CP-nets.

3.4.2 Ordering Queries

Because dominance testing is hard in many cases, Boutilier et al. [16] also introduced a weaker, incomplete method of reasoning with CP-nets. Rather than asking whether a flipping sequence exists between a pair of outcomes, the method searches for local rules that would contradict the existence of such a sequence. Given a CP-net N and a pair of outcomes o and o', o is said to be *consistently orderable* over o' if $N \not\models o' \succ o$. They call this search for local contradictions an *ordering query* and show that it can be completed in linear time in the number of variables, provided CPTs are complete. (The complexity of ordering queries for incomplete CP-nets seems to be an open problem. However, the authors note that they suspect it is hard.) Let $N \models_{\text{oq}} o \gg o'$ denote that o is consistently orderable over o' with respect to CP-net N. Note that $N \models_{\text{oq}} o \gg o' \implies N \not\models o' \succ o$.

However, when $N \not\models_{\text{oq}} o \gg o' \ \wedge \ N \not\models_{\text{oq}} o' \gg o$, it could still be the case that $N \models o \succ o'$, $N \models o' \succ o$, or $N \models o \parallel o'$.

To better understand the nature of this method of ordering, recall that a CP-net induces a partial order on outcomes. If o is *consistently orderable* over o', it means only that o is ordered before o' in *some* linear extension of the induced partial order (see Section 1.1.2). However, if o *dominates* o', then it follows that o is ordered before o' in *every* such linear extension.

3.4.3 Reductions and Heuristic Methods

In general, DT involves a search for a flipping sequence that connects the two outcomes. Any of the familiar search methods in AI, e.g., iteratively deepening depth-first search, can be employed. Boutilier et al. [16] introduced two methods of pruning the search tree, *suffix fixing* and *forward pruning*, that work in all cases, as well as a heuristic method, *least-variable flipping*, that is incomplete except for binary-valued tree-shaped CP-nets. In addition to the reduction to STRIPS-type planning [16], DT problems can also be reduced to model checking [92, 93] (similar to their reduction for the consistency problem; see Section 3.3). Finally, Li et al. [68] have proposed a heuristic approach to DT in acyclic, generally multivalued CP-nets that they call DT*. While the algorithm is inspired by A*, it does not seem to guarantee optimality; i.e., it does not always return a *shortest* flipping sequence.

3.5 Learning CP-nets

Recall from Section 1.2.3 that, aside from *direct construction* by the subject, which is problematic since it relies on introspection, learning a CP-net can take the form of *active elicitation* or *passive learning* from data. The earliest work on learning CP-nets seems to be that of Athienitou and Dimopoulos for the MPREF-2007 conference [6]. They introduced a passive learning algorithm that attempts to recover a CP-net that *entails* all examples

in a set of outcome comparison data. However, as observed by Lang and Mengin [65], *entailment* is an exceptionally strong requirement. Suppose a subject has a linear order on the outcome space, arising from some utility function (see 1.1.1). Arguably, then, the goal is not to "recover" some presumed original CP-net, but to learn a CP-net such that the observed comparison data are *consistent* with the learned model. They proposed three notions of consistency for CP-nets. Lang and Mengin were also among the first to study the complexity of the learning problem. To establish a lower bound, they proposed a simple class of so-called *separable CP-nets* (SCP-nets) for which the dependency graph is an antichain, i.e., a graph with no edges. In a later paper [66] they succeeded in proving that, while it was possible to answer in polynomial time whether there *exists* a binary-valued SCP-net that *entails* all examples, the problem of deciding whether there exists such a network that is *weakly consistent* with all examples is NP-complete.

Meanwhile, Koriche and Zanuttini explored a somewhat different learning problem, that of *active elicitation*. They framed the problem as one of Angluin-style learning, attempting to elicit CP-nets via adaptively generated *swap queries*, i.e., comparisons in which the two outcomes differ in the value of just one variable (as in flipping sequences) [58, 59]. Among their contributions was introducing the concept of *query complexity*. Rather than defining complexity in terms of computation size, they defined an *attribute efficient algorithm* as one for which the *number of queries* is polynomial in the number of variables. They also described Angluin-style *membership* and *equivalence queries* for CP-nets: the relationship between the two types of queries is analogous to that of ordering queries and dominance testing for reasoning with CP-nets. Koriche and Zanuttini showed that it was not possible to learn a CP-net with equivalence queries alone, but that membership queries were also required. They presented an algorithm showing it was possible to learn binary-valued tree-structured CP-nets in an efficient manner given this definition.

Dimopoulos et al. continued work on the problem of learning CP-nets passively from data, improving on their research from a few years before. They introduced an algorithm

that attempted to learn an acyclic, binary-valued CP-net that was *consistent* with all outcome comparisons in a preexisting database [29]. Their algorithm first attempted to identify variables over which the subject's preferences were unconditional (those with 0 parents), then variables with 1 parents, 2, and so on, until a node for each variable had been added to the model.

A crucial step for each node involves determining whether a prospective set of nodes could be the parents of the node under consideration. To determine this, the authors propose constructing a 2SAT instance, which is solvable in linear time, such that the solution provides a CPT and establishes a position in the network for the node that is consistent with all available comparison data. However, in some cases the algorithm may output failure instead of a CP-net. Additionally, in the worst case the algorithm could iterate an exponential number of times in the number of variables. They also proved that the problem of learning a CP-net consistent with comparison data was hard even for acyclic, binary CP-nets to which other simplifying assumptions had been applied.

Recently Guerin et al. [47, 48] have proposed an elicitation algorithm for learning CP-nets from user queries. The algorithm is similar in many respects to that of Dimopoulos and Athienitou, but employs active elicitation rather than passive learning. The algorithm is also distinctive in that it allows subjects to introspect on their preferences by asking for a default, most-preferred value for each variable.

One of the problems inherent in learning preferences from human subjects is the possibility of *noise* or comparison data that are ultimately inconsistent through transitive closure. Liu et al. [70] propose maximizing the number of outcome comparisons that can be included using a *branch-and-bound* approach. The method they propose—which is not restricted to binary-valued domains—first learns a *preference graph* for the subject, then constructs a CP-net from the preference graph. They claim that their algorithm runs in time polynomial in the size of the preference graph. However, recall that the size of the preference graph is exponential in the number of features. Moreover, because the algorithm

employs a branch-and-bound method, it seems unlikely that it is polynomial even in the size of the preference graph; it seems more likely that the worst-case running time is actually doubly exponential in the number of features. In their conclusion, Liu et al. mention the possibility of exploiting an *approximation method* rather than branch-and-bound for learning CP-nets from possibility inconsistent data. While this is an interesting proposal, I am not yet aware of any algorithm that exploits such a method or of theoretical results on the complexity of the problem of approximating CP-nets.

Finally, Cornelio et al. have explored the possibility of updating a learned CP-net after additional data have been collected, without recreating it entirely [25, 26].

3.6 Experiments with CP-nets

Proposed CP-net algorithms are typically evaluated with theoretical proofs, experiments, or both; Table 3.3 summarizes several studies involving CP-net learning or reasoning and the methods of evaluation. Ideally, algorithms would be evaluated at least in part using actual CP-nets obtained from human subjects. Unfortunately, no such datasets are presently available [3]. Datasets from which CP-nets can be learned are also problematic. The PrefLib preference data repository [75], for example, provides only two combinatorial preference datasets. One consists of approval ballots and ratings for candidates in the 2002 French election; the other consists of hotel reviews submitted to the Trip Advisor travel site. However, it is unclear how these could be used to construct CP-nets for an experiment, and to date neither dataset has been cited in any CP-net study. The SUSHI preference dataset [53], consisting of user's ratings, ranking, and the feature composition of different types of sushi, has been adapted by at least one team of authors for their CP-net learning algorithms [69, 70]. However, this sort of adaptation, while fairly common in the computational social choice and preference learning communities, involves numerous data modeling decisions that can undermine the validity of such experiments [81, 84]. While Allen et al. [3] describe a rigorous psychological experiment on whether human prefer-

Table 3.3: Evaluation Methods for Proposed CP-net Algorithms

Study	Reasoning	Learning	Proofs	Synthetic Data	Repurposed Data
Allen [1]	✓	✓	✓	✓	
Bigot et al. [11]	✓		✓		
Bigot et al. [12]		✓		✓	
Boutilier et al. [16]	✓		✓		
Dimopoulos et al. [29]		✓	✓		
Eckhardt and Vojtáš [33, 34]		✓		✓	
Guerin et al. [47]		✓		✓	
Koriche and Zanuttini [58, 59]		✓	✓		
Kronegger et al. [61]	✓		✓	✓	
Li et al. [68]	✓		✓	✓	
Liu et al. [69, 70]		✓	✓	✓	✓
Santhanam et al. [92]	✓		✓	✓	

ences can be consistently modeled with CP-nets, the results of that experiment are not yet available. Therefore, due to the lack of suitable real-world data, researchers often rely on synthetic datasets. However, as discussed in detail in Chapter 4, synthetic data are also problematic due to naïve generation methods.

3.7 Extensions to the Formalism

We conclude with a brief discussion of a few of the many proposed extensions to the CP-net formalism—GCP-nets, TCP-nets, *m*CP-nets, and PCP-nets.

- **General CP-nets (GCP-nets)** dispense with the explicit graphical structure of classical CP-nets and consist only of a set of conditional preference rules. Such a formalism is helpful for networks that may contain cycles and be highly complex in the interrelationships among nodes. As such, GCP-nets can facilitate the proof of certain complexity results and were employed both in the PSPACE-completeness proofs of Goldsmith et al. [42, 43] and in the FPT results of Kronegger et al. [61].

- **Tradeoffs-enhanced CP-nets (TCP-nets)**, introduced by Brafman et al. [19], extends the CP-net formalism by allowing the model to reflect the relative importance assigned to the features. In addition to the directed edges indicating conditional dependencies, the model includes two additional types of edges: *importance arcs*, directed edges showing that one variable is more important to the subject's satisfaction than the other, and *conditional importance arcs*, indicating the relative importance of variables given the values assigned to other nodes in the graph. While the proposed formalism is highly expressive, its greater complexity (at least for human users) has perhaps limited its adoption.

- ***m*CP-nets** were conceived by Rossi et al. [89] as a formalism for representing the preferences of multiple agents. The preferences of m individual agents are represented as a *partial CP-net*, and the semantics of the aggregated *m*CP-net are related to voting. Recently, Lukasiewicz and Malizia [72] have studied the computational complexity of various problems involving *m*CP-nets.

- **Probabilistic CP-nets (PCP-nets)** were proposed by Cornelio [26] and later refined by Bigot et al. [11] and Cornelio et al. [25]. Whereas CP-nets are deterministic, PCP-nets assign a probability to every conditional preference rule. PCP-nets are presently limited to $\mathcal{O}$*-legal* structures with strict preferences over binary domains and complete CPTs. Such networks assume that the probabilities of each node are independent and hence that the probabilities of edges can be calculated using the product rule. Cornelio [26] has also explored the connection between PCP-nets and Bayesian networks. Based on this close relationship with Bayesian networks, Bigot et al. [12] have subsequently shown that PCP-nets can be learned in polynomial time from a set of *optimal* outcomes when the dependency graph is tree-shaped.

Chapter 4 Generating CP-nets Uniformly at Random

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 [55]—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 [62]. However, 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 black-box 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 [3, 75]. Principled methods exist to generate simulated data in social choice and preference handling using *generative cultures* [10, 73, 100]. Such cultures have their drawbacks and limitations [81, 84], but provide a first step in experimentation for fields where data are hard to gather. While generative cultures over strict, linear 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.

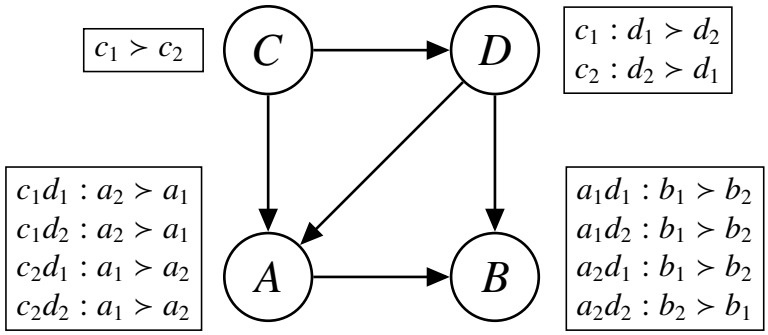
This chapter introduces a method for generating acyclic CP-nets uniformly at random. A key idea of this method is that the structure of a CP-net is equivalent to a tuple of sets representing the parents of nodes in the network. We will consider how to enumerate all such *dagcodes*, as these tuples are known [99],[1] and how to calculate the number of CP-nets—the possible graphs and conditional preference tables (CPTs)—that extend a partial dagcode. The resulting novel recurrence makes it possible to generate the graph and CPTs, node by node, such that all CP-nets with a given domain size and bound on indegree are equiprobable.

Section 4.1 highlights two problems, bias and degeneracy, that result from commonly used naïve generation methods. Section 4.2 shows how to encode and avoid degeneracy in the CPTs. Section 4.3 explains how to encode and count the dependency graphs. These results are then brought together in Section 4.4 to create an algorithm that samples the space of CP-nets uniformly. Section 4.5 shows how to sample uniformly from outcomes and the set of dominance testing problem instances. Note that in this chapter it is assumed that domains are *homogeneous* but possibly multivalued, i.e., $d = d_1 = \cdots = d_n$, where $d_i = |\mathrm{Dom}(X_i)|$, for all $X_i \in \mathcal{V}$, and that the CPTs are *complete*, i.e., have d^m rules, one for every assignment to the m parents, $m = |\mathrm{Pa}(X_i)|$. Notation is discussed in detail in Chapter 2; in particular, see Table 2.2.

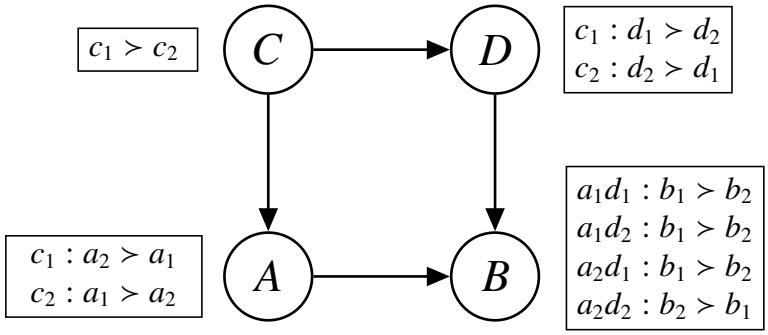
4.1 Naïve Generation, Bias, and Degeneracy

If one wants to generate CP-nets without regard for the resulting distribution, many simple random methods exist. For example, initialize a CP-net with n nodes, no edges, and empty CPTs; choose a random subset of pairs (X_h, X_i), $h < i$, inserting an edge from each X_h to X_i; generate a CPT for each X_i with $d^{|\mathrm{Pa}(X_i)|}$ rules, each a random permutation of the d values of X_i; and randomly permute the n labels. One suspects that something along these lines is meant when a paper states, "We generated 1000 CP-nets at random." However, this naïve

[1] I use *dagcode* rather than *DAG code* (as in [99]) so as to better emphasize whether I am referring to the encoding or to the DAG itself.



(a) Degenerate CP-net



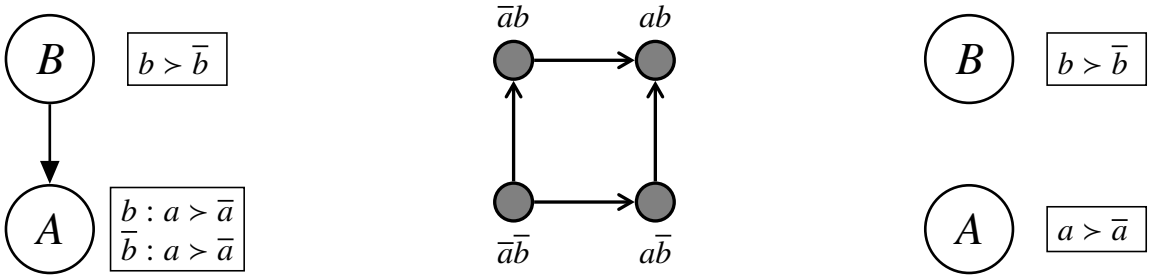
(b) Equivalent Non-Degenerate CP-net

Figure 4.1: An Example of Degeneracy in CP-nets

approach to generation leads to two problems, *degeneracy* and *bias*. Let us first consider the problem of *degeneracy*, which occurs when one or more dependencies in the graph are not reflected in the conditional preference rules (CPRs).

Example 17. *Consider the CP-net in Figure 4.1. The edge* (D, A) *in Figure 4.1a indicates that the preference over the values of A depends on the value of D. However, in examining the CPT of A closely, one can observe that the preference over A does not in fact depend on D. The preferences can thus be represented by the simpler CP-net shown in Figure 4.1b.*

Degeneracy in synthetic datasets is problematic for two reasons. First, dependencies in the graph, such as edge (D, A) in Example 17, can be *fictional*; that is, the presence of an edge in the graph does not necessarily express any factual information about the induced preference order. Second, if degeneracy can occur, multiple, apparently different CP-net models can map to the same induced preference order.



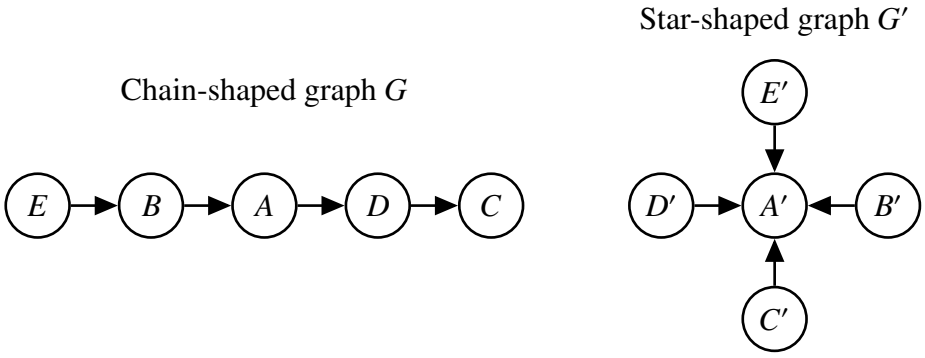
(a) One Edge (b) Induced PG (c) No Edge

Figure 4.2: Degeneracy Can Violate Basic Assumptions of an Experiment

Example 18. *Suppose a researcher wants to test a new DT algorithm to understand how running time varies as the number of edges in the network increases. (Also, he thinks a nice bar chart will impress reviewers.) As a first step, he generates two sets of CP-nets using a naïve approach. The CP-nets in each set have binary domains and two nodes; the first set has just one edge, and the second set has no edge. However, if each CPR is assigned in the manner of a coin flip, an expected 50% of the CP-nets in the first set will be degenerate like the one in Figure 4.2a. As such, its induced preference graph, shown in Figure 4.2b, will be identical to that of the no-edge CP-net in Figure 4.2c. Thus, one of the basic assumptions of the experiment, that the two sets induce different preference orders, is violated.*

Naïve generation methods also result in another problem that can violate the basic assumptions of experiments: *statistical bias*. To understand this bias, consider the following dependency graphs and their associated CP-net counts[2] as shown in Figure 4.3. Both graphs have 5 nodes and 5 edges, but the number of CP-nets associated with each graph differs greatly. Observe that for the chain-shaped graph on the left, when $d = 2$, there are just two ways to choose each of the n CPTs such that they are consistent with the dependency graph. The CPT of root E could be $[e_1 \succ e_2]$ or $[e_2 \succ e_1]$. The other nodes, each of which has only one parent, also have two (non-degenerate) possibilities for their CPT; e.g., CPT(A) could be $[b_1 : a_1 \succ a_2,\ b_2 : a_2 \succ a_1]$ or $[b_1 : a_2 \succ a_1,\ b_2 : a_1 \succ a_2]$. However, in the

[2]The CP-net counts for G and G' are $\psi_d(0)(\psi_d(1))^4$ and $(\psi_d(0))^4\psi_d(4)$, respectively, where $\psi_d(m)$ is the number of non-degenerate CPTs with m parents and d-ary domains, as discussed in Section 4.2.



(a) Dependency Graphs that Differ in Maximum Indegree

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

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

Figure 4.3: How Naïve Generation Can Lead to Bias

case of the star-shaped graph on the right, CPT(A') has $d^4 = 16$ rules, each with $d! = 2$ possible orderings. In all, over one million CP-nets have the graph on the right, while only 32 have the graph on the left. Further observe that the ratio of this imbalance very rapidly increases with the domain size d. Thus, if the algorithm above in fact generated the two graphs with equal likelihood, it would grossly oversample CP-nets with the first graph, while correspondingly undersampling those with the second.

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

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

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

Figure 4.4: CPT and Corresponding Boolean Function

4.2 Counting and Generating the CPTs

The notion of a degenerate CPT introduced in Section 4.1 can be generalized with the help of a *bijection* (a mapping that is *one-to-one* and *onto*) with discrete multivalued functions. One can model each CPT(X_i) as a function $f : \{0, \ldots, d-1\}^m \rightarrow \{0, \ldots, d!-1\}$, where $m = |\text{Pa}(X_i)|$. The inputs correspond to the values of the m parents of X_i. The output corresponds to one of the $d!$ orders of the domain of X_i.

Observe that if variables are binary ($d = 2$), f is a Boolean function. In 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 outputs 0 and 1. One can thus model the degenerate CPT of node A from Example 17 with the truth table in Figure 4.4.

If variables are multivalued, the mapping is similar, as illustrated in Figure 4.5 for a CPT with two parents and three-valued domains. Note that the inputs to the multivalued function are in the range $(0\,..\,d-1)$, while the outputs are in the range $(0\,..\,d!-1)$. For mapping the outputs, one can use Lehmer codes [67] (see also the discussion of the factorial number system by Knuth [56, 3.2.2 Alg. P]), as illustrated in the table on the right-hand side of Figure 4.5 for $d = 3$.

Thus, any CPT can be encoded as an equivalent *function vector* F of length d^m. Note that the values in the input columns of Figures 4.4 and 4.5 are only for the benefit of the human reader and need not be represented explicitly. The entries in a function vector are ordered according to a radix d positional numbering system, where each digit corresponds to the value of a parent variable. Of course, to produce the corresponding CPT from a vector, one would also need to know the labels of the variable, parent variables, and domains.

CPT(C)
$a_1b_1 : c_1 \succ c_2 \succ c_3$
$a_1b_2 : c_1 \succ c_2 \succ c_3$
$a_1b_3 : c_1 \succ c_3 \succ c_2$
$a_2b_1 : c_1 \succ c_2 \succ c_3$
$a_2b_2 : c_1 \succ c_2 \succ c_3$
$a_2b_3 : c_1 \succ c_3 \succ c_2$
$a_3b_1 : c_1 \succ c_2 \succ c_3$
$a_3b_2 : c_1 \succ c_2 \succ c_3$
$a_3b_3 : c_3 \succ c_2 \succ c_1$

In_1	In_2	F
0	0	0
0	1	0
0	2	1
1	0	0
1	1	0
1	2	1
2	0	0
2	1	0
2	2	5

Decimal rank	Factoradic numeral	Lehmer code	Ranking $\succ^i$ of Dom(X_i)
0	$000_!$	(0, 1, 2)	$x_1 \succ x_2 \succ x_3$
1	$010_!$	(0, 2, 1)	$x_1 \succ x_3 \succ x_2$
2	$100_!$	(1, 0, 2)	$x_2 \succ x_1 \succ x_3$
3	$110_!$	(1, 2, 0)	$x_2 \succ x_3 \succ x_1$
4	$200_!$	(2, 0, 1)	$x_3 \succ x_1 \succ x_2$
5	$210_!$	(2, 1, 0)	$x_3 \succ x_2 \succ x_1$

Figure 4.5: Multivalued CPT and Mapping

The mapping helps us formalize the notion of degeneracy introduced in Section 4.1.

Definition 19 (Degeneracy)**.** *A function $f(u)$ is* vacuous[3] *in variable u_k if and only if its output never depends on u_k; i.e., for all $u \in \{0, \ldots, d-1\}^m$,*

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

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

Let us denote by $\phi_d(m)$ the total number of distinct CPTs for a node with m parents. Let us denote by $\chi_d(m)$ the number of those that are degenerate, and by $\psi_d(m)$ the number that are non-degenerate. It follows immediately from Definition 19 that

$$\phi_d(m) = \chi_d(m) + \psi_d(m). \tag{4.1}$$

First consider binary domains, $d = 2$. Because CPTs and Boolean functions are in one-to-one correspondence, $\phi_2(m)$ is equivalent to the number of Boolean functions of m inputs, and $\psi_2(m)$ is equivalent to the number of non-degenerate Boolean functions. Hu

[3]Such a variable is sometimes said to be *vacated* or *fictional* [79].

Table 4.1: Values of $\phi_2(m)$ and $\psi_2(m)$ for $m = 0$ to 5

m	$\phi_2(m)$	$\chi_2(m)$	$\psi_2(m)$	$\frac{\chi_2(m)}{\phi_2(m)}$	$\frac{\psi_2(m)}{\phi_2(m)}$
0	2	0	2	$0.000\overline{0}$	$1.000\overline{0}$
1	4	2	2	$0.500\overline{0}$	$0.500\overline{0}$
2	16	6	10	$0.375\overline{0}$	$0.625\overline{0}$
3	256	38	218	0.1484	0.8516
4	65,536	942	64,594	0.0144	0.9856
5	4,294,967,296	325,262	4,294,642,034	0.0001	0.9999

[51, §2] (see also the work of Harrison [50] and O'Connor [79]) proved that for Boolean functions

$$\phi_2(m) = 2^{2^m}, \tag{4.2}$$

and

$$\psi_2(m) = \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} 2^{2^k}. \tag{4.3}$$

Table 4.1 shows the values of $\phi_2(m)$ [98, A001146] $\psi_2(m)$ [98, A005530], and $\psi_2(m)$ [98, A000371] for small m, as well as the ratios of $\psi_2(m)$ and $\chi_2(m)$ to $\phi_2(m)$, approximated to four decimal digits. From these ratios, one may conjecture the results in the limit, also proved by Hu [51, §10],

$$\lim_{m\to\infty} \frac{\chi_2(m)}{\phi_2(m)} = 0, \tag{4.4}$$

$$\lim_{m\to\infty} \frac{\psi_2(m)}{\phi_2(m)} = 1. \tag{4.5}$$

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

Theorem 20 (Number of CPTs)**.** *For every non-negative integer d, m,*

$$\phi_d(m) = d!^{d^m}. \tag{4.6}$$

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

Is-CPT-Degenerate(N, X_i)

Input: CP-net N
Node X_i
Output: returns **true** if CPT(X_i) is degenerate, **false** if non-degenerate

1: **for all** $X_h \in \mathrm{Pa}(X_i)$ in CP-net N **do** ▷ Iterate over parents
2: *vacuous* ← **true** ▷ Assume for now X_h is fictional
3: **for all** $x_k^h \in \mathrm{Dom}(X_h)$, $k > 1$ **do** ▷ All domain values except x_1^h
4: **if** $\mathrm{CPT}(X_i \,|\, X_h = x_k^h) \neq \mathrm{CPT}(X_i \,|\, X_h = x_1^h)$ **then**
5: *vacuous* ← **false** ▷ Output depends on X_h, so
6: **break** ▷ move on to next parent
7: **end if**
8: **end for**
9: **if** *vacuous* **then**
10: **return true** ▷ Since CPT(X_i) is vacuous in parent X_h, it is degenerate
11: **end if**
12: **end for**
13: **return false** ▷ CPT(X_i) depends on all parents; thus it is non-degenerate

Figure 4.6: Algorithm: Decide Whether a CPT Is Degenerate

Theorem 21 (Number of Non-degenerate CPTs)**.** *For every non-negative integer d, m,*

$$\psi_d(m) = \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} d!^{d^k}. \tag{4.7}$$

Theorem 22 (Convergence to Non-degeneracy)**.** *For every non-negative integer d, m,*

$$\lim_{m\to\infty} \frac{\chi_d(m)}{\phi_d(m)} = 0, \tag{4.8}$$

$$\lim_{m\to\infty} \frac{\psi_d(m)}{\phi_d(m)} = 1. \tag{4.9}$$

The proofs of Theorems 21 and 22 very closely follow the rather lengthy proofs of Hu [51][§10] for Boolean functions, except that ϕ_d is needed in place of ϕ_2; the primary change is to replace every occurrence of 2^{2^k} with $d!^{d^k}$.

Figure 4.6 describes an algorithm that decides whether a CPT is degenerate. The loop in Line 1 iterates over the m parents of X_i. If the preference over X_i is the same for all values of parent X_h,

$$\mathrm{CPT}(X_i \,|\, X_h = x_1^h) = \mathrm{CPT}(X_i \,|\, X_h = x_2^h) = \cdots = \mathrm{CPT}(X_i \,|\, X_h = x_m^h), \tag{4.10}$$

then the CPT is vacuous in X_h. In that case the inner loop completes with variable *vacuous* still set to **true**, and the program exits at Line 10 to report that the CPT is degenerate. However, if Equation 4.10 does not hold, then *vacuous* will be set to **false** at Line 9, the inner loop will terminate early, and execution will proceed to the next parent variable. If the main loop terminates, the CPT depends on every parent variable; thus, it is non-degenerate.

Because every CPT has a corresponding function vector F of length d^m, as discussed above, it is possible to test F for degeneracy directly without having to keep track of the parent labels or to parse expressions such as $a_2 b_3 : c_1 \succ c_3 \succ c_2$. The algorithm described in Figure 4.7 returns **true** if such a vector is degenerate.

Theorem 23 (Degeneracy Can Be Decided Efficiently)**.** Is-Function-Degenerate *runs in polynomial time.*

Proof. The outermost loop at Line 1 executes at most m times, and the nested **for** loop at Line 3 executes at most $d - 1$ times. The **while** loop at Line 7 requires some discussion. The variables r and s iterate over indices corresponding to all d-ary inputs of the form

$$(\alpha_0, \ldots, \alpha_{h-1}, 0, \alpha_{h+1}, \ldots, \alpha_{m-1}),$$
$$(\alpha_0, \ldots, \alpha_{h-1}, k, \alpha_{h+1}, \ldots, \alpha_{m-1}),$$

where $\alpha_j \in \{0, \ldots, d-1\}$, for $j \in \{1, \ldots, m-1\}$, $j \neq k$. Observe that r and s are non-negative, strictly increase on each iteration (see Lines 11–12 and Lines 15–16), and are bounded by d^m. Thus, the **while** loop at Line 7 executes at most d^{m-2} times, and the overall running time of the algorithm is $O(md^{m-1})$. Note that the size of the input is $O(d^m)$, i.e., the length of vector F. Therefore, the running time is polynomial in the size of the input. ❑

Note that in the proof the assumption is made, as usual, that arithmetic and memory operations can be performed in constant time. However, because the elements of F are integers in the range $\{0, \ldots, d! - 1\}$, this assumption may not hold if d is arbitrarily large. Recall our assumption that d is a small constant. Also, as a practical matter, observe that $d!$

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

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

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

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

Figure 4.7: Algorithm: Decide Whether Function Vector F is Degenerate

Table 4.2: Odds of Generating a Degenerate Function at Random on a Given Attempt

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

can be represented by a 64-bit unsigned integer for $d \leq 20$, since $20! < 2^{64}$. Note also the proof of O'Connor [79], that deciding whether a Boolean function is vacuous in a variable (and hence degenerate) is Co-NP-complete. However, this assumes the input takes the form of arbitrary Boolean expressions, whereas in our case the input is already exponential in m, since it is assumed that CPTs are complete.

Let us now consider how to leverage these results to generate non-degenerate CPTs in an efficient, uniformly random manner. For tiny values of d and m, one can choose uniformly from a modest-sized table of non-degenerate functions (e.g., $\psi_2(4) = 64594$). For larger values, one can use *rejection sampling*, generating a random integer in the range $(0\,..\,d! - 1)$ for each of the d^m elements of vector F and repeating this process in the *unlikely* event (e.g., < 0.0001 for $m > 4$ and very rapidly converging to 0 as m increases) that the result is degenerate (see Table 4.2). With probability $\psi_d(m)/\phi_d(m)$, asymptotic to 1, a non-degenerate CPT is obtained on a given attempt. Finally, observe that it is possible to generate *all* non-degenerate CPTs by generating all $\phi_d(m)$ vectors F and outputting the corresponding CPT only when Is-Function-Degenerate(F) answers **false**.

4.3 Encoding and Counting Dependency Graphs

This section considers how to model the dependency graph as a *dagcode* [99], inspired by Prüfer codes for labeled trees [60]. The encoding makes it easier to count the number of ways to complete a partially constructed DAG in order to avoid bias. In this section the dagcode is first treated as an abstraction and then related to the dependency graph.

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** ▷ Iterate over dagcode: A_j is the parent set
5: $\quad i \leftarrow \max\left(Q \setminus \bigcup_{k=1}^{j} A_k\right)$ ▷ of X_i, where i is the largest unused label
6: $\quad$ **for all** $h \in A_j$ **do**
7: $\quad\quad$ insert edge to X_i from its parent X_h
8: $\quad$ **end for**
9: $\quad Q \leftarrow Q \setminus \{i\}$
10: **end for**
11: output DAG G

Figure 4.8: Algorithm: Generate a DAG from its Dagcode

Definition 24 (Dagcode)**.** *For any positive integer n, a* dagcode $A = \langle A_1, \ldots, A_{n-1} \rangle$ *is a tuple of* $n - 1$ *subsets* $A_j \subset \{1, \ldots, n\}$ *that satisfy the* cardinality constraint

$$\left| \bigcup_{k \leq j} A_k \right| \leq j \tag{4.11}$$

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

Observe from Definition 24 that tuples $\langle \{1\}, \{1, 3\} \rangle$ and $\langle \{3\}, \emptyset \rangle$ are valid dagcodes (in which $n = 3$), but $\langle \{1, 2\}, \emptyset \rangle$ and $\langle \emptyset, \{1, 2, 3\} \rangle$ are not, since each violates the cardinality constraint. Steinsky [99] proved that dagcodes correspond one-to-one with DAGs and described efficient algorithms for converting dagcodes to DAGs and vice versa. The algorithm shown in Figure 4.8 maps an encoding A to its corresponding graph G.[4]

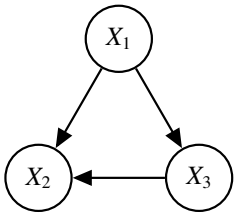
Applied to CP-nets, each subset $A_j \subset \{1, \ldots, n\}$ in the dagcode corresponds to the parents of some node X_i in the dependency graph: i.e., $h \in A_j \implies X_h \in \text{Pa}(X_i)$. Note

[4] I have modified the algorithm of Steinsky [99] slightly so as to remove the node with the *largest* rather than the smallest label. One can confirm that this change has no effect on the one-to-one correspondence between dagcodes and DAGs. Removing the node with the smallest label offers a closer analogue to Prüfer codes. However, DAGs, unlike (undirected) trees, impose a topological ordering on the nodes. By removing the node with the largest label, I map an encoding such as $\langle \{1\}, \{1, 2\}, \{1, 2, 3\} \rangle$ to a graph in which the node labels are topologically ordered $1 \blacktriangleright 2 \blacktriangleright 3 \blacktriangleright 4$, which I find more aesthetically appealing than $4 \blacktriangleright 3 \blacktriangleright 2 \blacktriangleright 1$.

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



The subsets $\{1\}$ *and* $\{1,3\}$ *indicate that one node has parent* X_1 *and another has parents* X_1 *and* X_3*; the third (implicit) node is a root. The mapping from parent sets to their children can be recovered from* Dagcode-to-DAG *(Figure 4.8) [99, 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 if and only if

$$|A_j| \leq c \tag{4.12}$$

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.

The generation method in Section 4.4 depends on counting the number of *extensions* to a partially specified graph. Consider a partial encoding $A_{<3} = \langle\{1\},\{2\},\_\_\rangle$ of a graph with

$n = 4$ nodes and bound $c = 1$ on 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 $\{4\}$.[5] One can generalize this as follows.

Definition 26 (Partial dagcode)**.** *A* partial dagcode $A_{<j} = \langle A_1, \ldots, A_{j-1}, \_, \ldots, \_ \rangle$ *is a dagcode for which only elements* A_1 *through* A_{j-1} *have been specified, such that*

$$\left| \bigcup_{\ell \leq k} A_\ell \right| \leq k \tag{4.13}$$

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

A partial dagcode is said to respect a bound c on indegree *when*

$$|A_k| \leq c \tag{4.14}$$

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

The algorithm shown in Figure 4.9 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, All-DAGs$(n, c, 1, 0, \emptyset, A_{<1})$ is called.

Theorem 27. All-DAGs *generates each DAG exactly once.*

Proof. Because dagcodes are in one-to-one correspondence with DAGs [99, Cor. 1], it suffices to show that each dagcode is generated exactly once. For this let us use the *recursion invariant:* Each time Line 1 is reached, $A_{<j}$ is valid; that is, for all k, $1 \leq k < j$, $\left|\bigcup_{\ell \leq k} A_\ell\right| \leq k$ and $|A_k| \leq c$ to satisfy Equations 4.13 and 4.14. The proof will show that under this assumption, Figure 4.9 generates each A_j such that the invariant holds for $A_{<j+1}$.

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

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

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

Inputs:

n	number of nodes
c	bound on 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:     DAGCODE-TO-DAG(A_<n)
3:     return
4:  end if
5:  for all s, t ≥ 0, s ≤ q, s + t ≤ c, q + t ≤ j do
6:     for all S ⊆ U, |S| = s do
7:        for all T ⊆ Ū, |T| = t do
8:           A_j ← S ∪ T; include A_j with A_<j to form A_≤j
9:           ALL-DAGs(n, c, j + 1, q + t, U ∪ A_j, A_≤j)
10:       end for
11:    end for
12: end for
```

Figure 4.9: Algorithm: Generate All DAGs that Extend Dagcode $A_{<j}$

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

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

From All-DAGs it is possible to derive a new recurrence for the number of DAGs that is more easily extended to CP-nets than those of Robinson [87] and Steinsky [99]. Let us denote by $a_{n,c}$ the *number of DAGs* (respectively dagcodes) with n nodes and bound c on indegree, and by $a_{n,c}(j,q)$ the number of extensions to a partial dagcode $A_{<j}$, where $q \equiv \left|\bigcup_{k<j} A_k\right|$. That is, $a_{n,c}(j,q)$ is the number of ways to choose the remaining elements $A_j, \ldots, A_{n-1}$ such that the cardinality and indegree constraints in Equations 4.13 and 4.14 are satisfied.

Theorem 28 (Number of DAGs)**.** *For all non-negative integers n and c,*

$$a_{n,c} = a_{n,c}(1,0). \tag{4.15}$$

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

$$a_{n,c}(j,q) = \sum_{\substack{s\geq 0,\ t\geq 0,\\ s\leq q,\ s+t\leq c,\\ q+t\leq j}} \binom{q}{s}\binom{n-q}{t} a_{n,c}(j+1, q+t). \tag{4.16}$$

For $j = n$*,*

$$a_{n,c}(j,q) = 1. \tag{4.17}$$

Proof. (Strong induction.) In the proof of Theorem 27, the proof employed a form of strong induction on j increasing. To show that Equation 4.16 is correct, let us again use strong induction, this time on j decreasing.

Base case ($j = n$): One DAG is generated at Line 3; hence, $a_{n,c}(n,q) = 1$ for all q, as claimed in Equation 4.17.

Inductive hypothesis: Assume $a_{n,c}(j',q')$ gives the correct count for $j' > j$ and all q'. The proof will show that the resulting count for $a_{n,c}(j,q)$ is also correct. Observe that, whatever the size of set $U \subset V$, the loop at Line 6 iterates over the $\binom{q}{s}$ ways to choose s elements from U. Similarly, the loop at Line 7 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

Table 4.3: Number of DAGs $a_{n,c}$ with n Nodes and Bound c on Indegree

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

Line 5, one can take the sum of the DAGs generated in the loop body, obtaining the result given in Equation 4.16.

Finally, observe that all dagcodes parameterized by n, c extend the fully *un*specified dagcode $A_{<1} = \langle \_, \ldots, \_ \rangle$, for which $j = 1$ and $q = 0$. Thus, $a_{n,c} = a_{n,c}(1, 0)$, the result given in Equation 4.15. ❑

One can verify that for DAGs with unbounded indegree ($c \geq n - 1$), the recurrence yields the sequence 1, 1, 3, 25, 543, 29281, 3781503, 1138779265, ..., as expected ([98, A003024]). Table 4.3 gives values of $a_{n,c}$ from Equation 4.15 for $n = 1$ to 7 and $c < n$.

4.4 Generating CP-nets

The insights of Section 4.2 can be used to extend ALL-DAGs (Figure 4.9) to obtain a new algorithm that generates ALL-CP-NETS, presented in Figure 4.10. CP-nets with the same dependency graph differ if any rule of a CPT differs. To generate all combinations of CPTs, one needs only introduce a new innermost loop iterating over the possibilities, as described at the end of Section 4.2. Note that since the dagcode is partial, there is not yet enough information to construct the CPT: the parents are known, but the label of the child to which they belong and its domain values are not. However, observe that sufficient information is available to iterate over the corresponding function vectors F_j, since the number of parents ($|A_j| = s + t$) and the size (d) of every domain is known, so we do that instead.

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

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

1: **if** $j = n$ **then**
2: BUILD-CP-NET($A_{<n}, \boxed{F_{<n}}$)
3: **return**
4: **end if**
5: **for all** $s, t \geq 0,\ s \leq q,\ s + t \leq c,\ q + t \leq j$ **do**
6: **for all** $S \subseteq U,\ |S| = s$ **do**
7: **for all** $T \subseteq V \setminus U,\ |T| = t$ **do**
8: **if** $j > 0$ **then**
9: $A_j \leftarrow S \cup T$
10: include A_j with $A_{<j}$ to form $A_{\leq j}$
11: **end if**
12: **for all** vectors F_j of length $d^{|A_j|}$ with elements in the range $(0 \,..\, d! - 1)$ **do**
13: **if not** IS-FUNCTION-DEGENERATE(F_j) **then**
14: ALL-CP-NETS($n,\ c,\ d,\ j + 1,\ q + t,\ U \cup A_j,\ A_{\leq j},\ F_{\leq j}$)
15: **end if**
16: **end for**
17: **end for**
18: **end for**
19: **end for**

Note: The boxes highlight the differences from the algorithm in Figure 4.9.

Figure 4.10: Algorithm: Generate All CP-nets that Extend $A_{<j}$

Each F_j is included in a tuple $F = \langle F_0, \ldots, F_{n-1} \rangle$ that I call a *cpt-code*. (The expressions $F_{<j}$ and $F_{\leq j}$, analogous to $A_{<j}$ and $A_{\leq j}$, are used here for a *partial cpt-code*.) Since a root node is implicit in the dagcode, F contains an additional element F_0 corresponding to that node's CPT,[6] and ALL-CP-NETS is invoked with $j = 0$ instead of 1:

$$\text{ALL-CP-NETS}(n, c, d, 0, 0, \emptyset, \langle \_, \ldots, \_ \rangle, \langle \_, \ldots, \_ \rangle). \tag{4.18}$$

When $j = n$, the encoding is complete: A and F fully and uniquely characterize a CP-net

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

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

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

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

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

Note: The boxes highlight the differences from the algorithm in Figure 4.8.

Figure 4.11: Algorithm: Construct CP-net from its Encoding

N. BUILD-CP-NET is then called, as shown in Figure 4.11 (analogous to DAGCODE-TO-DAG in Figure 4.8) to decode it—the DAG from A, the CPTs from F.

Theorems 27 and 28 can similarly be extended to CP-nets.

Theorem 29. ALL-CP-NETS *generates each CP-net exactly once.*

Proof. Observe that ALL-CP-NETS (Figure 4.10) is identical to ALL-DAGS (Figure 4.9) insofar as the graph is concerned. In the proof of Theorem 27, it has already been shown that each DAG is generated just once and that the algorithm terminates. Thus ALL-CP-NETS generates CP-nets for every possible dependency graph with n nodes and bound c on indegree.

The principle difference from All-CP-nets is the inclusion of a new innermost loop at Line 12 iterating over all possible function vectors F_j, such that F_j is non-degenerate. Note that these correspond to all possible CPTs for the current node via the mapping described in Section 4.2. Further note that each possible CPT for the root node is also generated in the innermost loop, since the algorithm is called with $j = 0$. Thus, if $A_{<j}$ and $F_{<j}$ are valid, $A_{\leq j}$ and $F_{\leq j}$ will also be valid, and each A_j and F_j will be generated exactly once, for all j, such that $0 \leq j < n$. Therefore every non-degenerate CP-net with n nodes, bound c on indegree, and d-ary domains will be generated exactly once. ❑

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

Theorem 30 (Number of CP-nets)**.** *For all non-negative integers n, c, and d,*

$$a_{n,c,d} = a_{n,c,d}(0, 0). \tag{4.19}$$

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

$$\sum_{\substack{s\geq 0,\ t\geq 0,\\ s\leq q,\ s+t\leq c,\\ q+t\leq j}} \binom{q}{s}\binom{n-q}{t} \psi_d(s+t)\ a_{n,c,d}(j+1, q+t). \tag{4.20}$$

For $j = n$*,*

$$a_{n,c,d}(j, q) = 1. \tag{4.21}$$

Note that the loop at Line 12 executes $\psi_d(s + t)$ times, since the indegree of the node modeled by A_j is $s + t$. Otherwise, the proof is nearly identical to that of Theorem 28.

Proof. (Strong induction.) *Base case* ($j = n$): One CP-net is generated at Line 3; hence, $a_{n,c,d}(n, q) = 1$ for all q, as in Equation 4.21.

Inductive hypothesis: Assume $a_{n,c,d}(j', q')$ gives the correct count for $j' > j$ and all q'. The proof will show that the resulting count for $a_{n,c,d}(j, q)$ is also correct. The loop at Line 6 iterates over the $\binom{q}{s}$ ways to choose s elements from U. The loop at Line 7 iterates over the

Table 4.4: Number of Binary CP-nets with Complete CPTs and Unbounded Indegree

Nodes	Number of CP-nets
1	2
2	12
3	488
4	481776
5	157549032992
6	4059976627283664056256
7	52425344846017796047472951749050356669657 6

$\binom{n-q}{t}$ ways to choose t elements from $\overline{U}$. The innermost loop at Line 12 executes $\phi_d(s+t)$ times. However, the call to Is-Function-Degenerate (Figure 4.7) in Line 13 guarantees that Line 14 is reached just $\psi_d(s+t)$ times, once for each non-degenerate CPT. The number of CP-nets generated in the body of the outermost loop at Line 5 depends on s and t, which differ on each iteration. Thus, for all (s,t) one can take the sum of the CP-nets generated in the loop body, to obtain Equation 4.20.

Finally, observe that all CP-nets parameterized by n, c, and d extend the empty dagcode $A_{<0}$ and empty cpt-code $F_{<0}$, for which $j=0$ and $q=0$. Thus, $a_{n,c,d} = a_{n,c,d}(0,0)$, the result given in Equation 4.19. ❑

Table 4.4 shows the number of binary CP-nets with unbounded indegree ($c \geq n-1$) up to 7 nodes (cf. Sloane [98, A250110]). Table 4.5 gives some intuition as to the magnitudes of $a_{n,c,d}$ as bounds on indegree and size of domains increase. From the values, it is evident that generating all CP-nets is feasible only for tiny n, c, and d. To generate larger *random* instances, I propose an efficient method that relies on Equation 4.20. Algorithm Random-CP-net, as shown in Figure 4.12, generates a dagcode one A_j at a time, such that all *CP-nets* (as opposed to DAGs) are equally likely. To satisfy the cardinality constraint, the algorithm keeps track of node labels $U = \bigcup_{k<j} A_k$ that already occur in $A_{<j}$, choosing s labels for A_j from U and the other t from $\overline{U}$, subject to constraints on cardinality and indegree. It also chooses a non-degenerate function F_j for the CPT (see Section 4.2). To avoid bias, (s,t)

Table 4.5: Values of $a_{n,c,d}$ for Small Values of n, c, and d

n	$c = 0$	$c = 1$	$c = 2$	$c = 3$	$c = 4$	$c = 5$
			$d = 2$			
1	2					
2	4	12				
3	8	128	488			
4	16	2000	56240	481776		
5	32	41472	1.31×10^{7}	1.95×10^{9}	1.58×10^{11}	
6	64	1.08×10^{6}	5.21×10^{9}	2.06×10^{13}	2.76×10^{17}	4.06×10^{21}
7	128	3.36×10^{7}	3.16×10^{12}	4.50×10^{17}	1.54×10^{24}	6.71×10^{32}
			$d = 3$			
1	6					
2	36	2556				
3	216	2.43×10^{6}	7.73×10^{10}			
4	1296	3.63×10^{9}	7.79×10^{18}	3.16×10^{32}		
5	7776	7.46×10^{12}	1.67×10^{27}	5.67×10^{54}	1.70×10^{96}	
6	46656	1.95×10^{16}	6.28×10^{35}	2.70×10^{77}	4.91×10^{160}	1.25×10^{286}
7	279936	6.20×10^{19}	3.65×10^{44}	2.70×10^{100}	4.52×10^{225}	5.95×10^{476}
			$d = 4$			
1	24					
2	576	1.59×10^{7}				
3	13824	2.38×10^{13}	5.79×10^{29}			
4	331776	5.61×10^{19}	7.01×10^{52}	4.99×10^{118}		
5	7.96×10^{6}	1.82×10^{26}	1.81×10^{76}	1.88×10^{208}	5.39×10^{472}	
6	1.91×10^{8}	7.50×10^{32}	8.17×10^{99}	1.89×10^{298}	3.14×10^{827}	7.01×10^{1886}
7	4.59×10^{9}	3.76×10^{39}	5.71×10^{123}	3.98×10^{388}	5.81×10^{1182}	5.85×10^{3301}

is chosen such that all extensions to $A_{<j}$ are equally likely, using a table precomputed by Compute-Distribution (Figure 4.13). Build-CP-net (Figure 4.11) outputs the result.

Theorem 31. *For all non-negative integers n, c, and d,* Random-CP-Net(n, c, d) *generates each CP-net N with uniform probability* $P(N) = 1/a_{n,c,d}$.

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

$$\binom{q_j}{s}\binom{n-q_j}{t}\psi_d(s+t)\,\frac{a_{n,c,d}\left(j+1, q_j+t\right)}{a_{n,c,d}\left(j, q_j\right)}. \tag{4.22}$$

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 uniformly at random

1: $F_0 \leftarrow$ random constant function with $d!$ outputs ▷ For CPT of a root node
2: $U \leftarrow \emptyset$
3: $q \leftarrow 0$
4: **for** $j \leftarrow 1$ to $n-1$ **do** ▷ Iterate over dagcode
5: $s, t \leftarrow$ values in cols. 1–2 of a row of $\texttt{DIST}_{n,c,d}(j, q)$
selected randomly according to the weights in col. 3 ▷ Weighted selection
6: $S \leftarrow$ subset of size s selected randomly from U
7: $T \leftarrow$ subset of size t selected randomly from $\overline{U}$
8: $A_j \leftarrow S \cup T$
9: $U \leftarrow U \cup T$
10: $q \leftarrow q + t$
11: **repeat**
12: $F_j \leftarrow$ random function with $|A_j|$ inputs, $d!$ outputs ▷ CPT for current node
13: **until** F_j is non-degenerate ▷ Note the odds in Table 4.2
14: **end for**
15: BUILD-CP-NET(A, F) ▷ See Figure 4.11

Figure 4.12: Algorithm: Generate a CP-net Uniformly at Random

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

$$\frac{1}{\binom{q_j}{s}} \frac{1}{\binom{n-q_j}{t}} \frac{1}{\psi_d(s+t)}. \tag{4.23}$$

Multiplying Equations 4.22 and 4.23 and simplifying gives us the probability of generating A_j and F_j given U_j in Lines 5–13:

$$P(A_j, F_j | U_j) = \frac{a_{n,c,d}\left(j+1, q_j + t\right)}{a_{n,c,d}\left(j, q_j\right)} = \frac{a_{n,c,d}\left(j+1, q_{j+1}\right)}{a_{n,c,d}\left(j, q_j\right)}, \tag{4.24}$$

since $q_j + t = q_{j+1}$ for $j = 1$ to $n-1$ (Line 10). Since A and F uniquely characterize a CP-net, $P(N) = P(A, F)$. Altogether, iterating through all values of j in the **for** loop at Line 4, the probability of generating N is:

$$P(N) = P(F_0)P(A_1F_1|U_1)P(A_2F_2|U_2) \cdots P(A_{n-2}F_{n-2}|U_{n-2})P(A_{n-1}F_{n-1}|U_{n-1}) \tag{4.25}$$

$$= \frac{1}{d!} \frac{a_{n,c,d}(2, q_2)}{a_{n,c,d}(1, q_1)} \frac{a_{n,c,d}(3, q_3)}{a_{n,c,d}(2, q_2)} \cdots \frac{a_{n,c,d}(n-1, q_{n-1})}{a_{n,c,d}(n-2, q_{n-2})} \frac{a_{n,c,d}(n, q_n)}{a_{n,c,d}(n-1, q_{n-1})}. \tag{4.26}$$

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

Figure 4.13: Algorithm: Compute Tables for Uniform CP-net Generation

One can use Equation 4.20 to verify that

$$a_{n,c,d}(0, 0) = d!\, a_{n,c,d}(1, 0). \tag{4.27}$$

Also, $q_1 = |\bigcup_{k<1} A_k| = 0$. One can thus rewrite the first term of Equation 4.26 as

$$P(F_0) = \frac{1}{d!} = \frac{a_{n,c,d}(1, q_1)}{a_{n,c,d}(0, 0)}. \tag{4.28}$$

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

$$P(N) = \frac{1}{a_{n,c,d}(0, 0)} = \frac{1}{a_{n,c,d}} \tag{4.29}$$

which proves the case. ❑

Theorem 32. COMPUTE-DIST *(Figure 4.13) runs in time and space polynomial in the number of nodes n.*

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

Algorithm Random-CP-Net is also efficient. Random subset sampling and proportional (i.e., weighted) sampling can be performed efficiently [21, 55, 3.4.2], and with high probability the inner loop will execute just once, as discussed in Section 4.2 (see Table 4.2 in particular).

4.5 Generating Outcomes and DT Problem Instances

Generating outcomes, pairs of outcomes, and DT problem instances is straightforward, but it is discussed here briefly for the sake of completeness. To sample uniformly at random from a set of outcomes $\mathcal{O} = \mathrm{Dom}(X_i)\times\cdots\times\mathrm{Dom}(X_n)$, one need only select, independently, a value $x^i_{j_i}$ uniformly from each domain $\mathrm{Dom}(X_i)$ and combine these to form the outcome $o = x^1_{j_1} x^2_{j_2} \cdots x^n_{j_n}$.

To sample uniformly from the set of distinct pairs of outcomes $\mathcal{O}^2_{n,d}$, one can simply select two outcomes o and o', $o \in \mathcal{O}_{n,d}$, $o' \in \mathcal{O}_{n,d}$ as before, and repeat selection of o' in the unlikely event that $o = o'$.

To sample uniformly at random from pairs with Hamming distance h, $\mathcal{O}^2_{n,d\,|\,h}$, one can select o from $\mathcal{O}$ in the usual manner, select a random subset U of size h from variables $\mathcal{V}$, and for each variable $X_k \in U$, flip the value of $o[k]$ to another value selected uniformly randomly from $\mathrm{Dom}(X_k)$ such that $o'[k] \neq o[k]$.

Finally, since it follows from Definition 13 that the space of DT problem instances is

$$\mathrm{DT}\,(\mathcal{N}_{n,c,d}, \mathcal{O}_{n,d\,|\,h}) = \mathcal{N}_{n,c,d} \times \mathcal{O}^2_{n,d\,|\,h}\,, \tag{4.30}$$

to select an instance uniformly at random, one needs only to generate a pair of outcomes $(o, o') \in \mathcal{O}^2_{n,d\,|\,h}$ as described above and then generate a CP-net $N \in \mathcal{N}_{n,c,d}$ uniformly randomly using algorithm Random-CP-net(n, c, d) as described in Section 4.4. The resulting

set of triples $\mathcal{I} = \{(N_1, o_1, o'_1), \ldots, (N_t, o_t, o'_t)\}$, $t = |\mathcal{I}|$, is thus a representative sample of $\mathrm{DT}(\mathcal{N}_{n,c,d}, \mathcal{O}_{n,d\,|\,h})$.

4.6 Conclusion

This chapter has presented an efficient and provably uniformly random method for generating CP-nets. The method allows for bounds on indegree and multivalued domains. The recurrence of Theorem 30 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 Figure 4.13 to $s+t = 1$.

The method described in this chapter has been implemented in C++ using the GnuMP library (Granlund et al. 2014), allowing generation of thousands of CP-nets per second. The code is available under a free and open source license.

Chapter 5 Depth-Limited Dominance Testing

Despite their advantages and conceptual beauty, one of the chief objections to CP-nets is that the problem of *dominance*—deciding whether one outcome is better than another with respect to the network—is computationally hard. As discussed in Section 3.4.1, deciding dominance in CP-nets is known to be PSPACE-complete in general [42], and in certain instances it is hard even to verify a solution [16]. Except for special cases (e.g., tree-shaped CP-nets), dominance testing in CP-nets necessitates a search, with possible backtracking, for a path in the exponentially larger induced preference graph of the CP-net.

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

This technique of improving the efficiency of algorithms through learning a set of parameters for which the algorithm is well-behaved is widely used in practical applications and is known as algorithm configuration [52]. Using algorithm configuration to specify search strategies based on easily computable properties of input instances has led to many practical refinements to heuristics for known hard problems such as SAT [105] and planning [86].

Section 5.1 offers a brief review of the problem, related research, and useful notation. Section 5.2 discusses an experiment that exhaustively explores tiny cases ($n \leq 4$ nodes) to more fully understand *expected flipping length*. Sections 5.3 and 5.4 discuss experiments

that employ sampling to gain insights into expected flipping length in larger problem instances. Section 5.5 shows that very long transitive sequences are unreliable if the learned model is too noisy. Section 5.6 introduces the notion of *limited dominance* along with algorithms for *depth-limited dominance testing* (DLDT). A conclusion follows in Section 5.7.

5.1 Preliminaries

Many decision-making applications involve finding the optimal, i.e., most preferred, outcome.[1] Section 1.2 discussed the possibility of an assistive robot that purchases its busy client a sandwich from a deli. Certain ingredients may be unavailable (e.g., tuna salad), thus constraining the domains of some features that describe the set of possible alternatives. If the client's preferences can be modeled as a CP-net, choosing the most preferred sandwich for her is computationally easy, even if some domains are constrained [16] (see Section 3.2), and this is an advantage of CP-nets over other compact preference models, such as soft constraints [32] and preference trees (P-trees) [71].

Section 1.2 also discussed the related problem of selecting the best sandwich from a set S of preassembled, wrapped sandwiches, $S \subset \mathcal{O}$, where $|S| \ll |\mathcal{O}|$. In that case it may be necessary to perform a pairwise comparison of all $\binom{|S|}{2}$ preassembled alternatives. In CP-nets the method of deciding whether an arbitrary outcome o is preferred to another outcome o' is known as *dominance testing* (DT). DT is also employed by some CP-net learning algorithms, such as those of Dimopoulos et al. [29] and Guerin et al. [47], as discussed in Section 3.5.

Figure 5.1 revisits another example, from Section 2.4. Suppose the same assistive robot also plans activities for its busy client, using this CP-net as a model of her preferences. The robot can take into account the weather forecast, availability of friends, whether a table can be reserved at the table tennis club, and so on. Would the client prefer a *fair day cycling*

[1] Note that in some settings, such as multi-participant decision making with diverses stakeholders, e.g., negotiated decisions, such optima are not well defined. Recall, however, that the preferences in this work are assumed to be those of an individual.

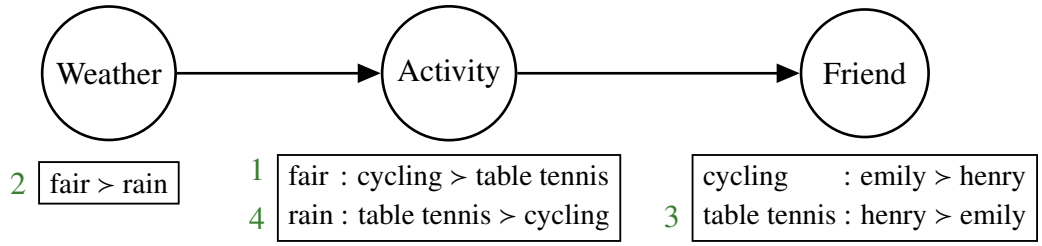


Figure 5.1: CP-net Describing Client's Preferences on Activities (Figure 2.3 Revisited)

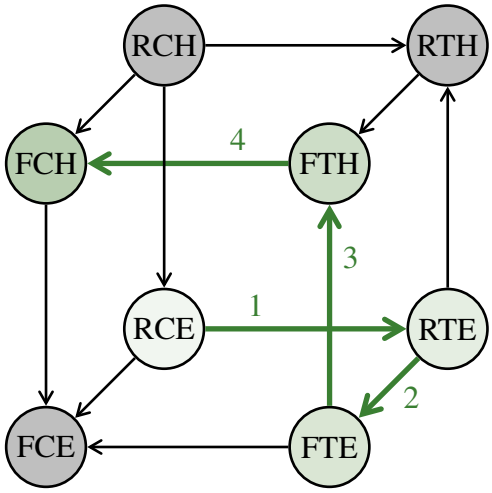


Figure 5.2: Flipping Sequence in Induced Preference Graph (Figure 2.4 Revisited)

with Henry or a *rainy day cycling with Emily*? Because the alternatives differ in the value of more than one variable, the conditional preference rules (CPRs) of the CP-net do not afford a direct comparison. However, with the help of the annotated Figures 5.1 and 5.2 one can observe that the CP-net induces the transitive sequence $\langle$*rain, cycling, emily*$\rangle \prec \langle$*rain, table tennis, emily*$\rangle \prec \langle$*fair, table tennis, emily*$\rangle \prec \langle$*fair, table tennis, henry*$\rangle \prec \langle$*fair, cycling, henry*$\rangle$. From this, the robot is able to reason that the client would prefer a *fair day cycling with Henry* and sends Henry a message on her behalf asking if he would like to go cycling that day.

Such transitive sequences are known as *flipping sequences*. In general, DT involves the search for such a sequence from the less to the more preferred outcome along a path in the preference graph induced by the rules of the network. Note that the branching factor of this search is $O(dn)$ since at any outcome in H one can flip at most n variables to any

Table 5.1: Computational Difficulty of Dominance Testing (Table 3.2 Revisited)

Graph	Domains	CPTs	DT Complexity	Running Time	References
Directed Forest	Binary	Complete	P	$O(n)$	[11]
Directed Forest	Binary	*	P	$O(n^2)$	[16]
Directed Forest	*	*	NP-hard	?	[16]
Polytree	Binary	*	P	$O(2^{2c}n^{2c+3})$	[16]
DPSCG	Binary	*	NP-complete	?	[16]
Max-δ-connected	Binary	*	NP-complete	?	[16]
*	*	*	PSPACE-complete	?	[42]

*No restriction

of the $d - 1$ different values in their domain. Such paths can be exponentially long [16], and computing the length of a longest path is itself hard [42]. Moreover, even if this length is known in advance or polynomially bounded in the number of variables, it is still hard in general to tell whether o is preferred to o' [61]. Tractable subclasses of the problem exist. For instance, if the network has the shape of a directed tree, with binary variables and complete CPTs, dominance can be answered in linear time in the number of nodes [11]. DT can also be performed in linear time in any CP-net if the outcomes differ in the value of just one variable. (See Table 5.1, revisited from Section 3.4.1.)

Some methods of pruning the search tree are known. Boutilier et al. [16] showed how to use a method known as *suffix fixing* (SF) to prune regions of the search tree that cannot contain a solution. They also introduced a best-first search strategy known as *least variable flipping* (LVF); however, LVF is complete only for restricted subclasses such as tree and polytree CP-nets. Moreover, Li et al. [68] showed in their experiments that LVF failed to find an increasing proportion of the solutions as n increased, with about 20% incompleteness for acyclic binary CP-nets with 20 nodes. The latter also proposed a specialized heuristic DT algorithm known as DT*, which has been adapted for use in the experiments described in this chapter. A number of reductions have also been proposed for DT, including automated planning [16] and model checking [92]; such approaches rely on heuristics inherent in the respective solver, which is usually treated as a black box.

Table 5.2: Cardinalities of $\mathcal{O}^2$ and $\mathrm{DT}(\mathcal{N}, \mathcal{O})$

$\left\lvert\mathcal{O}^2\right\rvert = \lvert\mathcal{O}\rvert^2$	$\lvert\mathrm{DT}(\mathcal{N}, \mathcal{O})\rvert = \lvert\mathcal{N}\rvert\,\lvert\mathcal{O}\rvert^2$
$\lvert\mathcal{O}_n\rvert = 2^n$	$\left\lvert\mathcal{O}_n^2\right\rvert = (2^n)^2 = 2^{2n}$
$\left\lvert\mathcal{O}_{n,d}\right\rvert = d^n$	$\left\lvert\mathcal{O}_{n,d}^2\right\rvert = (d^n)^2 = d^{2n}$
$\left\lvert\mathcal{O}_{n\,\vert\,h}^2\right\rvert = \lvert\mathcal{O}_n\rvert\binom{n}{h} = 2^n\binom{n}{h}$	$\left\lvert\mathcal{O}_{n,d\,\vert\,h}^2\right\rvert = d^n(d-1)^h\binom{n}{h}$
$\left\lvert\mathrm{DT}(\mathcal{N}_{n,d\,\vert\,c}, \mathcal{O}_{n,d})\right\rvert = d^{2n}a_{n,c,d}$	$\left\lvert\mathrm{DT}(\mathcal{N}_{n,d\,\vert\,c}, \mathcal{O}_{n,d\,\vert\,h})\right\rvert = d^n(d-1)^h\binom{n}{h}a_{n,c,d}$

While good heuristics can reduce the effective branching factor, the depth of solutions and the depth to which an algorithm explores the search tree when no solution exists are of considerable importance. Note that while Boutilier et al. [16] proved flipping lengths could be exponential in n in chain CP-nets with multivalued domains and incomplete CPTs, Kronegger et al. [61], found in their randomly generated GCP instances that flipping length was usually of the same order of magnitude as n. However, it is unclear if either of these results extend to binary CP-nets or to those with complete tables. We also desire a more nuanced understanding of flipping lengths that can be applied to DT heuristics. In particular, if one can predict the flipping length of DT instances in advance, then this enables us to constrain search depth to some depth k at which, say, 99% of the solutions are expected to be found. This enables a heuristic approach in which one sacrifices finding solutions to a very small number of problems that require exhaustive search, but in return is able to find the vast majority of the solutions with fewer computational resources.

Some notation introduced in Chapter 2 is useful in discussing DT. $\mathcal{N}_n$ is the set of all complete, acyclic CP-nets on n binary variables. $\mathcal{N}_{n,d\,\vert\,c}$ is the set of complete CP-nets on n d-ary variables and bound c on indegree in the dependency graph, and $a_{n,c,d}$ denotes the cardinality of this set. $\mathcal{O}_n^2$ is the set of all pairs of outcomes on n binary variables. $\mathrm{HD}(o, o')$ denotes the Hamming distance, i.e., the number of variables in which o and o' differ. $\mathcal{O}_{n,d\,\vert\,h}^2$ is the set of all pairs of outcomes on n d-ary variables with Hamming distance h. $\mathrm{DT}(\mathcal{N}, \mathcal{O})$ is the set of all DT problem instances (N, o, o'), $N \in \mathcal{N}$, $o \in \mathcal{O}$, $o' \in \mathcal{O}$ under consideration.

DT($\mathcal{N}, \mathcal{O}, \mathcal{C}$) is the set of such instances that also satisfy condition $\mathcal{C}$. FL(N, o, o') is the flipping length, i.e., the length ℓ of the shortest path from o' to o in the induced preference graph H. If no flipping sequence exists, ℓ is undefined, written $\ell = \infty$. MFL($\mathcal{N}, \mathcal{O}$) is the *mean flipping length* for DT($\mathcal{N}, \mathcal{O}$). Let us denote by $o[i]$ the value of variable X_i in outcome o, and by $o[-i]$ the assignment to the other $n - 1$ variables. The function $[P]$, where P is a Boolean predicate, evaluates to 1 if P is true and 0 if P is false. Chapter 4 showed how to compute $|\mathcal{N}_{n,d\,|\,h}| = a_{n,c,d}$ (Theorem 30). To this one can add the helpful identities in Table 5.2 for outcomes, pairs of outcomes, and DT instances, which are easily verified. Further notation is discussed in Chapter 2; see Table 2.2 in particular.

5.2 Experiment 1: An Exhaustive Consideration of Tiny Cases

To understand the problem as fully as possible for small n, I first studied *all* DT instances up to $n = 4$ binary variables—123,334,656 instances.[2] For this experiment, I thus generated the sets $\mathcal{N}_1$ to $\mathcal{N}_4$ consisting respectively of all binary acyclic CP-nets with 1 to 4 nodes. I uncompacted each CP-net $N \in \mathcal{N}_n$ to obtain its induced preference graph H, and applied the Floyd–Warshall all-pairs-shortest-path algorithm [24, 37, 101] to H to determine the flipping length FL(N, o', o) for all pairs of outcomes $\mathcal{O}_n^2$. I then aggregated solutions according to the Hamming distance between outcomes and other prospective parameters.

The preference graph and its relationship to a CP-net is discussed in Sections 1.2.1 and 1.3 and Section 2.4 (see also Definition 11 and Figures 1.5 and 2.4). Figure 5.3 provides an overview of how the DT instances are generated and solved for this experiment. The outer loop iterates from $n = 1$ to 4, calling All-CP-nets in Line 3 to generate the complete set of acyclic binary CP-nets with n nodes, as discussed in Chapter 4 (Figure 4.10 in particular). Each CP-net N, in turn, is then uncompacted into its induced preference graph H, as modeled by a sparse adjacency matrix M using the algorithm Uncompact described in Figure 5.4. Note that the columns and rows of the matrix correspond to all outcomes $\mathcal{O}$.

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

TINY-CASES

Output: database of DT instances and flipping lengths

1: initialize database
2: **for** $n \leftarrow 1$ **to** 4 **do**
3: **for all** $N \in \{\text{ALL-CP-NETS}(n, n-1, 2, 0, 0, \emptyset, A_{<1}, F_{<1})\}$ **do**
4: $M \leftarrow \text{UNCOMPACT}(N)$
5: $\mathcal{L} \leftarrow$ Floyd-Warshall-All-Pairs-Shortest-Paths(M)
6: store $(n, N, \mathcal{L})$ in database
7: **end for**
8: **end for**
9: **return** database

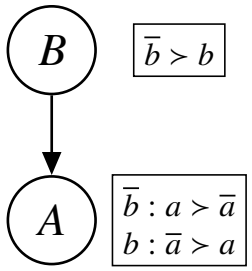
Figure 5.3: Initial Experiment: All DT Instances Up to $n = 4$

UNCOMPACT(N)

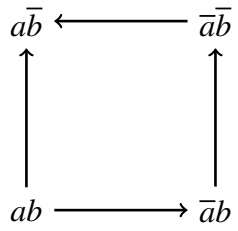
Input: N CP-net
Output: M Adjacency matrix of corresponding preference graph

1: $\mathcal{O} \leftarrow \bigtimes_{X_i \in \mathcal{V}} X_i$
2: initialize $|\mathcal{O}| \times |\mathcal{O}|$-matrix M
3: **for** $o \leftarrow \text{First}(\mathcal{O})$ **to** $\text{NextToLast}(\mathcal{O})$ **do**
4: **for** $o' \leftarrow \text{Next}(o)$ **to** $\text{Last}(\mathcal{O})$ such that $\text{HD}(o, o') = 1$ **do** ▷ upper triangular
5: $X_i \leftarrow$ variable that differs in o and o' ▷ since $\text{HD}(o, o') = 1$
6: $u \leftarrow o[\text{Pa}(X_i)]$ ▷ assignment to parents
7: $R \leftarrow$ ranking $\succ^i$ from $\text{CPT}(X_i \mid \text{Pa}(X_i) = u)$
8: **if** $(o[X_i], o'[X_i]) \in R$ **then** ▷ $u : o[i] \succ o'[i]$
9: $M[o', o] = 1$
10: **else if** $(o'[X_i], o[X_i]) \in R$ **then** ▷ $u : o'[i] \succ o[i]$
11: $M[o, o'] = 1$
12: **end if**
13: **end for**
14: **end for**
15: **return** M

Figure 5.4: Algorithm: Uncompact CP-net to Obtain Preference Graph



(a) CP-net N



(b) Induced Preference Graph H

	$\overline{a}\overline{b}$	$\overline{a}b$	$a\overline{b}$	ab
$\overline{a}\overline{b}$	0	0	1	0
$\overline{a}b$	1	0	0	0
$a\overline{b}$	0	0	0	0
ab	0	1	1	0

(c) Adjacency Matrix M that Models H

	$\overline{a}\overline{b}$	$\overline{a}b$	$a\overline{b}$	ab
$\overline{a}\overline{b}$	0	∞	1	∞
$\overline{a}b$	1	0	2	∞
$a\overline{b}$	∞	∞	0	∞
ab	2	1	1	0

(d) Floyd–Warshall Output $\mathrm{FL}(N, o, o')$

	$\overline{a}\overline{b}$	$\overline{a}b$	$a\overline{b}$	ab
$\overline{a}\overline{b}$	0	0	1	0
$\overline{a}b$	1	0	1	0
$a\overline{b}$	0	0	0	0
ab	1	1	1	0

(e) Answers to DT Instances $[N \models o \prec o']$

Figure 5.5: CP-net, Adjacency Matrix, and Preference Graph

In constructing the matrix, the assumption is made that the variables $X_i \in \mathcal{V}$ are ordered by their indices i, $1 \leq i \leq n$, and that $\overline{x_i}$ is ordered before x_i in each domain $\mathrm{Dom}(X_i)$, for all i. It is further assumed that outcomes are ranked lexicographically, such that o is ordered before o' if there exists h, $1 \leq h \leq n$ such that $o[h]$ is ordered before $o'[h]$ in the domain of X_h and that $o[j] = o'[j]$ for all j, $1 \leq j < h$. Let us denote respectively by $\mathrm{First}(\mathcal{O})$, $\mathrm{Last}(\mathcal{O})$, and $\mathrm{NextToLast}(\mathcal{O})$ the minimum, maximum, and penultimate outcomes according to this lexicographic ranking, and by $\mathrm{Next}(o)$ each outcome that covers (immediately follows) o. Note that if the outcomes are represented as binary numerals, then $\mathrm{First}(\mathcal{O}) \equiv 0$, $\mathrm{Last}(\mathcal{O}) \equiv 2^n - 1$, $\mathrm{NextToLast}(\mathcal{O}) \equiv 2^n - 2$, and $\mathrm{Next}(o) = o + 1$ for all o, $0 \leq o \leq 2^n - 2$.

An entry $M[o, o'] = 1$ in the adjacency matrix indicates that the preference graph contains an edge from o to o'. The outcomes o and o' thus differ in the value of just one variable, and a conditional preference rule (CPR) of the CP-net directly specifies that $o' \succ o$. Otherwise, if $M[o, o'] = 0$, then there is no edge from o to o' in H. Figure 5.5a–c provides a simple example of a CP-net N with 2 nodes, its induced preference graph H and

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

$$|\mathrm{DT}\,(\mathcal{N}_4, \mathcal{O}_4 \mid \mathrm{HD}(o, o') = h \wedge \mathrm{FL}(N, o, o') = \ell\,)|$$

Figure 5.6: Number of DT Solutions Given Hamming Distance and Flipping Length

corresponding adjacency matrix M.[3] Applying the Floyd–Warshall algorithm produces the output shown in Figure 5.5d. Note that the elements of that matrix give the flipping lengths $\mathrm{FL}(N, o, o')$ for each pair of outcomes. An entry of ∞ indicates that there is no improving flipping sequence from o to o'. While, for technical reasons, flipping sequences of length 0 are excluded, the values of 0 along the diagonal indicate that $o = o'$. From the matrix of flipping lengths it is then straightforward to obtain statistics, such as the mean flipping length for a set of CP-nets or mean flipping length given some specified parameter, such as Hamming distance. One can also obtain the answers to all DT instances by cloning the matrix and replacing each ∞ with a 0 and subsequently each non-0 element with a 1, as shown in Figure 5.5e.

Figure 5.6 summarizes the results for DT problem instances with 4 variables that entail dominance. The rows of the table correspond to the flipping length ℓ and the columns correspond to the Hamming distance h. Each entry at position (ℓ, h) corresponds to the number of DT problems with that flipping length and Hamming distance,

$$|\mathrm{DT}(\mathcal{N}_4, \mathcal{O}_4 \mid \mathrm{HD}(o, o') = h \wedge \mathrm{FL}(N, o, o') = \ell)|\,,$$

$o \in \mathcal{O}_4$, $o' \in \mathcal{O}_4$, $N \in \mathcal{N}_4$, $\ell \in \mathbb{Z}^+$. The entries that are left blank correspond to a count of 0. Since no DT instance had a flipping length $\ell > 6$, the blank rows for higher values of ℓ are

[3] Note that in the figure, A maps to X_2 and B to X_1.

not shown. Also, note that DT instances that did not entail dominance are excluded from the table, since in that case the flipping length ∞ is undefined.

The table entries shown in Figure 5.6 for $n = 4$ (and also those for $n = 2$ and $n = 3$, which are similar) provide important insights into the space of DT problems that can be extended to all CP-nets through simple proofs. First, notice that if the Hamming distance is 1, then the flipping length, if defined, is also 1.

Theorem 33 (Flip Comparisons)**.** *For every DT instance* (N, o, o') *that entails dominance,*

$$\mathrm{HD}(o, o') = 1 \textit{ if and only if } \mathrm{FL}(N, o, o') = 1. \tag{5.1}$$

Proof. Note that the claim follows directly from the *ceteris paribus* semantics of CP-nets. Consider that if $\mathrm{HD}(o, o') = 1$, there exists a variable X_i such that $o[i] \neq o'[i]$ and $o[-i] = o'[-i]$. Thus, we need only check the rule in $\mathrm{CPT}(X_i)$ that corresponds to the values of the parents of X_i in the dependency graph, $u : \succ^i$, where $u = o[\mathrm{Pa}(X_i)]$, if such a rule exists. If CPTs are complete, $u : \succ^i$ exists, and in all cases the rule will be unique. In the rule exists, we need only check the respective ordering of $o[i]$ and $o'[i]$ in $\succ^i$. If $(o[i], o'[i]) \in \succ^i$, then $o \succ o'$ and $\mathrm{FL}(N, o, o') = 1$. Otherwise, if $(o[i], o'[i]) \in \succ^i$ or, in the case of incomplete tables, if no such rule exists, then $o \not\succ o'$ and the flipping length is undefined. ❑

Since DT instances with Hamming distances of 0 and 1 do not necessitate a search, we can henceforth limit attention to Hamming distances in the interval $(2\,..\,n)$.

Next observe that the flipping length is always *at least* the Hamming distance.

Theorem 34 (Range of HD)**.** *For every DT instance* (N, o, o') *that entails dominance*

$$\mathrm{FL}(N, o, o') \geq \mathrm{HD}(o, o'). \tag{5.2}$$

Proof. $\mathrm{FL}(N, o, o') \not< \mathrm{HD}(o, o')$ since each flip changes the value of just one variable. However, the value of a variable can change more than once in a flipping sequence if the assignment to parents changes. The proof is by example. Note that in the flipping sequence in Figure 5.2, the value of Activity changes twice. ❑

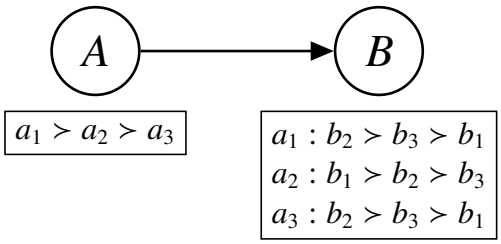
Next observe that if the Hamming distance is even (respectively odd), the flipping length, if a path exists from o' to o in H exists, will also be even (respectively odd). A definition is helpful in stating this formally.

Definition 35 (Parity)**.** *The* parity *of outcomes o and o' is the parity of their Hamming distance. If* $\mathrm{FL}(o, o') \bmod 2 = 0$*, the parity of o and o' is* even*; otherwise, it is* odd*.*

Theorem 36. *For every binary-valued DT instance (N, o, o') that entails dominance, the parity of the flipping length is the parity of the outcomes.*

Proof. It suffices to observe that in each flip (o_t, o_{t+1}), $0 \leq t < \ell$, $o' = o_0$, $o = o_\ell$, the parity of o_t and o (the goal) is inverted, provided that the variables are binary. A sequence from o' to o thus has an even (odd) number of flips if and only if $\mathrm{HD}(o, o')$ is even (odd). ❑

Note however that this result does not generalize to multivalued variables, since for $d \geq 3$, a flip does not necessarily change the Hamming distance $\mathrm{HD}(o_t, o)$. The proof is a simple counter-example. Consider the three-valued CP-net below.



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

From the values in each column h of the table in Figure 5.8, one can compute the mean flipping length $\hat{\ell}$ given Hamming distance h, as shown in Figure 5.7. From the data one can observe that the mean flipping length $\mathrm{MFL}_h(\mathcal{N}_4, \mathcal{O}_4)$ is close to the Hamming distance. Furthermore, by calculating the ratio of each count to the total counts in each column and

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

Figure 5.7: Mean Flipping Length $\hat{\ell} = \text{MFL}(\mathcal{N}_4, \mathcal{O}_4 \mid \text{HD}(o, o') = h)$

computing the cumulative sum on ℓ ascending, we obtain the empirical distribution shown in Figure 5.8. Moreover, since the data for $n \leq 4$ reflect *all* DT instances, for these we have the cumulative density function (c.d.f.) of the flipping length ℓ itself, conditioned on the value of h.

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

Figure 5.8: Cumulative Density Function (c.d.f.) Resulting from Figure 5.6

This suggests a useful notion in searching for a flipping sequence. Consider a setting in which we are performing DT on an instance (N, o, o') and have already searched the implicit preference graph to a depth of k. With the help of a table such as the one in Figure 5.8, we could then compute the probability that a path from o' to o of length $\ell \geq k+2$ exists. For example, suppose the algorithm had searched to a depth of 4 flips for a DT instance with $n = 4$, $\text{HD}(o, o') = 2$. We would then know that the odds that $N \models o \succ o'$ were only about 2 in 1000. We could then either continue the search to depth $k + 2$ or, if the cost of computational resources was too high, halt the search and report that it was *very likely* the case, but not guaranteed, that $o \not\succ o'$.

In addition to Hamming distance, observe that the *average path length* of the dependency graph $\text{APL}(G)$ (see Definition 4) also serves as an important parameter for estimating flipping length. In graphs for which the value of $\text{APL}(G)$ is relatively low (such as DAGs of unbounded indegree), the values of flipping length also tend to be lower (closer to h). Con-

versely, in graphs for which the value of APL(G) is relatively high, one can observe that flipping sequences tend to be longer. The three-dimensional bar chart in Figure 5.9 illustrates the mean flipping length $\hat{\ell}$ given average path length $L = \text{APL}(G)$ of the dependency graph and the Hamming distance $h = \text{HD}(o, o')$ of the pair of outcomes.[4]

Note that the values of h and L are not distributed uniformly across problem instances; Hamming distances h, for example, are distributed binomially. Figures 5.10a and 5.10b illustrate the distribution of h with respect to $\mathcal{O}_4^2$, and the distribution for APL(G) with respect to $\mathcal{N}_4$.

Finally, let us consider the *maximum* value that FL(N, o, o') can take. In the analysis of $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$, and $\mathcal{N}_4$ (see Figure 5.6 for $\mathcal{N}_4$), the longest flipping lengths for any DT instance were 1, 2, 4, and 6, respectively. In each case, a *chain* CP-net induced a sequence of that length.[5] Recall from Section 2.2 that a chain is a directed tree with just one leaf. Thus, every node except the root in the dependency graph of a chain CP-net has just one parent, and every node except the leaf has just one child. Chain CP-nets also feature prominently in the proofs of Boutilier et al. [16] that flipping lengths can be $\Theta(n^2)$ for complete, binary, acyclic CP-nets and $\Omega(2^{n/2})$ for CP-nets with incomplete tables and multivalued domains. One can also observe that chains have the maximum *average path length* of any graph [49]. Thus, because of their relatively long flipping lengths, chain CP-nets are of interest to us despite the fact that, because chains are trees, DT can be conducted for such problems in polynomial time (see Table 5.1).

I generated binary chain CP-nets with $n = 1$ to 12 nodes, uncompacting each[6] into its corresponding preference graph and applying the Floyd–Warshall algorithm as in Figure 5.3 to compute the diameter of H. For $n = 1$ to 12, I found that binary chain CP-nets had maximum flipping lengths of 1, 2, 4, 6, 9, 12, 16, 20, 25, 30, 36, and 42, consistent with the formula $\lfloor (n+1)^2/4 \rfloor$, as in Sloane [98, A002620]. In all of my experiments involving

[4] For $n = 4$ the range of APL(G) does not include the value $\frac{3}{4}$.

[5] Note that in some cases other graphs had maximum flipping lengths as long as those of chain CP-nets.

[6] Because every chain CP-net on n binary nodes with complete CPTs is the same up to symmetry, it is only necessary to generate one instance for each integer n.

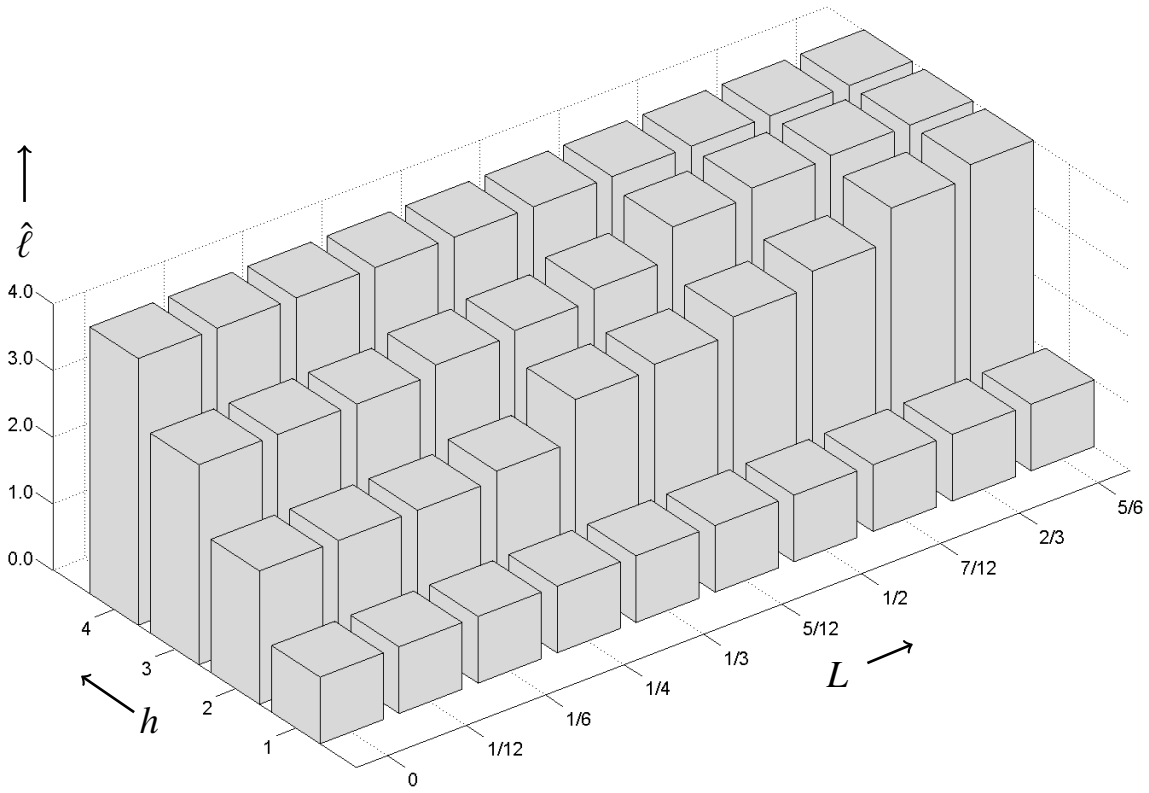
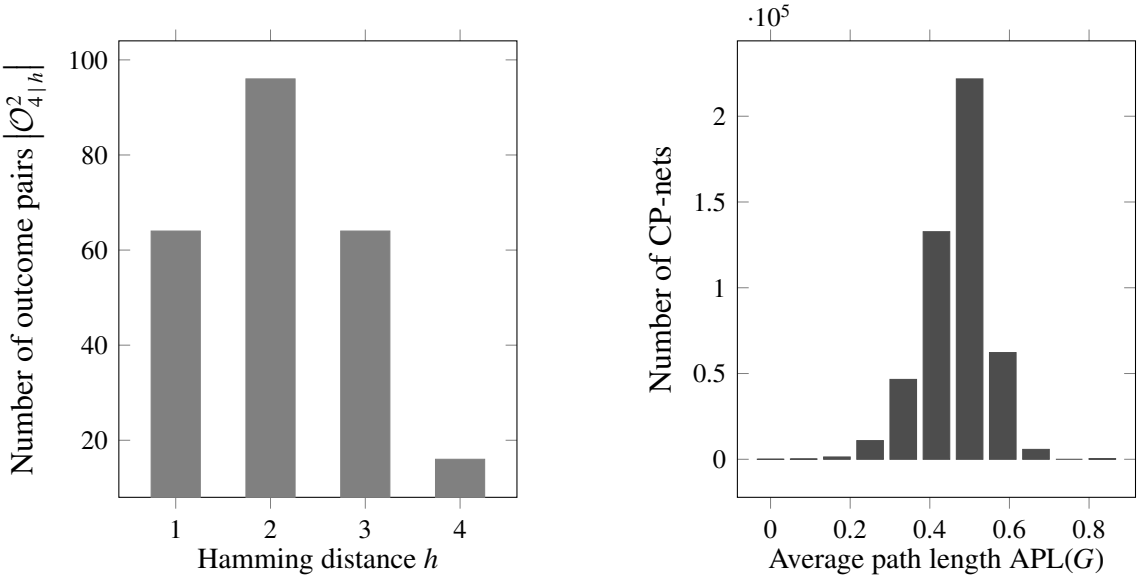


Figure 5.9: Mean Flipping Length $\hat{\ell}$ as a Function of HD h and APL L



(a) Distribution of HD

(b) Distribution of APL

Figure 5.10: Distribution of Parameter Values over DT Problem Instances ($n = 4$)

complete binary[7] acyclic CP-nets, including those described in the sections that follow, I have not yet encountered a flipping length that exceeds these values for a given n. Hence, the following can be stated as an interesting open problem.

Conjecture 37. *Let (N, o, o') be an arbitrary DT instance, where N is a complete CP-net on n binary variables. Then, for all n, N, o, o', the longest flipping length is*

$$\max(\mathrm{FL}(N, o, o')) = \left\lfloor \frac{1}{4}(n+1)^2 \right\rfloor. \tag{5.3}$$

Note that this is consistent with the $\Theta(n^2)$ asymptotic *lower* bound shown by Boutilier et al. [16]. If the conjecture holds, $\Theta(n^2)$ would also be a tight *upper* bound. An immediate result would be that dominance testing in such CP-nets would be "only" NP-complete.

5.3 Experiment 2: Sampling CP-nets and Solving DT for all Outcomes

the exhaustive analysis of tiny cases suggests that a flipping sequence, if it exists, is probably not much longer than the Hamming distance between the two outcomes. Does this result extend to larger values of n and d? Unfortunately, the exhaustive analysis of the sort found in Section 5.2 is infeasible for $n > 4$, because $|\mathcal{N}_5| = a_{5,4,3} = 157549032992$, and for multivalued domains $d > 2$ even for 3 nodes because $|\mathcal{N}_{3,3}| = a_{3,2,3} = 77274933336$ (see Tables 4.4 and 4.5). Thus, for larger cases one must rely on random sampling. Fortunately, with algorithm Random-CP-net (see Figure 4.12) one can sample the space of DT problem instances DT $(\mathcal{N}_{n,d\,|\,c}, \mathcal{O}_{n,d\,|\,h})$ provably uniformly randomly (see Sections 4.4 and 4.5).

On the systems available to us, the method of uncompacting a CP-net (see Figure 5.4) and applying the Floyd–Warshall algorithm is feasible up to $n = 9$ for binary domains. For the second experiment, I thus generated 100 binary-valued CP-nets with n ranging from 5 to 9. Because it is assumed that indegree is bounded by a small constant (see Section 2.4), I also varied this bound from $c = 1$ to $c = 4$, such that $c < n$. The generation method of Chapter 4 does not let us specify a bound on APL; however, it can be shown that APL

[7] The bound does not hold for multivalued domains in general.

decreases as the bound c on indegree increases. Thus, by varying c, one can observe the affect of APL L indirectly. Figure 5.11 summarizes the procedure for the experiment for CP-nets with 5–9 binary variables.

Tables 5.3 and 5.4 for 9 nodes correspond to the one shown in Figure 5.6 for 4 nodes. In this case, however, the counts are for the sampled instances rather than for all DT instances. Also, for variables with 5 or more nodes, the results have been aggregated by the bound on indegree. Note once again that most flipping lengths are relatively close to the Hamming distance. Also, note that as c increases, APL decreases, and flipping lengths tend to become even more tightly compacted to the Hamming distance. Table 5.5 shows how the mean flipping length varies as a function of the number of variables n, bound on indegree c, and Hamming distance h (shown only at the extremes and in the middle).

SMALL-CASES

Output: database of DT instances and flipping lengths

```
1:  initialize database
2:  d ← 2                                                    ▷ binary domains
3:  for n ← 5 to 9 do                                        ▷ number of features
4:     for c ← 1 to 4 do                                     ▷ indegree bounded by small constant
5:        𝒩 ← ∅                                              ▷ generate a set 𝒩 of 100 CP-nets
6:        for t ← 1 to 100 do
7:           𝒩 ← 𝒩 ∪ RANDOM-CP-NET(n, c, d)                 ▷ Figure 4.12
8:        end for
9:        for all N ∈ 𝒩 do                                   ▷ solve DT for all O²_{n,d}
10:          M ← UNCOMPACT( N )                            ▷ preference graph
11:          ℒ ← Floyd-Warshall-All-Pairs-Shortest-Paths( M )   ▷ flipping lengths
12:          store (n, c, d, N, ℒ) in database
13:       end for
14:    end for
15: end for
16: return database
```

Figure 5.11: Second Experiment: Sample CP-nets, Solve for All Outcome Pairs

Table 5.3: Number of DT Solutions Given HD and FL ($n = 9$; $c = 1$ and $c = 2$)

$c = 1$								
ℓ	$h = 2$	3	4	5	6	7	8	9
25	0	0	0	1	0	0	0	0
24	0	0	4	0	0	0	0	0
23	0	0	0	9	0	0	0	0
22	0	0	8	0	12	0	0	0
21	0	0	0	48	0	16	0	0
20	0	0	44	0	104	0	0	0
19	0	0	0	202	0	120	0	0
18	0	0	152	0	558	0	0	0
17	0	80	0	1058	0	432	0	0
16	64	0	1204	0	1997	0	0	0
15	0	1168	0	4362	0	928	0	0
14	512	0	7024	0	5055	0	0	0
13	0	5952	0	16309	0	1896	0	0
12	2048	0	23400	0	20549	0	0	0
11	0	15984	0	51610	0	19804	0	0
10	4480	0	60584	0	76520	0	20718	0
9	0	37056	0	132888	0	99784	0	16800
8	10112	0	131112	0	223220	0	108332	0
7	0	74672	0	298704	0	341624	0	0
6	19968	0	253984	0	694896	0	0	0
5	0	131328	0	1011008	0	0	0	0
4	32704	0	1100480	0	0	0	0	0
3	0	906496	0	0	0	0	0	0
2	552704	0	0	0	0	0	0	0
$c = 2$								
ℓ	$h = 2$	3	4	5	6	7	8	9
25	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0
20	0	0	1	0	0	0	0	0
19	0	1	0	8	0	2	0	0
18	0	0	14	0	30	0	4	0
17	0	8	0	94	0	58	0	0
16	6	0	158	0	290	0	33	0
15	0	203	0	851	0	412	0	7
14	94	0	1837	0	2206	0	321	0
13	0	1707	0	7813	0	2954	0	90
12	516	0	11952	0	17816	0	1608	0
11	0	8036	0	42090	0	22445	0	265
10	2068	0	48054	0	81215	0	20089	0
9	0	28123	0	140292	0	106837	0	14930
8	7043	0	136226	0	258900	0	111016	0
7	0	74623	0	362912	0	383570	0	0
6	18952	0	313374	0	815841	0	0	0
5	0	161057	0	1192476	0	0	0	0
4	39560	0	1263972	0	0	0	0	0
3	0	993672	0	0	0	0	0	0
2	574848	0	0	0	0	0	0	0

Table 5.4: Number of DT Solutions Given HD and FL ($n = 9$; $c = 3$ and $c = 4$)

$c = 3$								
ℓ	$h = 2$	3	4	5	6	7	8	9
25	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0
17	0	0	0	4	0	2	0	0
16	0	0	13	0	24	0	3	0
15	0	18	0	96	0	47	0	1
14	9	0	228	0	367	0	36	0
13	0	280	0	1681	0	680	0	10
12	123	0	3674	0	5851	0	633	0
11	0	3479	0	19562	0	11557	0	177
10	1193	0	29654	0	56318	0	16179	0
9	0	21310	0	121814	0	108505	0	18376
8	6032	0	137546	0	304463	0	145982	0
7	0	82642	0	463686	0	513799	0	0
6	22154	0	416183	0	1076618	0	0	0
5	0	215657	0	1513546	0	0	0	0
4	52419	0	1515996	0	0	0	0	0
3	0	1114108	0	0	0	0	0	0
2	602816	0	0	0	0	0	0	0
$c = 4$								
ℓ	$h = 2$	3	4	5	6	7	8	9
25	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0
15	0	0	0	3	0	1	0	0
14	0	0	16	0	23	0	2	0
13	0	47	0	186	0	49	0	0
12	41	0	696	0	724	0	41	0
11	0	983	0	4476	0	2522	0	31
10	413	0	10452	0	20579	0	7902	0
9	0	10268	0	63580	0	71631	0	18795
8	3788	0	93478	0	252293	0	161735	0
7	0	68424	0	454199	0	602739	0	0
6	21149	0	458186	0	1297174	0	0	0
5	0	255975	0	1817701	0	0	0	0
4	65124	0	1768986	0	0	0	0	0
3	0	1239046	0	0	0	0	0	0
2	632864	0	0	0	0	0	0	0

Table 5.5: Mean Flipping Length Given n, c, and h (for $n = 5$ to 9)

Hamming distance $h = 2$

n	h	$c = 1$	2	3	4
5	2	2.24	2.17	2.16	2.15
6	2	2.25	2.23	2.22	2.18
7	2	2.36	2.24	2.26	2.23
8	2	2.42	2.29	2.30	2.27
9	2	2.43	2.34	2.35	2.33

Hamming distance $h = \lceil n/2 \rceil$

n	h	$c = 1$	2	3	4
5	3	3.32	3.24	3.20	3.17
6	3	3.40	3.37	3.32	3.24
7	4	4.67	4.49	4.41	4.31
8	4	4.91	4.67	4.58	4.43
9	5	6.07	5.92	5.73	5.51

Hamming distance $h = n$

n	h	$c = 1$	2	3	4
5	n	5.00	5.01	5.00	5.00
6	n	6.00	6.01	6.01	6.00
7	n	7.00	7.02	7.00	7.00
8	n	8.00	8.03	8.01	8.00
9	n	9.00	9.06	9.02	9.00

5.4 Experiment 3: A Consideration of Larger Instances

As n increases, it is no longer practical to uncompact the CP-net into its preference graph or to use the Floyd–Warshall algorithm. To gain insight into the flipping lengths for larger cases, one thus has to sample from the outcome space as well as the space of CP-nets.

Figure 5.13 describes the third experiment. For selected Hamming distance h, I generated and stored 100 outcome pairs in set $\mathcal{P}$. Recall that in the first two experiments, flipping lengths depended significantly on Hamming distance. Also recall that values of h are not distributed evenly across the space of outcome pairs $\mathcal{O}^2_{n,d}$. Instead, the peak of the binomial distribution seen in Figure 5.10a for $n = 4$ nodes becomes much sharper as n increases, as seen in Figure 5.12 for $n = 15$. The loop at Line 5 thus chooses values of h at the both extremes of the distribution as well as in the middle. Consider that if we did not explicitly constrain h and sampled from $\mathcal{O}^2_{15}$, for example, instead of $\mathcal{O}^2_{15\,|\,2}$, then the probability of obtaining a pair of outcomes with Hamming distance 2 on a given attempt would be

$$P(\mathrm{HD}(o, o') = 2) = \frac{|\mathcal{O}^2_{15\,|\,2}|}{|\mathcal{O}^2_{15}|} = \frac{2^{15}\binom{15}{2}}{(2^{15})^2} \approx 0.0032; \tag{5.4}$$

thus stratified sampling, as in the approach here, is important.

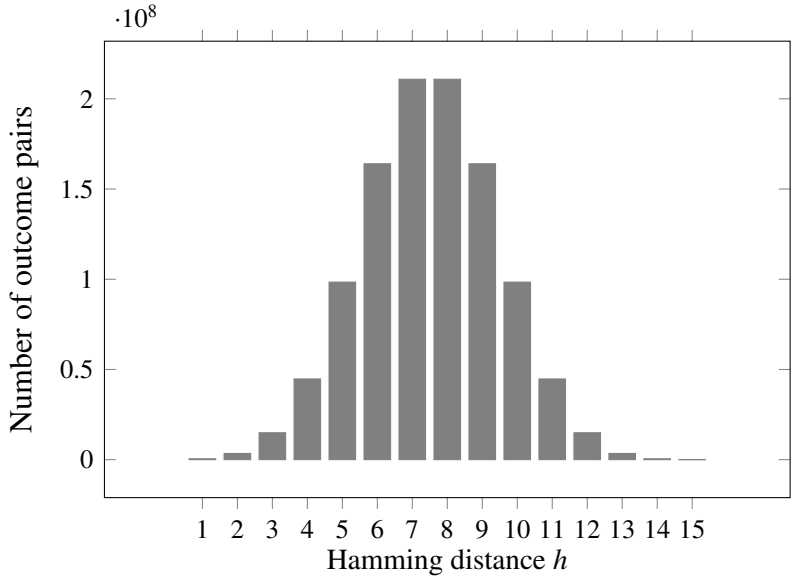


Figure 5.12: Distribution of HD for $n = 15$

LARGER-CASES

Output: database consisting of DT instances and flipping lengths

```
1:  initialize database
2:  d ← 2                                                        ▷ binary domains
3:  for n ← 10 to 15 do                                          ▷ number of features
4:      P ← ∅
5:      for h ∈ {2, ⌈n/2⌉, n} do                                 ▷ Hamming distance
6:          for t ← 1 to 100 do                                  ▷ generate 100 outcome pairs
7:              P ← P ∪ random outcome pair (o, o′) ∈ O²_{n,d|h}
8:          end for
9:      end for
10:     for c ← 1 to 4 do                                        ▷ indegree bounded by small constant
11:         N ← ∅
12:         for t ← 1 to 100 do                                  ▷ generate 100 CP-nets
13:             N ← N ∪ RANDOM-CP-NET(n, c, d)
14:         end for
15:     end for
16:     for all N ∈ N do                                         ▷ iterate over sampled CP-nets
17:         for all (o, o′) ∈ P do                               ▷ iterate over sampled outcome pairs
18:             ℓ ← ITERATIVE-DT*(N, o, o′, ∞)                   ▷ compute flipping length
19:             store (n, c, d, h, N, o, o′, ℓ) in database
20:         end for
21:     end for
22: end for
23: return database
```

Figure 5.13: Third Experiment: Sample Outcome Pairs and CP-nets

GENERATE-CP-NET is then called to generate 100 CP-nets for each value of n and c as for the first two experiments. The nested loops at Line 16 and Line 17 iterate over all DT instances $\mathcal{N} \times \mathcal{P}$, such that the same outcome pairs are applied to every CP-net N. Finally, observe that Line 18 calls DEPTH-LIMITED-DT* rather than the Floyd–Warshall algorithm to perform DT and determine the flipping length for each instance. DEPTH-LIMITED-DT*, adapted from the DT* algorithm of Li et al. [68], is discussed in Section 5.6. Note that for this experiment the depth limit k is set to ∞ and search continues indefinitely until a solution is found or the algorithm reports **false** indicating that no flipping sequence exists at any depth. I performed a similar experiment for multivalued domains of size $d = 3$.

Table 5.6: Mean Flipping Length Given n, c, and h (Binary Variables, n = 10 to 15)

Hamming distance $h = 2$

n	h	$c = 1$	2	3	4
10	2	2.75	2.48	2.27	2.34
11	2	2.57	2.47	2.42	2.43
12	2	2.52	2.40	2.57	2.52
13	2	2.44	2.58	2.52	2.51
14	2	2.60	2.54	2.58	2.56
15	2	2.81	2.38	2.58	2.59

Hamming distance $h = \lceil n/2 \rceil$

n	h	$c = 1$	2	3	4
10	5	6.50	6.47	6.14	5.79
11	6	7.82	7.61	7.33	7.00
12	6	7.82	8.03	7.62	7.07
13	7	9.61	9.19	8.82	8.38
14	7	10.21	9.45	8.93	8.41
15	8	10.86	10.73	10.01	9.53

Hamming distance $h = n$

n	h	$c = 1$	2	3	4
10	10	10.00	10.12	10.04	10.00
11	11	11.00	11.26	11.08	11.01
12	12	12.00	12.32	12.07	12.01
13	13	13.00	13.37	13.14	13.02
14	14	14.00	14.38	14.10	14.07
15	15	15.00	15.39	15.10	15.04

The experiment is the same as shown in Figure 5.13, except that it assigns variable d to 3 in Line 2 and that it specifies in Line 3 that n ranges from 5 to 10 instead of 10 to 15. Tables 5.6 and 5.7 show results similar to those in the previous experiment for smaller values of n.

Table 5.7: Mean Flipping Length Given n, c, and h (Multivalued Variables $d = 3$)

Hamming distance $h = 2$

n	h	$c = 1$	2	3	4
5	2	2.42	2.42	2.37	2.27
6	2	2.40	2.55	2.45	2.31
7	2	2.52	2.74	2.54	2.46
8	2	2.72	2.97	2.62	2.55
9	2	2.76	2.62	2.66	2.61
10	2	2.71	2.78	2.74	2.59

Hamming distance $h = \lceil n/2 \rceil$

n	h	$c = 1$	2	3	4
5	3	3.67	3.66	3.42	3.38
6	3	3.96	3.80	3.52	3.47
7	4	5.52	5.47	4.81	4.72
8	4	5.80	5.71	5.09	4.78
9	5	7.16	7.08	6.42	5.97
10	5	8.37	7.28	6.67	6.19

Hamming distance $h = n$

n	h	$c = 1$	2	3	4
5	5	5.83	5.64	5.42	5.43
6	6	7.19	6.80	6.66	6.51
7	7	8.69	8.16	7.74	7.55
8	8	10.20	9.41	8.85	8.72
9	9	11.14	10.65	9.99	9.90
10	10	12.68	11.97	11.13	10.91

5.5 Are Preferences Really Transitive?

To this point, we have assumed, except for the brief discussion in Section 1.1.3, that preferences are transitive. That is, if a subject prefers $o \succ o'$ and $o' \succ o''$, then we reason that the subject also prefers $o \succ o''$. This assumption is the basis for constructing flipping sequences via the *ceteris paribus* rules of the CP-net. However, let us briefly reconsider Peter C. Fishburn's example [36] from Chapter 1. This time, instead of a long sequence of cups of coffee, each with one more grain of sugar than its predecessor, consider that we have a similarly long sequence of beverages described by multiple features, such as sweetness, temperature, provenance, and the presence of varied subtle flavorings. The subject's preference over the combinatorial domain is described with a CP-net, and a DT algorithm assures us that the subject will prefer beverage o to o' due to very long flipping sequence, as shown in Figure 5.14. We saw in Sections 5.2–5.4 that flipping sequences much longer than the Hamming distance are rare. However, sequences of length $O(n^2)$ certainly exist even in binary CP-nets with complete tables, and Boutilier et al. [16] showed that in certain cases flipping lengths can be exponential in n.

Recall that each flip in the sequence is entailed by a particular CPR. As discussed in Chapter 1, CP-nets are considered to be a static, deterministic formalism. That is, it is assumed that for each outcome pair that differs in just one feature, the subject always makes the same choice. One should keep in mind, however, that this determinism is a *modeling* decision, not an intrinsic property of the subject's underlying preferences. Whether N is constructed by the subject, elicited through queries, or learned from data in any of the ways discussed in Sections 1.2.3 and 3.5, it is reasonable to allow for the possibility that

Figure 5.14: An Exceptionally Long Flipping Sequence

Table 5.8: Noise Model for Maximum Reliable Flipping Lengths

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

the model reflects some margin of error, or *noise*. If such errors are small, they can be safely ignored. However, the presence of noise turns out to be particularly problematic for very long flipping sequences.

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

$$p = (1 - \epsilon)^{\ell}, \tag{5.5}$$

which converges to 0 as ℓ tends to infinity.[8] Assuming $p \geq 0.5$ and solving for ℓ gives us

$$\ell \leq \left\lfloor \frac{-1}{\log_2(1 - \epsilon)} \right\rfloor. \tag{5.6}$$

Table 5.8 shows the longest reliable values of ℓ for varying noise levels ϵ.

The problem of noise is of course not limited to CP-nets. *Any* predictive model, if it admits too much noise, becomes unreliable. The tendency of very long transitive sequences to accumulate noise, however, is a further reason to limit search depth.

5.6 Depth-Limited Dominance Testing

Sections 5.2–5.5 can be summarized as follows: *The deeper the search in DT, the lower the probability that a path exists, and the higher the probability that the path is unreliable.* It thus seems reasonable in many settings to limit search depth, as we now discuss.

[8] Here I consider only a single flipping sequence from o' to o. Multiple paths from o' to o of length ℓ are possible, as in Figure 5.2, but this complicates the analysis.

Definition 38 (Depth-limited dominance)**.** *Let* (N, o, o') *be an arbitrary DT instance. For any positive integer k, N entails the* depth-k dominance *of o over o′,*

$$N \models o \succ^k o', \tag{5.7}$$

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

We call the search for such a sequence *depth-limited dominance testing* (DLDT). A general algorithm for conducting DLDT is described in Figure 5.15. The algorithm takes as its input a DT instance, specified depth limit k, and a reference to a blackbox subroutine *DTSolver* that returns **true** if a path of length $\ell \leq k$ exists and *may* return **unknown** if the cutoff is reached before the search tree is fully explored. The loop in Line 2 iterates over feasible flipping lengths from $\ell = \mathrm{HD}(o, o')$ to k in increments of 2 (see Theorems 34 and 36). The algorithm returns the length $\ell \leq k$ of a flipping sequence from o' to o, if such a sequence exists, or ∞ if $o \not\succ^k o'$.

Let us briefly discuss two algorithms that can serve as the *DTSolver* subroutine. Section 5.6.1 shows how to adapt the DT* algorithm proposed by Li et al. [68]. Section 5.6.2 then proposes SAT-DT, a novel approach that solves DT instances of specified flipping length via reduction to Boolean satisfiability (SAT). The algorithms can easily be adapted to return the flipping sequence as well if that is desired.

5.6.1 Depth-Limited DT*

DT* employs a heuristic approach to dominance testing. A priority queue is employed, with a heuristic function on any outcome o^* to guide the search. Nodes on the fringe of the search tree with the lowest positive values of $f(o^*)$ are searched first; negative values of $f(o)$, however, rule out any possibility of finding a solution. Pseudocode, adapted from Li et al. [68], is provided in Figure 5.16. The boxes emphasize changes that are central to the iterative approach employed here for DLDT. Note that if the depth limit ℓ is reached,

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

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

1: $s \leftarrow 2$ if N is binary valued and 1 otherwise
2: **for** $\ell \leftarrow \mathrm{HD}(o, o')$ **to** k **step** s **do** ▷ Iterative search depth
3: $\quad$ *answer* $\leftarrow$ *DTSolver* (N, o, o', ℓ) ▷ Invoke subroutine
4: $\quad$ **if** *answer* = **true then**
5: $\quad\quad$ **return** ℓ ▷ Return flipping length ℓ indicating $N \models o \succ^{\ell} o'$
6: $\quad$ **else if** *answer* = **false then**
7: $\quad\quad$ **return** ∞ ▷ No solution exists at any depth, so halt search
8: $\quad$ **end if**
9: **end for**
10: **return** ∞ ▷ No solution with flipping length $\ell \leq k$

Figure 5.15: Generic Algorithm: Depth-Limited Dominance Testing

no successor nodes in the search tree will be added to the priority queue. If this occurs, the algorithm returns **unknown**. If all solutions can be ruled out, it returns **false**.

5.6.2 DT-SAT

The section concludes with a reduction of DLDT to SAT, as shown in Figure 5.17. For this reduction outcomes are modeled as *states* and flips as *actions* that transition between states, employing the *satisfiability as planning* (SATPlan) method of Kautz and Selman [54]. The variable ω denotes a Boolean formula in Conjunctive Normal Form (CNF), i.e., a conjunction of *clauses*, each a disjunction of *literals*. Initially ω is *empty*, with an assumed truth value of **true**, which we denote with the assignment $\omega \leftarrow \top$. To *write* a clause ξ means to conjoin it to ω to form a new formula, $\omega \leftarrow \omega \wedge \xi$. When all clauses have been written thus, a *SAT solver* is called. It is assumed that the solver returns **true** if ω is satisfiable and **false** otherwise.

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

Input: N CP-net
o Goal outcome
o' Start outcome
k Depth limit

Output: result **true** if $N \models o \succ^k o'$, **false** if $N \models o' \not\succ o$; else **unknown**

```
1:  if f(o') < 0 then                                          ▷ Heuristic function [68]
2:     return false
3:  end if
4:  cutoff ← false
5:  insert (o', 0) into priority-queue with priority f(o')
6:  while priority-queue ≠ ∅ do
7:     (o', ℓ) ← remove-first(priority-queue)
8:     if o' = o then                                          ▷ Goal test
9:        return true
10:    end if
11:    for all X_i ∈ V do
12:       if improvable(o', X_i) ∧ X_i ∉ any-matching-suffix(o', o) then
13:          o'' ← single-flip(o', X_i)
14:          if not-repeated(o'') ∧ f(o'') ≥ 0 then
15:             if ℓ ≥ k then                  ▷ Can we flip and not exceed depth limit?
16:                cutoff ← true               ▷ Thus will not search complete graph
17:             else
18:                insert (o'', ℓ + 1) into priority-queue with priority f(o'')
19:             end if
20:          end if
21:       end if
22:    end for
23: end while
24: if cutoff = true then
25:    return unknown
26: else
27:    return false
28: end if
```

Figure 5.16: Solver Algorithm: Depth-Limited DT*

DT-SAT(N, o, o', ℓ)

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

1: $\omega \leftarrow \top$
2: **for** $i \leftarrow 1$ **to** n **do** ▷ Assert initial and final states
3: $\quad \omega \leftarrow \omega \wedge z_{0,i,o'[i]}$
4: $\quad \omega \leftarrow \omega \wedge z_{t,i,o[i]}$
5: **end for**
6: **for** $t \leftarrow 1$ **to** ℓ **do** ▷ Just one state occurs for all t, i
7: $\quad$ **for** $i \leftarrow 1$ **to** n **do**
8: $\quad\quad \omega \leftarrow \omega \wedge \underset{1 \le j \le d}{\mathfrak{JustOne}}\ z_{t,i,j}$
9: $\quad$ **end for**
10: **end for**
11: **for** $t \leftarrow 1$ **to** $\ell - 1$ **do**
12: $\quad$ **for** $i \leftarrow 1$ **to** n **do**
13: $\quad\quad$ **for** distinct $i, j \le d$ **do**
14: $\quad\quad\quad \omega \leftarrow \omega \wedge \alpha_{t,i,j,k} \Rightarrow z_{t,i,j} \wedge z_{t+1,i,k}$ ▷ Implications of actions
15: $\quad\quad\quad$ **for** $h \leftarrow 1$ **to** n s.t. $h \neq i$ **do** ▷ Framing rules
16: $\quad\quad\quad\quad$ **for** $q \leftarrow 1$ **to** d **do**
17: $\quad\quad\quad\quad\quad \omega \leftarrow \omega \wedge \alpha_{t,i,j,k} \wedge z_{t,h,q} \Rightarrow z_{t+1,h,q}$
18: $\quad\quad\quad\quad\quad \omega \leftarrow \omega \wedge \alpha_{t,i,j,k} \wedge \neg z_{t,h,q} \Rightarrow \neg z_{t+1,h,q}$
19: $\quad\quad\quad\quad$ **end for**
20: $\quad\quad\quad$ **end for**
21: $\quad\quad$ **end for**
22: $\quad$ **end for**
23: $\quad \omega \leftarrow \omega \wedge \underset{1 \le i \le n, 1 \le j,k \le d}{\mathfrak{JustOne}}\ \alpha_{t,i,j,k}$
24: $\quad$ **for each** CPR in N of the form $u : x_j^i \succ x_k^i$ **do**
25: $\quad\quad \omega \leftarrow \omega \wedge z_{t,p_1,u_1} \wedge \cdots \wedge z_{t,p_m,u_m} \Rightarrow \neg \alpha_{t,i,j,k}$ ▷ Disallow flip unless CPR permits
26: $\quad$ **end for**
27: **end for**
28: *answer* $\leftarrow$ SAT-solver(ω)
29: **if** *answer* = **true then**
30: $\quad$ **return true**
31: **else**
32: $\quad$ **return unknown**
33: **end if**

Figure 5.17: Solver Algorithm: DT-SAT

States

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

$$z_{t,i,j} \iff o_t[i] = x_j^i, \tag{5.8}$$

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

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

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

Actions

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

$$\alpha_{t,i,j,k} \implies z_{t,i,j} \wedge z_{t+1,i,k}, \tag{5.9}$$

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

$$\alpha_{t,i,j,k} \wedge z_{t,h,q} \implies z_{t+1,h,q} \tag{5.10}$$

$$\alpha_{t,i,j,k} \wedge \neg z_{t,h,q} \implies \neg z_{t+1,h,q} \tag{5.11}$$

Modeling the CPRs

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

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

$$z_{t,p_1,u_1} \wedge z_{t,p_2,u_2} \wedge \cdots \wedge z_{t,p_m,u_m} \Rightarrow \neg\alpha_{t,i,j,k}. \tag{5.12}$$

5.7 Conclusion

An oft-cited objection to the use of CP-nets is that the problem of dominance testing, deciding whether one outcome is preferred to another, is NP-hard. Moreover, the proofs of dominance, so-called flipping lengths, can be exponentially long. In the experiments described in this chapter, I have shown that while long sequences exist, they are rarer than sometimes supposed and tend to occur most often for dependency graphs with long average path lengths. Expected flipping length also depends on factors such as the number of variables, maximum indegree of nodes in the dependency graph, domain size, and the Ham-

ming distance of the outcomes. I have shown that limiting search depth is often reasonable and showed how to achieve this by adapting an existing heuristic solver to perform iteratively deepening, depth-limited search, as well as through a reduction to SAT to leverage the heuristics in modern solvers. In future work, I hope to show how to leverage parameters such as average path length and Hamming distance to enable portfolio approaches to dominance testing.

Chapter 6 Local Search for Learning Tree-Shaped CP-nets

This chapter *proposes a novel method for learning tree-shaped CP-nets from choice data.* It is sometimes claimed that CP-nets are "easy to elicit" [16]. That is, we explain CP-nets to the user, and she introspects on her preferences and writes down the CP-net that best describes her decision-making process when comparing alternatives. This is a fairly common assumption in the CP-net literature [18, 30]. 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 [39, 40, 97, 103].

On the other hand, when presented with alternatives, people often do 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 elicitation algorithms depend on *choice data*—sets of binary choices from some domain. Some algorithms assume that the choices have been made prior to run time [23, 64], 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 the user about their preferences directly [23, 29, 47] or Angluin-style learning queries [58]. (See Sections 1.2.3 and 3.5.)

Both in psychological experiments and in many preference data sets, such as those in the PrefLib repository [75], one can find examples of noisy and inconsistent preferences. For example, Kamishima and Akaho [53] 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 directly only minutes before. There are many explanations for such phenomena, including a lack of computational power to compute optima consistently, and a very human desire for variety

[76]. To our knowledge only Liu et al. [70] have addressed the problem of learning CP-nets from noisy data. However, their algorithm explicitly builds a preference graph on the outcomes, which is exponentially larger than the CP-net. The algorithm we present implicitly represents the preference graph by only constructing the CP-net and is far more efficient with respect to space.

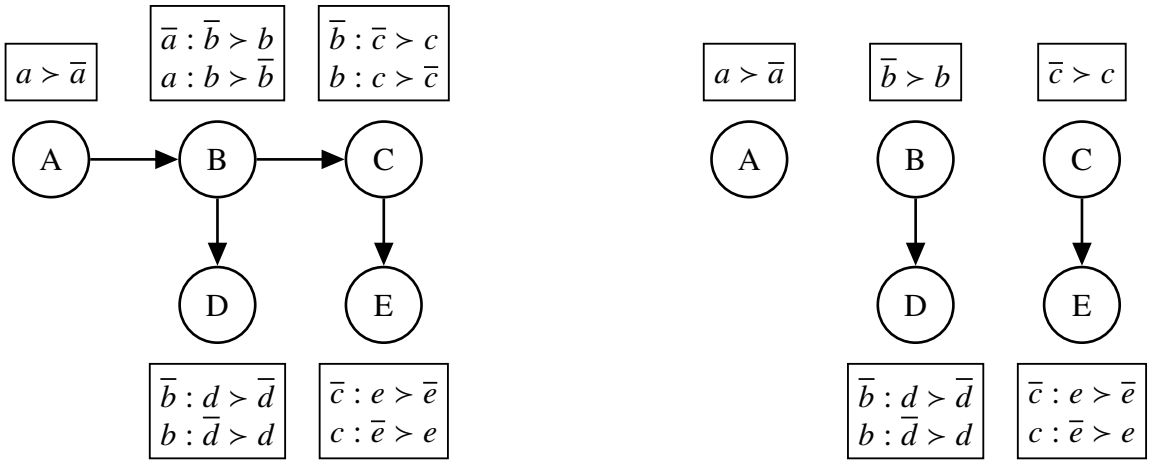
The approach that we take in this chapter falls into the category of passive learning, in which the agent unobtrusively records the user's choices and then fits a CP-net model to the observed pairwise comparison data. Note that this approach to learning differs from approaches in which "big data" from large populations are systematically mined in an effort to predict user behavior. In our case, the data arise, for example, from an individual for whom we are trying to customize a particular system.

Recall that many problems involving CP-nets and their variants, including learning a CP-net that is consistent with comparison data (Section 3.5) and using that CP-net to determine which of two arbitrary outcomes is preferred (Section 3.4), are known to be computationally hard. However, the dominance problem is known to be easy for CP-nets for which the dependency structure is a tree. Here we restrict our attention to this subclass of CP-nets in order to take advantage of efficient dominance testing.

In Section 6.1 we review local search as well as graph theoretic concepts that are needed to explain our approach. In Section 6.2 we offer a novel encoding for tree-shaped CP-nets that is useful in defining a neighborhood in which local search can be conducted. In Section 6.3 we consider how to evaluate prospective models with respect to a set of possibly noisy of example data. We discuss the learning algorithm itself in Section 6.4. In Section 6.5 we evaluate the algorithm with experiments. A conclusion follows in Section 6.6.

6.1 Background

A *tree-shaped* CP-net is a conditional preference network (see Definition 9) for which the dependency graph takes the shape of a *directed forest*. The term *tree* in this context is



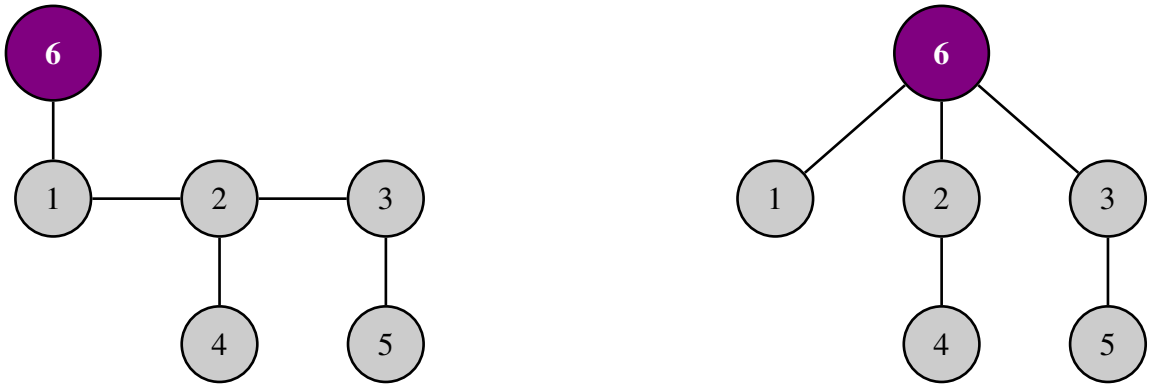
(a) CP-net N_1 (Connected Digraph) (b) CP-net N_2 (Disconnected)

Figure 6.1: Tree-shaped CP-nets

potentially confusing, since the digraph may have more than one connected component; our terminology follows that of Boutilier et al. [16]. Recall from Section 2.2 that a *directed tree* is a digraph $G = (V, E)$ such that, for just one node s, called the *root*, and every other node t, where $s \in V$, $t \in V$, $s \neq t$, there exists just one path from s to t. In contrast, a *directed forest* is a digraph in which each node has at most one parent; thus, a directed tree is also a directed forest, but the converse does not hold. One can observe that the number of *roots* is the same as the number of connected components or directed trees that make up the directed forest. Figure 6.1 depicts two tree-shaped CP-nets, one in which the dependency graph is a directed tree, the other in which it is a directed forest.

Unless otherwise noted, a CP-net in this chapter is assumed to be tree-shaped. We also assume here that the domain of each variable is binary and that each conditional preference table (CPT) of the network is complete, i.e., contains a conditional preference rule (CPR) for each assignment to the variables that label the node's parents in the dependency graph, which in this case will be either a single parent, or the empty set for nodes that are roots.

In this chapter, we represent a directed forest as an *un*directed tree. A *tree*, when used without any modifier, refers to a *labeled*, *undirected* graph $G = (V, E)$ in which any two vertices, $u \in V$, $v \in V$, $\{u, v\} \in E$ $u \neq v$, are connected by just one path. Similarly, a



(a) Tree T_1 (b) Tree T_2

Figure 6.2: Rooted Trees

forest, if not qualified as *directed*, is a union of disjoint, labeled, undirected trees. We may distinguish one node of a tree as a *root*. In that case, the tree is a *rooted tree*; otherwise, the tree is assumed to be *unrooted*. In this chapter, since we work with both directed and undirected graphs, we will refer to directed graphs as *digraphs* and to undirected graphs simply as *graphs*. Figure 6.2 depicts two rooted trees.

We denote by $\mathcal{E} = \{E_1, E_2, \ldots, E_\ell\}$ a set of *example data*. We also refer to these data as *choice data* or *outcome comparisons*. Each element $E_t = (o_t, o'_t)$ of the set is an outcome pair, $o_t \in \mathcal{O}$, $o'_t \in \mathcal{O}$, in which the first element is preferred to the second, i.e., $o_t \succ o'_t$, for all t, $1 \leq t \leq \ell$. Such data are intended as a model of the observed choices of an individual user over discrete time t. In general such data may be *inconsistent*; that is, the transitive closure of $\mathcal{E}$ may result in a cycle in which some outcome o is seen to be preferred to itself.

When we say our goal is to *learn* a CP-net from data $\mathcal{E}$, we mean that we want to identify, from among a set of models $\mathcal{N}$, one that is *most consistent with*, or *best explains*, $\mathcal{E}$, as described in Section 6.3. Since in general it is infeasible to search the whole space of $|\mathcal{N}|$ models, we will rely on a heuristic approach known as *local search*, in which the *fitness* $f(N)$ of a proposed model N is compared with that of its *neighbors*, Neigh(N), as discussed in Section 6.4. The term *neighborhood* refers to the set of models Neigh(N) $\cup$ $\{N\}$. If a model is as fit as any model in its neighborhood, i.e., $f(N) \geq f(N')$ for all $N' \in$ Neigh(N),

then we say that it is *locally optimal*. If a model is as fit as any model in $\mathcal{N}$, it is furthermore a *global optimum.* If two or more neighbors are equally fit and are as fit as their respective neighbors, we say they belong to a *plateau*. For further discussion of local search, the reader is referred to a textbook such as that of Russell and Norvig [91].

The GSAT [95] and WalkSAT [96] algorithms for Boolean satisfiability serve as inspirations for the method that we describe in Section 6.4. We are not aware of any previous work applying local search to CP-nets or related formalisms. Various methods for learning CP-nets are discussed in Section 3.5. The three algorithms that are most relevant to the problem that we describe here are those of Dimopoulos et al. [29], which assumes that example data are consistent and reports failure if no model satisfies $\mathcal{E}$; of Liu et al. [70], which like ours identifies optima in the presence of possibly noisy examples, but requires exponential space, since it explicitly constructs the preference graph while our method extends to larger problem instances; and finally of Bigot et al. [12], which discusses how to learn tree-shaped *probabilistic* conditional preference networks (PCP-nets) from *optimal outcomes*, while our method learns a deterministic model from pairwise comparison data.

6.2 Encoding Tree-shaped CP-nets

In Section 4.3 we showed how to encode acyclic dependency graphs as dagcodes. For tree-shaped CP-nets, however, a simpler encoding is available that facilitates local search by making it more straightforward to define neighbors. Notice that the dependency graph $G = (V, E)$ of a so-called tree-shaped CP-net is equivalent to 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 each tree in G_F from its root. Furthermore, one may notice that there is a nice correspondence between trees and forests of rooted trees: 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 trees T_1 and T_2 in Figure 6.2 correspond to the dependency graphs of N_1 and N_2 in Figure 6.1 according to this mapping.

Once the digraph of a tree-shaped CP-net with n nodes has been modeled as a tree with $n + 1$ nodes, it can be further modeled as a vector of integers known as a *Prüfer code*.[1] As Arthur Cayley proved in the 1800s, for every integer k, the number of unrooted trees with k labeled nodes is k^{k-2}. Heinz Prüfer later showed how to encode each such instance as 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 $L_j \in \{1, \ldots, k\}$, $1 \leq j \leq k - 2$. Any unrooted labeled tree T can be mapped to its corresponding Prüfer code L, and vice versa, by straightforward algorithms that we present in Figures 6.3 and 6.4, which closely follow Kreher and Stinson [60, 3.3].

To encode the CPTs, we could of course use the mapping described in Section 4.4. Again, however, a simpler mapping is available since we are dealing with tree-shaped CP-nets with binary variables. Recall from Equation 4.3 that the number of non-degenerate CPTs of a node with m parents is

$$\psi_2(m) = \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} 2^{2^k}. \qquad \text{(4.3 revisited)}$$

Moreover, since each node has at most 1 parent, Equation 4.3 evaluates to

$$\psi_2(0) = \psi_2(1) = 2 \tag{6.1}$$

for every CPT of a tree-shaped CP-net composed of binary variables.[2] We can thus represent the CPTs of a tree-shaped CP-net as a vector B of n integers $B[i] = 0$ or $B[i] = 1$, $1 \leq i \leq n$, according to the mapping

$$\begin{aligned} \boxed{\overline{x_i} \succ x_i} \quad &\longleftrightarrow \quad B[i] = 0 \\ \boxed{x_i \succ \overline{x_i}} \quad &\longleftrightarrow \quad B[i] = 1 \end{aligned} \tag{6.2}$$

[1] Note that a Prüfer code of a tree with n nodes consists of $n - 2$ integers, each ranging from 1 to n. There are thus n^{n-2} trees on n labeled nodes. However, since the tree-shaped dependency graphs described by Boutilier et al. [16] actually correspond to forests (!), we have to add an additional node $n + 1$ to the representation. Thus, in the case of so-called tree-shaped CP-nets, the corresponding Prüfer code actually consists of $n - 1$ integers ranging from 1 to $n + 1$.

[2] Note that $\psi_d(0) \neq \psi_d(1)$ for $d > 2$.

TREE-TO-PRÜFER-CODE(E, n)

Input: E undirected edges of the tree
n number of labeled nodes
Output: L Prüfer code consisting of $n-2$ integers

```
1:  d ← 0 vector of length n
2:  for all {j, k} ∈ E do                ▷ compute the degrees d of the n nodes
3:      d[j] ← d[j] + 1
4:      d[k] ← d[k] + 1
5:  end for
6:  for i ← 1 to n − 2 do                ▷ on i^th iteration we have a tree on n + 1 − i nodes
7:      j ← n
8:      while d[j] ≠ 1 do                ▷ find j, the largest labeled node of degree 1
9:          j ← j − 1
10:     end while
11:     k ← n
12:     while {j, k} ∉ E do              ▷ find edge {j, k} ∈ E
13:         k ← k − 1
14:     end while
15:     L[i] ← k                         ▷ store label k in code
16:     d[j] ← d[j] − 1                  ▷ reduce the degrees j and k
17:     d[k] ← d[k] − 1
18:     E ← E \ {{j, k}}                 ▷ and delete that edge
19: end for
20: return L = ⟨L[i], ..., L[n − 2]⟩
```

Figure 6.3: Algorithm: Tree to Prüfer Code [60]

for each node X_i with no parents and

$$\boxed{\begin{array}{l}\overline{x_h} : \overline{x_i} \succ x_i \\ x_h : x_i \succ \overline{x_i}\end{array}} \quad \longleftrightarrow \quad B[i] = 0 \qquad \boxed{\begin{array}{l}\overline{x_h} : x_i \succ \overline{x_i} \\ x_h : \overline{x_i} \succ x_i\end{array}} \quad \longleftrightarrow \quad B[i] = 1 \tag{6.3}$$

for each remaining node X_i with a single parent X_h, $X_h \in \mathcal{V}$, $X_i \in \mathcal{V}$, $1 \leq h \leq n$, $1 \leq i \leq n$.

By combining the encodings for the digraph and CPTs, it is possible to model every tree-shaped CP-net N as a *treecode* (L, B) consisting of a Prüfer code L with $n-1$ integers $L_j \in \{1, \ldots, n+1\}$, $1 \leq j \leq n-1$, for the dependency graph, and a *bit vector* B with n bits for the CPTs. The algorithms of Figures 6.5 and 6.6 show how to map an arbitrary

PRÜFER-CODE-TO-TREE(L, n)

Input: L Prüfer code consisting of $n-2$ integers
n eventual number of labeled nodes
Output: E directed edges composing the tree

1: $E \leftarrow \emptyset$
2: $L[n-1] \leftarrow 1$
3: $d \leftarrow$ vector of n 1s
4: **for** $i \leftarrow 1$ **to** $n-2$ **do** ▷ compute degrees from L
5: $d[L[i]] \leftarrow d[L[i]] + 1$
6: **end for**
7: **for** $i \leftarrow 1$ **to** $n-1$ **do**
8: $j \leftarrow n$
9: **while** $d[j] \neq 1$ **do** ▷ find j, the largest labeled node of degree 1
10: $j \leftarrow j-1$
11: **end while**
12: $k \leftarrow L[i]$
13: $d[j] \leftarrow d[j]-1$ ▷ reduce the degrees of j and k in L
14: $d[k] \leftarrow d[k]-1$
15: $E \leftarrow E \cup \{\{j,k\}\}$ ▷ insert the edge into E
16: **end for**
17: **return** E

Figure 6.4: Algorithm: Prüfer Code to Tree [60]

tree-shaped CP-net N to its corresponding treecode (L, B), and vice versa. Also, Figure 6.7 completes the examples in Figures 6.1 and 6.2 by supplying the corresponding treecodes.

In the theorems and proofs that follow, we denote by ${}^{T}\mathcal{N}_n$ the set of all complete tree-shaped CP-nets on n binary variables and by $\mathcal{T}_n$ the corresponding set of treecodes

$$\mathcal{T}_n = \mathcal{L}_n \times \mathcal{B}_n, \tag{6.4}$$

where $\mathcal{L}_n$ denotes the set of all Prüfer codes $L = \langle L_1, \ldots, L_{n-1} \rangle$ with $n-1$ integers ranging from 1 to $n+1$ and $\mathcal{B}_n$ is the set of all bit vectors $B = \langle B_1, \ldots, B_n \rangle$ with n bits $B_i \in \{0, 1\}$. We denote by $\mathrm{TC} : {}^{T}\mathcal{N}_n \rightarrow \mathcal{T}_n$ the function mapping tree-shaped CP-nets to treecodes, as implemented by TREE-CP-NET-TO-TREECODE as shown in Figure 6.5.

The algorithms presented in Figures 6.5 and 6.6 map treecodes to CP-nets, and vice-versa, suggesting that the algorithms are inverses. We now formalize this relationship.

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

Input: N Tree-shaped CP-net with binary variables
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 mapping in (6.2) and (6.3)
3: **end for**
4: relabel nodes V in N with the integers 1 to n
5: $R \leftarrow$ nodes of digraph G of N that are roots
6: $V \leftarrow V \cup \{n+1\}$
7: $E \leftarrow E \cup \{(n+1, r)\}$ for all $r \in R$
8: make edges E undirected
9: $L \leftarrow$ TREE-TO-PRÜFER-CODE($E, n+1$) ▷ Figure 6.3
10: **return** (L, B)

Figure 6.5: Algorithm: Tree-shaped CP-net to Treecode

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

Input: L Prüfer code
B Bit vector encoding the CPTs
Output: N Tree-shaped CP-net

1: $n \leftarrow$ length of B
2: assert that L has length $n-1$
3: $V \leftarrow \{1, \ldots, n+1\}$
4: $E \leftarrow$ PRÜFER-CODE-TO-TREE(L, n) ▷ Figure 6.4
5: $G = (V, E)$
6: traverse G from node $n+1$ assigning directions to edges in order of traversal
7: $V \leftarrow V \setminus \{n+1\}$ ▷ delete node $n+1$ and its edges
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 mapping in (6.2) and (6.3)
11: **return** N

Figure 6.6: Algorithm: Treecode to Tree-shaped CP-net

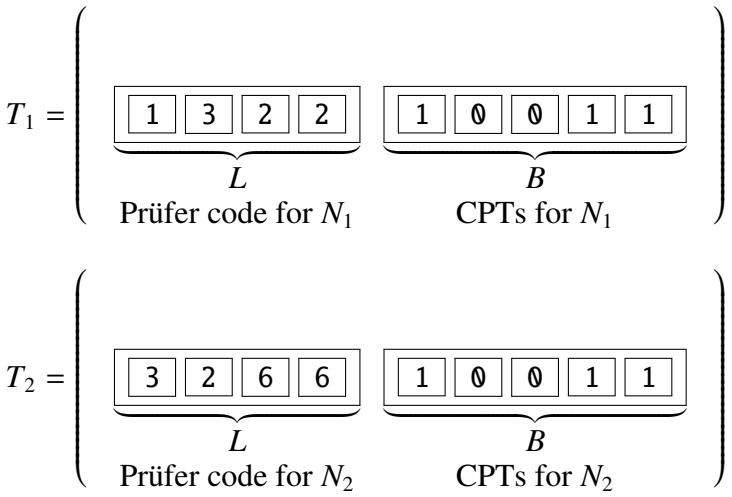


Figure 6.7: Treecodes Corresponding to the Tree-shaped CP-nets in Figure 6.1

Theorem 39 (Treecode Bijection)**.** TC *is a bijection.*

Proof. Let N and N' denote arbitrary tree-shaped CP-nets on n binary variables, where $N \in {}^{T}\mathcal{N}_n$, $N' \in {}^{T}\mathcal{N}_n$, $n > 0$.

Suppose $\mathrm{TC}(N) = \mathrm{TC}(N')$. If so, N and N' must have the same Prüfer code L, hence the same digraph G. They also must have the same bit vector B; hence, the CPTs of N and N' must also be the same for each node. Thus, $N = N'$.

Next, let T be an arbitrary treecode in $\mathcal{T}_n$. Recall that for every Prüfer code $L = \langle L_1, \ldots, L_{n-1} \rangle$, where $L_j \in \{1, \ldots, n+1\}$, $1 \leq i \leq n$, there exists a labeled, unrooted tree with $n+1$ nodes with labels $V = \{1, \ldots, n+1\}$. Since $n > 0$, there exists a unique node with largest label $n+1$. The graph, as a tree, must be connected and free of cycles, so we will always be able to traverse it from node $n+1$ to every other node, applying directions to each edge in the direction of traversal. We can then delete node $n+1$ along with its edges. Consider that the resulting digraph will be a directed forest that can serve as the dependency graph G of a tree-shaped CP-net N. Furthermore, by relabeling each node i as X_i and specifying that $\mathrm{Dom}(X_i) = \{\overline{x_i}, x_i\}$, $1 \leq i \leq n$, the digraph will be in its canonical form. Further consider that once digraph G is available, we will know the parent, if any, of each node X_i, allowing us to obtain its non-degenerate CPT from B_i via the mapping in

Table 6.1: Number of Tree-Shaped CP-Nets (Respectively Treecodes) with n Binary Nodes

n	$\lvert\mathcal{T}_n\rvert$
1	2
2	12
3	128
4	2000
5	41472
6	1075648
7	33554432
8	1224440064
9	51200000000
10	2414538435584

Equations 6.2 and 6.3. It follows that for all $T = (L, B)$, $T \in \mathcal{T}_n$, $L \in \mathcal{L}_n$, $B \in \mathcal{B}_n$, $n > 0$, there exists $N \in {}^{T}\mathcal{N}$ such that $\mathrm{TC}(N) = T$.

Therefore, tree-shaped CP-nets are in one-to-one correspond with treecodes. ❑

Theorem 40 (Number of Tree-shaped CP-nets)**.** *For every non-negative integer n, the number of tree-shaped CP-nets of n binary variables is*

$$\left|{}^{T}\mathcal{N}_n\right| = 2^n(n+1)^{n-1}. \tag{6.5}$$

Proof. The set of treecodes $\mathcal{T}_n$ consists of all combinations $\mathcal{L}_n \times \mathcal{B}_n$. Thus, $|\mathcal{T}_n| = |\mathcal{L}_n||\mathcal{B}_n|$. Since a Prüfer code $L \in \mathcal{L}_n$ consists of $n-1$ integers ranging from 1 to $n+1$ inclusive, $|\mathcal{L}_n| = (n+1)^{n-1}$. Also, there are $|\mathcal{B}_n| = 2^n$ bit vectors of length n. We have already shown that ${}^{T}\mathcal{N}_n$ and $\mathcal{T}_n$ are in one-to-one correspondence. Therefore, $|{}^{T}\mathcal{N}_n| = 2^n(n+1)^{n-1}$. ❑

Table 6.1 gives the computed value of Equation 6.5 for $n = 1$ to 10 (also see Sloane [98, A097629]). Note that, as expected, the values are identical to the number of acyclic CP-nets with n binary variables and bound 1 on indegree, $a_{n,1,2}$ (see Theorem 30 in Section 4.4).

With the help of the encoding and bijection, it is then straightforward to generate all tree-shaped CP-nets on n binary variables, as illustrated by the pseudocode in Figure 6.8.

ALL-TREE-SHAPED-CP-NETS(n)

Input: n Number of binary variables
Output: $\mathcal{N}$ Set of all tree-shaped CP-nets

$\mathcal{N} \leftarrow \emptyset$
$L \leftarrow$ vector of $n-1$ 1s ▷ first treecode lexicographically
$B \leftarrow$ vector of n 0s
first $\leftarrow (L, B)$
repeat
  $N \leftarrow$ TREECODE-TO-TREE-CP-NET(L, B)
  $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$
  $(L, B) \leftarrow$ NEXT-TREECODE-LEXICOGRAPHICALLY(L, B, n) ▷ see pseudocode below
until $(L, B) =$ first ▷ when the figurative odometer finally rolls over
return $\mathcal{N}$

NEXT-TREECODE-LEXICOGRAPHICALLY(L, B, n)

Input: L Prüfer code of length $n-1$
  B bit vector of length n
  n positive integer corresponding to the number of variables
Output: (L, B) Next treecode according to lexicographic order

$k \leftarrow n$ ▷ start with rightmost bit of B
while $k > 0$ **do**
  $B[k] \leftarrow B[k] + 1$ ▷ increment binary digit
  if $B[k] < 2$ **then** ▷ and carry if necessary
    break
  end if
  $B[k] \leftarrow 0$
  $k \leftarrow k - 1$ ▷ move on to the next most significant digit
end while
if B is the 0 vector **then** ▷ we reached all 1s and rolled over, so now get next L
  $k \leftarrow n - 1$
  while $k > 0$ **do**
    $L[k] \leftarrow L[k] + 1$ ▷ increment $(d+1)$-ary digit
    if $L[k] < n + 2$ **then** ▷ and carry if necessary
      break
    end if
    $L[k] \leftarrow 1$ ▷ Prüfer code elements start at 1 not 0
    $k \leftarrow k - 1$ ▷ move on to the next most significant digit
  end while
end if
return (L, B)

Figure 6.8: Algorithm: Generate All Tree-Shaped CP-nets

RANDOM-TREECODE(n)

Input: n Number of binary variables
Output: T Random Treecode in $\mathcal{T}_n$

1: initialize vectors L and B to respective lengths $n-1$ and n
2: **for** $j \leftarrow 1$ **to** $n-1$ **do** ▷ generate Prüfer code L
3: $L[k] \leftarrow$ element drawn uniformly randomly from $\{1, \ldots, n+1\}$
4: **end for**
5: **for** $j \leftarrow 1$ **to** n **do** ▷ generate bit vector B
6: $B[k] \leftarrow$ element drawn uniformly randomly from $\{0, 1\}$
7: **end for**
8: **return** (L, B)

Figure 6.9: Algorithm: Generate Random Tree-Shaped CP-net

The strategy is to order the treecodes lexicographically. In this lexicographic ordering, we first compare the Prüfer code L. If the two Prüfer codes are the same, then we compare the bit vectors B. In each case we first compare the first element of the vector. If it is the same, we move on to the second, and so on, until we reach the end of the vector. Generating all treecodes is then equivalent to generating all mixed-radix numerals, as discussed in Knuth [57, 7.2.1.1]. As each successive treecode is generated in this manner, we call TREECODE-TO-TREE-CP-NET (Figure 6.6) to obtain the corresponding CP-net and add it to the set to be returned. After generating all $(n+1)^{n-1}2^n$ treecodes, (L, B) will "roll back over" to the first treecode in the lexicographic ordering in Line 9, not unlike an odometer, and the algorithm will terminate, returning ${}^{T}\mathcal{N}_n$

We can also generate random treecodes (hence random tree-shaped CP-net models) uniformly with respect to ${}^{T}\mathcal{N}_n$, as shown in Figure 6.9. Note that since we can choose each element of L and B independently from their respective ranges $\{1, \ldots, n+1\}$ and $\{0, 1\}$, the resulting instances are provably uniform to the extent that we can generate integers uniformly at random.

6.3 Evaluating a Learned Model

A natural idea in discussing CP-net learning is to start with a CP-net N_T (a training model) and generate from it a set of outcome comparisons $\mathcal{E} = \left\{(o_1, o_1'), (o_2, o_2'), \ldots, (o_\ell, o_\ell')\right\}$ such that the training model entails every example, i.e., $N_T \models o_t \succ o_t'$ for all t, $1 \leq t \leq \ell$. The idea is then to attempt to learn a CP-net N_L that recovers the original, such that $N_L = N_T$, or, if the set of examples is too limited, at least to find a CP-net that entails all of the comparisons in $\mathcal{E}$.

This approach suffers from two significant problems. First, it assumes that all comparisons are *entailed* by a CP-net. Significantly, this conflicts with the assumption, common in economics, that preferences arise from an underlying utility function $u : \mathcal{O} \to \mathbb{R}$. In that case, every pair of outcomes is comparable provided their respective utilities can be computed: if $u(o) \geq u(o')$, then $o \succeq o'$. Recall that CP-nets, in contrast, induce a *partial* order on $\mathcal{O}$; thus, in general, some pairs of outcomes will be incomparable with respect to the CP-net, $N \models o \parallel o'$. Rather than trying to recover some presupposed ideal CP-net or to learn a model that *entails* every comparison, a more sophisticated approach, as argued by Lang and Mengin [65, 66], is to find a model that is *consistent* with every comparison. (See also the discussion in Section 3.5.)

Since this point often seems to be misunderstood, we offer an example. Suppose there are 3 variables with values 0 and 1; i.e., $\mathrm{Dom}(X_i) = \{0, 1\}$, $X_i \in \{X_1, X_2, X_3\}$. Further suppose that the user's preferences on the outcomes arise from the utility function

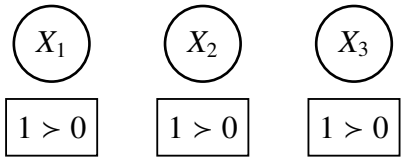
$$u(o) = \sum_{i=1}^{n} 2^{o[n-i-1]}. \tag{6.6}$$

We can then see that

$$000 \prec 001 \prec 010 \prec 011 \prec 100 \prec 101 \prec 110 \prec 111. \tag{6.7}$$

From this we could construct $\mathcal{E}$ consisting of all $\left|\mathcal{O}_3^2\right| = 28$ pairwise comparisons, such that $u(o) > u(o') \implies o \succ o'$. However, through exhaustive analysis, it can be shown that there

is no tree-shaped CP-net on $\mathcal{V}$, indeed, no *CP-net* on $\mathcal{V}$, that entails all 28 comparisons.[3] Nevertheless, consider that the CP-net N_0



while it does not entail (*agree* with) all of the comparisons, at least does not *disagree* with any of them; i.e., for all $(o, o') \in \mathcal{E}$, $N_0 \not\models o' \succ o$. In this sense, N_0 can be said to be *consistent* with $\mathcal{E}$, although no CP-net *entails* every example in $\mathcal{E}$.

Second, the notion of learning as recovering an original CP-net or entailing all examples fails to account for the possibility of *noise*. Suppose the preferences of a particular user *did* in fact arise from CP-net N_0, shown above. Suppose that each day, the user must choose between a pair of distinct outcomes, o and o', drawn from the set 000 to 111 inclusive. His observed choices from Monday through Thursday are

$$\mathcal{E}_{t=4} = \{(101, 100),\ (111, 101),\ (111, 011),\ (100, 000)\}. \tag{6.8}$$

Note that every choice is consistent with CP-net N_0. On Friday, he is again asked to choose between 100 and 101, the same two outcomes he was given on Monday. His preference has not changed; he still prefers $101 \succ 100$ on account of CPT(X_1). However, he picks 100 by mistake. The system dutifully adds (100, 101) to the data to obtain

$$\mathcal{E}_{t=5} = \{(101, 100),\ (111, 101),\ (111, 011),\ (100, 000),\ (100, 101)\}. \tag{6.9}$$

Observe that the resulting preference order is no longer consistent, since $101 \succ 100 \succ 101$. Thus, there is no possibility of finding a CP-net, or for that matter a deterministic utility function, that is consistent with every example in $\mathcal{E}$.

This suggests the idea of framing CP-net learning as an optimization problem: *Can we identify a model that maximizes agreement with example data?* Provided we have an

[3] Of course, one could define a variable for which the domain contained all 8 values, 000 to 111. The total ordering can then be entailed by a CP-net with a single node, but in that case no CP-net is needed.

objective function f to assess the fitness of a model N with respect to the example data, the learning problem can be stated as

$$N^* = \arg\max_N f(N, \mathcal{E}). \tag{6.10}$$

Many different notions of *fitness* (optimality) could be considered for the objective function. For this chapter, we use a simple one, as follows. Let *agree* denote the examples for which the CP-net model entails the *same* ordering as in $\mathcal{E}$; Conversely, let *disagree* denote those for which the model entails the *opposite* ordering,

$$\text{agree} = \{(o, o') : (o, o') \in \mathcal{E} \wedge N \models o \succ o'\} \tag{6.11}$$

$$\text{disagree} = \{(o, o') : (o, o') \in \mathcal{E} \wedge N \models o' \succ o\}, \tag{6.12}$$

computed by applying dominance testing to each element in $\mathcal{E}$. Given a non-empty set of example comparisons $\mathcal{E}$, we define the *fitness* (score) of a prospective model N as

$$f(N, \mathcal{E}) = \frac{|\text{agree}| - |\text{disagree}|}{|\mathcal{E}|}. \tag{6.13}$$

Note that if $(o, o') \in \mathcal{E}$ and $N \models o \parallel o'$, the pair (o, o') is not included in either *agree* or *disagree*, but nevertheless reduces the fitness score, which in general can range between -1 and 1 inclusive. To complete the examples given above, the fitness of N_0 on all pairs of the ordering given in Equation 6.7 is $(12 - 0)/28 \approx 0.4286$. The fitness scores of N_0 for $\mathcal{E}_{t=4}$ in Equation 6.8 and $\mathcal{E}_{t=5}$ in Equation 6.9 are $(4 - 0)/4 = 1$ and $(4 - 1)/5 = 0.6$, respectively.

6.4 Learning via Local Search

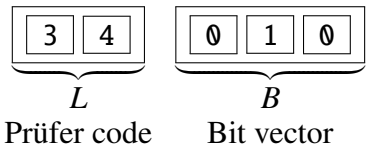
A chief advantage of the encoding described in Section 6.2 is that it provides a straightforward way to define neighbors for local search. A pair of tree-shaped CP-nets are *neighbors* if *their corresponding treecodes differ in just one element*. Formally,

$$\text{Neigh}(N) = \left\{ N' : \text{HD}\left(\text{TC}(N), \text{TC}(N')\right) = 1,\ N' \in {}^{T}\mathcal{N} \right\}. \tag{6.14}$$

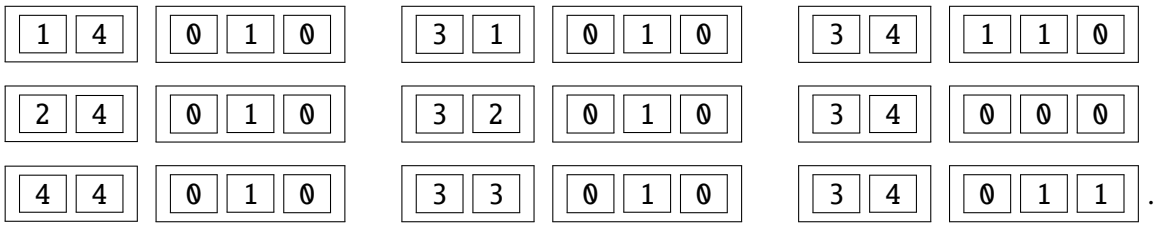
It is also helpful to define the neighbors of a treecode T:

$$\text{Neigh}(T) = \{\, T' : \text{HD}\,(T,\, T') = 1,\ T' \in \mathcal{T}\,\}. \tag{6.15}$$

For example, given treecode



the neighbors are



Theorem 41 (Number of Neighbors)**.** *Let n be any positive integer and N be any tree-shaped CP-net on n binary variables. Then the number of neighbors of N is*

$$|\text{Neigh}(N)| = n^2. \tag{6.16}$$

Proof. Consider that to obtain $\text{TC}(N') = T' = (L', B')$ from $\text{TC}(N) = T = (L, B)$ we can change the value of just one element in (L, B). If we modify a value in Prüfer code L, there are $n-1$ elements to choose from and for each there are n new values available, since $L'[j] \in \{1, \ldots, n+1\}$ and $L'[j] \neq L[j]$; i.e., the new value must differ from the current value. Similarly, if we modify B, there are n elements to choose from, but only one new value, since $B'[k] \in \{0, 1\}$ and $B'[k] \neq B[k]$. Thus, the number of treecodes $T' = (L', B')$, such that $\text{HD}(T, T') = 1$, is $(n-1)n+n = n^2$. Finally, since TC is a bijection, $|\text{Neigh}(N)| = n^2$. ❑

The algorithm Walk-CP-net, presented in Figure 6.10, combines these ideas to enable local search for learning tree-shaped CP-nets. Inspired by the GSAT [95] and Walk-SAT [96] algorithms for solving Boolean satisfiability problems, Walk-CP-net attempts to

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

Input:	$\mathcal{E}$	Example data
	n	Number of variables
	π	Probability of taking a random walk
	max-strikes	Counter to help detect plateaux
	max-restarts	Limit of random restarts before algorithm terminates
Output:	N^*	Fittest tree-shaped CP-net encountered

1: $N^* \leftarrow$ null model $N^\perp$ with fitness score $f(N^\perp, \cdot) = -\infty$
2: **for** restarts $\leftarrow 1$ **to** max-restarts **do**
3: $N \leftarrow$ TREECODE-TO-TREE-CP-NET(RANDOM-TREECODE(n)) ▷ Figures 6.6, 6.9
4: strikes $\leftarrow 0$
5: **repeat**
6: **if** $f(N, \mathcal{E}) > f(N^*, \mathcal{E})$ **then**
7: $N^* \leftarrow N$ ▷ The new model is also the new best model
8: strikes $\leftarrow 0$ ▷ New height reached, so also reset counter
9: **if** $f(N^*, \mathcal{E}) = 1$ **then**
10: **return** N^* ▷ N^* entails all examples, hence also globally best
11: **end if**
12: **end if**
13: **if** random value in $[0, 1] \leq \pi$ **then**
14: $N \leftarrow$ random element of Neigh(N) ▷ Random walk
15: **else**
16: previous-fitness $\leftarrow f(N, \mathcal{E})$
17: $N \leftarrow \arg\max_{N'} f(N', \mathcal{E})$ s.t. $N' \in$ Neigh(N)) ▷ Hill climbing
18: **if** $f(N, \mathcal{E}) \leq$ previous-fitness **then** ▷ Possible plateau
19: strikes $\leftarrow$ strikes + 1
20: **end if**
21: **end if**
22: **until** strikes = max-strikes ▷ Random restart to get off plateau
23: **end for**
24: **return** N^* ▷ Give up and return the best model encountered

Figure 6.10: Algorithm: Walk-CP-net

find a balance between random and greedy behavior. The algorithm starts by generating a random model. By default it is the best model yet encountered, so we distinguish it as N^*. With probability π, a user-selected parameter in the range $[0, 1]$, the algorithm then either randomly walks (Line 14) or attempts hill-climbing (Line 17). In the case of a random walk, the algorithm replaces N with a randomly selected neighbor. In the case of greedy behavior, the algorithm performs dominance testing (DT) on $\mathcal{E}$ to evaluate all n^2 neighbors. However, while DT is NP-hard for CP-nets in general, it is easy for tree-shaped CP-nets on binary variables, the class of models that we consider here.

If hill climbing is possible, WALK-CP-NET chooses a neighbor with the highest score. Note that if N is locally optimal, the fitness level may stay the same or even decrease. In that case, it is possible that the search has reached a plateau. To avoid becoming stranded there, the algorithm increments a *strikes* counter. However, if fitness is strictly better, and the new model turns out to be the best yet encountered, the counter is reset to 0 in Line 8.

When the counter eventually reaches *max-strikes*, a user-selected parameter, the algorithm performs a *random restart*, choosing a (potentially distant) model uniformly at random. The number of random restarts is also controlled by a user-selected parameter, *max-restarts*. When this limit is also eventually reached, the algorithm terminates, returning N^*, the best model encountered. The algorithm can also terminate early, in Line 10, if it encounters a model that entails all examples in $\mathcal{E}$. In that case, N^* is provably a global optimum, so we halt the search.

6.5 Experiments

As discussed in Section 3.6, real-world data relevant to CP-nets are in short supply, and the datasets that are occasionally used for CP-net learning are problematic. Thus, as with most research involving CP-nets, we used synthetic data to evaluate our algorithm. Figure 6.11 describes how the choice data $\mathcal{E}$ are generated. The algorithm first generates an archetypal tree-shaped CP-net N_T uniformly at random. DT is then employed to identify pairs in $\mathcal{O}_n^2$

GENERATE-CHOICE-DATA(n, ℓ, p, q)

Input: n Number of binary variables
ℓ Number of comparisons to return; i.e., $|\mathcal{E}| \approx \ell$
p Probability of answering $o \succ o'$ or $o' \succ o$ randomly when $o \parallel o'$
q Probability of answering incorrectly; i.e., $o' \succ o$ when actually $o \succ o'$
Output: $\mathcal{E}$ Example data with ℓ comparisons specifying $o \succ o'$ for all $(o, o') \in \mathcal{E}$
N_T Tree-shaped CP-net used to generate $\mathcal{E}$

$N_T \leftarrow$ TREECODE-TO-TREE-CP-NET(RANDOM-TREECODE(n))
for all $o \in \mathcal{O}_n$ or subset of $\mathcal{O}_n$ of size $\left\lfloor \sqrt{\ell} \right\rfloor$ **do**
  for all $o' \in \mathcal{O}_n$ or subset of size $\left\lceil \sqrt{\ell} \right\rceil$ such that $o \neq o'$ **do**
    if $N_T \models o \succ o'$ **then** ▷ model specifies $o \succ o'$
      $u \leftarrow o$
      $v \leftarrow o'$
    else if $N_T \models o' \succ o$ **then** ▷ model specifies $o' \succ o$
      $u \leftarrow o'$
      $v \leftarrow o$
    else ▷ model specifies $o \parallel o'$
      $x \leftarrow$ uniform random number in [0, 1]
      if $x < p/2$ **then** ▷ With probability p, we flip a coin,
        $u \leftarrow o$ ▷ to choose between $o \succ o'$
        $v \leftarrow o'$
      else if $x < p$ **then**
        $u \leftarrow o'$ ▷ and $o' \succ o$.
        $v \leftarrow o$
      else
        continue ▷ The rest of the time, we do not give any answer if $o \parallel o'$.
      end if
    end if
    $y \leftarrow$ uniform random number in [0, 1]
    if $y < q$ **then** ▷ With probability q
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(v, u)\}$ ▷ answer differently than expected.
    else
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(u, v)\}$ ▷ The rest of the time, answer as intended.
    end if
  end for
end for
$\mathcal{E} \leftarrow$ random sample of size ℓ selected uniformly from $\mathcal{E}$
return $(\mathcal{E}, N)$

Figure 6.11: Algorithm: Generate Choice Data

for which N_T entails a preference. When feasible, we iterate over all pairs of outcomes. When $|\mathcal{O}_n^2|$ is too large, we specify some number ℓ of outcome pairs to be selected from $\mathcal{O}_n^2$ at random.

DT is applied to each resulting problem instance (N_T, o, o') to determine an ordering, if one exists. If the CP-net does *not* entail an ordering, i.e., if $o \parallel o'$, then by default the algorithm does not include the pair in $\mathcal{E}$. However, this behavior can be customized by setting a user-specified parameter p. With probability $p \in [0, 1]$, when $N_T \models o \parallel o'$, the algorithm will *guess* $o \succ o'$ or $o' \succ o$ at random. (The rest of the time, i.e., with probability $1 - p$, the algorithm does not report a preference for pairs that are incomparable with respect to N_T.) This behavior is analogous to that of a subject who sometimes has to choose between two outcomes even though he has little information to indicate which is better. Note, however, that choosing in this randomized manner introduces noise into the choice data, since if $p > 0$, the resulting transitive closure of $\mathcal{E}$ is likely to contain a cycle. Noise can also be introduced via a separate parameter, q. With probability $q \in [0, 1]$, the algorithm will *misreport* the preference, analogous to a subject who on occasion makes an incorrect choice. Again, note that even small, non-zero values of q are likely to cause $\mathcal{E}$ to be inconsistent. While it is possible to specify non-zero values of p and q simultaneously, we did not do this for our experiments.

For the first experiment, we generated an original CP-net N_T on $n = 4$ variables. The choice data $\mathcal{E}$ consisted of all pairs (out of 120 possible) for which the original CP-net entailed a preference. We generated 100 such datasets and ran the Walk-CP-net learning algorithm on the data multiple times to compare the performance of each learned model to that of the original model N_T. Since no noise was introduced into the example data, the original model always had a fitness score of 1.0. We ran the learning algorithm 100 times for each of the 100 data sets, varying the *max-restarts* parameter. The bar chart in Figure 6.12 shows the mean fitness score achieved for 10, 50, 100, and 500 restarts, respectively.

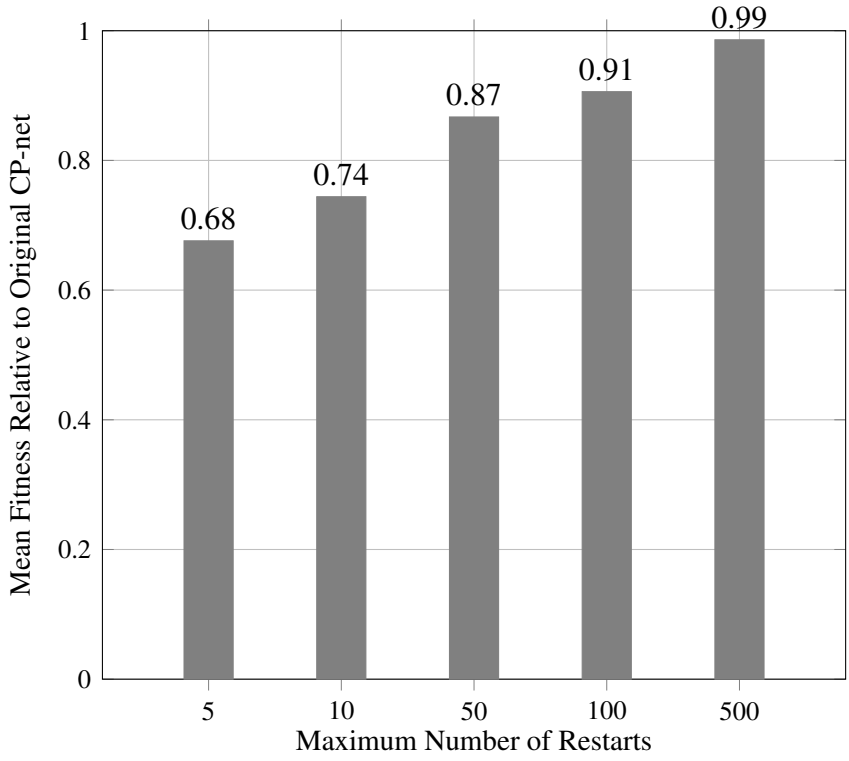


Figure 6.12: Walk-CP-net Experiment 1 (No Noise)

For the second experiment, we used parameter p to introduce varying degrees of noise. Note that once noise is introduced, it is possible—in fact, likely—that some tree-shaped CP-net N^* *other than the original CP-net* N_T offers a better fit to $\mathcal{E}$. To provide an *oracle* for the learning problem, we thus used the algorithm FIND-BEST-TREE-SHAPED-CP-NET shown in Figure 6.13. The oracle iterates over all $N \in {}^{T}\mathcal{N}_n$ tree-shaped CP-nets to find a model that has maximum fitness with respect to $\mathcal{E}$. The learned model N_L is then compared to the model obtained from the oracle rather than to the original or the one selected by the oracle. Note that when too much noise is introduced, it is often the case that no model performs well, including the original. We generated 100 datasets with noise level $p = 0.02$ and ran the learning algorithm 100 times on each, again varying the *max-restarts* parameter. The bar chart in Figure 6.14 shows the mean fitness score achieved *relative to the oracle* for 10, 50, 100, and 500 restarts, respectively.

FIND-BEST-TREE-SHAPED-CP-NET($\mathcal{E}$, n)

Input: $\mathcal{E}$ Example data
n Number of binary variables
Output: N Optimal tree-shaped CP-net with respect to $\mathcal{E}$

1: $\mathcal{N} \leftarrow$ ALL-TREE-SHAPED-CP-NETS(n)
2: BestScore $\leftarrow -\infty$
3: **for all** $N \in \mathcal{N}$ **do**
4: agree = $\{(o, o') : (o, o') \in \mathcal{E} \wedge N \models o \succ o'\}$
5: disagree = $\{(o, o') : (o, o') \in \mathcal{E} \wedge N \models o' \succ o\}$
6: score = $\dfrac{|\text{agree}| - |\text{disagree}|}{|\mathcal{E}|}$
7: **if** score > BestScore **then**
8: BestScore $\leftarrow$ score
9: BestModel $\leftarrow N$
10: **end if**
11: **end for**
12: **return** BestModel

Figure 6.13: Algorithm: Find a Best Tree-Shaped CP-net Globally

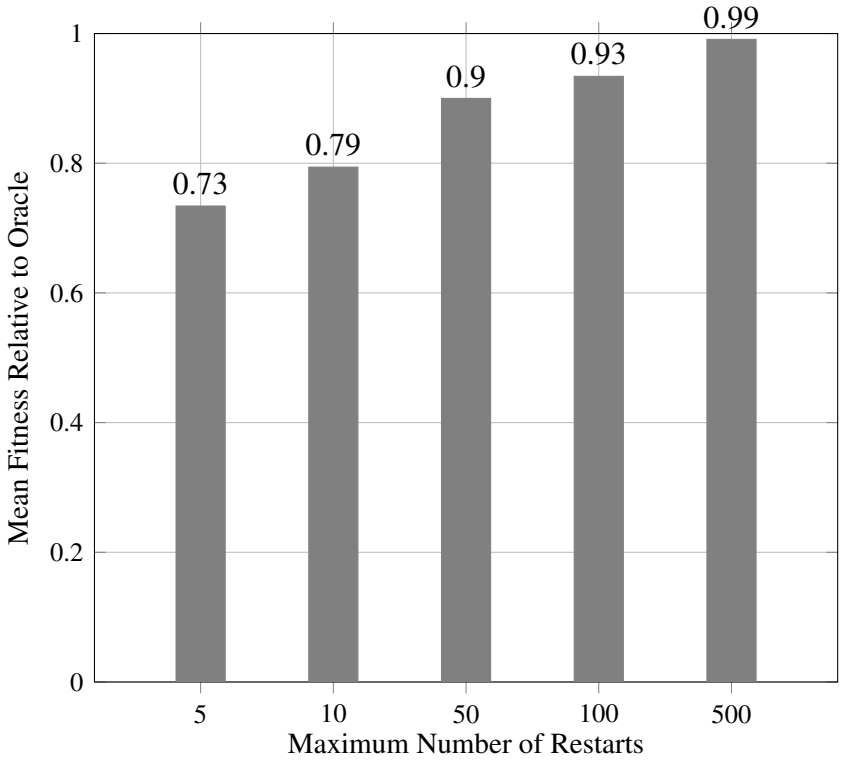


Figure 6.14: Walk-CP-net Experiment 2 (Noise Level $p = 0.02$)

The third experiment was similar to the second, except that we set noise parameter q, rather than p. That is, rather than randomly "guessing" an ordering for some incomparable pairs of outcomes, the answer for an *entailed* pair of outcomes is *reversed* with probability q. The bar chart in Figure 6.15 shows the mean fitness score achieved *relative to the oracle* for $q = 0.03$. Keep in mind that the comparison metric is the fitness score relative to the oracle for each generated data set $\mathcal{E}$, and that when example data are noisy, even the oracle usually achieves a fitness score less than 1.0.

Finally, we generated larger problem instances (Table 6.2). Since consulting the oracle was infeasible for these, we did not include noise. For this experiment we generated tree-shaped CP-nets with 10, 15, and 20 nodes, with 100 CP-nets in each set. For each CP-net, we generated a choice set $\mathcal{E}$ consisting of 45 comparisons entailed by the original CP-net. We then repeatedly ran WALK-CP-NET for each of the 100 datasets, increasing the maximum number of restarts by about 50% on each iteration. We continued this until we reached a

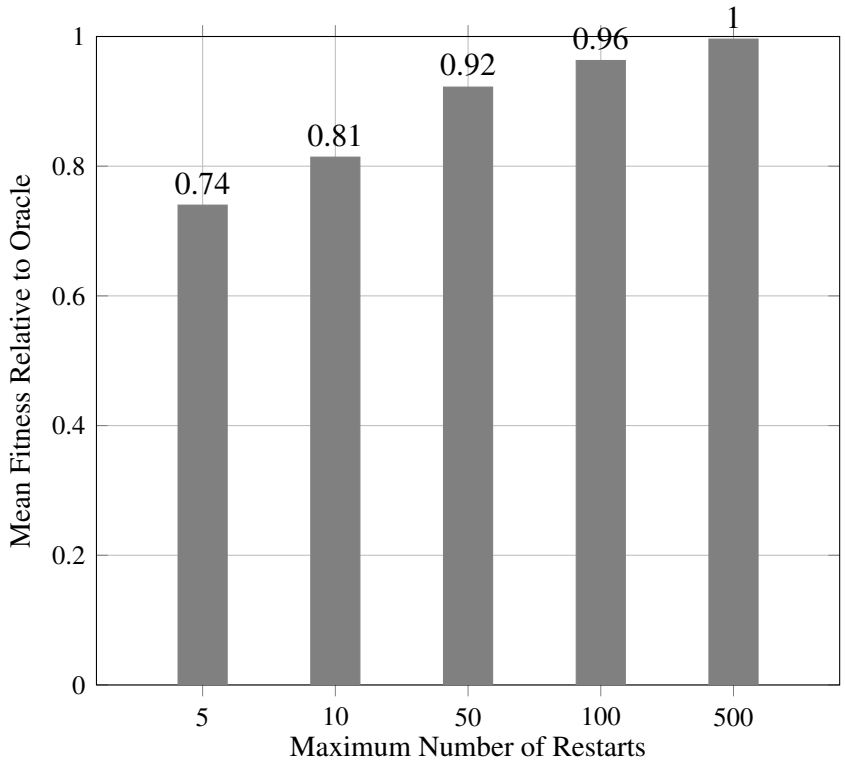


Figure 6.15: Walk-CP-net Experiment 3 (Noise Level $q = 0.03$)

Table 6.2: Walk-CP-net Experiment 4 (No Noise)

Variables	Restarts	Mean Fitness	Mean DT Calls[†]
10	10	0.86	4
10	15	0.89	8
10	22	0.92	14
10	33	0.94	25
10	49	0.95	43
15	10	0.77	9
15	15	0.80	16
15	22	0.83	29
15	33	0.85	52
15	49	0.87	89
15	73	0.88	149
15	109	0.90	250
15	163	0.92	410
15	244	0.93	670
15	366	0.94	1090
15	549	0.95	1762
20	10	0.72	14
20	15	0.75	27
20	22	0.77	49
20	33	0.80	88
20	49	0.82	151
20	73	0.83	254
20	109	0.85	425
20	163	0.86	701
20	244	0.88	1150
20	366	0.90	1873
20	549	0.90	3031
20	823	0.91	4882
20	1234	0.91	7809
20	1851	0.93	12459
20	2276	0.93	19819

[†]To the nearest thousand

fitness threshold of 0.95 or (in the case of $n = 20$ nodes) reached a preset time limit. We also recorded the number of calls to the DT solver, in this case the DT-Tree algorithm [16].

6.6 Conclusion

In this chapter we have considered the problem of learning CP-nets from noisy, possibly inconsistent choice data using local search. We proposed a novel and elegant encoding for tree-shaped CP-nets and showed how to use this encoding to define a neighborhood for local search. Tree-shaped CP-net models are of interest because, once such models are learned, dominance testing can be conducted with them in polynomial time, in contrast to CP-nets in general, for which dominance testing is known to be NP-hard. We described a local search algorithm inspired by Walk-SAT that allows for a balance between greedy improvement and randomized search. The experiments suggest that the approach is resilient in the presence of noise. Our work represents first steps toward a difficult and important problem for which no satisfactory solution yet exists. In future work, we plan to extend our encoding to tree-shaped CP-nets with multivalued variables and to polytrees, and to explore whether other encodings, such as the dagcodes and cpt-codes described in Chapter 4, could also be used to define neighbors for local search.

Chapter 7 Conclusion

In this dissertation I have addressed three of the major problems involving conditional preference networks: *reasoning* with CP-nets to determine the ordering of an arbitrary pair of outcomes, *learning* CP-nets from pairwise comparison data, and *generating* CP-nets uniformly at random. In Chapters 4 and 6, I introduced two novel encodings for CP-nets, one that generalizes to acyclic models with constraints on indegree and multivalued domains and a separate encoding that is specific to tree-shaped CP-nets with binary domains and complete tables. The encodings enabled us to study the expected flipping lengths of dominance testing instances in Chapter 5 and also to define neighbors for learning CP-nets via local search in Chapter 6. A better understanding of the dominance and reasoning problems, in turn, enables us to develop more effective heuristics and opens the door to new areas of research involving CP-nets, such as applying portfolio techniques to learning and reasoning with CP-nets and analyzing the average time complexity of algorithms. I hope to give these problems further attention in future research. In future research, I also hope to develop methods of learning CP-nets that generalize to more complex models. We observed that databases of CP-nets from human subjects are not yet available. This hinders research for the computational preferences and social choice communities as a whole. The problem is remedied only through learning algorithms that scale to real-world problems. Finally, I hope to integrate CP-nets into real-world applications, in particular those involving assistive technologies and intelligent environments.

Bibliography

[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 *Proceedings of the Fourth International Conference on Algorithmic Decision Theory (ADT)*, 2015.

[4] Thomas E Allen. I prefer to eat.... In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[5] Thomas E Allen, Judy Goldsmith, Hayden E Justice, Nicholas Mattei, and Kayla Raines. Generating cp-nets uniformly at random. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[6] Fani Athienitou and Yannis Dimopoulos. Learning CP-networks: a preliminary investigation. *Procedings 3rd Multidisciplinary Work. on Advances in Preference Handling (PREF'07)*, 2007.

[7] Reyhan Aydoğan, Tim Baarslag, Koen V Hindriks, Catholijn M Jonker, and Pınar Yolum. Heuristic-based approaches for CP-nets in negotiation. In *Complex Automated Negotiations: Theories, Models, and Software Competitions*, pages 113–123. Springer, 2013.

[8] Fahiem Bacchus and Adam Grove. Graphical models for preference and utility. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 3–10. Morgan Kaufmann Publishers Inc., 1995.

[9] José L Balcázar, Antoni Lozano, and Jacobo Torán. The complexity of algorithmic problems on succinct instances. In *Computer Science*, pages 351–377. Springer, 1992.

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

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

[12] Damien Bigot, Jérôme Mengin, and Bruno Zanuttini. Learning probabilistic CP-nets from observations of optimal items. In *STAIRS*, pages 81–90, 2014.

[13] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM (JACM)*, 44(2):201–236, 1997.

[14] Stefano Bistarelli, Ugo Montanari, Francesca Rossi, Thomas Schiex, Gérard Verfaillie, and Hélene Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3):199–240, 1999.

[15] Stefano Bistarelli, Fabio Fioravanti, and Pamela Peretti. Using CP-nets as a guide for countermeasure selection. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, pages 300–304. ACM, 2007.

[16] C. Boutilier, R.I. Brafman, C. Domshlak, H.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.

[17] Craig Boutilier, Ronen I. Brafman, Holger H. Hoos, and David Poole. Reasoning with conditional ceteris paribus preference statements. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 71–80, 1999.

[18] Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. Preference-based constrained optimization with CP-nets. *Computational Intelligence*, 20(2):137–157, 2004.

[19] Ronen I. Brafman and Carmel Domshlak. Introducing variable importance tradeoffs into CP-nets. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI'02, pages 69–76, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-897-4.

[20] Ronen I Brafman, Enrico Pilotto, Francesca Rossi, Domenico Salvagnin, Kristen Brent Venable, and Toby Walsh. The next best solution. In *AAAI*, 2011.

[21] Karl Bringmann and Konstantinos Panagiotou. Efficient sampling methods for discrete distributions. In *Automata, Languages, and Programming*, pages 133–144. Springer, 2012.

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

[23] Yann Chevaleyre, Frédéric Koriche, Jérôme Lang, Jérôme Mengin, and Bruno Zanuttini. Learning ordinal preferences on multiattribute domains: The case of CP-nets. In *Preference Learning*, pages 273–296. Springer, 2011.

[24] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. ISBN 0-070-13151-1.

[25] C. Cornelio, J. Goldsmith, N. Mattei, F. Rossi, and K. B. Venable. Updates and uncertainty in CP-nets. In *26th Australasian Joint Conference on Artificial Intelligence*, 2013. To Appear.

[26] Cristina Cornelio. Dynamic and probabilistic CP-nets. Master's thesis, University of Padua, 2012.

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

[28] Sanjoy Dasgupta. Learning polytrees. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 134–141. Morgan Kaufmann Publishers Inc., 1999.

[29] Yannis Dimopoulos, Loizos Michael, and Fani Athienitou. Ceteris paribus preference elicitation with predictive guarantees. In *Proceedings of the 21st International Joint conference on Artificial intelligence (IJCAI-09)*, IJCAI'09, pages 1890–1895, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[30] C. Domshlak and R.I. Brafman. CP-nets: Reasoning and consistency testing. In *Proceedings 8th International Conference on Principles and Knowledge Representation and Reasoning (KRR)*, 2002.

[31] Carmel Domshlak. On recursively directed hypercubes. *The Electronic Journal of Combinatorics*, 9(1):R23, 2002.

[32] Carmel Domshlak, Steve Prestwich, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Hard and soft constraints for reasoning about qualitative conditional preferences. *Journal of Heuristics*, 12(4-5):263–285, 2006.

[33] Alan Eckhardt and Peter Vojtáš. How to learn fuzzy user preferences with variable objectives. In *Proceedings of IFSA/EUSFLAT*, pages 938–943, 2009.

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

[35] Peter Fishburn. Preference structures and their numerical representations. *Theoretical Computer Science*, 217(2):359–383, 1999.

[36] Peter C. Fishburn. Intransitive indifference in preference theory: A survey. *Operations Research*, 18(2):pp. 207–228, 1970. ISSN 0030364X. URL `http://www.jstor.org/stable/168680`.

[37] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, June 1962. ISSN 0001-0782. doi: 10.1145/367766.368168. URL `http://doi.acm.org/10.1145/367766.368168`.

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

[39] Gerd Gigerenzer. *Gut Feelings: The Intelligence of the Unconscious*. Penguin, 2007.

[40] Malcolm Gladwell. *Blink: The Power of Thinking Without Thinking*. Back Bay Books, 2007.

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

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

[43] Judy Goldsmith, Jérôme Lang, Miroslaw Truszczyński, and Nic Wilson. The computational complexity of dominance and consistency in CP-nets. In *Procedings IJCAI*, 2005.

[44] C. Gonzales and P. Perny. Gai networks for utility elicitation. In *Procedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR-2004)*, pages 224–233. AAAI Press, June 2004.

[45] Gary Gordon and Elizabeth McMahon. A greedoid polynomial which distinguishes rooted arborescences. *Proceedings of the American Mathematical Society*, 107(2): 287–298, 1989.

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

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

[48] Joshua T. Guerin. *Graphical Models for Decision Support in Academic Advising*. PhD thesis, University of Kentucky, 2012.

[49] László Gulyás, Gábor Horváth, Tamás Cséri, and George Kampis. An estimation of the shortest and largest average path length in graphs of given density. *arXiv preprint arXiv:1101.2549*, 2011.

[50] Michael A Harrison. *Introduction to Switching and Automata Theory*, volume 65. McGraw-Hill, 1965.

[51] Sze-Tsen Hu. *Mathematical Theory of Switching Circuits and Automata*. University of California Press, 1968.

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

[53] Toshihiro Kamishima and Shotaro Akaho. Nantonac collaborative filtering: A model-based approach. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 273–276, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-906-0. doi: 10.1145/1864708.1864765. URL `http://doi.acm.org/10.1145/1864708.1864765`.

[54] Henry Kautz and Bart Selman. Planning as satisfiability. In *ECAI-92*, pages 359–363. Wiley, 1992.

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

[56] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998. ISBN 0-201-89685-0.

[57] Donald E. Knuth. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms Part 1*. Addison-Wesley, 2011. ISBN 0-201-03804-8.

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

[59] Frédéric Koriche and Bruno Zanuttini. Learning conditional preference networks with queries. In *IJCAI-09*, pages 1930–1935, 2009.

[60] Donald L. Kreher and D.R. Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, 1999. ISBN 978-0-849-33988-2.

[61] Martin Kronegger, Martin Lackner, Andreas Pfandler, and Reinhard Pichler. A parameterized complexity analysis of generalized CP-nets. In *Proceedings AAAI*, pages 1091–1097, 2014.

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

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

[64] Jérôme Lang and Jérôme Mengin. Learning preference relations over combinatorial domains. *Procedings NMR*, pages 207–214, 2008.

[65] Jéròme Lang and Jéròme Mengin. Learning preference relations over combinatorial domains. In *NMR-08*, 2008.

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

[67] Derrick H Lehmer. Teaching combinatorial tricks to a computer. In *Combinatorial Analysis*, volume 10 of *Procedings of Symposia in Applied Mathematics*, pages 179–193, 1960.

[68] Minyi Li, Quoc Bao Vo, and Ryszard Kowalczyk. Efficient heuristic approach to dominance testing in CP-nets. In *Proceedings AAMAS*, pages 353–360, 2011.

[69] Juntao Liu, Zhijun Yao, Yi Xiong, Wenyu Liu, and Caihua Wu. Learning conditional preference network from noisy samples using hypothesis testing. *Knowledge-Based Systems*, 40:7–16, 2013.

[70] Juntao Liu, Yi Xiong, Caihua Wu, Zhijun Yao, and Wenyu Liu. Learning conditional preference networks from inconsistent examples. *Knowledge and Data Engineering, IEEE Transactions on*, 26(2):376–390, 2014.

[71] Xudong Liu and Miroslaw Truszczynski. Reasoning with preference trees over combinatorial domains. In *Algorithmic Decision Theory*, pages 19–34. Springer, 2015.

[72] Thomas Lukasiewicz and Enrico Malizia. On the complexity of *m*CP-nets. In Dale Schuurmans and Michael Wellman, editors, *Proceedings of the 30th National Conference on Artificial Intelligence AAAI 2016 Phoenix, Arizona USA February 1217 2016*. AAAI Press, February 2016.

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

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

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

[76] Leigh McAlister and Edgar Pessemier. Variety seeking behavior: An interdisciplinary review. *Journal of Consumer Research*, pages 311–322, 1982.

[77] Richard E Nisbett and Timothy D Wilson. Telling more than we can know: Verbal reports on mental processes. *Psychological review*, 84(3):231, 1977.

[78] Loran F. Nordgren and A.P. Dijksterhuis. The devil is in the deliberation: thinking too much reduces preference consistency. *Journal of Consumer Research*, 36(1): 39–46, 2009.

[79] Luke O'Connor. Nondegenerate functions and permutations. *Discrete Applied Mathematics*, 73(1):41–57, 1997. ISSN 0166-218X. doi: http://dx.doi.org/10.1016/0166-218X(95)00117-A. URL `http://www.sciencedirect.com/science/article/pii/0166218X9500117A`.

[80] Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Incompleteness and incomparability in preference aggregation. In *IJCAI*, volume 7, pages 1464–1469, 2007.

[81] Anna Popova, Michel Regenwetter, and Nicholas Mattei. A behavioral perspective on social choice. *Annals of Mathematics and Artificial Intelligence*, 68(1–3):1–26, 2013.

[82] George Rebane and Judea Pearl. The recovery of causal poly-trees from statistical data. In *Proceedings UAI*, 1987.

[83] M. Regenwetter, B. Grofman, A.A.J. Marley, and I. Tsetlin. *Behavioral Social Choice*. Cambridge University Press, Cambridge, UK, 2006.

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

[85] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[86] J. Rintanen and C. O. Gretton. Computing upper bounds on lengths of transition sequences. In *Proceedings IJCAI*, pages 2365–2372, 2013.

[87] Robert W Robinson. Counting labeled acyclic digraphs. In F. Harary, editor, *New directions in the theory of graphs: proceedings*, pages 239–273. Academic Press, 1973.

[88] K. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Education, 7th edition, 2012. ISBN 978-0-073-38309-5.

[89] F. Rossi, K. B. Venable, and T. Walsh. *m*CP nets: Representing and reasoning with preferences of multiple agents. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pages 729–734. AAAI Press, 2004. ISBN 0-262-51183-5. URL `http://dl.acm.org/citation.cfm?id=1597148.1597265`.

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

[91] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. ISBN 0136042597, 9780136042594.

[92] G. R. Santhanam, S. Basu, and V. Honavar. Dominance testing via model checking. In *AAAI*, 2010.

[93] Ganesh Ram Santhanam, Samik Basu, and Vasant Honavar. Crisner: A practically efficient reasoner for qualitative preferences. *arXiv preprint arXiv:1507.08559*, 2015.

[94] Ganesh Ram Santhanam, Samik Basu, and Vasant Honavar. Representing and reasoning with qualitative preferences: Tools and applications. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(1):1–154, 2016.

[95] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *AAAI*, volume 92, pages 440–446, 1992.

[96] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532. AMS, 1995.

[97] Eldar Shafir, Itamar Simonson, and Amos Tversky. Reason-based choice. *Cognition*, 49(1):11–36, 1993.

[98] N J A Sloane. The On-Line Encyclopedia of Integer Sequences. http://oeis.org, March 2016. Accessed: 2016-03-20.

[99] 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`.

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

[101] Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, January 1962. ISSN 0004-5411. doi: 10.1145/321105.321107. URL `http://doi.acm.org/10.1145/321105.321107`.

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

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

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

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

Vita

Thomas E. Allen received his Bachelor of Science in Information and Computer Science from the Georgia Institute of Technology with Highest Honors in 1991. In 1994 he received his Master of Divinity from the Southern Baptist Theological Seminary and subsequently served as pastor to congregations in Georgia and Kentucky. In 2011, he began graduate studies in computer science at the University of Kentucky. He has served as a Research Assistant to Dr. Judy Goldsmith and as a Teaching Assistant in the Computer Science Department. He was awarded the Thaddeus B. Curtz Memorial Scholarship Award in 2012 and was a recipient of a Graduate School Academic Year Fellowship in 2013–2014 and 2015–2016 and the Verizon Fellowship in Fall 2014.

Professional Publications

Thomas E. Allen, Judy Goldsmith, Hayden Elizabeth Justice, Nicholas Mattei, Kayla Raines. Generating CP-nets Uniformly at Random. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (AAAI-16), 2016.

Thomas E. Allen, Muye Chen, Judy Goldsmith, Nicholas Mattei, Anna Popova, Michel Regenwetter, Francesca Rossi, Christopher Zwilling. Beyond Theory and Data in Preference Modeling: Bringing Humans into the Loop. In *Proceedings of the Fourth International Conference on Algorithmic Decision Theory* (ADT 2015), Lecture Notes in Artificial Intelligence. Springer, 2015.

Thomas E. Allen. CP-nets with Indifference. (Invited paper.) In *Proceedings of the 51st Annual Allerton Conference on Communication, Control, and Computing*, IEEE, 2013.

Joshua T. Guerin, Thomas E. Allen, and Judy Goldsmith. Learning CP-net Preferences Online from User Queries. In *Proceedings of the Third International Conference on Algorithmic Decision Theory* (ADT 2013), Lecture Notes in Artificial Intelligence. Springer, 2013.

Thomas Eugene Allen