More Than the Sum of Its Parts

Compact Preference Representation Over Combinatorial Domains

Joel Uckelman

More Than the Sum of Its Parts

Compact Preference Representation Over Combinatorial Domains

ILLC Dissertation Series DS-2009-12

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 904
1098 XH Amsterdam
phone: +31-20-525 6051
fax: +31-20-525 5206
e-mail: illc@uva.nl
homepage: http://www.illc.uva.nl/

More Than the Sum of Its Parts

Compact Preference Representation Over Combinatorial Domains

Academisch Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof.dr. D.C. van den Boom
ten overstaan van een door het college voor
promoties ingestelde commissie, in het openbaar
te verdedigen in de Agnietenkapel
op vrijdag 11 december 2009, te 10.00 uur

door

Joel David Uckelman

geboren te Carroll, Iowa, Verenigde Staten van Amerika

Promotiecommissie:

Promotor: Prof. dr. K.R. Apt
Co-promotor: Dr. U. Endriss

Overige leden:
Prof. dr. J.F.A.K. van Benthem
Prof. dr. G. Dari-Mattiacci
Prof. dr. J. Lang
Prof. dr. B. Löwe
Prof. dr. F. Rossi
Prof. dr. M.J. Wooldridge

Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Universiteit van Amsterdam

The work comprising this dissertation was supported by a GLoRiClass fellowship funded by the European Commission (Early Stage Research Training Mono-Host Fellowship MEST-CT-2005-020841).

Cover design by Matt Kuhns.
Printed and bound by Ipskamp Drukkers.

ISBN: 978–90–5776–203–1

For Sara

Contents

Acknowledgments

The first person I wish to thank is someone I can't identify by name: whomever wrote the course description for my high school's freshman speech class. This course description was so frightening for someone who disliked public speaking as much as I did that I decided to fulfill the speech requirement the only other way possible—by joining the debate team. Somehow it escaped me that this was not a clever way to evade the requirement; instead of giving five speeches in a one-quarter course, I spent every fall and winter Saturday giving speeches... and kept doing it for three more years, long after I'd fulfilled the requirement. Our coach, Bob Galligan, (who possibly wrote the scary course description) introduced me to philosophy, and debate was what convinced me to major in philosophy at Iowa State University.

In my second semester at Iowa State, I had the good fortune of taking the introductory logic course from Bill Robinson. We did natural deduction for propositional and quantifier logic, with one lecture at the end on modal logic. I was hooked, and Prof. Robinson graciously offered to do an independent study course with me on modal logic the next semester. Later, he (and my friend, Josh Kortbein) encouraged me take the two-semester mathematical logic course offered by the math department, which put me on the path to becoming a logician. Roger Maddux's math logic course opened my eyes to how much I didn't know. I'm grateful that Prof. Maddux took into account that I was a non-mathematician, since with my background at the time, the work was quite hard for me. It was in his course that I first understood how to prove anything, a skill without which I'd have no dissertation. (Prof. Maddux also started me, unbeknownst to him, on my habit of hoarding scratch paper on which to do proofs.)

My advisor at the University of Wisconsin–Madison, Mike Byrd, deserves special thanks, on four counts: First, he showed me the best example I have ever seen of how to teach. Mike drew in even the students who didn't want to be there, not by gimmicks or by watering down the material, but by sheer force of enthusiasm. As we walked to the first lecture of his introductory logic course

(for which I was several times his TA), he remarked to me in his matter-of-fact way, "This is the seventieth time I've taught this course", and smiled. Mike set an inspiring example. Second, Mike didn't just tell me when I'd made a mistake, he showed me exactly where and gave me a counterexample. I've always been grateful for his dedication to providing feedback; it helped me to mind the details, in a field where details are everything. Third, Mike did all this at a difficult time in his life, when no one would have blamed him for having his mind on matters other than his students. Finally, Mike suggested that I apply to Amsterdam, which is how I came to be at the ILLC.

From my time in Madison, I'd also like to thank Madeleine Arseneault, Joey Baltimore, Sara Chant, Paul Dunn, Zach Ernst, Matt Ferkany, Tim Hansel, Fred Harrington, Michael Humiston, Holly Kantin, Cora Lee Kleinhenz, John Koolage, Gene Marshall, Corey Mather, Margaret Moore, Greg Novack, Tasia Persson, Alan Rubel, Ben Sachs, Eric Stencil, Joel Velasco, Andrea Veltman, and Matt Vickery, for good times and giving me a lot to think about. In particular, I thank Greg for some interesting discussions about Arrow's Theorem (which we still need to do something about!).

In Amsterdam, there are many people to thank: At the ILLC, our administrative staff—Marjan Veldhuisen, Tanja Kassenaar, Jessica Pogorzelski, Karen Gigengack, Ingrid van Loon, and Peter van Ormondt—have been of help on more occasions than I can count. Rene Goedman, our doorman at Euclides, eagerly subverted the rules to smooth our path and was always ready with some words of wit (or to listen to me complain). My officemates Olivier Roy, Stefan Bold, Katja Rybalko, Yurii Khomski, Brian Semmes, Umberto Grandi, Andreas Witzel, and Jonathan Zvesper, were always ready to listen to an idea, offer advice, or share a laugh. Stefan and Andi were also always ready to show me an interesting web site in case I needed (or did not need) a distraction. Others at the ILLC, without whom it would not have been the same: Stephané Airiau, Edgar Andrade-Lotero, Martin Bentzen, Nick Bezhanishvili, Dave Cochran, Inés Crespo, Cédric Dégremont, Tejaswini Deoskar, Ioanna Dimitriou, Fenrong Liu, Raquel Fernández, Hartmut Fitz, Gaëlle Fontaine, Caroline Foster, Amélie Gheerbrant, Sujata Ghosh, Nina Gierasimczuk, Patrick Girard, Davide Grossi, Jens Ulrik Hansen, Daisuke Ikegami, Tikitu de Jager, Szymon Klarman, Jarmo Kontinen, Wouter Koolen-Wijkstra, Lena Kurzen, Olivia Ladinig, Raul Leal Rodriguez, Henrik Nordmark, Eric Pacuit, Daniele Porello, Petter Remen, Federico Sangati, Leigh Smith, Marc Staudacher, Jakub Szymanik, Reut Tsarfaty, Fernando Velazquez-Quesada, Jacob Vosmaer, Jelle Zuidema, and probably some people I've neglected to list. Also in Amsterdam, but not at the ILLC: Jill Woodward and Martijn Buisman. Martijn has been a true friend, and all the better, a fanatical devotee of good beer which I would not have found on my own.

Two people with whom I write open-source software contributed a great deal to helping me stay sane while working on this dissertation: Michael Kiefte and Brent Easton. The usual suspects—Nate Ellefson, Matt Kuhns, Tom Plagge,

Matt Potter—also played a significant role on this front, as well as Paul Thelen, who offered some valuable perspective late at night (for me, not him). Additional thanks go out to Paul for his hospitality during AAAI 2008 in Chicago, to Brent for his hospitality after KR 2008 in Sydney, and to Matt Kuhns for designing this dissertation's cover.

Thanks are due to Peter van Ormondt for translating my abstract into Dutch, for dealing with the UvA's accounting office on my behalf in connection with the Cabal project, and for general commiseration.

Thanks to Vangelis Markakis for discussions about approximation, and the insight that I should try block matrices in the alternate proofs of Theorems 3.4.2 and 3.4.3; to Tuomas Sandholm for helpful pointers (and oracular pronouncements) about branch-and-bound; to Vince Conitzer, Judy Goldsmith, and Jörg Rothe for numerous discussions at numerous conferences; to Jon Stewart for helping me sort out a particularly nasty bug in my branch-and-bound solver; to Leen Torenvliet and Peter van Emde Boas for always being eager to answer my questions whenever I poked my head into their office (though Leen, being a determinist, would say he had no choice), and to Peter for help with editing my samenvatting, for spotting typos, and for providing references from his gargantuan library.

Thanks to Brammert Ottens for solving a resource allocation question I had—it's gratifying to have another person deem a question of yours interesting enough to spend time on it—and to Sara Ramezani for investigating an extension of some branch-and-bound heuristics I worked on. I learned a lot from working (and talking) with both of you, more, I think, than you learned from me.

Thanks are due to my coauthors, Yann Chevaleyre, Ulle Endriss, Jérôme Lang, and Andreas Witzel. Much of the core of this thesis is the result of joint work with them. A nod goes to Yann for the difficult proof of Theorem 4.4.13; for this one Ulle and I mostly just minded the details. It's been good working with all of you. Memories of an afternoon spent in front of a chalkboard with Nicolas and Ulle devising resource allocation examples (though what we were doing didn't work out) and, with the same two, drinking cider in a pub in Madrid, are ones I won't soon forget.

I want to thank Yann and Nicolas (and Akin Kazakci) for sharing their office with me when I visited LAMSADE in April 2008, and Jérôme for helping to organize my visit to IRIT in Toulouse in May. While in Toulouse, Sylvain Bouveret's assistance was indispensable—without his help, I would have had neither meals nor a place to stay, let alone a view of the Canal du Midi from my window. Sylvain's (and Marianne's) hospitality made my stay in Toulouse not just possible, but enjoyable. Thanks also to Elise Bonzon, Florence Dupin de Saint-Cyr, and Sylvia Estivie for interesting discussions while I was at IRIT, and to Alexis Tsoukiàs, Denis Bouyssou, and Guillaume Ravilly-Abadie for the same while I was at LAMSADE.

I am grateful to my committee, Johan van Benthem, Giuseppe Dari-Mattiacci, Jérôme Lang, Benedikt Löwe, Francesca Rossi, and Mike Wooldridge, for taking the

time to read my dissertation (and in the case of Benedikt and Jérôme, providing extensive suggestions), and to Krzysztof Apt for being my promotor and for arranging for me the account I needed at CWI in order to conduct the experiments in Chapter 6.

Benedikt Löwe has been particularly helpful to me while at the ILLC: One summer day in 2006, not long after I started the work you are holding, he called (!) to offer me the position in the GLoRiClass project which provided my funding. Both before and after that, we have had more discussions about more things than I can say, from set theory to German election law to typesetting, in addition to his offering some insight into How Things Work.

My advisor, Ulle Endriss, has, I think, had a harder job than I would have liked him to have had. Ulle has shown superhuman patience with my way of working and with my intransigence about fixing things which are un- or poorly explained, always gently nudging me in the right direction, and never gave up on Chapter 6, long after I had despaired of ever finishing it. Working with Ulle has always been easy, and that's much appreciated. I'm thankful to have had him as my advisor.

Finally, I want to thank my wife, Sara, not only for her ceaseless encouragement, but for telling me that I could quit if I wanted to—it was that which gave me the strength to finish.

Amsterdam
October, 2009

Chapter 1

Introduction

> The whole is more than the sum of its parts.
>
> *Metaphysics*, Book VIII, 1045a10
> Aristotle

> "I would prefer not to."
>
> *Bartleby, The Scrivener. A Story of Wall-Street*
> Herman Melville

> The miserable wasteland of multidimensional space was first brought home to me in one gruesome solo lunch hour in one of MIT's sandwich shops. "Wholewheat, rye, multigrain, sourdough or bagel? Toasted, one side or two? Both halves toasted, one side or two? Butter, polyunsaturated margarine, cream cheese or hummus? Pastrami, salami, lox, honey cured ham or Canadian bacon? Arugula, iceberg, romaine, cress or alfalfa? Swiss, American, cheddar, mozzarella, or blue? Tomato, gherkin, cucumber, onion? Wholegrain, French, English or American mustard? Ketchup, piccalilli, tabasco, soy sauce? Here or to go?"
>
> *Balliol College Annual Record, 2001*
> Myles Aston

What are preferences? Why have them over a combinatorial domain? Why do we want to represent them compactly? Why represent them at all? We begin this dissertation by unpacking its subtitle and addressing these questions.

Compact **Preference** Representation Over Combinatorial Domains

Any entity not wholly indifferent to the state of the world has preferences. I prefer ales to pilsners, my cat prefers to be petted in one direction over the other, 131 million people expressed their preference for President of the United States by voting in the 2008 general election [Federal Election Commission, 2009]. These are *ordinal* preferences, ranking one alternative ahead of another. *Cardinal* preferences, which assign values to alternatives, are ubiquitous as well: All monetary transactions involve cardinal preferences as prices. I would pay $300

for a camera with features X and Y, and €2 for a coffee in Paris. Stock markets collect the cardinal preferences of investors; auctions do the same for bidders. It is true (though possibly trite): Preferences are everywhere.

Compact **Preference Representation** Over Combinatorial Domains

The ubiquity of preferences in our interactions with each other brings about the need for us to express them. When you place a bid in an auction on the Internet auction site eBay, your bid encodes what you are willing to pay for the item being auctioned. My cat bats my hand away when he'd rather I leave him alone. I say to my dinner companions that I would rather eat at the Thai than the Indonesian restaurant. Voters mark ballots for their preferred candidates. In all of these cases, there is a mechanism by which individuals—agents—translate their preferences from whatever form they take inside their heads into a form which is visible to others. This external form is the *representation* of an agent's preferences. Representations matter: Arguably, the outcome of the 2000 U.S. Presidential election was due to a faulty preference representation method.

Compact Preference Representation Over **Combinatorial Domains**

If I have a basket of fruit and offer you a piece, then what I am asking you to do is express your preferences over single pieces of fruit. The domain—that is, the set of alternatives—is the contents of the basket. If my basket contains an apple, a banana, a cherry, a fig, a grapefruit, a lime, a mango, a nectarine, an orange, and a peach, then it will not be overly difficult for you give your entire preference *order* over the pieces of fruit. For example, you might say that

$$G > C > A > P > B > F > N > M > O > L,$$

where $>$ is to be read as "is preferred to". I might need your full preference order because I am offering fruit to others as well, and I want to ensure that no one is stuck with their last choice. Now consider what happens if I am giving away not just single pieces of fruit, but arbitrary collections of it. Originally, the possible outcomes for you were ten—for each single piece of fruit, you could be given it. Now, the space of potential outcomes has grown exponentially: You could be given any of the 1023 ($= 2^{10} - 1$) combinations of the pieces of fruit in the basket. We have moved from a simple domain to a *combinatorial* one.

Compact Preference Representation Over Combinatorial Domains

We continue with the fruit basket example: If I need your complete preference ordering over all 1023 nonempty subsets of fruit in order to make a decision about what fruit to give you, then we are facing a serious problem. You will surely not want to rank each of the 1023 nonempty subsets of the fruit in the basket even

if you are able to do so; moreover, I will not want to wait for you to do it, nor would I want to deal with such a torrent of information even if you were able to produce it quickly. If I added another ten pieces of fruit to my basket, the number of subsets of fruit would already exceed one million, and with 300 pieces of fruit, there are more subsets ($2^{300} \approx 2 \times 10^{90}$) than atoms in the observable universe ($\approx 10^{80}$). The problem being described here is known as *exponential blowup*, and is a perennial issue when the space of alternatives has a combinatorial structure. What is needed here is a more compact way of expressing your preferences over the subsets of fruit, one which does not require you to list your ordering explicitly, but rather takes advantage of the structure of your preferences and in so doing permits you to convey them concisely. Even for computerized agents, handling preferences over combinatorial domains can quickly become unmanageable without good representations.

Having unpacked the subtitle, it should be clear why compact preference representation over combinatorial domains is needed. Now on to the title: A little reflection on common experience reveals that we often have complex preferences, even over multiples of the same type of thing. For example, while I might be willing to pay $3 for my first ice cream cone, it is unlikely that I will place as high a value on a second, third, or fourth cone. At some point, I might even refuse to accept additional ice cream cones offered to me at no cost. The upshot is that my—and probably also, your—preferences over ice cream cones are such that I will value n cones less than n times the value I place on *one* cone. My preferences are *subadditive*.

Examples pointing in the opposite direction exist as well: Adjacent plots of land may be worth more together than individually. Rights to use sections of railway tracks are more valuable in combination when they link desirable locations. Matching trucks with truckloads to prevent trucks from traveling empty in one direction makes the loads more valuable together than singly. Complete sets of baseball cards are worth more than the individual cards comprising them. Takeoff and landing rights at airports are useless if not matched. All of these are examples of *superadditive* preferences. (For more examples of complex preferences see [Ball, Donohue, and Hoffman, 2006; Caplice and Sheffi, 2006; Cantillon and Pesendorfer, 2006; Sandholm, 2007].)

What these cases all have in common is that the goods involved interact to affect their value as a collection. Together, they have value which is more (or less) than the sum of the values of the their constituent parts—and this is where the difficulty of preference representation over combinatorial domains lies. *Modular* preferences, ones where the values of goods are independent from one another, are simple to represent. There is no need to introduce any conceptual heavy machinery to handle them, they may be written concisely in an obvious way, and almost all computational problems involving them are easy. But as soon as we move away from modular preferences—and, as we have seen, nonmodular preferences

are found abundantly in the real world—we are immediately confronted with representation problems. These are the problems we tackle in this dissertation.

We return once more to the fruit basket example, to give a taste of how the structure of an agent's preferences can be exploited to dramatically simplify their representation. As mentioned above, modular preferences are easy to represent. If the values of the fruits are independent for some agent, then we can represent his preferences over bundles of fruit by writing the agent's value for each piece of fruit:

$$\{(A,1),(B,3),(C,2),(F,2),(G,1),(L,3),(M,4),(N,2),(O,1),(P,3)\}$$

Then, all we need to do to determine this agent's value for any bundle of fruit is to sum the values of the individual pieces contained in it. For example, the bundle $\{B,F,G\}$, containing the banana, the fig, and the grapefruit, has value 6 for an agent with these preferences.

An agent might well have more complex structure to his preferences than this. For example, he might be allergic to cherries, and so any bundle containing cherries is worse than one without cherries. He might intend to cook a dessert which uses both limes and mangoes, but has no use for one without the other. Or he might not care what fruit he gets, so long as he gets at least one piece. Look again at the example of modular preferences over the fruit. We assigned a symbol—a propositional variable—to each piece of fruit, and a value which accrues to the agent for receiving that piece of fruit—for making that propositional variable true. This suggests an extension, by which we use more complex logical formulas instead of just single propositional variables. In this way, we can say $\neg C$ if we want to avoid cherries, $L \wedge M$ if we get extra value from receiving the lime and mango together, and $A \vee B \vee C \vee F \vee G \vee L \vee M \vee N \vee O \vee P$ to say that we want at least one piece of fruit, but we don't care which one. So,

$$\{(\neg C,1),(L \wedge M,5)\}$$

could be the preferences of an agent who wants to avoid cherries and to get the lime and mango together. These kinds of languages, languages of weighted formulas, are the method of compact preference representation which we pursue in this dissertation.

Chapter Overview

The aim of this dissertation is to explore the possibilities of a particular formalism for compact preference representation over combinatorial domains—sets of weighted formulas, known as *goalbases*. The overall structure may be seen in Figure 1.1. A chapter at the head of an arrow relies on results from the chapter at the tail of the same arrow.

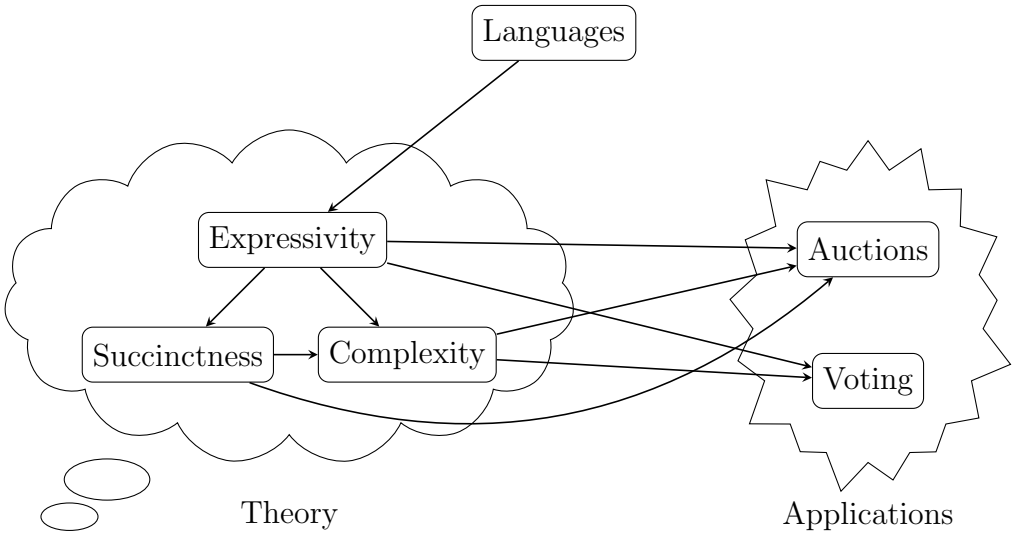


Figure 1.1: The structure of this dissertation.

Chapter 1, Introduction is the chapter you are reading now.

Chapter 2, Languages introduces the basic formalism and notation used throughout this dissertation. In particular, we define goalbases, goalbase languages, and the various restrictions which may be placed on them; we show how goalbases generate utility functions, and thereby represent cardinal preferences; and we give a wide-ranging overview of other preference representation languages, both ordinal and cardinal.

Part I, Theory is the heart of this dissertation, where we explore the properties of the goalbase languages defined in Chapter 2. In particular, we examine in detail the expressivity, succinctness, and complexity of each language.

Chapter 3, Expressivity takes up a basic question about each goalbase language defined in Chapter 2, namely: Which utility functions are expressible in each language? We show that many goalbase languages correspond exactly to classes of utility functions having well-known properties. Along the way, we also prove some results about the variety of available representations in certain languages. In particular, we show that some goalbase languages have exactly one representation for each utility function they are able to represent, a property we call *unique representations*.

Chapter 4, Succinctness considers how compactly our goalbase languages are able to represent utility functions. Here, we present numerous pairwise

comparisons between goalbase languages, in some cases showing that one language is exponentially more succinct than another. Due to our systematic approach, this chapter contains hundreds of results, conveniently summarized in several tables.

Chapter 5, Complexity classifies goalbase languages according to the computational complexity of deciding various questions concerning goalbases in those languages. For many (though not all) goalbase languages, the decision problem MAX-UTIL, which asks whether an alternative exists which produces at least a given level of utility, is NP-complete. Similarly, the problem MIN-UTIL, which asks whether all alternatives yield at least some minimum amount of utility, is coNP-complete for many of the more expressive languages. Thirdly, we consider the problem MAX-CUF, which deals with maximizing collective utility, rather than individual utility as MAX-UTIL does, and again find that for many—though, significantly, not all—goalbase languages, MAX-CUF is NP-complete. Finally, we consider an alternative version of MAX-UTIL, which asks about true atoms in optimal states instead of the existence of states yielding at least a given amount of utility.

The chapters in Part I are based on and extend work presented at the AAAI-2007 Workshop on Preference Handling for Artificial Intelligence (AiPref-2007) [Uckelman and Endriss, 2007], the 11th International Conferences on Principles of Knowledge Representation and Reasoning (KR-2008) [Uckelman and Endriss, 2008b], and in the journal article "Representing Utility Functions via Weighted Goals" [Uckelman, Chevaleyre, Endriss, and Lang, 2009]. In turn, the latter includes some results due to Chevaleyre, Endriss, and Lang [2006]. Section 5.7 extends work presented at the AAAI-2008 4th Multidisciplinary Workshop on Advances in Preference Handling (MPREF-2008) [Uckelman and Witzel, 2008].

Part II, Applications highlights two areas in which goalbase languages may be used to good effect.

Chapter 6, Auctions is the first of our two chapters showing applications of goalbase languages. Auctions are a common way of selling goods. Unfortunately, sequential auctions—auctions where individual goods are sold consecutively—are inefficient when the values of goods being sold are interdependent. Combinatorial auctions are a method of auctioning all goods simultaneously, so that synergies among goods may be taken into account. In this chapter, we discuss existing bidding languages for combinatorial auctions, suggest the use of goalbase languages for bids, present two algorithms for solving the Winner Determination Problem for combinatorial auctions when using goalbases as bids, and give experimental results for these algorithms.

All save Section 6.3 of this chapter is based on and extends work presented at the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2008) [Uckelman and Endriss, 2008a].

Chapter 7, Voting points toward elections as another area in which goalbase languages may be useful. We consider two main problems: First, many voting methods are insufficiently expressive to capture the preferences of voters. Second, voting methods intended to choose only a single winner acquire undesirable properties when modified to produce multiple winners, as in elections for committees. In this chapter, we argue that using goalbases as ballots has potential both as a way of extending the expressivity of single-winner voting methods, and also as a way of handling the combinatorial nature of voters' preferences in multi-winner voting.

Chapter 8, Conclusion is at the end, summarizing what we have shown and suggesting some avenues for further work.

Chapter 2

Languages

2.1 Introduction

In this chapter we present the basic notation and terminology which is used throughout this dissertation, as well as an overview of previous work on preference representation.

2.2 Notation

In Section 2.2.1, we define some fundamental notions from propositional logic and give names to certain classes of propositional formulas. In Section 2.2.2 we introduce utility functions, goalbases, and goalbase languages. Goalbase languages are the utility representation framework which we study in Chapters 3–7.

2.2.1 Propositional Logic

Though we expect that the reader is already familiar with propositional logic, we define it here for the sake of completeness.

Definition 2.2.1 (Propositional Formulas)**.** The set $\mathcal{PS}$ is a fixed, finite set of propositional variables. We write $\mathcal{PS}_n$ to indicate that $|\mathcal{PS}| = n$. Given a particular $\mathcal{PS}$:

- Each $p \in \mathcal{PS}$ is a formula.
- If φ is a formula, then $\neg\varphi$ is a formula.
- If φ and ψ are formulas, then $\varphi \wedge \psi$ and $\varphi \vee \psi$ are formulas.
- $\top$ and $\bot$ are formulas.

Let $\mathcal{L}_{\mathcal{PS}}$ be the language of propositional logic over $\mathcal{PS}$. That is, $\mathcal{L}_{\mathcal{PS}}$ is the set of all formulas generated by the atoms in $\mathcal{PS}$. The technical results found here apply to formulas that contain only the connectives $\neg$, $\wedge$, and $\vee$. We omit $\rightarrow$ (implication) as a Boolean connective because is it succinctly definable in terms of $\neg$ and $\vee$. Equivalence and XOR we do not consider here; they are not obviously useful for our purposes, though their inclusion might result in more succinct languages.

Definition 2.2.2 (Propositional Models)**.** A *model* is a set $M \subseteq \mathcal{PS}$. The satisfaction relation $\models$ for models and formulas is defined as follows:

$$\begin{array}{lll} M \models \top. & & \\ M \not\models \bot. & & \\ M \models p & \text{iff} & p \in M. \\ M \models \neg\varphi & \text{iff} & M \not\models \varphi. \\ M \models \varphi \wedge \psi & \text{iff} & M \models \varphi \text{ and } M \models \psi. \\ M \models \varphi \vee \psi & \text{iff} & M \models \varphi \text{ or } M \models \psi. \end{array}$$

We give names to some types of propositional formulas:

Definition 2.2.3 (Types of Formulas)**.**

- An *atom* is a member of $\mathcal{PS}$.
- A *literal* is an atom or its negation.
- A *clause* is a disjunction of literals.
- A *cube* is a conjunction of literals.
- A *positive* X is a satisfiable formula of type X that is free of negations.
- A *strictly positive* X is a non-tautologous positive X.
- A k-X is an X with at most k occurrences of atoms.
- A *complete cube* is a cube having every atom as a subformula exactly once.
- A *Horn clause* is a clause with at most one positive literal.

When discussing positive clauses, positive cubes, and positive formulas, we frequently abbreviate these to *pclauses*, *pcubes*, and *pforms*, respectively. Additionally, we call strictly positive cubes and strictly positive formulas *spcubes* and *spforms*, respectively. (The term *spclauses* is redundant because every positive clause is falsifiable.) Atoms are 1-spclauses, 1-spcubes, and 1-spformulas (and also 1-pclauses, 1-pcubes, and 1-pformulas), while literals are 1-clauses, 1-cubes, and

1-formulas. Clauses, cubes, and formulas are ω-clauses, ω-cubes, and ω-formulas, respectively, which is to say that the formulas may be of any finite length.[1] Complete cubes are also known as *state formulas*, since any complete cube is true in precisely one state. Note that by convention $\bigwedge \emptyset = \top$ and $\bigvee \emptyset = \bot$, from which follows that $\top$ is the unique 0-pcube and $\bot$ the unique 0-clause. The notation $X + \top$ indicates the set of formulas $X \cup \{\top\}$ (e.g., pclauses $+ \top$ is the set containing all pclauses along with $\top$).

Definition 2.2.4 (State Formulas)**.** If $X \subseteq \mathcal{PS}$, then define $\bar{X} = \mathcal{PS} \setminus X$, and $\neg X = \{\neg p \mid p \in X\}$. Then $\bigwedge(M \cup \neg\bar{M})$ is the *state formula* corresponding to the model M.

For example, if $\mathcal{PS} = \{a, b, c, d\}$, then the state formula for the model $\emptyset$ is $\neg a \wedge \neg b \wedge \neg c \wedge \neg d$ and for $\{a, b\}$ is $a \wedge b \wedge \neg c \wedge \neg d$. Notice that $M' \models \bigwedge(M \cup \neg\bar{M})$ iff $M = M'$.

2.2.2 Utility Functions, Goalbases, and Languages

We are interested in utility functions over combinatorial domains that are the Cartesian product of several binary domains. A generic representation of this kind of domain is the set of all possible models for propositional formulas over a fixed language with a finite number of propositional variables (the dimensionality of the combinatorial domain).

Definition 2.2.5 (Utility Functions)**.** A *utility function* is a mapping $u\colon 2^{\mathcal{PS}} \to \mathbb{R}$.

Because the utility functions we consider have sets as their domain, and propositional models are sets, utility functions can be thought of as mapping models to their values.

Definition 2.2.6 (Weighted Goals and Goalbases)**.** A *weighted goal* is a pair (φ, w), where φ is a formula in the language $\mathcal{L}_{\mathcal{PS}}$ and $w \in \mathbb{R}$. A *goalbase* is a finite multiset $G = \{(\varphi_i, w_i)\}_i$ of weighted goals.

Goals are typically required to be satisfiable formulas. We will see in Chapter 3 that for the languages studied here this restriction does not affect expressive power, though the presence of unsatisfiable formulas can affect the computational complexity of some decision problems, as discussed in Chapter 5. When a particular goalbase is under consideration, we write w_φ to mean the weight of formula φ in that goalbase. $\mathrm{For}(G)$ is the set of formulas in G. $\mathrm{Var}(\varphi)$ is the set of propositional variables in the formula φ and $\mathrm{Var}(G) = \bigcup_{\varphi \in \mathrm{For}(G)} \mathrm{Var}(\varphi)$.

[1] Strictly speaking, we should write, e.g., $<\omega$-cubes instead of ω-cubes, but we abuse notation for the sake of brevity and because all formulas are assumed to have finite length.

Definition 2.2.7 (Generated Utility Functions)**.** A goalbase G and an *aggregation function* $F\colon \mathbb{N}^{\mathbb{R}} \to \mathbb{R}$ *generate* a utility function $u_{G,F}$ mapping each model $M \subseteq \mathcal{PS}$ to $u_{G,F}(M) = F(w \mid (\varphi, w) \in G \text{ and } M \models \varphi)$.

Aggregation functions map multisets of reals to reals.[2] Because multisets are unordered structures, any aggregation function will necessarily be associative and commutative over weights.[3] In this dissertation, we restrict ourselves to two aggregation functions, Σ and max, the summation and maximum functions, respectively. When $F = \Sigma$, the utility function generated from a goalbase G is

$$u_{G,\Sigma}(M) = \sum_{\substack{(\varphi,w)\in G \\ M\models\varphi}} w,$$

which is to say that the value of a model is the sum of weights of formulas made true in that model. For example, if $\mathcal{PS} = \{p, q, r\}$, then the goalbase $G_1 = \{(p \vee q \vee r, 2), (p \wedge q, 1), (p \wedge r, 1), (q \wedge r, 1), (p \wedge q \wedge r, -2)\}$ generates the utility function $u\colon X \mapsto \min(3, 2{\cdot}|X|)$. When $F = \max$, the utility function generated from a goalbase G is

$$u_{G,\max}(M) = \max_{\substack{(\varphi,w)\in G \\ M\models\varphi}} w.$$

In other words, the value of a model is the same as the largest weight had by any formula which is true in that model.

These two aggregation functions may produce dramatically different utility functions from the same goalbase. E.g., if $G = \{(a, 1) \mid a \in \mathcal{PS}\}$, then $u_{G,\max}$ is the simple unit-demand utility function ($u(X) = 1$ if $X \neq \emptyset$, 0 otherwise) while $u_{G,\Sigma}$ is the simple additive utility function ($u(X) = |X|$). In case max is used, we assume $\max(\emptyset) = -\infty$; it is often useful to include, say, $(\top, 0)$ in any goalbase intended for use with max so as to obtain utility functions defined for all states.

[2]The definition of aggregation function given here differs subtly from that given by Chevaleyre et al. [2006], Uckelman and Endriss [2007], Uckelman and Endriss [2008a], Uckelman and Endriss [2008b], Uckelman and Witzel [2008], and Uckelman et al. [2009], in that all of these write the aggregation function as $F\colon 2^{\mathbb{R}} \to \mathbb{R}$ when in practice the reader is meant to understand the aggregation function as operating on multisets. (Lafage and Lang [2000] do the same with their definition of disu_P as does Lang [2004] with the definition of F_1, F_2, and F_3 for R_{wg}.) In particular, these authors often write sums of weights as $\sum\{w \mid \textit{stuff}\}$, when what is intended is $\sum_{\textit{stuff}} w$. We have striven to avoid this ambiguity in the present work; in the event that we have failed, please in all cases read sums of weights as sums of *multisets* (rather than sets) of weights. That is, $\sum\{1, 1\} = 2 \neq 1$.

[3]If the domain were arbitrary-length tuples of reals instead of multisets of reals ($\mathbb{R}^*$ instead of $\mathbb{N}^{\mathbb{R}}$), then there could be aggregators which are sensitive to the order in which formula weights are aggregated. Since goalbases are unordered structures, there is no compelling reason to be concerned with the order in which weights are aggregated, so we limit the domains of aggregators to multisets.

Definition 2.2.8 (Goalbase Equivalence)**.** Two goalbases G and G' are equivalent with respect to an aggregation function F (written $G \equiv_F G'$) iff they define the same utility function. That is, $G \equiv_F G'$ iff $u_{G,F} = u_{G',F}$.

Goalbases provide a framework for defining different languages for representing utility functions. Any restriction we might impose on goals (e.g., we may only want to allow clauses as formulas) or weights (e.g., we may not want to allow negative weights) and any choice we make regarding the aggregator F give rise to a different language. An interesting question, then, is whether there are natural goalbase languages (defined in terms of natural restrictions) such that the utility functions they generate enjoy simple structural properties. (This is indeed the case, as seen in Chapter 3.)

Definition 2.2.9 (Languages and Classes of Utility Functions)**.** Let $\Phi \subseteq \mathcal{L}_{\mathcal{PS}}$ be a set of formulas, $W \subseteq \mathbb{R}$ a set of weights, and F an aggregation function. Then $\mathcal{L}(\Phi, W, F)$ is the set of all goalbases formed by formulas in Φ with weights from W to be aggregated by F, and $\mathcal{U}(\Phi, W, F)$ is the class of utility functions generated by goalbases belonging to $\mathcal{L}(\Phi, W, F)$. More generally, we write $\mathcal{U}(\mathcal{L})$ to mean the class of utility functions generated by goalbases in the language $\mathcal{L}$.

In order to keep the reader from being overwhelmed by indices, we may sometimes omit F and write u_G, $\equiv$, $\mathcal{L}(\Phi, W)$, and $\mathcal{U}(\Phi, W)$ in preference to $u_{G,F}$, $\equiv_F$, $\mathcal{L}(\Phi, W, F)$, and $\mathcal{U}(\Phi, W, F)$ when context makes clear which aggregation function F is.

Regarding weights, we study the restriction to the positive reals ($\mathbb{R}^+$) as well as the general case ($\mathbb{R}$). For complexity questions we will restrict our attention to the rationals ($\mathbb{Q}$). We restrict formulas by their structure, according to the types of formula defined in Definition 2.2.3. For example, the language $\mathcal{L}(\mathit{cubes}, \mathbb{R}^+, \Sigma)$ consists of all goalbases which contain only positively-weighted cubes, and are aggregated using summation. Many more examples of languages will be seen in Chapter 3, where we investigate language expressivity.

We may occasionally wish to combine goalbases. For this purpose, we define a notion of goalbase summation.

Definition 2.2.10 (Goalbase Summation)**.** If G, G' are goalbases, then

$$G \oplus G' = \left\{ \left(\varphi, \sum_{(\varphi,a)\in G} a + \sum_{(\varphi,b)\in G'} b \right) \;\middle|\; \varphi \in \mathrm{For}(G \cup G') \right\}$$

is their sum.

Note that $\oplus$ *does not* combine formulas which are semantically equivalent but syntactically distinct. E.g., $\{(p,1)\} \oplus \{(p \wedge p, 1)\} \neq \{(p,2)\}$. Combining equivalent weighted formulas would involve first checking for equivalence, which we wish to avoid because equivalence checking is coNP-complete in the general case.

We define a notion of uniform substitution for formulas and goalbases, which we occasionally need when transforming them:

Definition 2.2.11 (Uniform Substitution)**.** If $\varphi, \psi_1, \ldots, \psi_k, \chi_1, \ldots, \chi_k$ are formulas, then $\varphi[\psi_1/\chi_1, \ldots, \psi_k/\chi_k]$ is the result of (simultaneously) substituting ψ_i for every occurrence of χ_i as a subformula in φ. If G is a goalbase, then $G[\psi_1/\chi_1, \ldots, \psi_k/\chi_k]$ is the result of applying the substitution to each $(\varphi, w) \in G$.

Finally, a note on non-binary domains: Due to the applications we have in mind for goalbase languages, resource allocation, auctions, voting—all binary in the sense that an agent has an item or does not, or a candidate is a winner or not—we have restricted our variables to have binary domains. Moreover, restriction to binary domains is a natural one when working with propositional logic. Nonetheless, it is possible to simulate in our framework variables which take on a larger (but still finite) set of values, by coding single many-valued variables into multiple binary-valued ones. For example, a three-valued variable X could be decomposed into atoms x_0, x_1, where x_i represents the ith bit of X's value (assuming that its three values are enumerated $0, 1, 2$). In this case, we have one "extra" state, the one where $x_0 \wedge x_1$ is true, corresponding to no value in X's domain, due to the fact that the size of X's domain is not a power of two. This overhang can be adjusted for in several ways, e.g., by giving $x_0 \wedge x_1$ a large negative weight so that all "invalid" models are dominated by the "valid" ones. (The alternative proof of Theorem 5.5.6 on p. 107 shows an example of this trick, though in a different context.) The number of new binary variables required to "binarize" any variable X is $\lceil \log |\mathrm{dom}\, X| \rceil$; so long as the size of the domain of X does not vary with $|\mathcal{PS}|$, our succinctness and complexity results in Chapters 4 and 5 will be unaffected by the use of variables with larger domains. Similarly, our expressivity results in Chapter 3 carry over to many-valued variables, though we must caution that the naturalness of representations may be lost in translation.

2.3 Related Languages

There are a wide variety of languages for expressing preferences. In this section, we survey a selection of them. Some, such as CP-nets, are fundamentally ordinal languages. Others, such as penalty logic, weighted description logics, bidding languages, and generalized additive functions, are cardinal. Still others, such as valued constraint satisfaction problems, as well as some of the languages discussed by Lafage and Lang [2000], build ordinal preferences upon cardinal components.

2.3.1 CP-Nets

CP-nets are a formalism devised by Boutilier, Brafman, Geib, and Poole [1997] and refined by Boutilier, Brafman, Hoos, and Poole [1999a] for specifying conditional

ceteris paribus preferences in a compact fashion.[4] A ceteris paribus preference for a over b means that *all else being equal*, a is preferred to b. For example, it might be the case that, in the absence of other differences, I prefer the amplifier which has 11 as its maximum volume to one which goes only to 10. A conditional preference for a over b depends on some given state c. Conditional preferences are common in situations where multiple issues must be resolved, the canonical example being a diner who prefers white wine if the main course is fish, but red wine if the main course is beef. Putting these two together, a conditional *ceteris paribus* preference is one where the ordering over the domain of one option depends on how some subset of the other options are resolved.

A CP-net is a directed graph where each vertex X_i is a variable, there is an edge from X_i to X_j if the value of X_j depends on the value of X_i, and associated with each variable X_i is a conditional preference table specifying which total preorder over the domain of X_i is applicable given assignments to the variables on which X_i depends.

To illustrate this, we repeat an example given by Boutilier, Brafman, Domshlak, Hoos, and Poole [2004, Example 3]. Suppose that I am dressing for a fancy occasion and can choose black or white pants, a black or red shirt, and a black or white jacket. I unconditionally prefer the black jacket to the white, and the black pants to the white, but my preference for what shirt I wear depends on the combination of jacket and pants I will wear. Figure 2.1(a) shows one possible CP-net representing my conditional preferences over the colors of my jacket, pants, and shirt; Figure 2.1(b) shows the preference order over alternatives induced by that CP-net. Notice that the induced order is not total—for example, the black jacket with white pants and white shirt is incomparable to the white jacket with white pants and red shirt—and in general there may be many total orders which are compatible with the induced preorder.

In the previous example, the dependency graph is acyclic, but this is not an essential feature of CP-nets. For example, I may wish to wear socks given that I am wearing shoes instead of sandals, but also vice versa. However, once we permit cycles, we are no longer guaranteed to have a corresponding preference order.

Under some conditions, we can use CP-nets to efficiently answer ordering and dominance queries—whether a total ordering exists in which outcome o is strictly better than o', and whether in every total ordering o is strictly better than o', respectively [Boutilier et al., 2004, Section 4]. Individual CP-nets are intended for the representation of individual preferences; however, there has also been a great deal of work on aggregating the CP-nets of multiple agents in order to find group preferences [Xia, Lang, and Ying, 2007b,a; Xia, Conitzer, and Lang, 2008; Lang and Xia, 2009; Xia and Lang, 2009; Rossi, Venable, and Walsh, 2004; Pilotto, Rossi, Venable, and Walsh, 2009; Apt, Rossi, and Venable, 2008].

[4] *CP* here stands for either *ceteris paribus* or *conditional preference*. Due to an unfortunate naming collision, colored Petri nets, which are used for modeling transition systems, are also sometimes referred to as *CP-nets*.

For a thorough discussion of the properties of CP-nets, see the survey by Boutilier et al. [2004]. CP-nets have also been extended and modified in various ways: TCP-nets permit the expression of the relative importance of variables [Brafman and Domshlak, 2002]; CI-nets generalize one aspect of TCP-nets by permitting statements about the importance of sets of variables [Bouveret, Endriss, and Lang, 2009]; UCP-nets add utilities by combining GA-decompositions of utility functions (see Section 2.3.8) with CP-nets [Boutilier, Bacchus, and Brafman, 2001].

2.3.2 Penalty Logic

Knowledge bases are sets of propositional formulas intended to represent collections of "known" information, possibly derived from multiple sources of varying reliability. As with any set of propositional formulas, a knowledge base may be inconsistent. (This could occur, for example, if some formulas were generated by a malfunctioning sensor and others were generated by properly functioning ones.) We may wish to make inferences from an inconsistent knowledge base nonetheless, but in order to do so we need some method for dealing with inconsistency to prevent us from deriving nonsense. Pinkas [1991] devised penalty logic to address this problem. Penalty logic augments the formulas in a knowledge base with weights, which indicate the cost of falsifying the associated formula.

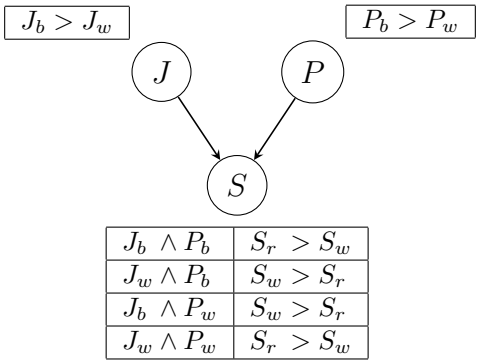
Following Dupin de Saint-Cyr, Lang, and Schiex [1994], a *penalty knowledge base* is a finite multiset of weighted propositional formulas (φ, w) where $w \in \mathbb{R}^+ \cup \{+\infty\}$. The *cost* of a model M given a penalty knowledge base PK is the sum of the penalties of the formulas in PK which M violates,

$$k_{PK}(M) = \sum_{\substack{(\varphi,w)\in PK \\ M\models\neg\varphi}} w,$$

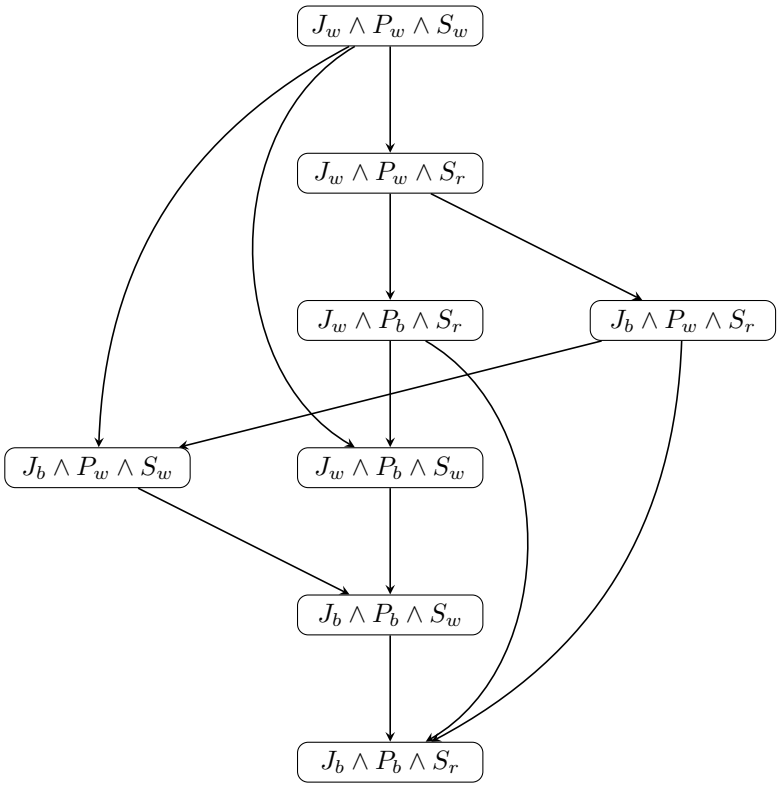
and a *preferred interpretation* is a minimum-cost model.

For example, suppose I am knocked unconscious in a cycling accident and when I awake I see what appear to be majestic snow-capped mountains. I have as a knowledge base $PK = \{(h, 10), (m, 1), (h \rightarrow \neg m, +\infty)\}$, where h stands for "I am in Holland" and m stands for "I see mountains". Having left my office in Amsterdam by bike, I have a strong belief that I am still in Holland, and I have an inviolable belief that there are no mountains there. I believe I see mountains, but realize that my vision might be unreliable due to the accident. According to the penalties in my knowledge base, the model with minimal cost is $\{h\}$, which says that I am in Holland, but I am not really seeing mountains.

In addition to minimal-cost models, we might be interested in minimal-cost subtheories—that is, minimal-cost consistent subsets of PK—or even minimal-cost subtheories consistent with some given formula φ. The cost function k_{PK} induces a total order $\leq_{PK}$ on the subsets of PK. Using this, we can define a nonmonotonic inference relation $\mid\!\sim^c_{PK}$, where $\varphi \mid\!\sim^c_{PK} \psi$ iff for every $S \subseteq PK$



(a) A CP-net.



(b) The induced preference order.

Figure 2.1: What to wear?

which is $\leq_{PK}$-maximal among the φ-consistent subtheories of PK, $S \cup \{\varphi\} \models \psi$. Returning to the example, it is easy to see that the minimal-cost h-consistent subtheory is $\{(h, 10), (h \rightarrow \neg m, +\infty)\}$. (However, in the general case, deciding whether $\varphi \mathrel{|\!\sim^{c}_{PK}} \psi$ is $\mathsf{\Delta}^{\mathsf{p}}_{2}$-complete [Cayrol, Lagasquie-Schiex, and Schiex, 1998].)

There is a simple translation between penalty knowledge bases and sum-aggregated goalbases: A penalty knowledge base $PK = \{(\varphi_1, w_1), \ldots, (\varphi_k, w_k)\}$ may be translated into a goalbase $G = \{(\neg\varphi_1, -w_1), \ldots, (\neg\varphi_k, -w_k)\}$; having done that, it will be the case that $k_{PK}(M) = -u_{G,\Sigma}(M)$ for all models M.

2.3.3 Weighted and Distance-Based Logics for Cardinal Disutility

Lafage and Lang [2000] introduce a framework for cardinal preferences using what they call *weighted logics*. A *preference profile* $\langle P, K\rangle$ is a *preference base* P (in our terms, a goalbase) and a consistent set of formulas K called *integrity constraints*. Worlds (models) are considered possible if they satisfy all of the integrity constraints ($w \models K$); the set of possible worlds is denoted by $\mathrm{Mod}(K)$. An individual's preference ordering over possible worlds is induced by a disutility function $\mathrm{disu}_P\colon \mathrm{Mod}(K) \rightarrow [0, +\infty]$ such that $\mathrm{disu}_P(w) = *\{\alpha_i \mid w \models \neg\varphi_i\}$, where $*\colon [0, +\infty] \times [0, +\infty] \rightarrow [0, +\infty]$ is any operation which is commutative, nondecreasing, associative, and for which 0 is an additive identity (for all a, $a * 0 = a$). The collective disutility function disu_P for a collection of individual preference bases $(P_1, \ldots, P_n)$ is $\mathrm{disu}_P(w) = \Diamond_{i=1}^{n} \mathrm{disu}_{P_i}(w)$, where $\Diamond\colon [0, +\infty]^n \rightarrow [0, +\infty]$ is any operation which is nondecreasing in each argument and commutative. Each pair of operators $\langle *, \Diamond\rangle$ defines a weighted logic. Penalty logic is a special case of this framework, which results from setting $* = +$.

Lafage and Lang [2000] also define an alternative measure of disutility based on distances. A *distance* $d\colon \Omega \times \Omega \rightarrow \mathbb{N}$ (also known as a *metric*) is a mapping which is nonnegative ($d(w, w') \geq 0$), identifies indiscernibles ($d(w, w') = 0$ iff $w = w'$), is symmetric ($d(w, w') = d(w', w)$), and satisfies the triangle inequality ($d(w, w'') \leq d(w, w') + d(w', w'')$). Here the w are considered to be possible worlds. Further, d is extended to cover the distance between a world and a propositional formula, as well as between two formulas:

$$d(w, \varphi) = \min_{w \in \mathrm{Mod}(\varphi)} d(w, w') \qquad\qquad d(\varphi, \psi) = \min_{\substack{w \models \varphi \\ w' \models \psi}} d(w, w')$$

Finally, the distance between a possible world w and a set of (unweighted) goals G is the distance from w to each $\varphi \in G$, aggregated by $*$,

$$d(w, G) = \mathop{\Large *}_{\varphi \in G} d(w, \varphi)$$

which we identify with $\mathrm{disu}_G(w)$. Collective disutility is aggregated as it was with weighted goals. Lafage and Lang suggest using the Hamming distance between

worlds for the metric d. (We consider the consequences of this with regard to committee elections in Section 7.3.2.)

2.3.4 Propositional Languages for Ordinal Preferences

Coste-Marquis, Lang, Liberatore, and Marquis [2004] consider the use of several different propositional preference representation languages for generating preference orderings, aiming to find ones which are more succinct than explicitly listing $M \geq M'$ for each such pair of alternatives. These languages do not originate with Coste-Marquis et al.—for their sources, see [Coste-Marquis et al., 2004]—but are presented there in a uniform fashion for the purpose of comparing their expressivity and succinctness. We enumerate and describe these languages here. All languages mentioned here are distinct; we do not provide examples of their use here, due to their number.

$\boldsymbol{R}_{\mathbf{penalties}}$ The preference relation $R_{\text{penalties}}$ is the one induced by a penalty goalbase, as described in Section 2.3.2.

$\boldsymbol{R}_{\mathbf{H}}$ The preference relation R_{H} is based on the (weighted) Hamming distance between models. The Hamming distance between two models M and M' is the number of variables which would need to have their values negated to convert M into M', and the distance between a model M and a formula φ is the minimum distance between M and any M' such that $M' \models \varphi$:

$$\mathrm{d}_{\text{H}}(M, \varphi) = \min_{M' \models \varphi} \mathrm{d}_{\text{H}}(M, M').$$

From there, we can extend the notion of distance to compare models with goalbases, so that

$$\mathrm{d}_{\text{H}}(M, G) = \sum_{(\varphi, w) \in G} w \cdot \mathrm{d}_{\text{H}}(M, \varphi),$$

which induces an ordering $\preceq^{\text{H}}_G$ where $M \preceq^{\text{H}}_G M'$ iff $\mathrm{d}_{\text{H}}(M, G) \geq \mathrm{d}_{\text{H}}(M', G)$. (Notice that the preference ordering runs in the direction opposite to that of the distance ordering.)

Coste-Marquis et al. define three preference orderings which treat the weights of goals as priorities, rather than as values where the satisfaction of one goal may compensate for the violation of another. (That is, goals are not fungible: no amount of satisfied goals with priority 2 will compensate for a single violated goal with priority 1.)

$\boldsymbol{R}^{\mathbf{bestout}}_{\mathbf{prio}}$ The best-out ordering $R^{\text{bestout}}_{\text{prio}}$ is defined by

$$\mathrm{r}_G(M) = \min_{\substack{(\varphi, w) \in G \\ M \not\models \varphi}} w$$

and so $M \succeq_G^{\text{bo}} M'$ iff $\text{r}_G(M) \geq \text{r}_G(M')$.

$R_{\text{prio}}^{\text{discrimin}}$ The discrimin ordering $R_{\text{prio}}^{\text{discrimin}}$ is defined by

$$\begin{aligned}\text{discr}_G^+(M, M') &= \{\varphi \mid (\varphi, w) \in G, M \models \varphi, M' \not\models \varphi\} \\ \text{discr}_G(M, M') &= \text{discr}_G^+(M, M') \cup \text{discr}_G^+(M', M)\end{aligned}$$

so that $M \succ_G^{\text{discrimin}} M'$ iff

$$\begin{aligned}\min\{w \mid (\varphi, w) \in G, \varphi \in \text{discr}_G^+(M, M')\} \\ < \min\{w \mid (\varphi, w) \in G, \varphi \in \text{discr}_G^+(M', M)\}\end{aligned}$$

and $M \succeq_G^{\text{discrimin}} M'$ iff $M \succ_G^{\text{discrimin}} M'$ or $\text{discr}_G(M, M') = \emptyset$.

$R_{\text{prio}}^{\text{leximin}}$ The leximin ordering $R_{\text{prio}}^{\text{leximin}}$ is defined by

$$d_k(M) = |\{(\varphi, w) \in G \mid M \models \varphi \text{ and } w = k\}|$$

so that $M \succ_G^{\text{leximin}} M'$ iff there exists a k such that $d_k(M) > d_k(M')$ and for all $j < k$, $d_j(M) = d_j(M')$; and $M \succeq_G^{\text{leximin}} M'$ iff $M \succeq_G^{\text{leximin}} M'$ or $d_k(M) = d_k(M')$ for some k.

Coste-Marquis et al. also define two preference orderings based on conditional logics. Here, each goal φ has a context χ. A conditional goal $\chi \colon \varphi$ is satisfied by an ordering $\geqslant$ iff the set of $\geqslant$-maximal models where χ holds are a subset of those models where φ holds; a set of conditional goals is satisfied by $\geqslant$ when all of its members are.

$R_{\text{cond}}^{\text{S}}$ The "standard" conditional preference relation $R_{\text{cond}}^{\text{S}}$ is defined as $M \succeq_G^{\text{cond,S}} M'$ iff every ordering $\geqslant$ which satisfies G has $M \geqslant M'$.

$R_{\text{cond}}^{\text{Z}}$ The Z-ranking preference relation $R_{\text{cond}}^{\text{Z}}$ is much more fine-grained than $R_{\text{cond}}^{\text{S}}$, but also much more complex due to its procedural definition. In addition to a set of conditional goals G, we also have a set of hard constraints K. A conditional goal $\varphi \colon \psi$ is *tolerated* by a set of conditional goals $\{\varphi_1 \colon \psi_1, \ldots, \varphi_k \colon \psi_k\}$ and set of hard constraints K iff $\varphi \wedge \psi \wedge \bigwedge_{i=1}^{k}(\varphi_i \rightarrow \psi_i) \wedge \bigwedge K$ is satisfiable. Then, we build a partition $R_1, \ldots, R_j$ of G as in Figure 2.2.

Use the R_i to define a rank function where $\text{rank}(\varphi \colon \psi) = i$ when $\varphi \colon \psi \in R_i$. Let $G' = \{(\varphi \rightarrow \psi, \mathit{maxrank} - \text{rank}(\varphi \colon \psi) + 1) \mid \varphi \colon \psi \in G\}$. Then, define $M \succeq_G^{\text{cond,Z}} M'$ iff $M \succeq_{G'}^{\text{bo}} M'$.

Finally, Coste-Marquis et al. introduce a preference ordering for *ceteris paribus* preferences:

$k := 0$
$R := G$
repeat
  $k := k + 1$
  $R_k := \emptyset$
  for all $\varphi\colon \psi \in R$ **do**
    if $\varphi\colon \psi$ is tolerated by $R \setminus \{\varphi\colon \psi\}$ **then**
      $R_k := R_k \cup \{\varphi\colon \psi\}$
      $R := R \setminus \{\varphi\colon \psi\}$
    end if
  end for
until $R = \emptyset$
$maxrank := k$

Figure 2.2: Algorithm for partitioning G for use in determining $R^{\mathrm{Z}}_{\mathrm{cond}}$.

$\boldsymbol{R_{\mathbf{cp}}}$ If φ, ψ, χ are formulas and $V \supseteq \mathrm{Var}(\psi) \cup \mathrm{Var}(\chi)$ is a set of variables , then the *ceteris paribus desire* $\varphi\colon \psi > \chi[V]$ means that given φ, $\psi \wedge \neg\chi$ is preferred to $\neg\psi \wedge \chi$, where the values of variables not in V are considered irrelevant. Similarly, indifference between $\psi \wedge \neg\chi$ and $\neg\psi \wedge \chi$ given ψ is indicated by $\varphi\colon \psi \sim \chi[V]$. The set of preference desires is denoted by $\mathcal{D}_P$, the set of indifference desires by $\mathcal{D}_I$, and a *ceteris paribus* goalbase $G = \mathcal{D}_P \cup \mathcal{D}_I$. A single desire $D = \varphi\colon \psi > \chi[V]$ induces a preference order $>_D$ where for models M, M', $M >_D M'$ iff $M \models \varphi \wedge \psi \wedge \neg\chi$, $M' \models \varphi \wedge \neg\psi \wedge \chi$, and M, M' agree on $\mathcal{PS} \setminus V$; the same conditions hold for indifference desires and $\sim_D$. The ordering $\succeq^{\mathrm{cp}}_G$ is defined so that $M \succeq^{\mathrm{cp}}_G M'$ iff there is a finite chain of models $M = M_0, M_1, \ldots, M_{k-1}, M_k = M'$ such that for each $0 \leq i < k$, there is some desire $D \in G$ for which $M_i >_D M_{i+1}$ or $M_i \sim_D M_{i+1}$.

Lang [2004] additionally defines R_{basic}, $R_{\subseteq}$, R_{card}, R_{wg}, and R_{d}:

$\boldsymbol{R_{\mathbf{basic}}}$ simply distinguishes states which satisfy a single goal from states which do not: $G = \{\varphi\}$ and models M, M' are such that $M \succ^{\mathrm{basic}}_G M$ iff $M \models \varphi$ and $M' \not\models \varphi$. Hence R_{basic} is very limited, permitting the representation of dichotomous preferences only.

$\boldsymbol{R_{\subseteq}}$ refines R_{basic}: $G = \{\varphi_1, \ldots, \varphi_k\}$ and models M, M' are such that $M \succeq^{\subseteq}_G M'$ iff $\{\varphi_i \in G \mid M \models \varphi_i\} \supseteq \{\varphi_i \in G \mid M' \models \varphi_i\}$. $R_{\subseteq}$ is the Pareto ordering on states; $\succ^{\subseteq}_G$-maximal states are ones where no further goals may be satisfied.

$\boldsymbol{R_{\mathbf{card}}}$ additionally counts the satisfied goals, but otherwise treats them interchangeably: $G = \{\varphi_1, \ldots, \varphi_k\}$ and models M, M' are such that $M \succeq^{\mathrm{card}}_G M'$ iff

$|\{\varphi_i \in G \mid M \models \varphi_i\}| \geq |\{\varphi_i \in G \mid M' \models \varphi_i\}|$. Hence, R_{card} produces a total order, while $R_{\subseteq}$ will in many cases be partial.

$\boldsymbol{R_{\text{wg}}}$ is a further generalization of weighted goal languages to three aggregators, F_2 which aggregates the weights of satisfied goals, F_3 which aggregates the weights of unsatisfied goals, and F_1 which aggregates the outputs of F_2 and F_3. This gives us

$$u_G^{F_1,F_2,F_3}(X) = F_1(F_2(\alpha \mid (\varphi,\alpha) \in G, X \models \varphi), F_3(\alpha \mid (\varphi,\alpha) \in G, X \not\models \varphi))$$

in symbols.[5] When considered this way, when F_1 is the projection function for its second argument ($\forall \alpha, \beta,\ F_1(\alpha,\beta) = \beta$), F_2 is arbitrary, and $F_3 = +$, we get penalty logic; when F_1 is the projection function for its first argument, $F_2 \in \{+, \max\}$, and F_3 is arbitrary, we get the sum- and max-aggregated goalbase languages considered in the present work. The induced ordering $\succeq_G^{\text{wg}}$ is simply the natural ordering on the utilities of the states as determined by G.

$\boldsymbol{R_{\text{d}}}$ The relation R_{d} is defined similarly to the distance-based measure given by Lafage and Lang [2000]. Let d be a pseudo-distance metric. (The difference between a distance metric and a pseudo-distance metric is that a pseudo-distance metric need not respect the triangle inequality.) The distance between two models $d(M, M')$ is as above, as is the distance between a model and a formula $d(M, \varphi)$. Rather than computing disutilities as Lafage and Lang [2000] do, however, Lang computes utilities—given a set of formulas G, the utility of a model M is $-d(M, G)$—and defines $\succeq_{G,d}^{\text{d}}$ as the natural ordering of models M induced by $-d(M, G)$.

Note that for both R_{wg} and R_{d}, a cardinal preference structure underlies the ordinal one.

These orderings show the great diversity of ways in which ordinal preferences may be represented, and especially the versatility of goalbases for inducing orders. The investigations which we carry out in Chapters 3–5 regarding the expressivity, succinctness, and complexity of goalbase languages for representing cardinal utility could be repeated for any of the ordinal languages mentioned here.

[5] This differs in two ways from [Lang, 2004, Section 3.3.2], which has

$$u_G^{F_1,F_2,F_3}(X) = F_1(F_2(\{\alpha \mid (\varphi,\alpha) \in G, X \models \varphi\}, F_3(\{\alpha \mid (\varphi,\alpha) \in G, X \not\models \varphi\})))$$

instead. In the main text, we do not collect weights as sets, in order to prevent the unwanted disappearance of duplicate weights. (For a discussion of this, see also p. 12, footnote 2.) The second difference is subtle: This way, F_2 has the value of F_3 as one of its arguments, whereas in our main text F_2 and F_3 are computed independently. Likewise, we could make the output of F_2 an input for F_3 if we wanted the aggregated value of the unsatisfied formulas to depend on the aggregated value of the satisfied formulas. Ultimately this difference matters only if we wish to take both satisfied and unsatisfied goals into account simultaneously.

2.3.5 Weighted Description Logics

Ragone, Noia, Donini, Sciascio, and Wellman [2009a,b] introduce a framework similar to our own, except that formulas from description logic are used instead of those from propositional logic. Description logics are knowledge representation languages which provide ways to reason about concepts, roles, and individuals. Concepts are sets of objects, individuals are particular named objects, and roles are relations among concepts. E.g., `Sandwich` is a concept, `joelsLunch` is an individual, and `hasCheese` is a role. Description logics provide operators for intersection and union of concepts: $\texttt{Sandwich} \sqcap \texttt{Soup}$ is the (presumably empty) concept which contains all objects which are both soups and sandwiches at the same time, while $\texttt{Sandwich} \sqcup \texttt{Soup}$ is the concept containing anything which is either a soup or a sandwich. Role restrictions may be quantified:

$$\texttt{Sandwich} \sqcap \exists \texttt{hasCheese}.\top$$

is the concept of a sandwich with cheese, and similarly,

$$\texttt{Sandwich} \sqcap \exists \texttt{hasCheese}.\top \sqcap \forall \texttt{hasCheese}.\texttt{Münster}$$

is the concept containing the sandwiches with Münster on them. One concept may subsume another: $\texttt{BlueCheese} \sqsubseteq \texttt{Cheese}$. An ontology $\mathcal{T}$ is a set of description logic formulas indicating how concepts are related. (For more information on description logics, see [Baader, Calvanese, McGuinness, Nardi, and Patel-Schneider, 2007].)

Ragone et al. form preference sets $\mathcal{P}$ of weighted concepts $\langle P, v\rangle$, where the value of a model A is

$$\sum \{v \mid \langle P, v\rangle \in \mathcal{P} \text{ and } A \sqsubseteq_{\mathcal{T}} P\}.$$

That is, every weighted concept which subsumes the concept represented by the model A (according to the ontology $\mathcal{T}$) contributes its weight to the overall utility.

Ragone et al. [2009a, Theorem 2] consider a problem analogous to our MINUTIL decision problem (*cf.* Definition 5.3.2 and Section 5.5), and determine that the complexity of finding minimal models is the same as the complexity of deciding satisfiability for the particular description logic in use. (In some cases, the complexity will go far beyond anything we consider here, since there are description logics for which SAT is, e.g., PSPACE-complete.)

2.3.6 Boolean Games

A Boolean game is one in which each player has a goal, specified as a propositional formula, and a subset of $\mathcal{PS}$ over which he has exclusive control. (For example, a player might have $p \vee (q \wedge r)$ as his goal, but control the values of the variables q and s.) A player receives a payoff of 1 for satisfying his goal, and 0 otherwise.

Bonzon, Lagasquie-Schiex, Lang, and Zanuttini [2009] propose an extension called L-Boolean games, in which the payoff for each player is determined by some compact preference representation language L; Bonzon et al. consider CP-nets and prioritized goalbases using the discrimin, leximin, and best-out relations as candidates for L. Independently, Mavronicolas, Monien, and Wagner [2007] extend Boolean games to use weighted formulas; and Dunne, van der Hoek, Kraus, and Wooldridge [2008] extend Boolean games to a cooperative setting.

2.3.7 Valued Constraint Satisfaction Problems

Related to penalty logic and other propositional ordinal representations are *valued constraint satisfaction problems* (VCSPs). A constraint satisfaction problem (CSP) is a tuple $\langle V, D, C\rangle$, where $V = \{x_1, \ldots, x_n\}$ is a set of variables, $D = \{d_1, \ldots, d_n\}$ a set of domains of values for the variables, and C a set of constraints. A constraint c is a pair $\langle V_c, R_c\rangle$, where $V_c \subseteq V$ and $R_c \subseteq \prod_{x_i \in V_c} d_i$. A *valuation* function $v\colon V \to \bigcup D$ maps each variable x_i to an element of its domain d_i. A constraint $\langle\{x_1, \ldots, x_k\}, R\rangle$ is satisfied by a valuation v when $(v(x_1), \ldots, v(x_k)) \in R$. (A CSP where all variables have binary domains amounts to specifying constraints as cubes.) Interesting CSPs are ones which are overconstrained, and in such cases it is NP-hard to decide whether any given subset of constraints is satisfiable.

As defined by Bistarelli, Montanari, Rossi, Schiex, Verfaillie, and Fargier [1999], a VCSP is a CSP augmented with a *valuation structure* $\langle E, \circledast, \succ\rangle$, where $\succ$ totally orders the set E, there is a $\succ$-minimum element $\top \in E$ and a $\succ$-maximum element $\bot \in E$, and $\circledast$ is an associative, commutative binary operator on E which satisfies identity ($\forall a \in E, a \circledast \bot = a$), monotonicity ($\forall a, b, c \in E, a \succeq b$ implies $(a \circledast c) \succeq (b \circledast c)$), and has an absorbing element ($\forall a \in E, a \circledast \top = \top$). That is, the valuation structure is a totally ordered commutative monoid with a monotonic operator. Each constraint c is then labeled with an element of E, which indicates the importance of violating c.

So, a VCSP $\mathcal{P} = \langle V, D, C, S, \varphi\rangle$, where $S = \langle E, \circledast, \succ\rangle$ is a valuation structure and the function $\varphi\colon C \to E$ maps constraints to their valuations. The overall valuation $\mathcal{V}$ of the VCSP $\mathcal{P}$ given an assignment A for some subset of variables $W \subseteq V$ is

$$\mathcal{V}_{\mathcal{P}}(A) = \bigcircledast_{\substack{c \in C,\, V_c \subseteq W \\ A \text{ violates } c}} \varphi(c).$$

Because $\circledast$ is an operator on E, $\mathcal{V}_{\mathcal{P}}(A)$ will also be some element of E. Since $\succ$ totally orders E, $\mathcal{V}_{\mathcal{P}}$ induces a total ordering on the allocations A. The valuation $\mathcal{V}_{\mathcal{P}}(A)$ indicates the overall quality of the allocation A according to $\succ$.

VCSPs subsume penalty goalbases restricted to cubes (let $E = \mathbb{N} \cup \{+\infty\}$, $\circledast = +$, $\bot = 0$, $\top = +\infty$, and $\succ = >$) as well as leximin goalbases restricted to cubes (let $E = \{0,1\}^* \cup \{\top\}$, $\circledast = \cup$, $\bot = \emptyset$, $\top =$ the symbol $\top$, and $\succ =$ the lexicographical ordering on binary strings).

2.3.8 Generalized Additive Independence

A utility function u over $\mathcal{PS}$ is *generalized additive decomposable* over a given collection of subsets $P_1, \ldots, P_k$ which covers $\mathcal{PS}$ if there are $u_i \colon 2^{P_i} \to \mathbb{R}$ such that for all states $X \subseteq \mathcal{PS}$,

$$u(X) = \sum_{i=1}^{k} u_i(X \cap P_i).$$

Clearly, every u is GA-decomposable over the trivial cover $P_1 = \mathcal{PS}$—just let $u_1 = u$. When u is GA-decomposable over the singleton cover $\{p_1\}, \ldots, \{p_n\}$, then u is *additive*. More interesting cases are where a utility function may be decomposed into the sums of utility functions over several (possibly overlapping) nonsingleton subsets of $\mathcal{PS}$. For example, suppose that $\mathcal{PS} = \{a, b, c, d\}$. Then the following complexly-structured utility function u is GA-decomposable over $\{a, b\}, \{a, c, d\}, \{b\}$ using the following three utility functions

$$u_{\{a,b\}}(X) = \begin{cases} 3 & \text{if } X = \{a, b\} \\ 1 & \text{if } X = \{a\} \\ 0 & \text{otherwise} \end{cases} \qquad u_{\{a,c,d\}}(X) = \begin{cases} 1 & \text{if } X = \{a, c, d\} \\ 0 & \text{otherwise} \end{cases}$$

$$u_{\{b\}}(X) = \begin{cases} -2 & \text{if } X = \{b\} \\ 2 & \text{otherwise} \end{cases}$$

which together sum to the value of u:

	$u_{\{a,b\}}$	$u_{\{a,c,d\}}$	$u_{\{b\}}$			$u_{\{a,b\}}$	$u_{\{a,c,d\}}$	$u_{\{b\}}$	
$u(\emptyset)$	$= 0$	$+ 0$	$+ 2$	$= 2$	$u(\{d\})$	$= 0$	$+ 0$	$+ 2$	$= 2$
$u(\{a\})$	$= 1$	$+ 0$	$+ 2$	$= 3$	$u(\{a, d\})$	$= 1$	$+ 0$	$+ 2$	$= 3$
$u(\{b\})$	$= 0$	$+ 0$	$- 2$	$= -2$	$u(\{b, d\})$	$= 0$	$+ 0$	$- 2$	$= -2$
$u(\{a, b\})$	$= 3$	$+ 0$	$- 2$	$= 1$	$u(\{a, b, d\})$	$= 3$	$+ 0$	$- 2$	$= 1$
$u(\{c\})$	$= 0$	$+ 0$	$+ 2$	$= 2$	$u(\{c, d\})$	$= 0$	$+ 0$	$+ 2$	$= 2$
$u(\{a, c\})$	$= 1$	$+ 0$	$+ 2$	$= 3$	$u(\{a, c, d\})$	$= 1$	$+ 1$	$+ 2$	$= 4$
$u(\{b, c\})$	$= 0$	$+ 0$	$- 2$	$= -2$	$u(\{b, c, d\})$	$= 0$	$+ 0$	$- 2$	$= -2$
$u(\{a, b, c\})$	$= 3$	$+ 0$	$- 2$	$= 1$	$u(\{a, b, c, d\})$	$= 3$	$+ 1$	$- 2$	$= 2$

Here, decomposing u into $u_{\{a,b\}}$, $u_{\{a,c,d\}}$, and $u_{\{b\}}$ reveals some structure in u which is not apparent on the surface, and also provides some space savings over the explicit representation of u. Note, however, that the utility functions which form a GA-decomposition may still be arbitrarily complex over their restricted domains. GA-decomposition may also be done for utility functions over variables with more than just binary domains, though this requires a more general definition than we have given here; for that, see [Gonzales, Perny, and Queiroz, 2006, Definition 1].

GA-decomposition was introduced by Fishburn [1970] and has more recently been used by Gonzales and Perny [2004] and Gonzales et al. [2006] for constructing GAI-nets, and by Brafman, Domshlak, and Kogan [2004] for eliciting preferences using GA-decomposable CP- and TCP-nets; related notions are the conditional additive independence of Bacchus and Grove [1995] and conditional expected utility independence of La Mura and Shoham [1999].

All utility functions representable in $\mathcal{L}(\mathit{forms}, \mathbb{R}, \Sigma)$ have a natural (though possibly suboptimal) GA-decomposition, namely the ones suggested by the goalbases which generates it: Given a goalbase G, the utility function $u_{G,\Sigma}$ is GA-decomposable over $\mathrm{Var}(\varphi_1), \ldots, \mathrm{Var}(\varphi_k)$ for $(\varphi_i, w_i) \in G$, using the $u_{\{(\varphi_i,w_i)\},\Sigma}$ as the component utility functions.

2.3.9 Coalitional Games

A coalitional game is one in which a group of agents receives some payoff for joint action. The payoff received depends on which agents join the coalition. Formally, a coalitional game with transferable utility $\langle N, v\rangle$ is a set of agents N and a valuation function $v\colon S \subseteq N \to \mathbb{R}$ which indicates the value of any coalition S to its members. The game specifies only how much utility a coalition receives, not how its members should divide it; this is what distinguishes a coalitional game with transferable utility from one without.

Ieong and Shoham [2005] introduce marginal contribution nets (MC-nets) as a way of modeling coalitional games with transferable utility. An MC-net is a set of rules of the form $\varphi \to w$, where φ is a cube and $w \in \mathbb{R}$. A rule $\varphi \to w$ is said to apply to a coalition S iff all of the positive literals in φ and none of the negative literals in φ are members of S; the value of a coalition S is the sum of weights of all rules which apply to S. It is easy to see that the language of MC-nets is exactly $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma)$ in disguise. Elkind, Goldberg, Goldberg, and Wooldridge [2009] further generalize basic MC-nets to general MC-nets, by additionally permitting arbitrary Boolean connectives in their rules. See Section 4.3 for further discussion of MC-nets.

2.3.10 Bidding Languages

Auctions are a method of allocating items and costs to bidders. Bidders in auctions need some way of expressing their valuations to the auctioneer; the method by which they do this is called a bidding language. Any bidding language may be thought of as a scheme for representing cardinal preferences over sets of goods. Bidding languages for traditional single-item auctions tend to be simple, as in single-item auctions it is only possible to express preferences which are modular. (When the left shoe is auctioned separately from the right shoe in a single-item auction, there is no way to tell the auctioneer *through your bids* that the value you place on one shoe depends on whether you win the auction for the other shoe.)

Combinatorial auctions are a type of auction in which all items are sold simultaneously, rather than sequentially as in traditional auctions. (For a discussion of combinatorial auctions, see Section 6.2.) Because combinatorial auctions simultaneously auction many items, the possibility arises for bidders to express preferences over bundles of items. The simplest bidding language, in which a bidder lists every possible combination of goods along with the price he is willing to pay for each one, clearly permits the full range of expression (limited only by the divisibility of the currency being used) but is too verbose to be used for any but the smallest auctions. A bidder wishing to bid in a ten-item combinatorial auction would need to list his price for 1023 bundles (the $2^{10} - 1$ nonempty subsets of ten items), which is surely beyond the desire, if not the capacity of any human bidder; and an auction with a hundred items would overwhelm even a computerized bidder were it forced to place explicit bids.

We might try to improve the explicit form somewhat by adopting the convention that any bundle which has no listed value is assumed to be worth nothing to the bidder; however, this will be cold comfort for bidders who place a nonzero value for every single-item bundle, as it will save them no effort at all. Clearly, we need a less naïve approach, one which saves space by taking advantage of the internal structure of bidders' preferences. Rather than assuming that the value of an unlisted bundle is zero, we might instead assume that it has the value of its greatest-valued subset. This bidding language, known as the XOR language, is demonstrably better than the explicit form. Further space efficiency may be gained by assigning not the value of its single highest-valued subset to a bundle, but rather by taking the greatest sum of values of subsets which partition it. This language, known as the OR language, is even more space-efficient than the XOR language, but cannot express all utility functions, and moreover computing with it is more difficult than with the XOR language. Further variations of these languages have been studied—for example, the OR* language, which is the OR language with dummy items, and the OR-of-XORs language, which permits XOR bids to be ORed together—and are discussed in detail by Nisan [2006]. We discuss the family of OR/XOR languages further in Section 6.3.1 and examine their succinctness with respect to our own goalbase languages in Section 6.3.3.

In addition to the OR/XOR family of bidding languages, some logic-based bidding languages have been proposed. Hoos and Boutilier [2000] introduce what they call *CNF bids*, which are weighted positive formulas in conjunctive normal form, as well as *extended CNF bids*, where a *k-of* operator is introduced into the language. (The formula k-of(S) is satisfied by any subset $S' \subseteq S$ where $|S'| \geq k$.) Hoos and Boutilier's CNF bidding language is $\mathcal{L}(\mathit{CNF}, \mathbb{R}^+, \Sigma)$ in our terms.

Boutilier and Hoos [2001] introduce *generalized logical bids* (GLBs), formed as follows: $\langle p, w\rangle$ is a bid for any good $p \in \mathcal{PS}$ and weight $w \in \mathbb{R}^+ \cup \{0\}$, and if b_1 and b_2 are bids, then $\langle b_1 \wedge b_2, w\rangle$, $\langle b_1 \vee b_2, w\rangle$, and $\langle b_1 \oplus b_2, w\rangle$ are bids also. Let $\Phi(b)$ be the formula formed by stripping all weights from a bid b. A bid will be satisfied or not, depending on the allocation of items. Satisfaction conditions are

defined recursively:

$$\sigma(\Phi(p), A) = \begin{cases} 1 & \text{if } A \text{ allocates } p \text{ to the bidder} \\ 0 & \text{otherwise} \end{cases}$$
$$\sigma(\Phi(\varphi \wedge \psi), A) = \min(\sigma(\varphi, A), \sigma(\psi, A))$$
$$\sigma(\Phi(\varphi \vee \psi), A) = \max(\sigma(\varphi, A), \sigma(\psi, A))$$
$$\sigma(\Phi(\varphi \oplus \psi), A) = \max(\sigma(\varphi, A), \sigma(\psi, A))$$

The value of a bid given an allocation is also defined recursively: Let $\Psi(b, A)$ be the value of bid b given allocation A. Then:

$$\Psi(\langle p, w\rangle) = w \cdot \sigma(p, A),$$
$$\Psi(\langle b_1 \wedge b_2, w\rangle) = \Psi(b_1, A) + \Psi(b_2, A) + w \cdot \sigma(\Phi(b_1) \wedge \Phi(b_2), A),$$
$$\Psi(\langle b_1 \vee b_2, w\rangle) = \Psi(b_1, A) + \Psi(b_2, A) + w \cdot \sigma(\Phi(b_1) \vee \Phi(b_2), A),$$
$$\Psi(\langle b_1 \oplus b_2, w\rangle) = \max(\Psi(b_1, A), \Psi(b_2, A)) + w \cdot \sigma(\Phi(b_1) \vee \Phi(b_2), A).$$

The $\vee$ and $\oplus$ connective differ not in their truth conditions, but in how they combine the values of "inner" bids, which is why Boutilier and Hoos call $\oplus$ "valuative XOR" (VXOR). The GLB language does not contain a logical XOR connective, or any other nonmonotone connective.

To see how GLBs work, consider the bid $\langle\langle a, 1\rangle \wedge \langle b, 1\rangle, 2\rangle$. This bid expresses that the bundles $\{a\}$ and $\{b\}$ are worth 1, while the bundle $\{a, b\}$ is worth 4 (1 each for a and b, plus an extra 2 for their combination). Similarly, $\langle\langle a, 1\rangle \vee \langle b, 1\rangle, 2\rangle$ gives 3 for $\{a\}$ and $\{b\}$, and 4 for both, while $\langle\langle a, 1\rangle \oplus \langle b, 1\rangle, 2\rangle$ gives 3 for each of $\{a\}$, $\{b\}$, and $\{a, b\}$.

For a further discussion of auctions and bidding languages, see Sections 6.2 and 6.3.

Part I

Theory

Chapter 3

Expressivity

3.1 Introduction

An important feature of any preference representation language is the range of preferences which can be represented in it. This available range is known as the *expressivity* of the language, and this chapter is devoted to determining the expressivity of numerous goalbase languages.

From the point of view of the theorist, we want to know how expressive goalbase languages are, both for their own sake and because these expressivity results are necessary for proving succinctness results in Chapter 4. As a user of goalbase languages, knowing the expressivity of goalbase languages can help us choose the language which is best suited for our application. Can our language of choice represent all functions belonging to a given class of utility functions which interests us? Not all languages are equally expressive and not all applications require full expressivity. Excess expressivity is often undesirable, because highly expressive languages tend to be computationally more demanding to reason about.

We are interested in *correspondence results* between languages and classes of functions. For instance, a very simple result which we present shows that the language we obtain by restricting formulas to *literals* and by using summation to aggregate weights can express all *modular* utility functions, and only those.

An interesting property closely related to expressivity is *uniqueness* of representation. A language has the uniqueness property with respect to a given class of utility functions if it has no more than one way of representing any function from that class. This is an interesting property because it suggests that the language in question is parsimonious in its expressivity. Syntactically rich languages often lack the uniqueness property. Non-uniqueness may be considered wasteful or useful, depending on the intended application—but from a theoretical point of view uniqueness is a property which greatly simplifies proofs (especially those in Chapter 4).

This chapter is divided into two main parts, the first covering the expressivity of languages using the sum aggregator, the second covering those using the max aggregator. Within each section, we present some useful goalbase transformations, followed by the expressivity results themselves. Most of the expressivity results are correspondences between goalbase languages and classes of utility functions. Our expressivity results for sum languages are summarized in Section 3.4.4 and Figure 3.1, and in Section 3.5.4 and Figure 3.2 for max languages.

3.2 Preliminaries

We note here some properties of utility functions to which we make frequent reference:

Definition 3.2.1 (Properties of Utility Functions)**.** Suppose that u is a utility function. Then:

- u is *normalized* iff $u(\emptyset) = 0$.
- u is *nonnegative* iff $u(X) \geq 0$ for all X.
- u is *monotone* iff $u(X) \geq u(Y)$ for all $X \supseteq Y$.
- u is *submodular* iff $u(X \cup Y) \leq u(X) + u(Y) - u(X \cap Y)$ for all $X, Y \subseteq \mathcal{PS}$.
- u is *supermodular* iff $u(X \cup Y) \geq u(X) + u(Y) - u(X \cap Y)$ for all $X, Y \subseteq \mathcal{PS}$.
- u is *modular* iff u is both sub- and supermodular.
- u is *subadditive* iff $u(X \cup Y) \leq u(X) + u(Y)$ for all $X \cap Y = \emptyset$, $X, Y \subseteq \mathcal{PS}$.
- u is *superadditive* iff $u(X \cup Y) \geq u(X) + u(Y)$ for all $X \cap Y = \emptyset$, $X, Y \subseteq \mathcal{PS}$.
- u is *additive* iff u is both sub- and superadditive.
- u is a *unit-demand* valuation iff $u(X) = \max_{a \in X} u(\{a\})$ and u is normalized.
- u is a *simple unit-demand* valuation iff $u(X) = 1$ for all $X \neq \emptyset$ and u is normalized.

For further examples of utility function properties, see [Nisan, 2006] and [Lehmann, Lehmann, and Nisan, 2006a]. Note that modularity with normalization is equivalent to additivity, and that (super-) submodularity together with normalization implies (super-) subadditivity.

Additionally we, define the property k-additivity:

Definition 3.2.2. Let $\mathcal{PS}(k)$ be the set of all subsets of $\mathcal{PS}$ with at most k elements. A utility function u is *k-additive* if there exists a mapping $m\colon \mathcal{PS}(k) \to \mathbb{R}$ such that

$$u(X) = \sum_{\substack{Y \subseteq X \\ Y \in \mathcal{PS}(k)}} m(Y)$$

for each set $X \subseteq \mathcal{PS}$.

The k-additive functions play an important role in fuzzy measure theory [Grabisch, 1997] as well as in combinatorial auctions [Conitzer, Sandholm, and Santi, 2005] and distributed multiagent resource allocation [Chevaleyre, Endriss, Estivie, and Maudet, 2008a]. For example, if the variables in $\mathcal{PS}$ are used to model whether an agent owns certain resources, then k-additive utility functions naturally model situations where synergies among different resources are restricted to bundles of at most k elements.

The k-additivity property is a generalization of another property of utility functions, namely modularity. The 1-additive utility functions are exactly the modular ones: For a utility function to be modular, the value it assigns to each good must be independent of the bundle it appears in, and 1-additivity enforces just that. Modular utility functions are interesting precisely because they are very simple and have limited expressive power. As a result, they are frequently used in applications, e.g., in work on modeling negotiation between autonomous software agents [Rosenschein and Zlotkin, 1994].

Supermodularity (and its counterpart, submodularity) are widely used concepts in the economics literature [Moulin, 1988]. The traditional assumption in economics is that goods have decreasing marginal utility—I will be willing to pay less for the $(n+1)$th ice cream cone than the nth—which is to say that buyers' utility functions are submodular. Similarly, nonnegativity, normalization, and monotonicity are nearly universally assumed by economists. Unit-demand valuations are often encountered when dealing with items where only a single one is necessary to fulfil its purpose. (For example, the typical person has unit-demand valuations for items like stoves and mobile phones.)

Goalbases may sometimes contain formulas which fail to contribute value to the goalbase in any state. We call these formulas *superfluous*.

Definition 3.2.3 (Superfluity)**.** A weighted formula $(\varphi, w) \in G$ is *superfluous* under aggregator F if $G \equiv_F G \setminus \{(\varphi, w)\}$.

It is easy to see that for sum languages, the only superfluous formulas are those equivalent to $\bot$ or having zero weight; for every other formula, there would be at least one state which would change in value if it were removed. For max languages, the picture is more complex: While contradictions are still superfluous, it is possible for satisfiable formulas with nonzero weight to be superfluous and for formulas with zero weight not to be. We take up the issue of superfluity for max languages in detail in Section 3.5.1.

Some goalbase languages have a unique way of expressing a given utility function, while others allow for several alternative representations. Here we make this notion of uniqueness precise:

Definition 3.2.4 (Unique Representations)**.** A utility function u is *represented* in a language $\mathcal{L}$ if there exists a goalbase $G \in \mathcal{L}$ such that $u = u_G$. A utility function u is *uniquely represented* (modulo formula equivalence) in a language $\mathcal{L}$ if, given a set of formulas Φ containing one representative formula for each formula equivalence class in $\mathcal{L}$ (except $\bot$), there is a unique goalbase G such that $\mathrm{For}(G) \subseteq \Phi$, $u_G = u$, and G contains no superfluous formulas. A language $\mathcal{L}$ is said to have *unique representations* if every u represented in $\mathcal{L}$ is uniquely represented.

Any language which has unique representations can be thought of as minimal in the sense that any further restriction on permissible weighted formulas will lead to a reduction in expressivity. Effectively this means that the set of representatives of formula equivalence classes forms a minimal basis for the vector space in which the goalbases live. We discuss uniqueness for sum languages in Section 3.4.2, and for max languages in Theorem 4.5.3.

3.3 Related Work

In broad terms, the expressivity of languages has occupied mathematicians, and logicians in particular, for quite some time. The question of whether various figures can be constructed using only a compass and an unmarked straightedge was a particular favorite of the ancient Greeks, and is fundamentally a question of language expressivity. In this vein, Gauss' celebrated proof that a heptadecagon is constructable can be thought of as a positive expressivity result, while Wantzel's proof that arbitrary angles cannot be trisected shows that additional expressivity (e.g., substituting a marked ruler for an unmarked straightedge) is needed in order to admit such constructions. Expressivity has been a perennial issue in logic from early in the 20th century to the present day. "What can I say in this language?" is a natural question for logicians, and its pursuit has been fruitful. We can point to results concerning the expressive power of quantifier logics (e.g., that there is no first-order formula corresponding to finiteness) and the substantial literature on correspondences between formulas in modal logic and properties of classes of Kripke frames (Sahlqvist's Theorem being a particularly nice example of this [Chagrov and Zakharyaschev, 1997, Section 10.3]).

We now move closer to the topic at hand. The expressivity of some ordinal preference representation languages has been studied by Coste-Marquis et al. [2004, Theorem 1]. In particular, they examine the ability of eight logic-based languages to represent classes of preorders over finite sets of objects. All of the languages considered consist of sets of propositional formulas which are annotated with

additional information. This additional information may be a weight, a penalty, a priority, a distance, or a context, depending on the language. Boutilier et al. [2004] discuss CP-nets, a graph-based language for representing ordinal conditional preferences. Only consistent preference orders (and not even all of those) are representable using acyclic CP-nets, though more (including some intransitive ones) are representable if dependency cycles are allowed. Often CP-nets will produce only a preorder, with which many linear orders are compatible. For more about CP-nets, see Section 2.3.1.

Ieong and Shoham [2005, Proposition 1] prove that MC-nets are fully expressive, which is equivalent to one part of our Corollary 3.4.9. For a discussion of MC-nets, see Sections 2.3.9 and 4.3.

Nisan [2006] collects numerous results on the expressivity of the OR/XOR family of languages for representing utility functions. (These languages are discussed in detail in Section 6.3.1.) For example, the XOR language corresponds to the monotone utility functions, while the OR language represents all superadditive utility functions. Several succinctness results given there also entail that particular classes of utility functions are representable in certain OR/XOR languages.

Lehmann et al. [2006a] define a syntactic hierarchy of OR/XOR languages which they aim to characterize. The languages defined are OS (OR of singletons), XS (XOR of singletons), OXS (OR of XS), and XOS (XOR of OS); these are shown to have expressivity such that

$$\text{OXS} \subset \text{GS} \subset \text{SM} \subset \text{XOS} \subset \text{CF},$$

where GS is the class of gross substitutes valuations (valuations where the demand for an item does not decrease when the prices of other items increase), SM is the class of submodular valuations, and CF is the class of complement-free valuations ($u(X) + u(Y) \geq u(X \cup Y)$). Though they do not further characterize these languages, Lehmann et al. do show that there is a large gap between GS and SM by demonstrating that each m-item subclass of GS has measure zero in $(2^m - 1)$-dimensional Euclidean space, while the corresponding m-item subclass of SM has positive measure—which implies that XOS is a much more expressive language than OXS.

3.4 Expressivity of Sum Languages

Throughout this section, when the aggregator function is omitted from the notation, it is indented to be Σ.

3.4.1 Goalbase Equivalences

Recall (from Definition 2.2.8) that two goalbases G and G' are equivalent ($G \equiv G'$) iff they generate the same utility function (i.e., $u_G = u_{G'}$). The following

equivalences are, for convenience, stated as they are used later in this section, and not necessarily in their most general form.

Fact 3.4.1. *Given a goalbase G, formulas $\varphi, \varphi_1, \ldots, \varphi_k, \psi, \chi$, and weight $w \in \mathbb{R}$, the following equivalences hold:*

$$G \cup \{(\varphi \wedge \neg\psi, w)\} \equiv G \cup \{(\varphi, w), (\varphi \wedge \psi, -w)\} \tag{3.1}$$
$$G \cup \{(\varphi \vee \neg\psi, w)\} \equiv G \cup \{(\top, w), (\varphi, w), (\varphi \vee \psi, -w)\} \tag{3.2}$$
$$G \cup \{(\varphi \wedge (\psi \vee \chi), w)\} \equiv G \cup \{(\varphi \wedge \psi, w), (\varphi \wedge \chi, w), (\varphi \wedge \psi \wedge \chi, -w)\} \tag{3.3}$$
$$G \cup \{(\varphi \vee (\psi \wedge \chi), w)\} \equiv G \cup \{(\varphi \vee \psi, w), (\varphi \vee \chi, w), (\varphi \vee \psi \vee \chi, -w)\} \tag{3.4}$$
$$G \cup \{(\varphi_1 \wedge \ldots \wedge \varphi_k, w)\} \equiv G \cup \{(\neg\varphi_1 \vee \ldots \vee \neg\varphi_k, -w), (\psi, w), (\neg\psi, w)\} \tag{3.5}$$
$$G \cup \{(\varphi_1 \vee \ldots \vee \varphi_k, w)\} \equiv G \cup \{(\neg\varphi_1 \wedge \ldots \wedge \neg\varphi_k, -w), (\psi, w), (\neg\psi, w)\} \tag{3.6}$$
$$G \cup \{(\top, w)\} \equiv G \cup \{(\varphi, w), (\neg\varphi, w)\} \tag{3.7}$$

Each of these equivalences is easily verified by considering all possible combinations of truth values for φ, ψ, and χ.

3.4.2 Uniqueness

It is easy to see that many sum languages lack unique representations. Neither $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma)$ nor $\mathcal{L}(\mathit{clauses}, \mathbb{R}, \Sigma)$, for instance, have them: The two goalbases $\{(\top, 3), (p, 2)\}$ and $\{(p \wedge q, 5), (p \wedge \neg q, 5), (\neg p \wedge q, 3), (\neg p \wedge \neg q, 3)\}$ define the same utility function over $\mathcal{PS} = \{p, q\}$ in $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma)$, while the two goalbases $\{(\neg p, 1)\}$ and $\{(p \vee \neg p, 1), (p, -1)\}$ yield the same utility function in $\mathcal{L}(\mathit{clauses}, \mathbb{R}, \Sigma)$. However, two sum languages (and all of their sublanguages) do have unique representations, namely $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$ and $\mathcal{L}(\mathit{pclauses} + \top, \mathbb{R}, \Sigma)$, which we now set out to prove.

Theorem 3.4.2. *$\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$ has unique representations.*

Proof. Given any utility function u, the weight of each positive cube is uniquely determined: We must have $w_\top = u(\emptyset)$, because $\top$ is the only positive cube satisfied by the model $\emptyset$, and furthermore $w_{\bigwedge X} = u(X) - \sum_{Y \subset X} w_{\bigwedge Y}$ for any nonempty set X. □

The recursive definition of the weights given in the proof of Theorem 3.4.2 can be turned into a direct rule for computing weights by using the so-called *Möbius inversion* [Rota, 1964; Grabisch, 1997]:

$$w_{\bigwedge X} = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} \cdot u(Y). \tag{3.8}$$

As an example, consider the utility function

$$\begin{array}{llll} u(\emptyset) & = 0 & u(\{c\}) & = 2 \\ u(\{a\}) & = 1 & u(\{a,c\} & = 6 \\ u(\{b\}) & = 0 & u(\{b,c\} & = 0 \\ u(\{a,b\}) & = 0 & u(\{a,b,c\}) & = 7 \end{array}$$

over the three items a, b, c. We now compute the representation of u in $\mathcal{L}(pcubes, \mathbb{R}, \Sigma)$ using the Möbius inversion. The pattern formed by equation (3.8) is best seen by calculating the weight for $a \wedge b \wedge c$:

$$\begin{aligned} w_{a\wedge b\wedge c} = &- u(\emptyset) \\ &+ u(\{a\}) + u(\{b\}) + u(\{c\}) \\ &- u(\{a,b\}) - u(\{a,c\}) - u(\{b,c\}) \\ &+ u(\{a,b,c\}) \\ = &- 0 + 1 + 0 + 2 - 0 - 6 - 0 + 7 = 4 \end{aligned}$$

Because $|\{a, b, c\}| = 3$ is odd, the utility of odd-sized subsets of $\{a, b, c\}$ is added while the utility of even-sized subsets is subtracted. For the weights of even-length formulas, the opposite happens:

$$\begin{aligned} w_{a\wedge b} = &+ u(\emptyset) \\ &- u(\{a\}) - u(\{b\}) \\ &+ u(\{a,b\}) \\ = &+ 0 - 1 - 0 + 0 = -1 \end{aligned}$$

For the remaining weights, we have

$$\begin{aligned} w_{a\wedge c} &= + u(\emptyset) - u(\{a\}) - u(\{c\}) + u(\{a,c\}) = +0 - 1 - 2 + 6 = 3 \\ w_{b\wedge c} &= + u(\emptyset) - u(\{b\}) - u(\{c\}) + u(\{b,c\}) = +0 - 0 - 2 + 0 = -2 \\ w_a &= - u(\emptyset) + u(\{a\}) = -0 + 1 = 1 \\ w_b &= - u(\emptyset) + u(\{b\}) = -0 + 0 = 0 \\ w_c &= - u(\emptyset) + u(\{c\}) = -0 + 2 = 2 \\ w_{\bigwedge \emptyset} &= + u(\emptyset) = +0 = 0 \end{aligned}$$

which gives us the goalbase

$$\{(a, 1), (c, 2), (a \wedge b, -1), (a \wedge c, 3), (b \wedge c, -2), (a \wedge b \wedge c, 4)\}.$$

Note that we omit $\top$ and b, since $w_{\bigwedge \emptyset}$ and w_b turned out to be zero.

As can be seen from our example, the Möbuis inversion is not an efficient method of calculating weights, due not only to the fact that we have $2^{|\mathcal{PS}|}$ weights to calculate, but also because for each weight we must sum a number of terms exponential in the length of the formula being weighted.

Theorem 3.4.3. $\mathcal{L}(pclauses, \mathbb{R}, \Sigma)$ *has unique representations.*

Proof. Let u be any utility function represented by positive clauses with weights $w_{\bigvee X}$ (with nonempty sets $X \subseteq \mathcal{PS}$). Then for any $Y \subseteq \mathcal{PS}$, $u(Y)$ must be equal to the sum of the weights $w_{\bigvee X}$ for which X and Y have a nonempty intersection:

$$\begin{aligned} u(Y) &= \sum_{X \cap Y \neq \emptyset} w_{\bigvee X} \\ &= \sum_{\emptyset \subset X \subseteq \mathcal{PS}} w_{\bigvee X} - \sum_{\emptyset \subset X \subseteq \mathcal{PS} \setminus Y} w_{\bigvee X} \\ &= u(\mathcal{PS}) - \sum_{\emptyset \subset X \subseteq \mathcal{PS} \setminus Y} w_{\bigvee X} \end{aligned}$$

This shows that each weight $w_{\bigvee X}$ is uniquely determined: For singletons $X = \{p\}$, by setting $Y = \mathcal{PS} \setminus \{p\}$ in the above equation, we obtain $w_p = u(\mathcal{PS}) - u(\mathcal{PS} \setminus \{p\})$. For general sets X, using $Y = \mathcal{PS} \setminus X$, we then obtain

$$w_{\bigvee X} = u(\mathcal{PS}) - u(\mathcal{PS} \setminus X) - \sum_{\emptyset \subset X' \subset X} w_{\bigvee X'},$$

which completes the proof. □

By unraveling the recursive definition of the weights given in the proof above, we can also provide a direct rule for computing weights in $\mathcal{L}(pclauses, \mathbb{R}, \Sigma)$, similar to the Möbius inversion:

$$w_{\bigvee X} = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|+1} \cdot u(\mathcal{PS} \setminus Y) \tag{3.9}$$

Furthermore, we have the following corollary due to the fact that no positive clause is a tautology:

Corollary 3.4.4. $\mathcal{L}(pclauses + \top, \mathbb{R}, \Sigma)$ *has unique representations.*

Finally, we note that having unique representations is a property which is preserved in all sublanguages of any language having the property; hence, it follows that many other languages not explicitly mentioned in this section, e.g., $\mathcal{L}(atoms, \mathbb{R}^+, \Sigma)$, also have unique representations.

While we can use the Möbius inversion for demonstrating that $\mathcal{L}(pcubes, \mathbb{R}, \Sigma)$ has unique representations, and a similar construct for $\mathcal{L}(pclauses, \mathbb{R}, \Sigma)$, it is also possible to show this by application of a more general method. The problem of determining whether a utility function u has a unique representation in a given language $\mathcal{L}$ amounts to examining the system of linear equations which describes u in $\mathcal{L}$. A language $\mathcal{L}$ has $|\mathcal{L}/_{\equiv}| = m$ distinct nonequivalent formulas, and over a set of atoms $\mathcal{PS}$ there are $2^{|\mathcal{PS}|} = n$ states. Each state $i \in 2^{\mathcal{PS}}$ defines a constraint

$$a_{i1}w_1 + ... + a_{im}w_m = b_i$$

where $a_{ij} \in \{0, 1\}$, depending on whether formula j is true in state i; and $b_i = u(X_i)$, where X_i is the set of true atoms in state i. Taken together as matrices $\mathsf{Ax} = \mathsf{b}$, we have:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

That is, the w_i are the weights, the b_i are the values of the utility function, and the a_{ij} mark which formulas are true in which states. If $n = m$, i.e., if the number of formulas in the language over $\mathcal{PS}$ equals the number of states over $\mathcal{PS}$, then the matrix A will be square. This makes available to us a well-known fact from linear algebra, *viz.* that the determinant of the square matrix A is nonzero only when the system has a single, unique solution [Anton, 1994, Theorem 2.3.6].

Using this, we give an alternative proof of Theorem 3.4.3:

Proof. Suppose that G contains only positive clauses and generates u_n, where $n = |\mathcal{PS}|$. We can write one constraint for each state except $\emptyset$, so we have $2^n - 1$ constraints. (The constraint for $\emptyset$ can be omitted, since all positive clauses are false in that case.) We have the clause $\bigvee X$ for each nonempty $X \subseteq \mathcal{PS}$, so also $2^n - 1$ distinct nonequivalent clauses, and hence $2^n - 1$ variables for weights. The coefficient matrix formed by the states and formulas will be square ($2^n - 1$ rows and columns), so the strategy described above—proving that this matrix has nonzero determinant—is applicable.

Enumerate the positive clauses such that the index j codes for the positive clause $\varphi_j = \bigvee\{p_k \mid j \,\&\, 2^k \neq 0\}$, where '&' stands for *bitwise* conjunction. E.g., $\varphi_7 = p_0 \vee p_1 \vee p_2$, because $7 = 2^0 + 2^1 + 2^2$. Then, let $a_{ij} = 1$ if $i \,\&\, j \neq 0$, and $a_{ij} = 0$ otherwise. This sets $a_{ij} = 1$ iff clause j is true in state i. In other words, each row of A_n is a state, and the ones in a row mark the positive clauses which are true in that state.

Now observe that $\mathsf{A}_1 = \begin{bmatrix} 1 \end{bmatrix}$ and A_{n+1} is the block matrix

$$\mathsf{A}_{n+1} = \begin{bmatrix} & & & 0 & & & \\ & \mathsf{A}_n & & \vdots & & \mathsf{A}_n & \\ & & & 0 & & & \\ 0 & \ldots & 0 & 1 & 1 & \ldots & 1 \\ & & & 1 & & & \\ & \mathsf{A}_n & & \vdots & & \mathsf{1} & \\ & & & 1 & & & \end{bmatrix}$$

where $\mathsf{1}$ is a matrix of the appropriate size, with every element a 1. The additional rows in A_{n+1} (over A_n) are for states in which p_n, the new variable, is true. The middle row is the state where only p_n is true, and from there down p_n is true in

every state. With respect to the other variables, the states in the bottom half repeat the states in the top half. The additional columns in A_{n+1} are for positive clauses which contain p_n. The middle column is for p_n, the degenerate positive clause formed by that variable alone, and the columns thereafter repeat the first $2^{n-1} - 1$ columns with p_n as an additional disjunct. Therefore, the upper left and upper right blocks repeat A_n since no state there makes p_n true; the lower left block repeats A_n since no clause there contains p_n; and the lower right block is all ones because every state and clause there contains p_n.

Clearly, $\det(\mathsf{A}_1) = 1$. Suppose that $\det(\mathsf{A}_n) \neq 0$. To show that $\det(\mathsf{A}_{n+1}) \neq 0$, we will twice use the following fact about determinants of block matrices:

Fact 3.4.5. *For the block matrix* $\left[\begin{smallmatrix} \mathsf{A} & \mathsf{B} \\ \mathsf{C} & \mathsf{D} \end{smallmatrix}\right]$*,*

$$\det \begin{bmatrix} \mathsf{A} & \mathsf{B} \\ \mathsf{C} & \mathsf{D} \end{bmatrix} = \det(\mathsf{A}) \det(\mathsf{D} - \mathsf{C}\mathsf{A}^{-1}\mathsf{B})$$

where A *is* $m \times m$*,* B *is* $m \times n$*,* C *is* $n \times m$ *and* D *is* $n \times n$*.*

Slice A_{n+1} into blocks like so,

$$\mathsf{A} = \left[\begin{array}{ccc|ccccc} & \mathsf{A}_n & & 0 & & & & \\ & & & \vdots & & \mathsf{A}_n & & \\ & & & 0 & & & & \\ \hline 0 & \dots & 0 & 1 & 1 & \dots & 1 & \\ & & & 1 & & & & \\ & \mathsf{A}_n & & \vdots & & 1 & & \\ & & & 1 & & & & \end{array}\right] = \mathsf{B}$$

$$\mathsf{C} = \left[\begin{array}{ccc|ccccc} & \mathsf{A}_n & & 0 & & & & \\ & & & \vdots & & \mathsf{A}_n & & \\ & & & 0 & & & & \\ \hline 0 & \dots & 0 & 1 & 1 & \dots & 1 & \\ & & & 1 & & & & \\ & \mathsf{A}_n & & \vdots & & 1 & & \\ & & & 1 & & & & \end{array}\right] = \mathsf{D}$$

and note that $\mathsf{A} = \mathsf{A}_n = \mathsf{A}_n^{-1}$ (because the A_i are symmetric about their main diagonal). Further,

$$\mathsf{C}\mathsf{A}^{-1} = \begin{bmatrix} 0 & \dots & 0 \\ & \mathsf{A}_n & \end{bmatrix} \mathsf{A}^{-1} = \begin{bmatrix} 0 & \dots & 0 \\ & \mathsf{I} & \end{bmatrix}$$

$$\mathsf{C}\mathsf{A}^{-1}\mathsf{B} = \begin{bmatrix} 0 & \dots & 0 \\ & \mathsf{I} & \end{bmatrix} \begin{bmatrix} 0 & \\ \vdots & \mathsf{A}_n \\ 0 & \end{bmatrix} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \mathsf{A}_n & \\ 0 & & \end{bmatrix}$$

$$\mathsf{D} - \mathsf{C}\mathsf{A}^{-1}\mathsf{B} = \mathsf{1} - \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \mathsf{A}_n & \\ 0 & & \end{bmatrix} = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \mathsf{1} - \mathsf{A}_n & \\ 1 & & \end{bmatrix}$$

where I and $\mathsf{1}$ are an identity matrix and a matrix of ones, respectively, of the appropriate sizes.

Next, slice $\mathsf{D} - \mathsf{C}\mathsf{A}^{-1}\mathsf{B}$ into blocks like so,

$$\begin{array}{rl} \mathsf{A}' = & \left[\begin{array}{c|c} 1 & 1 \quad \ldots \quad 1 \end{array}\right] = \mathsf{B}' \\ \mathsf{C}' = & \left[\begin{array}{c|c} 1 & \\ \vdots & 1 - \mathsf{A}_n \\ 1 & \end{array}\right] = \mathsf{D}' \end{array}$$

and then by Fact 3.4.5, we have that

$$\begin{aligned} \det(\mathsf{D} - \mathsf{C}\mathsf{A}^{-1}\mathsf{B}) &= \det(\mathsf{A}') \det(\mathsf{D}' - \mathsf{C}'\mathsf{A}'^{-1}\mathsf{B}') \\ &= \det\begin{bmatrix}1\end{bmatrix} \det\left(1 - \mathsf{A}_n - \begin{bmatrix}1\\ \vdots \\ 1\end{bmatrix}\begin{bmatrix}1\end{bmatrix}\begin{bmatrix}1 & \ldots & 1\end{bmatrix}\right) \\ &= \det(1 - \mathsf{A}_n - 1) \\ &= \det(-\mathsf{A}_n) \\ &= -\det(\mathsf{A}_n). \end{aligned}$$

Applying Fact 3.4.5 a second time, we have that

$$\det(\mathsf{A}_{n+1}) = \det(\mathsf{A}_n) \det(\mathsf{D} - \mathsf{C}\mathsf{A}^{-1}\mathsf{B}) = -\det(\mathsf{A}_n)^2 = -1 \neq 0,$$

which completes the induction.

Having shown that $\det(\mathsf{A}_n)$ is nonzero, it follows that the system has exactly one solution. Therefore G is the unique positive-clause generator of u_n. □

The next proof demonstrates the generality of the method used in the previous proof. Here we use block matrices to show that $\mathcal{L}(\mathit{pcubes}\ {-\top}, \mathbb{R}, \Sigma)$ has unique representations. (This differs from Theorem 3.4.2 in that we omit $\top$ from the language, in order that this proof may parallel the previous one. If $\top$ were to be included, a similar proof could be carried out by adding to the A_n matrices a column for the formula $\top$ and a row for the state where all atoms are false, in order to keep these matrices square.)

Proof. Suppose that G contains only positive cubes and generates u_n. Proceed as above. Let the coefficient matrix A_n be such that

$$a_{ij} = \begin{cases} 1 & \text{if } i \mid j = i \\ 0 & \text{otherwise} \end{cases}$$

where '|' is bitwise disjunction. So $a_{ij} = 1$ iff the 'on' bits in j are a subset of the 'on' bits in i, which can be thought of as encoding the state description for constraint i, where each bit corresponds to one of the n atoms; j codes for the cube $\varphi_j = \bigwedge\{p_k \mid j \,\&\, 2^k = 1\}$.

Now observe that $\mathsf{A}_1 = \begin{bmatrix}1\end{bmatrix}$ and A_{n+1} is the block matrix

$$\mathsf{A}_{n+1} = \begin{bmatrix} & & & 0 & & & \\ & \mathsf{A}_n & & \vdots & & \mathsf{0} & \\ & & & 0 & & & \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ & & & 1 & & & \\ & \mathsf{A}_n & & \vdots & & \mathsf{A}_n & \\ & & & 1 & & & \end{bmatrix}$$

where $\mathsf{0}$ is a matrix of the appropriate size, with every element set to 0. Slice A_{n+1} into blocks like so,

$$\mathsf{A} = \left[\begin{array}{ccc|ccc} & & & 0 & & \\ & \mathsf{A}_n & & \vdots & \mathsf{0} & \\ & & & 0 & & \end{array}\right] = \mathsf{B}$$

$$\mathsf{C} = \left[\begin{array}{ccc|ccc} 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ & & & 1 & & & \\ & \mathsf{A}_n & & \vdots & & \mathsf{A}_n & \\ & & & 1 & & & \end{array}\right] = \mathsf{D}$$

noting that

$$\mathsf{C}\mathsf{A}^{-1}\mathsf{B} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \mathsf{0} & \\ 0 & & \end{bmatrix} = \mathsf{0}$$

because B is a zero matrix, and so $\det(\mathsf{D} - \mathsf{C}\mathsf{A}^{-1}\mathsf{B}) = \det(\mathsf{D} - \mathsf{0}) = \det(\mathsf{D})$.

Next, slice D into blocks like so:

$$\mathsf{A}' = \left[\begin{array}{c|ccc} 1 & 0 & \dots & 0 \end{array}\right] = \mathsf{B}'$$

$$\mathsf{C}' = \left[\begin{array}{c|ccc} 1 & & & \\ \vdots & & \mathsf{A}_n & \\ 1 & & & \end{array}\right] = \mathsf{D}'$$

Applying Fact 3.4.5 to D, we have

$$\begin{aligned} \det(\mathsf{D}) &= \det(\mathsf{A}')\det(\mathsf{D}' - \mathsf{C}'\mathsf{A}'^{-1}\mathsf{B}') \\ &= \det\begin{bmatrix}1\end{bmatrix} \det\left(\mathsf{A}_n - \begin{bmatrix}1\\ \vdots \\ 1\end{bmatrix}\begin{bmatrix}1\end{bmatrix}\begin{bmatrix}0 & \dots & 0\end{bmatrix}\right) \\ &= \det(\mathsf{A}_n - \mathsf{0}) \\ &= \det(\mathsf{A}_n). \end{aligned}$$

Applying Fact 3.4.5 a second time, we have that

$$\det(\mathsf{A}_{n+1}) = \det(\mathsf{A}_n)\det(\mathsf{D} - \mathsf{C}\mathsf{A}^{-1}\mathsf{B}) = \det(\mathsf{A}_n)\det(\mathsf{D}) = \det(\mathsf{A}_n)^2 = 1 \neq 0,$$

which completes the induction.

Having shown that $\det(\mathsf{A}_n)$ is nonzero, it follows that the system has exactly one solution. Therefore G is the unique positive-cube generator of u_n. □

This method can be used to investigate, for any language where the formulas and states give a square matrix, whether that language has the uniqueness property. Though calculating determinants appears much more involved, it is in fact easier to carry out than proving uniqueness by giving a formula for computing weights. Calculating the coefficient matrix is entirely mechanical and the number of different ways that it may be decomposed into blocks is finite and small; once armed with the general insight of how to proceed, it is not difficult to compute the needed determinants and resolve whether a language has unique representations. (Note that had $\det(\mathsf{A}_n) = 0$, that would have been sufficient to prove that the language in question lacked unique representations.) In contrast, finding a formula for weights is a matter of trial and error; while the result may be more pleasing, the process is less so.

3.4.3 Correspondences

We now address the following question: What class of utility functions can we model using a given language? As much as possible we will strive for exact characterization results that establish the correspondence between a natural goalbase language and a commonly used class of utility functions.

For the next proof, and indeed much of this chapter, we will make frequent use of the fact that whenever $\Phi \subseteq \Phi'$ and $W \subseteq W'$, then $\mathcal{U}(\Phi, W, F) \subseteq \mathcal{U}(\Phi', W', F)$.

Theorem 3.4.6. *Each of the following classes:*

- $\mathcal{U}(k\text{-}pcubes, \mathbb{R}, \Sigma)$
- $\mathcal{U}(k\text{-}clauses, \mathbb{R}, \Sigma)$
- $\mathcal{U}(k\text{-}cubes, \mathbb{R}, \Sigma)$
- $\mathcal{U}(k\text{-}pforms, \mathbb{R}, \Sigma)$
- $\mathcal{U}(k\text{-}pclauses\ +\top, \mathbb{R}, \Sigma)$
- $\mathcal{U}(k\text{-}forms, \mathbb{R}, \Sigma)$

is equal to the class of all k-additive utility functions.

Proof. Inspection of the definition of k-additivity shows that it is simply a notational variant of the language based on positive cubes of length $\leq k$: A k-additive function can be represented by a mapping $m\colon \mathcal{PS}_k \to \mathbb{R}$, and we can define a bijection f from such mappings m onto goalbases $G \in \mathcal{L}(k\text{-}pcubes, \mathbb{R}, \Sigma)$:

$$f\colon m \mapsto \{(p_1 \wedge \ldots \wedge p_k, \alpha) \mid m(\{p_1, \ldots, p_k\}) = \alpha\}.$$

Clearly $m = u_{f(m)}$. That is, $\mathcal{U}(k\text{-}pcubes, \mathbb{R}, \Sigma)$ is the class of all k-additive utility functions.

Next, we show that all six languages are expressively equivalent. By language inclusion, we have the following:

$$\begin{array}{ccccc} \mathcal{U}(k\text{-}cubes, \mathbb{R}, \Sigma) & \subseteq & \mathcal{U}(k\text{-}forms, \mathbb{R}, \Sigma) & \supseteq & \mathcal{U}(k\text{-}clauses, \mathbb{R}, \Sigma) \\ \cup\mathsf{I} & & \cup\mathsf{I} & & \cup\mathsf{I} \\ \mathcal{U}(k\text{-}pcubes, \mathbb{R}, \Sigma) & \subseteq & \mathcal{U}(k\text{-}pforms, \mathbb{R}, \Sigma) & \supseteq & \mathcal{U}(k\text{-}pclauses\ +\top, \mathbb{R}, \Sigma) \end{array}$$

Taken together, equivalences (3.1) and (3.3) from Fact 3.4.1 can be used to transform any goalbase in $\mathcal{L}(k\text{-}forms, \mathbb{R}, \Sigma)$ into an equivalent goalbase in $\mathcal{L}(k\text{-}pcubes, \mathbb{R}, \Sigma)$. (Use (3.3) to eliminate all disjunctions, then (3.1) to eliminate all negations. While these equivalences add more formulas, they never add *longer* formulas.) Thus, $\mathcal{U}(k\text{-}pcubes, \mathbb{R}, \Sigma) = \mathcal{U}(k\text{-}forms, \mathbb{R}, \Sigma)$, which collapses the leftmost four classes.

Equivalences (3.2) and (3.4) from Fact 3.4.1 can be used to transform any goalbase in $\mathcal{L}(k\text{-}forms, \mathbb{R}, \Sigma)$ into an equivalent goalbase in $\mathcal{L}(k\text{-}pclauses\ +\top, \mathbb{R}, \Sigma)$. (Use (3.4) to eliminate all conjunctions, then (3.2) to eliminate all negations.) Thus, $\mathcal{U}(k\text{-}pclauses, \mathbb{R}, \Sigma) = \mathcal{U}(k\text{-}forms, \mathbb{R}, \Sigma)$, which collapses the rightmost four classes.

This completes the proof, as all six classes have been collapsed together, and one of them has been shown to equal to the class of k-additive functions. □

The next lemma clarifies the effect that the ability to express tautologies in a language has on the class of utility functions that can be defined. Roughly speaking, any utility function u expressible in a language based on strictly positive formulas must be *normalized*, i.e., will satisfy $u(\emptyset) = 0$.

For the next proof, we make use of translations of utility functions. The (affine) translation function t_c shifts a utility function u such that $t_c(u(X)) = u(X) + c$ for all $X \subseteq \mathcal{PS}$. A property P is *invariant under translation* if, for all utility functions u, $c \in \mathbb{R}$, and $X \subseteq \mathcal{PS}$, u has property P iff $t_c(u)$ has property P.

Lemma 3.4.7. *Fix Φ as a strictly positive set of formulas and P a property of utility functions which is invariant under translation. Then $\mathcal{U}(\Phi, W, \Sigma)$ is the class of normalized utility functions with property P iff $\mathcal{U}(\Phi \cup \{\top\}, W, \Sigma)$ is the class of utility functions with property P.*

Proof. ($\Rightarrow$) Suppose that $\mathcal{U}(\Phi, W, \Sigma)$ is the class of normalized utility functions with property P. First, we show that every $u_G \in \mathcal{U}(\Phi \cup \{\top\}, W, \Sigma)$ has property P: Fix $u_G \in \mathcal{U}(\Phi \cup \{\top\}, W, \Sigma)$. $G \setminus \{(\top, w) \mid w \in W\} \in \mathcal{L}(\Phi, W, \Sigma)$ and so $u_{G\setminus\{(\top,w)|w\in W\}}$ has property P by hypothesis. $u_G = t_w\left(u_{G\setminus\{(\top,w)|w\in W\}}\right)$, so by invariance u_G has property P.

Next, we show that every u with property P is in $\mathcal{U}(\Phi \cup \{\top\}, W, \Sigma)$: Fix u with property P. The translation $t_{u(\emptyset)}(u)$ is normalized and has property P by

invariance, so $t_{u(\emptyset)}(u) \in \mathcal{U}(\Phi, W, \Sigma)$ by hypothesis. Let G represent $t_{u(\emptyset)}(u)$ in $\mathcal{L}(\Phi, W, \Sigma)$. Then $G \cup \{(\top, u(\emptyset))\} \in \mathcal{L}(\Phi \cup \{\top\}, W, \Sigma)$ and $u_{G\cup\{(\top,u(\emptyset))\}} = u$.

($\Leftarrow$) Suppose that $\mathcal{U}(\Phi \cup \{\top\}, W, \Sigma)$ is the class of utility functions with property P. First, we show that every $u_G \in \mathcal{U}(\Phi, W, \Sigma)$ is normalized and has property P: Fix $u_G \in \mathcal{U}(\Phi, W, \Sigma)$. Normalization follows due to Φ being strictly positive. $G \cup \{(\top, w)\} \in \mathcal{L}(\Phi \cup \{\top\}, W, \Sigma)$ for any $w \in W$, and by hypothesis $u_{G\cup\{(\top,w)\}}$ has property P. $u_{G\cup\{(\top,w)\}} = t_w(u_G)$, so by invariance has property P.

Next, we show that every normalized u with property P is in $\mathcal{U}(\Phi, W, \Sigma)$: Fix u normalized and with property P. For any $w \in W$, $t_w(u)$ has property P by invariance and so is in $\mathcal{U}(\Phi \cup \{\top\}, W, \Sigma)$. Let G represent $t_w(u)$ in $\mathcal{L}(\Phi \cup \{\top\}, W, \Sigma)$. Since Φ is strictly positive, $(\top, w) \in G$. Then $u_{G\setminus\{(\top,w)\}} = t_w^{-1}(t_w(u)) = u$, and so $u \in \mathcal{U}(\Phi \cup \{\top\}, W, \Sigma)$. □

Finally, it is easy to see that we have the following as a corollary:

Corollary 3.4.8. *Every utility function in $\mathcal{U}(\Phi, W, \Sigma)$ is normalized if Φ is strictly positive or $W = \{0\}$.*

Next we explore the class of k-additive utility functions for specific values of k. It is a well-known fact that *any* utility function is k-additive for some $k \in \mathbb{N}$ (certainly for $k = |\mathcal{PS}|$). (This is why we refer to general functions as ω-additive. See Chapter 2, note 1.) Our next result is therefore an immediate corollary of Theorem 3.4.6.

Corollary 3.4.9. *Each of the following classes:*

- $\mathcal{U}(\mathit{pcubes}, \mathbb{R}, \Sigma)$
- $\mathcal{U}(\mathit{clauses}, \mathbb{R}, \Sigma)$
- $\mathcal{U}(\mathit{cubes}, \mathbb{R}, \Sigma)$
- $\mathcal{U}(\mathit{pforms}, \mathbb{R}, \Sigma)$
- $\mathcal{U}(\mathit{pclauses} + \top, \mathbb{R}, \Sigma)$
- $\mathcal{U}(\mathit{forms}, \mathbb{R}, \Sigma)$

is equal to the class of all utility functions.

Corollary 3.4.10. *$\mathcal{U}(\mathit{literals}, \mathbb{R}, \Sigma)$ and $\mathcal{U}(\mathit{atoms} + \top, \mathbb{R}, \Sigma)$ are the class of all modular utility functions, and $\mathcal{U}(\mathit{atoms}, \mathbb{R}, \Sigma)$ is the class of all normalized modular utility functions.*

Proof. The set of 1-pcubes is equal to the set of atoms together with $\top$. Therefore, by Theorem 3.4.6, $\mathcal{U}(\mathit{atoms} + \top, \mathbb{R}, \Sigma)$ is the class of all modular functions. The class of 1-cubes is equal to the class of literals together with $\top$. But by equivalence (3.7) of Fact 3.4.1, literals alone have the same expressive power as literals together with $\top$. Hence, again by Theorem 3.4.6, $\mathcal{U}(\mathit{literals}, \mathbb{R}, \Sigma)$ is the class of all modular functions. The fact that $\mathcal{U}(\mathit{atoms}, \mathbb{R}, \Sigma)$ is the class of all normalized modular utility functions follows from Lemma 3.4.7. □

For the remainder of this section we consider languages where the set of weights is restricted to the positive reals. Clearly, the utility functions that can be so expressed will be *nonnegative*, i.e., $u(X) \geq 0$ for all $X \subseteq \mathcal{PS}$. The question is whether we can express *all* nonnegative utility functions in this manner.

Theorem 3.4.11. $\mathcal{U}(\mathit{cubes}, \mathbb{R}^+, \Sigma)$ *and* $\mathcal{U}(\mathit{forms}, \mathbb{R}^+, \Sigma)$ *are the class of all nonnegative utility functions.*

Proof. Clearly every $u \in \mathcal{U}(\mathit{forms}, \mathbb{R}^+, \Sigma)$, and by inclusion, also every $u \in \mathcal{U}(\mathit{cubes}, \mathbb{R}^+, \Sigma)$, is nonnegative. For the converse, suppose that u is nonnegative. Then define

$$G = \left\{ \left(\bigwedge M \cup \neg \bar{M}, u(M) \right) \;\middle|\; M \subseteq \mathcal{PS} \text{ and } u(M) \neq 0 \right\}$$

and observe that $u_G = u$ and that G contains only positively-weighted cubes. □

That is, general formulas as well as cubes are fully expressive over nonnegative utility functions when weights are required to be positive. As we shall see next, the same is not true for clauses.

Theorem 3.4.12. $\mathcal{U}(\mathit{clauses}, \mathbb{R}^+, \Sigma)$ *is a proper subset of all nonnegative utility functions.*

Proof. By definition, $\mathcal{L}(\mathit{clauses}, \mathbb{R}^+, \Sigma) \subset \mathcal{L}(\mathit{forms}, \mathbb{R}^+, \Sigma)$, so by Theorem 3.4.11, $\mathcal{U}(\mathit{clauses}, \mathbb{R}^+, \Sigma)$ contains only nonnegative utility functions. This utility function over $\mathcal{PS} = \{p, q\}$ demonstrates that the inclusion is strict:

$$u(X) = \begin{cases} 1 & X = \{p, q\} \\ 0 & \text{otherwise.} \end{cases}$$

The following five constraints must be satisfied if a $G \in \mathcal{L}(\mathit{clauses}, \mathbb{R}^+, \Sigma)$ exists which represents u:

$$w_p + w_q + w_{p \vee q} + w_{\neg p \vee q} + w_{p \vee \neg q} + w_{p \vee \neg p} = 1 \tag{3.10}$$
$$w_p + w_{\neg q} + w_{p \vee q} + w_{p \vee \neg q} + w_{\neg p \vee \neg q} + w_{p \vee \neg p} = 0 \tag{3.11}$$
$$w_{\neg p} + w_q + w_{p \vee q} + w_{\neg p \vee q} + w_{\neg p \vee \neg q} + w_{p \vee \neg p} = 0 \tag{3.12}$$
$$w_{\neg p} + w_{\neg q} + w_{\neg p \vee q} + w_{p \vee \neg q} + w_{\neg p \vee \neg q} + w_{p \vee \neg p} = 0 \tag{3.13}$$
$$w_\varphi \geq 0 \text{ for all clauses } \varphi \tag{3.14}$$

Together, constraints (3.11), (3.12), (3.13), and (3.14) force $w_\varphi = 0$ for every clause φ, contradicting (3.10). □

$\mathcal{L}(\mathit{clauses}, \mathbb{R}^+, \Sigma)$ seems not to characterize a natural class of functions. For (strictly) positive formulas with positive weights, on the other hand, we do obtain nice correspondences. Recall that a utility function u is *monotone* if, for all $X, Y \subseteq \mathcal{PS}$, $u(X) \leq u(Y)$ whenever $X \subseteq Y$.

Theorem 3.4.13. $\mathcal{U}(\mathit{spforms}, \mathbb{R}^+, \Sigma)$ *is the class of all normalized monotone utility functions.*

Proof. No strictly positive formula is a tautology, so every $u \in \mathcal{U}(\mathit{spforms}, \mathbb{R}^+, \Sigma)$ is normalized, and because all weights and formulas are positive, it is also monotone.

For the converse: Let u be an arbitrary normalized monotone utility function. We construct a $G \in \mathcal{L}(\mathit{spforms}, \mathbb{R}^+, \Sigma)$ for which $u_G = u$ as follows. Define a sequence of utility functions $u_1, \ldots, u_n$ such that

$$u_k(X) = \max\{u(X') \mid X' \subseteq X \text{ and } |X'| \leq k\}.$$

In this way, $u_1 = \max_{a \in X} u(\{a\})$ and $u_n = u$. Additionally, we define $u_0(X) = 0$ for all X, for convenience. Observe that we can use the these u_i to decompose u such that $u = \sum_{k=1}^{n}(u_k - u_{k-1})$, and so if we can construct a goalbase for each $u_k - u_{k-1}$, then the union of those goalbases will be a goalbase for u. Hereafter, we will abbreviate $u_k - u_{k-1}$ to u_k^*. To construct G_k, a goalbase for u_k^*, let $X_0 = \emptyset$ and $\left\langle X_1, \ldots, X_{\binom{n}{k}} \right\rangle$ be the set of size-k subsets of $\mathcal{PS}$, ordered so that $u_k^*(X_i) \leq u_k^*(X_j)$ for $i < j$. Then let

$$G_k = \left\{ \left(\bigvee_{j=i}^{\binom{n}{k}} \bigwedge X_j, u_k^*(X_i) - u_k^*(X_{i-1}) \right) \;\middle|\; 1 \leq i \leq \tbinom{n}{k} \right\}$$

from which it can easily, though tediously, be checked that $u_{G_k} = u_k^*$. (For example, if $\mathcal{PS} = \{a, b, c\}$ and $u(a) \leq u(b) \leq u(c)$, then $G_1 = \{(a \vee b \vee c, u(a)), (b \vee c, u(b) - u(a)), (c, u(c) - u(b))\}$. View items a, b, and c as substitutes, but with b conferring a bonus over a, and c a further bonus over b. This is the structure which can be seen in G_1. Higher-order G_is capture this same idea, but for sets of items larger than singletons.)

Finally, let $G = \bigcup_{k=1}^{n} G_k$. Now $u_G = u$, since for each k, $u_{G_k} = u_k^*$ and $\sum_{k=1}^{n} u_k^* = u_n = u$. Finally, observe that every formula in G is strictly positive; and all the weights $u_k^*(X_i) - u_k^*(X_{i-1})$ are nonnegative by virtue of the ordering declared over the X_i. □

Note that in the preceding theorem, we could also add *nonnegative* as a property, because normalization and monotonicity together imply nonnegativity. An application of Lemma 3.4.7 yields the following corollary:

Corollary 3.4.14. $\mathcal{U}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$ *is the class of all nonnegative monotone utility functions.*

Supermodularity seems not to correspond directly to a natural goalbase language, but we can characterize a large subclass.

Theorem 3.4.15. $\mathcal{U}(\mathit{pcubes}, \mathbb{R}^+, \Sigma)$ *is the class of all nonnegative utility functions satisfying the constraint* $\sum_{Y \subseteq X} (-1)^{|X \setminus Y|} \cdot u(Y) \geq 0$ *for all* $X \subseteq \mathcal{PS}$.

Proof. That $\mathcal{U}(pcubes, \mathbb{R}^+, \Sigma)$ is the class of all nonnegative utility functions satisfying $\sum_{Y \subseteq X} (-1)^{|X \setminus Y|} \cdot u(Y) \geq 0$ immediately follows from the fact that the weight of any positive cube is determined by the Möbius inversion as stated in equation (3.8). □

Note that the property corresponding to $\mathcal{U}(pcubes, \mathbb{R}^+, \Sigma)$ implies nonnegativity, monotonicity, and supermodularity. Nonnegativity and monotonicity follow from Corollary 3.4.14. For supermodularity, suppose that $G \in \mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$. Then

$$\begin{aligned} u_G(X \cup Y) &= \sum_{Z \subseteq X \cup Y} w_{\bigwedge Z} \\ &= \sum_{Z \subseteq X} w_{\bigwedge Z} + \sum_{Z \subseteq Y} w_{\bigwedge Z} - \sum_{Z \subseteq X \cap Y} w_{\bigwedge Z} + \sum_{\substack{Z \subseteq X \cup Y \\ Z \not\subseteq X, Y}} w_{\bigwedge Z} \\ &\geq \sum_{Z \subseteq X} w_{\bigwedge X} + \sum_{Z \subseteq Y} w_{\bigwedge Z} - \sum_{Z \subseteq X \cap Y} w_{\bigwedge Z} \\ &= u_G(X) + u_G(Y) - u_G(X \cap Y), \end{aligned}$$

which is equivalent to the supermodularity condition.

The utility function $u\colon X \mapsto \max(1, |X|)$ shows that there are supermodular utility functions which are not in $\mathcal{U}(pcubes, \mathbb{R}^+, \Sigma)$. As can easily be checked, if $\mathcal{PS} = \{p, q, r\}$, then expressing u in terms of positive cubes requires the use of a negative weight: $w_{p \wedge q \wedge r} = -1$. The previous theorem holds also for $\mathcal{U}(spcubes, \mathbb{R}^+, \Sigma)$ if "nonnegative" is replaced with "normalized".

Theorem 3.4.16. *$\mathcal{U}(pclauses, \mathbb{R}^+, \Sigma)$ is the class of all nonnegative utility functions satisfying the constraint $\sum_{Y \subseteq X} (-1)^{|X \setminus Y| + 1} \cdot u(\mathcal{PS} \setminus Y) \geq 0$ for all $X \subseteq \mathcal{PS}$.*

Proof. Follows from the fact that weights of positive clauses are determined by equation (3.9). □

Note that the property corresponding to $\mathcal{U}(pclauses, \mathbb{R}^+, \Sigma)$ implies monotonicity, normalization, and submodularity. Normalization and monotonicity follow from Theorem 3.4.13. To show submodularity, let $G \in \mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$ and let $X, Y \subseteq \mathcal{PS}$. For positive clauses φ, $X \cup Y \models \varphi$ together with $X \not\models \varphi$ implies $Y \models \varphi$. Furthermore, $X \not\models \varphi$ implies $X \cap Y \not\models \varphi$. Therefore:

$$\{(\varphi, w) \in G \mid X \cup Y \models \varphi \text{ and } X \not\models \varphi\} \subseteq \{(\varphi, w) \in G \mid Y \models \varphi \text{ and } X \cap Y \not\models \varphi\}$$

As all the weights w are positive, we immediately obtain the required inequality characterizing submodularity, namely $u_G(X \cup Y) - u_G(X) \leq u_G(Y) - u_G(X \cap Y)$.

An example which confirms that not all submodular utility functions belong to $\mathcal{U}(pclauses, \mathbb{R}^+, \Sigma)$ is the function $u\colon X \mapsto \min(2, |X|)$ for $\mathcal{PS} = \{p, q, r\}$. On the one hand, u is submodular; on the other we must have $w_{p \vee q \vee r} = -1$ if we

are to express u using positive clauses. The previous theorem holds also for $\mathcal{U}(pclauses + \top, \mathbb{R}^+, \Sigma)$ if "normalized" is replaced with "nonnegative".

The functions characterized by Theorem 3.4.15 are also known as *belief functions*, while those characterized by Theorem 3.4.16 are known as *plausibility functions* (when the functions are restricted to the interval $[0, 1]$) [Dempster, 1967; Shafer, 1976].

3.4.4 Summary

Our correspondence results for sum languages are summarized in Figure 3.1. In the figure, each node represents one language we examined, and an arrow from one node to another indicates that the tail language is included in the head language. Within each node, the expressivity of the language is given, according to the key below:

1	1-additive (modular)	m	monotone
k	k-additive	$\star$	plausibility function
ω	ω-additive (general)	†	belief function
n	normalized	$\subset$	proper subset of
$+$	nonnegative	$\subseteq$	subset of

Where $\subseteq$ (or $\subset$) is indicated, the language represents a (proper) subset of the class of utility functions with the given properties. In all other cases, the language represents exactly the class of utility functions with the given properties. The x-axis (increasing to the right) is the *cubes* axis, along which allowable cubes grow from length 1 up to ω; the y-axis (increasing into the page) is the *clauses* axis, also running from 1 to ω. The z-axis (decreasing upward) is the *positivity* axis and has three steps: strictly positive, positive, and general. Each language in the lower graph is a sublanguage of the corresponding language with general weights in the upper graph, but we have omitted these arrows for clarity.

We have not analyzed the interplay of bounding the length of formulas and restricting weights to positive reals in detail. By Theorem 3.4.6, any language restricting the length of formulas to at most k atoms can only generate k-additive utility functions. The opposite direction is less clear. While inspection of the proofs of Theorems 3.4.15 and 3.4.16 show that these results extend to the k-additive case in the expected manner, this is not so for Theorems 3.4.11 and 3.4.13. For instance, we do not know whether $\mathcal{U}(k\text{-}cubes, \mathbb{R}^+, \Sigma)$ is the class of all nonnegative k-additive functions or only a subclass thereof.

3.5 Expressivity of Max Languages

In many ways, sum is the most obvious aggregator to consider. However, there are certain classes of utility functions—in particular, single-minded and unit-demand

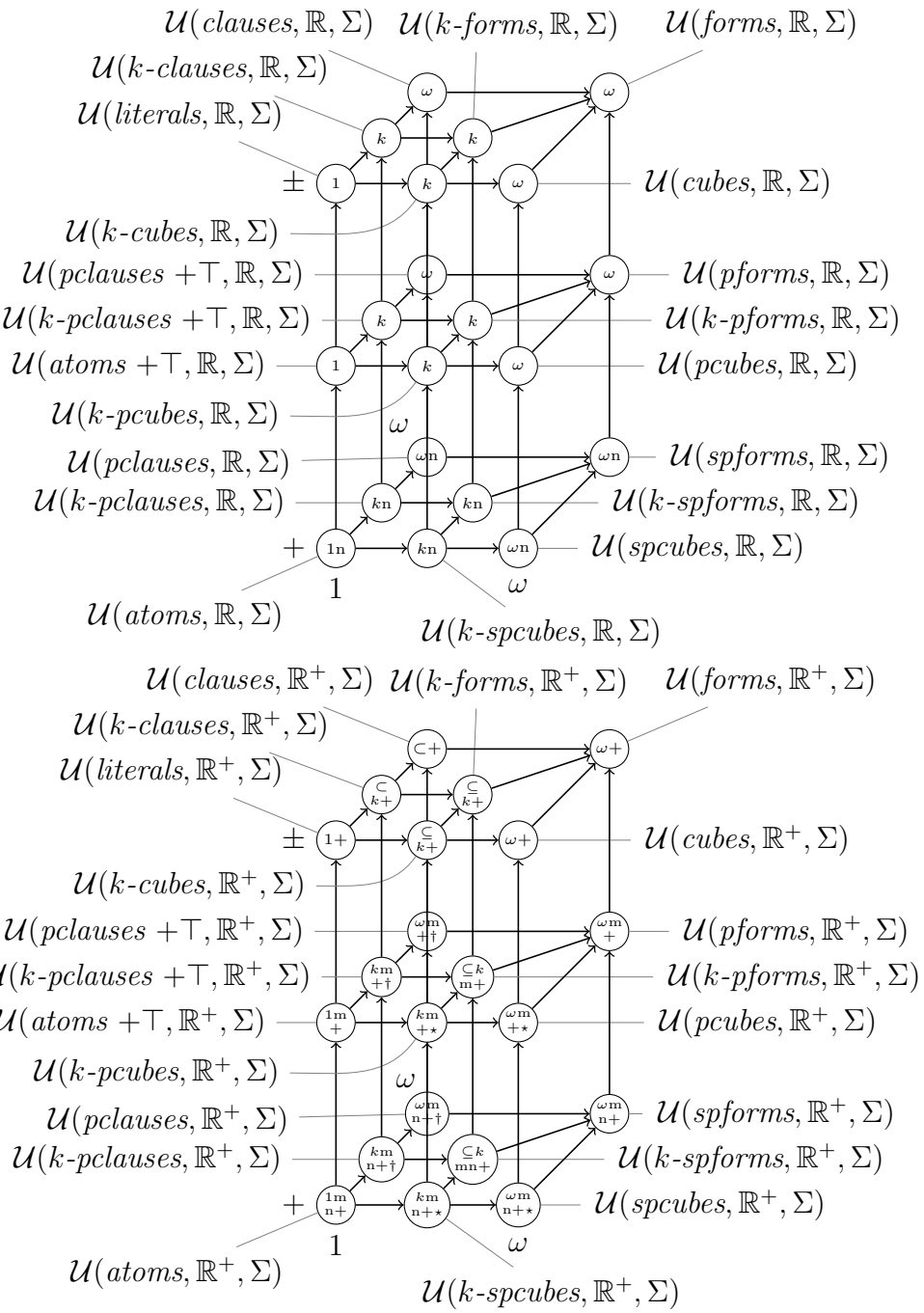


Figure 3.1: Summary of expressivity results for sum languages.

utility functions—which are not easily captured by sum languages. We turn now to max languages, both to exhibit languages which capture other classes of utility function and to serve as a study of how varying the aggregator affects the expressivity of goalbase languages.

We begin by establishing some simple results regarding equivalences between languages, which allow us to narrow down the range of languages to be considered in the remainder of the section. We then characterize the expressivity of the most important (distinct) languages.

When using a max language, only the weight of the most important goal satisfied by a given alternative matters. An example of a simple max language is $\mathcal{L}(atoms, \mathbb{R}, \max)$, which allows us to assign a value to any atomic proposition and where the utility of a state is equal to the value of the most valuable proposition which is true in that state. The utility functions in $\mathcal{U}(atoms, \mathbb{R}, \max)$ are known as *unit-demand valuations* (of which the aforementioned *simple* unit-demand valuation is a special case) in the literature on combinatorial auctions [Nisan, 2006]. We will return to prove that $\mathcal{U}(atoms, \mathbb{R}, \max)$ does indeed have this property in Theorem 3.5.10.

Throughout this section, when the aggregator function is omitted from the notation, it is intended to be max.

3.5.1 Superfluous Goals

For max languages, some weighted goals in a goalbase may never contribute to the utility of an alternative. Here we introduce terminology for speaking about this kind of situation and state some simple facts about potentially superfluous goals.

Definition 3.5.1 (Properties of Weighted Goals)**.**

- $(\varphi, w_\varphi) \in G$ is *dominated* if there exists a $(\psi, w_\psi) \in G$ such that $\varphi \models \psi$ and $w_\varphi < w_\psi$.
- $(\varphi, w_\varphi) \in G$ is *active* in a state M when $M \models \varphi$ and for all ψ, if $M \models \psi$ then $w_\psi \leq w_\varphi$.

Recall from Definition 3.2.3 that $(\varphi, w) \in G$ is superfluous if $G \equiv_F G \setminus \{(\varphi, w)\}$. Note that if there are no models for which (φ, w) is active, then the superfluity condition is fulfilled vacuously, so (φ, w) is superfluous in that case.

Fact 3.5.2. *Fix $(\varphi, w_\varphi) \in G$, and the aggregator as* max*. Then:*

1. *If (φ, w_φ) is dominated, then (φ, w_φ) is never active.*
2. *If (φ, w_φ) is never active, then (φ, w_φ) is superfluous.*
3. *If (φ, w_φ) is superfluous, for every state M where (φ, w_φ) is active, the set $\{(\psi, w_\psi) \mid M \models \psi$ and w_ψ is maximal$\}$ is not a singleton.*

Note that *superfluous* does not imply *never active*: For example, in the goalbase

$$\{(a, 1), (b, 1), (a \wedge b, 1)\},$$

the weighted formula $(a \wedge b, 1)$ is superfluous, but nonetheless active in the state $\{a, b\}$.

Since superfluous formulas are useless, they may be removed without harm. We can remove superfluous formulas from any $G \in \mathcal{L}(\mathit{forms}, \mathbb{R}, \max)$ as follows:

for all $(\varphi, w_\varphi) \in G$ **do**
  for all $(\psi, w_\psi) \in G$ **do**
    if $w_\varphi \leq w_\psi$ **then**
      if $\models \varphi \rightarrow \psi$ **then**
        $G := G \setminus \{(\varphi, w_\varphi)\}$
      end if
    end if
  end for
end for

Two aspects of this algorithm are noteworthy: First, we remove superfluous formulas one at a time. It would *not* be correct to remove all superfluous formulas in one step, as superfluity is defined relative to a particular goalbase and removal of a formula changes the goalbase. It is possible that some formulas which were superfluous will stop being so if another superfluous formula is removed. For example, all formulas in $\{(a, 1), (\neg\neg a, 1)\}$ are superfluous under max; however, removing both produces the empty goalbase, which is not equivalent to the original.

Second, this algorithm is not efficient, as it requires a quadratic number of calls to an UNSAT oracle; we would like to do better than a coNP algorithm. In the next section, we will see that the only max languages which are (expressively) interesting are based on cubes. This fact greatly reduces the complexity of deciding whether a formula is superfluous, since deciding whether one cube implies another is polynomial: Given two satisfiable cubes $\bigwedge X$ and $\bigwedge Y$, $\bigwedge X \rightarrow \bigwedge Y$ is a tautology iff the positive literals in Y are a subset of the positive literals in X and the negative literals in Y are a subset of the negative literals in X.

By way of comparison, it is easy to see that these concepts of domination and activity are not useful when considering sum languages. The formula $(a \wedge b, 1)$ is never active in the goalbase $\{(a, 2), (a \wedge b, 1)\}$, since $(a, 2)$ dominates it ($a \wedge b$ implies a, and $2 > 1$), but nonetheless it is not superfluous—we cannot remove it and retain an equivalent goalbase under the sum aggregator.

Fact 3.5.3. *Under the max aggregator, if G contains no superfluous formulas then every $(\varphi, w) \in G$ has a state in which it is uniquely active.*

3.5.2 Goalbase Equivalences

We are interested in comparing languages generated by different types of goalbases, in particular those generated from the following restrictions on formulas: *literals*, *cubes*, *clauses*, and *general* formulas using both conjunction and disjunction; as well as *positive* formulas and those including negation. Regarding weights, we want to consider *positive* and *general weights*. This gives rise to $4 \cdot 2 \cdot 2 = 16$ different languages. Here we establish several equivalences amongst languages and thereby show that we actually only need to consider a subset of all the languages that can be defined in this manner. Furthermore, if we are interested only in monotone utility functions, then we can further reduce the range of languages to consider.

We first show that disjunction is not an expressively helpful connective for max languages:

Theorem 3.5.4. $G \cup \{(\varphi_1 \vee \ldots \vee \varphi_n, w)\} \equiv_{\max} G \cup \{(\varphi_i, w) \mid 1 \leq i \leq n\}$.

Proof. Fix a state X. If $w_{\varphi_1 \vee \ldots \vee \varphi_n}$ is not the maximum w_ψ such that $X \models \psi$, then some other $(\psi, w_\psi) \in G$ is. Since $w_{\varphi_i} = w_{\varphi_1 \vee \ldots \vee \varphi_n}$, then $w_\psi > w_{\varphi_i}$ for all i. In this case, G alone determines the value of the left and right goalbases.

If $w_{\varphi_1 \vee \ldots \vee \varphi_n}$ is the maximum w_ψ such that $X \models \psi$, then some φ_i are such that $X \models \varphi_i$. For each such φ_i, we have $w_{\varphi_i} = w_{\varphi_1 \vee \ldots \vee \varphi_n}$ in the goalbase on the right, and so both the left and right have the same maximum. □

This tells us that with max as our aggregator, disjunctions as main connectives do not contribute to a language's expressivity. In fact, as any formula has an equivalent representation in disjunctive normal form, this tells us that disjunction can *never* increase the expressive power of a max language. In particular, from Theorem 3.5.4 we get the following equivalences between languages:

Corollary 3.5.5. *Fix $W \subseteq \mathbb{R}$. Then:*

1. $\mathcal{U}(\mathit{pclauses}, W, \max) = \mathcal{U}(\mathit{atoms}, W, \max)$,
2. $\mathcal{U}(\mathit{clauses}, W, \max) = \mathcal{U}(\mathit{literals}, W, \max)$,
3. $\mathcal{U}(\mathit{pforms}, W, \max) = \mathcal{U}(\mathit{pcubes}, W, \max)$,
4. $\mathcal{U}(\mathit{forms}, W, \max) = \mathcal{U}(\mathit{cubes}, W, \max)$.

Proof. For $\mathcal{U}(\mathit{forms}, W, \max) = \mathcal{U}(\mathit{cubes}, W, \max)$: Suppose that $(\varphi, w) \in G$. Without loss of generality, assume that $\varphi = \psi_1 \vee \ldots \vee \psi_n$ is in DNF. By Theorem 3.5.4, we may replace $(\psi_1 \vee \ldots \vee \psi_n, w)$ by $(\psi_1, w), \ldots, (\psi_n, w)$ and preserve goalbase equivalence. Repeating this for each original formula in G converts G to the language $\mathcal{L}(\mathit{cubes}, W, \max)$, since each ψ_i is a cube and the weights were left unchanged.

We use the same argument for each of the other cases, noting that the same transformation reduces a positive formula (in DNF) to a set of positive cubes, a clause to a set of literals, and a positive clause to a set of atoms. □

The same is *not* true for sum languages. E.g., under summation clauses are more expressive than literals: For $W = \mathbb{R}$, clauses can express *all* utility functions, while literals can express only modular functions (see Corollary 3.4.10). For each of the equivalences in Corollary 3.5.5, there is a set of weights W which violates it under summation.

Recall that a utility function u is called *monotone* if $M \subseteq M'$ implies $u(M) \leq u(M')$. Monotonicity is a reasonable assumption for many applications, in particular if propositional variables are interpreted as goods. Next we show that negation is not a helpful operation in case we are only interested in modeling monotone functions.

Theorem 3.5.6. *Fix G. Let X^+ be a set of positive literals, and X^- a set of negative literals, such that no atom appears in both. If $u_{G\cup\{(\bigwedge X^+\cup X^-,w)\},\max}$ is monotone, then*

$$G \cup \left\{\left(\bigwedge X^+ \cup X^-, w\right)\right\} \equiv_{\max} G \cup \left\{\left(\bigwedge X^+, w\right)\right\}.$$

Proof. There are two cases to consider: states which are supersets of X^+, and states which are not.

- Write u for $u_{G\cup\{(\bigwedge X^+\cup X^-,w)\},\max}$. In states $M \supseteq X^+$, we have that $M \models \bigwedge X^+$. It must be the case that $u(M) \geq w$ because $u(X^+) \geq w$ and u is monotone. Therefore, substituting $(\bigwedge X^+, w)$ for $(\bigwedge X^+ \cup X^-, w)$ cannot change the value of state M, since the value in M is already at least w.
- In states $M \not\supseteq X^+$, we have that both $\bigwedge X^+ \cup X^-$ and $\bigwedge X^+$ are false, and so cannot be active. Thus substituting $(\bigwedge X^+, w)$ for $(\bigwedge X^+ \cup X^-, w)$ cannot change the value at M, as inactive formulas do not affect the value of a utility function.

Therefore, in all states M we have that

$$u_{G\cup\{(\bigwedge X^+\cup X^-,w)\},\max}(M) = u_{G\cup\{(\bigwedge X^+,w)\},\max}(M).$$

□

The following result shows that we can further reduce the range of languages to consider if we limit ourselves to monotone utility functions. It follows immediately from Theorem 3.5.6 and Corollary 3.5.5. (Note that $\bigwedge \emptyset = \top$.)

Corollary 3.5.7. *Let* Mono *be the class of monotone utility functions. Fix $W \subseteq \mathbb{R}$. Then:*

1. $\mathcal{U}(\mathit{clauses}, W, \max) \cap \textsc{Mono} = \mathcal{U}(\mathit{atoms} + \top, W, \max) \cap \textsc{Mono}$.
2. $\mathcal{U}(\mathit{forms}, W, \max) \cap \textsc{Mono} = \mathcal{U}(\mathit{pcubes}, W, \max) \cap \textsc{Mono}$.

3.5.3 Correspondences

Corollary 3.5.5 tells us that the interesting languages, expressivity-wise, are those based on cubes, positive cubes, literals, and atoms. We prove that cubes are expressively complete for the full range of utility functions and that positive cubes correspond to the class of monotone functions:

Theorem 3.5.8. $\mathcal{U}(\mathit{cubes}, \mathbb{R}, \max)$ *is the class of all utility functions.*

Proof. Given a utility function u, define

$$G = \left\{ \left(\bigwedge X \cup \neg \bar{X}, u(X) \right) \;\middle|\; X \subseteq \mathcal{PS} \right\}$$

Since the formulas are the states, and as such are mutually exclusive, exactly one weight will be active in each state, and so $u(X) = u_G(X)$. □

Theorem 3.5.9. $\mathcal{U}(\mathit{pcubes}, \mathbb{R}, \max)$ *is the class of monotone utility functions.*

Proof. ($\Rightarrow$) Suppose that $G \in \mathcal{U}(\mathit{pcubes}, \mathbb{R}, \max)$ but u_G is not monotone. So there are states $M \subset M'$ such that $u_G(M') < u_G(M)$. Then there is a $(\varphi, u_G(M)) \in G$ which is active in M such that $M \models \varphi$ but $M' \not\models \varphi$. Since $M' \supset M$, then there is some $a \in M' \setminus M$ for which $\varphi \models \neg a$. Therefore, φ is not a positive formula, which contradicts the hypothesis that φ is a pcube.

($\Leftarrow$) If u is monotone, then let $G = \{(\bigwedge X, u(X)) \mid X \subseteq \mathcal{PS}\}$. Note that for $Y \subseteq X$, $u(Y) = w_{\bigwedge Y} \leq w_{\bigwedge X} = u(X)$ follows directly from the monotonicity of u. In state X, $u_G(X) = \max\{w_{\bigwedge Y} \mid Y \subseteq X\} = w_{\bigwedge X}$. Hence $u_G(X) = u(X)$. □

The class of unit-demand utility functions has no simple corresponding sum language, but does have a corresponding max language:

Theorem 3.5.10. $\mathcal{U}(\mathit{atoms}, \mathbb{R}, \max)$ *is the class of unit-demand utility functions.*

Proof. Suppose that u is a unit-demand valuation, which by definition means that $u(X) = \max_{a \in X} u(\{a\})$. Construct a $G \in \mathcal{U}(\mathit{atoms}, \mathbb{R}, \max)$ such that $G = \{(a, w) \mid a \in \mathcal{PS}, u(\{a\}) = w\}$. Then

$$u(X) = \max_{a \in X} u(\{a\}) = \max_{(a,w) \in G} w = u_{G,\max}(X).$$

Conversely, suppose that $G \in \mathcal{U}(\mathit{atoms}, \mathbb{R}, \max)$, and note that the same series of equivalences holds. □

We are not aware of a property of utility functions referred to in the literature which would characterize $\mathcal{U}(\mathit{literals}, \mathbb{R}, \max)$. The desired property is a generalization of the unit-demand valuation that also allows us to specify a value for *not* receiving a particular item.

By restricting the set of weights W we can capture classes of utility functions with a particular range. $\mathcal{U}(\mathit{pcubes}, \mathbb{R}^+, \max)$, for instance, is the class of nonnegative monotone functions. This class is known to be equal to $\mathcal{U}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$ (see Corollary 3.4.14).[1] This is a case where a syntactically simple language is more expressive with max than with sum.

On the other hand, some very simple classes of utility functions are hard to capture in structurally simple languages using max aggregation. For instance, recall that a utility function u is called *modular* iff $u(M \cup M') = u(M) + u(M') - u(M \cap M')$ for all $M, M' \in 2^{\mathcal{PS}}$. Modular functions are nicely captured by $\mathcal{U}(\mathit{literals}, \mathbb{R}, \Sigma)$ (see Corollary 3.4.10). However, there is no natural restriction to formulas that would allow us to characterize the modular functions under max aggregation. On the contrary, among the max languages considered here, only $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \max)$ can express all modular functions, and this language is so powerful that it can actually express *all* utility functions.

In particular, $\mathcal{L}(k\text{-}\mathit{pcubes}, \mathbb{R}, \max)$ misses some modular utility functions when $k < |\mathcal{PS}|$: Suppose that u is modular and nonnegative, and at least $k+1$ singleton states $\{p_1\}, \ldots, \{p_{k+1}\}$ have nonzero value. Then there is no $G \in \mathcal{L}(k\text{-}\mathit{pcubes}, \mathbb{R}, \max)$ such that $u(\{p_1, \ldots, p_{k+1}\}) = u_G(\{p_1, \ldots, p_{k+1}\})$, because $u_G(\{p_1, \ldots, p_{k+1}\})$ is the weight of some k-pcube (because max is our aggregator) and by assumption if $X \subset \{p_1, \ldots, p_{k+1}\}$ then $u(X) < u(\{p_1, \ldots, p_{k+1}\})$. The same modular utility functions are missing from $\mathcal{L}(k\text{-}\mathit{cubes}, \mathbb{R}, \max)$, due to the fact that the addition of negation to the language is not helpful for representing monotone utility functions (see Theorem 3.5.6). As a result, we have the following:

Theorem 3.5.11. *For all $k > j$, $\mathcal{U}(k\text{-}\mathit{pcubes}, W, \max) \supset \mathcal{U}(j\text{-}\mathit{pcubes}, W, \max)$ and $\mathcal{U}(k\text{-}\mathit{cubes}, W, \max) \supset \mathcal{U}(j\text{-}\mathit{cubes}, W, \max)$.*

3.5.4 Summary

Our correspondence results for max languages are summarized in Figure 3.2. In the figure, each node represents one language we examined, and an arrow from one node to another indicates that the tail language is included in the head language. Within each node, the expressivity of the language is given, according to the key below:

ω	ω-additive (general)	m	monotone
$+$	nonnegative	u	unit-demand
‡	general unit-demand	$\subset$	proper subset of

Where $\subset$ is indicated, the language represents a proper subset of the class of utility functions with the given properties. In all other cases, the language

[1] To be precise, $\mathcal{U}(\mathit{pcubes}, \mathbb{R}^+, \max) = \mathcal{U}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$ over total functions only. Because the max languages can also express partially defined functions returning $-\infty$ for some states while sum languages cannot, if we expand our consideration to partially-defined utility functions, then $\mathcal{U}(\mathit{pcubes}, \mathbb{R}^+, \max) \supset \mathcal{U}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$.

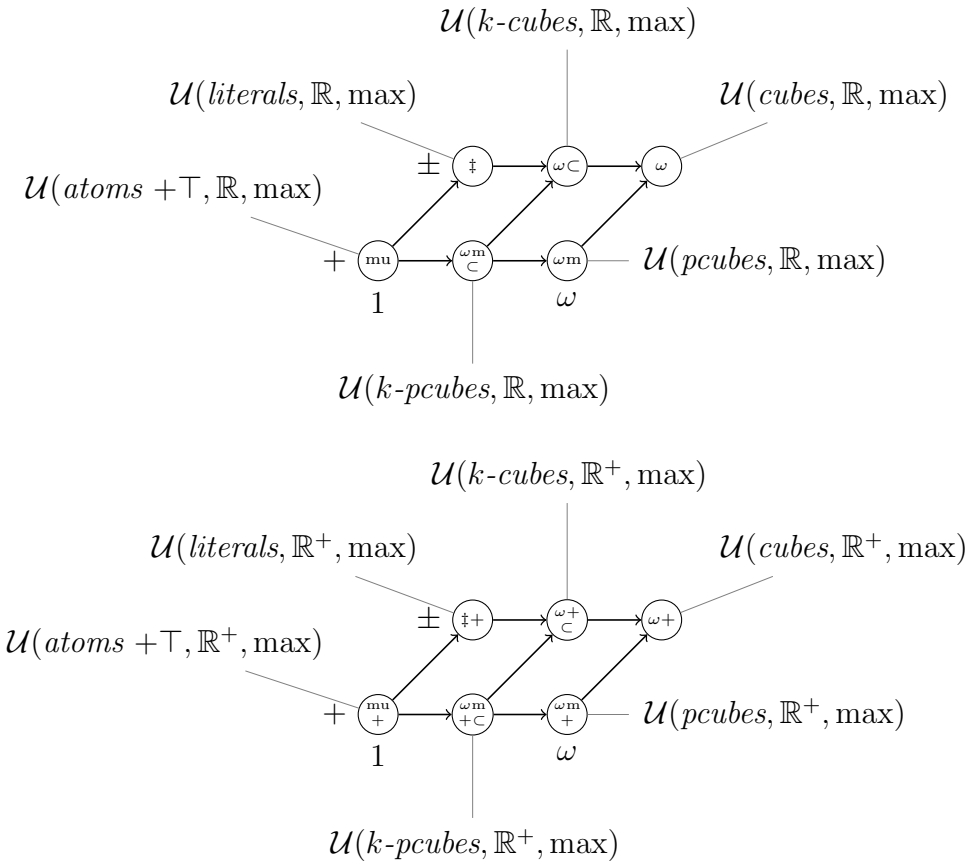


Figure 3.2: Summary of expressivity results for max languages.

represents exactly the class of utility functions with the given properties. The x-axis (increasing to the right) is the *cubes* axis, along which allowable cubes grow from length 1 up to ω; the y-axis (decreasing into the page) is the *positivity* axis and has two steps: positive and general. Each language in the lower graph is a sublanguage of the corresponding language with general weights in the upper graph, but we have omitted these arrows for clarity. Note that several languages do not appear in the figure due to being expressively equivalent to some language which does appear there. For max languages, this is due to the fact expressed in Corollary 3.5.5, namely that disjunction does not contribute expressivity to max languages.

3.6 Odds and Ends

In this section, we present two results which fit nowhere else, both of which bear on how weights affect expressivity. The first is the observation that the weights can be limited to the range of the representable utility functions if every state formula is in the language:

Theorem 3.6.1. *If Φ contains all state formulas and $F \in \{\Sigma, \max\}$, then $\mathcal{U}(\Phi, W, F)$ contains every utility function u such that* $\operatorname{ran} u \subseteq W$.

Proof. Let $G = \{(\bigwedge(M \cup \neg\bar{M}), u(M)) \mid M \subseteq \mathcal{PS}\}$. Clearly $u_{G,F} = u$ and $G \in \mathcal{L}(\Phi, W, \Sigma)$, since each formula in G is a state formula, and only weights in W are used. □

Next, we examine the relationship between integer-valued utility functions and noninteger weights:

Theorem 3.6.2. *If $u_{G,\Sigma} \colon 2^{\mathcal{PS}} \to \mathbb{Z}$ and G is minimal and contains any noninteger weight, then G contains at least three noninteger weights.*

Proof. If G contains exactly one noninteger weight, w_1 from (φ_1, w_1), then if $M \models \varphi_1$, $u_{G,\Sigma}(M) \notin \mathbb{Z}$, so if there are any, there must be at least two noninteger weights. If $(\varphi_2, w_2) \in G$ and $w_2 \notin \mathbb{Z}$, then by minimality of G we know that φ_1 and φ_2 are not equivalent, and so there must be a state M such that either $M \models \varphi_1 \wedge \neg\varphi_2$ or $M \models \neg\varphi_1 \wedge \varphi_2$. Hence if w_1 and w_2 are the sole noninteger weights, $u_{G,\Sigma}(M) \notin \mathbb{Z}$. Therefore, if G contains any noninteger weights, it must contain at least three of them. □

While we would like to continue the proof as an induction to show that no finite number of noninteger weights suffices, and hence no minimal representation of an integer-valued utility function will use noninteger weights, we cannot do so, for the following reason: By minimality, we know that the formulas in G are pairwise nonequivalent, and hence this gives us the needed model where exactly

one formula with a noninteger weight is true. But pairwise nonequivalence does not guarantee that for three formulas we can find a model which makes one of the formulas true and the other two false, and hence we do not find the needed model.

We expect that minimality does in fact entail that no noninteger weights are needed to represent integer-valued utility functions. For example, consider the goalbase $\{(\top, \frac{1}{2}), (a, \frac{1}{2}), (\neg a, \frac{1}{2})\}$, which represents $u(X) = 1$ using the minimal number of noninteger weights, but is itself clearly not minimal.

3.7 Conclusion

In this chapter we have characterized the expressivity of nearly all natural sum and max languages. The sum languages correspond to a wide variety of classes of utility functions, ranging from full (e.g., $\mathcal{L}(\mathit{forms}, \mathbb{R}, \Sigma)$) to rather limited (e.g., $\mathcal{L}(\mathit{atoms}, \mathbb{R}^+, \Sigma)$) expressivity, with a great many alternatives between (e.g., $\mathcal{L}(k\text{-}\mathit{cubes}, \mathbb{R}, \Sigma)$). The max languages are bipolar in their expressivity—there are fully-expressive languages (e.g., $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \max)$), extremely circumscribed ones ($\mathcal{L}(\mathit{atoms} + \top, \mathbb{R}^+, \max)$), and nothing in the middle. (The k-languages, which occupy the middle for sum, appear to have hardly more expressivity than atoms or literals for max.) There are sum or max languages corresponding to many common classes of utility functions, so it is likely that an appropriately expressive one may be found for any desired application. Additionally, we have demonstrated that some languages, such as $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$, $\mathcal{L}(\mathit{pclauses}, \mathbb{R}, \Sigma)$, and their sublanguages, have unique representations for all utility functions which are representable in them.

Chapter 4

Succinctness

4.1 Introduction

In this chapter, we consider how space-efficient languages are, both in absolute terms and relative to one another. This space-efficiency is known as *succinctness*. As with expressivity, the succinctness of a language is an important feature to consider when selecting the most suitable language for any application. The more succinct a language is, the less data will need to be conveyed and stored; on the other hand, we pay for high succinctness with increased computational complexity when we want to run queries on goalbases, as we shall see in Chapter 5.

Here we present several kinds of results. After introducing the definitions and notation necessary for talking about succinctness (Section 4.2), we consider related work (Section 4.3). In Section 4.4, we present many relative succinctness results for sum languages, grouped according to the methods used to prove them. For the impatient, a summary of all known succinctness results involving pairs of sum languages appears in Table 4.1. In Section 4.5, we do the same for max languages, though here the features of the max aggregator permit us to give some absolute succinctness results as well. Finally, in Section 4.6 we prove some succinctness results for pairs of languages where one uses max as its aggregator and the other sum, in order to give some insight into how the two families of languages compare.

4.2 Preliminaries

In this section, we define succinctness and present some basic, aggregator-independent facts about the succinctness relation. Because succinctness is a size notion, we must first specify how to measure the sizes of formulas and goalbases.

Definition 4.2.1 (Formula Length and Goalbase Size)**.** The *length of a formula* φ is the number of occurrences of atoms it contains. The *size of a weighted goal* (φ, w) is the length of φ plus the number of bits needed to store w (that is, $\log w$

bits). The *size of a goalbase* G, written as size(G), is the sum of the sizes of the weighted goals in G.

Observe that the size of a goalbase may differ from its cardinality: If $G = \{(a \wedge b, 1)\}$, then size(G) = 2 (or 3, if we are not neglecting the bit used for storing the weight) while $|G| = 1$.

Often we consider families of utility functions $\{u_n\}_{n\in\mathbb{N}}$, where for each n we have that $|\mathcal{PS}| = n$. Suppose that we have a corresponding family of goalbases $\{G_n\}_{n\in\mathbb{N}}$ for which $u_n = u_{G_n}$. Unless the number of bits required to represent the weights in G_n grows superexponentially in n, the size contributed by the weights can be safely ignored when considering how size(G_n) grows with n. Every family of utility functions considered here has weights which are independent of n, so we disregard the size of the weights in our succinctness results. (Superexponential growth in weights affects all languages equally.)

Frequently one language contains shorter representations of some utility functions than does another language. Here we offer a definition of relative succinctness to make this notion precise. This definition is similar to ones given by Cadoli, Donini, Liberatore, and Schaerf [2000] and Coste-Marquis et al. [2004]. Because we wish to compare languages which differ in expressive power, we define succinctness over only the expressive overlap of the languages being compared. This leads to some counterintuitive results for languages with little expressive overlap and makes the comparative succinctness relation intransitive, but it also permits us to make comparisons where the expressive overlap is substantial, though not total.

Definition 4.2.2 (Succinctness)**.** Let $\mathcal{L}(\Phi, W, F)$ and $\mathcal{L}(\Psi, W', F')$ be goalbase languages and $\mathcal{U}$ a class of utility functions for which every member is expressible in both languages. Then $\mathcal{L}(\Phi, W, F) \preceq_{\mathcal{U}} \mathcal{L}(\Psi, W', F')$ iff there exists a function $f\colon \mathcal{L}(\Phi, W, F) \to \mathcal{L}(\Psi, W', F')$ and a polynomial p such that for all $G \in \mathcal{L}(\Phi, W, F)$, if $u_{G,F} \in \mathcal{U}$ then $u_{G,F} = u_{f(G),F'}$ and size($f(G)$) $\leq p$(size(G)).

Read $\mathcal{L} \preceq_{\mathcal{U}} \mathcal{L}'$ as: $\mathcal{L}'$ is at least as succinct as $\mathcal{L}$ over the class $\mathcal{U}$. When $\mathcal{L}'$ is strictly more succinct than $\mathcal{L}$—that is, in no case are representations more than polynomially worse, and in at least one case, they are super-polynomially better in $\mathcal{L}'$—we write $\mathcal{L} \prec_{\mathcal{U}} \mathcal{L}'$. When we have nonstrict succinctness in both directions, we write $\mathcal{L} \sim_{\mathcal{U}} \mathcal{L}'$; when we have nonstrict succinctness in neither direction, i.e., incomparability, we write $\mathcal{L} \perp_{\mathcal{U}} \mathcal{L}'$. Whenever a succinctness relation appears unsubscripted (i.e., without an explicit class of comparison), then implicitly $\mathcal{U} = \{u_{G,F} \mid G \in \mathcal{L}\} \cap \{u_{G',F'} \mid G' \in \mathcal{L}'\}$, which is the *expressive intersection* of $\mathcal{L}$ and $\mathcal{L}'$.

This definition of succinctness is a generalization of those given by Chevaleyre et al. [2006] and Uckelman and Endriss [2007]. The definition from the former does not permit comparison of languages which differ in expressive power, while the latter fixes the class of comparison $\mathcal{U}$ as the expressive intersection of the two

languages. Later in this chapter, in Section 4.6, we illustrate some circumstances in which being explicit about the class of comparison is important.

Finding the succinctness relation between some pairs of goalbase languages is trivial, as when one language in a pair is a sublanguage of the other, or when the two languages have no expressive overlap. These cases can be dismissed without argument. Recall from Definition 3.2.4 that a goalbase language may have unique representations: If a utility function is representable in the language, then there is exactly one representation of it in the language. If $\mathcal{L}$ has unique representations and $\mathcal{L} \succ \mathcal{L}'$ is true, then we can show that $\mathcal{L} \succ \mathcal{L}'$ as follows:

Proof strategy. We present a family of utility functions $\mathcal{U}$, and construct a (small-ish, but not necessarily optimal) representation $G' \in \mathcal{L}'$ for each $u \in \mathcal{U}$. Then, we construct a representation $G \in \mathcal{L}$ for each $u \in \mathcal{U}$ where at least one G_u is exponentially larger than its corresponding G'_u. Because we know that $\mathcal{L}'$ has unique representations, we know that we can't find a smaller (or any other!) representation of u in $\mathcal{L}'$, so we have shown that $\mathcal{L} \succ \mathcal{L}'$. □

This is a handy proof strategy, one which we shall make use of many times in this chapter.

Here we state some basic properties of the succinctness relation, which we use frequently in our proofs, often without reference.

Fact 4.2.3. *For all languages $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$:*

1. *If $\mathcal{L}_1 \subseteq \mathcal{L}_2$, then $\mathcal{L}_1 \preceq \mathcal{L}_2$.*
2. *If $\mathcal{L}_1 \preceq \mathcal{L}_2$ and $\mathcal{L}_3 \subseteq \mathcal{L}_1$, then $\mathcal{L}_3 \preceq \mathcal{L}_2$.*
3. *If $\mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}_3$ and $\mathcal{L}_1 \prec \mathcal{L}_2 \preceq \mathcal{L}_3$, then $\mathcal{L}_1 \prec \mathcal{L}_3$.*
4. *If $\mathcal{L}_1 \perp \mathcal{L}_2$ and $\mathcal{L}_1 \cup \mathcal{L}_2 \subseteq \mathcal{L}_3$, then $\mathcal{L}_1, \mathcal{L}_2 \prec \mathcal{L}_3$.*
5. *If $\mathcal{L}_1 \sim \mathcal{L}_2$ and $\mathcal{U}(\mathcal{L}_1) = \mathcal{U}(\mathcal{L}_2)$, then $\mathcal{L}_1 \odot \mathcal{L}_3$ iff $\mathcal{L}_2 \odot \mathcal{L}_3$, where $\odot \in \{\sim, \succeq, \succ, \preceq, \prec, \perp\}$.*

Note that Fact 4.2.3.2 is useful contrapositively also, for deriving $\not\preceq$ results for superlanguages. For Fact 4.2.3.3, it would be inadequate to require that $\mathcal{L}_1 \preceq \mathcal{L}_2 \prec \mathcal{L}_3$ instead, since it could happen that $\mathcal{L}_1$ is too small to represent the utility functions which cause $\mathcal{L}_2 \prec \mathcal{L}_3$. Fact 4.2.3.5 expresses the notion that if two languages are equal in succinctness and expressivity, then they stand in the same succinctness relation with any third language.

Now we derive a simple succinctness result which applies to all languages which permit formulas of no more than a fixed, finite length.

Theorem 4.2.4. *For any fixed $k \in \mathbb{N}$, arbitrary set of formulas Ψ, and arbitrary sets of weights W, W' and aggregator F: If $\Phi \subseteq k$-forms, then $\mathcal{L}(\Phi, W, F) \succeq \mathcal{L}(\Psi, W', F)$.*

Proof. There are only $O(n^k)$ formulas of length k or less, and so any utility function u representable in $\mathcal{L}(\Phi, W, F)$ cannot have a representation more than polynomially larger than the best one in $\mathcal{L}(\mathit{forms}, \mathbb{R}, F)$. Hence, $\mathcal{L}(\Phi, W, F) \succeq \mathcal{L}(\mathit{forms}, \mathbb{R}, F)$. Furthermore, $\mathcal{L}(\mathit{forms}, \mathbb{R}, F) \supseteq \mathcal{L}(\Psi, W', F)$, so by Fact 4.2.3.2 we have that $\mathcal{L}(\Phi, W, F) \succeq \mathcal{L}(\Psi, W', F)$. □

As a consequence, any two languages with bounded-length formulas and the same aggregator are equally succinct over their expressive intersection.

Absolute succinctness, as its name implies, is not comparative. Rather, it deals with the size of a the smallest goalbase in a language which will represent a given utility function. To make *smallest* precise, we give a definition of goalbase minimality:

Definition 4.2.5. A goalbase $G \in \mathcal{L}$ is *minimal* if for all $G' \in \mathcal{L}$ such that $G' \equiv G$, $\text{size}(G) \leq \text{size}(G')$.

The difficulty of recognizing whether G is minimal, or of finding a G which is a minimal representative of some utility function u, is strongly dependent on what $\mathcal{L}$ and u are. In the general case, formulas in a minimal goalbase will be pairwise nonequivalent, but this is not a sufficient condition for minimality. For some languages (such as $\mathcal{L}(\mathit{atoms}, \mathbb{R}, \Sigma)$) and some classes of utility functions (e.g., modular) we can detect minimality and generate minimal representations easily, while for many richer languages how to do this is unobvious or unknown.

For max languages, in a minimal goalbase no formula is implied by any formula with a smaller or equal weight; for sum languages, formulas in a minimal goalbase will be pairwise non-equivalent.

For certain languages, due to the limited length of formulas which may appear in minimal goalbases, it will be the case that we need not distinguish between growth of $\text{size}(G)$ and growth of $|G|$.

Theorem 4.2.6. *Let $\mathcal{L}$ be a goalbase language. For each $n \in \mathbb{N}$, let φ_n be the longest formula in $\mathcal{L}$ in n variables with no shorter equivalent. Then if there is a polynomial p such that $\text{size}(\varphi_n) \in O(p(n))$, it follows that for all families of minimal goalbases $\{G_n\}_{n\in\mathbb{N}} \subseteq \mathcal{L}$, $\text{size}(G_n)$ is polynomial in n iff $|G_n|$ is polynomial in n.*

Proof. ($\Rightarrow$) This direction is obvious, since it is impossible to form exponentially-many formulas from polynomially-many atom instances.

($\Leftarrow$) Suppose that $|G_n| = p(n)$ is a polynomial. The longest formula φ_n in $\mathcal{L}$ in n variables has $\text{size}(\varphi_n) = q(n)$, for some polynomial q. The worst case is that each of the $p(n)$ formulas in G_n has size $q(n)$, for a total size of $p(n) \cdot q(n)$, which is still polynomial in n. □

In particular, this means that $\text{size}(G)$ and $|G|$ are interchangeable in terms of growth in $|\mathcal{PS}|$ for languages such as $\mathcal{L}(\mathit{cubes}, W, F)$ and $\mathcal{L}(\mathit{clauses}, W, F)$, but not necessarily for languages such as $\mathcal{L}(\mathit{forms}, W, F)$.

4.3 Related Work

Before proceeding to our results, we wish to point out some work which is relevant for language succinctness, as here there is much more in the literature which appears applicable to the problem at hand than there was in the previous chapter. The succinctness of representations in various languages, broadly speaking, has by now a rather long history, insofar as any problem can be said to have a long history in the still-young field of computer science.

Succinctness of representation plays an important role in complexity theory. The hardness of decision problems and runtime of algorithms is specified as a function of the size of their inputs (e.g., an $O(n^2)$ algorithm will, in the worst case, require a number of steps quadratic in the length of its input). As such, there is a trade-off between the succinctness of the input for and worst-case hardness of a given problem—the shorter the input, the fewer steps we may take to process it and still remain within the realm of tractability. The canonical example of this is representing input in unary instead of binary. Because the "tally" language is exponentially less succinct than binary, we have exponentially more time in which to process inputs in unary. An input with n binary digits will have $O(2^n)$ unary digits, while an $O(2^n)$ operation on that input in binary will be only an $O(n)$ operation on the same input in unary.

Succinctness has appeared in the study of Boolean circuits under the guise of circuit size. There is an enormous literature on Boolean circuits, in part due to the now-dashed hope that circuit complexity could shed light on whether $\mathsf{P} = \mathsf{NP}$. A cursory examination of Wegener's book on Boolean circuits [Wegener, 1987] reveals a wealth of results regarding upper and lower bounds on the size of Boolean circuits for computing various Boolean-valued functions. Because propositional formulas are themselves Boolean circuits, and likewise goalbases can be thought of as Boolean circuits with the aggregator as the output gate, results in this area are *prima facie* highly relevant for comparing the succinctness of goalbase languages. Unfortunately, the concerns of researchers working on Boolean circuits seem largely orthogonal to ours, as the properties which were investigated (such as bounded depth, bounded fan-in and -out) do not map neatly onto the properties which interest us. Nonetheless, we do make use of a circuit size result in our proof of Theorem 4.4.13 in this chapter.

There is also a significant literature on the effect that preprocessing may have on the complexity of decision problems. The general idea here is that some problem instances may have enough overlap that an advantage may be gained by preprocessing ("compiling") the common data. There is no limit on the time which may be spent in the compilation phase, only on the size of the compiled output, as compilation of the fixed part of the input is considered to happen offline. If the remaining decision problem, after compilation, is in a complexity class $\mathcal{C}$, then the original problem is said to be $\mathcal{C}$-compilable. Liberatore [2001] gives a thorough discussion of these notions. Most of the results there are negative,

showing that various problems are $\rightsquigarrow\mathcal{C}$-hard (i.e., they are $\mathcal{C}$-hard even after permitting unlimited preprocessing time on the fixed part of the problem).

In particular, Darwiche and Marquis [2004] give compilation results for propositional weighted bases, which are the penalty-logic version of our goalbases. The decision problems considered there, MODEL CHECKING and CLAUSAL INFERENCE, are not obviously applicable to our framework, as Darwiche and Marquis interpret their sets of weighted formulas as representing ordinal preferences, not cardinal preferences. Similarly, there are a great many succinctness results given by Coste-Marquis et al. [2004] for various ordinal preference notions, when represented using weighted propositional formulas. Here again, it is not clear whether or how these results might be applicable to our goalbase languages, as in these cases goalbases are being used to generate preorders, while in our case we are generating utility functions. Further work on knowledge base compilation may be found in [Cadoli, Donini, Liberatore, and Schaerf, 1996, 1999a; Cadoli, Palopoli, and Scarcello, 1999b; Cadoli et al., 2000; Cadoli, Donini, Liberatore, and Schaerf, 2002].

Though there are dissimilarities between using goalbases in ordinal and cardinal contexts, the compilation literature nonetheless points us towards strict succinctness proofs of the following sort:

Proof idea. Suppose that for the language $\mathcal{L}_1$, the decision problem BIPARTITE AARDVARK COLORING is $\mathcal{C}$-hard, but is known to be a member of the easier $\mathcal{C}'$ class for another language $\mathcal{L}_2$. Therefore, we know that if there were a polytime translation from $\mathcal{L}_1$ to $\mathcal{L}_2$, then we could decide BIPARTITE AARDVARK COLORING for $\mathcal{L}_1$ much faster by translating our input into $\mathcal{L}_2$ first and deciding BIPARTITE AARDVARK COLORING there instead. Because the $\mathcal{C}$-hardness of BIPARTITE AARDVARK COLORING for $\mathcal{L}_1$ makes this is impossible, we conclude that there can be no polytime translation of $\mathcal{L}_1$ into $\mathcal{L}_2$. □

The careful reader will notice that this proof idea is a dead end for us: It rules out a *polytime* translation from the harder language into the easier, but not a *polysize* translation. It follows from the impossibility of a polysize translation that $\mathcal{L}_1 \succ \mathcal{L}_2$, but no such thing follows from the impossibility of a polytime translation. It might well be the case that constructing a polysize translation involves iterating over all $2^{|\mathcal{PS}|}$ states for some goalbases. Compilation could help us here, by moving the translation step into the fixed part of the problem. Coste-Marquis et al. [2004, Table 1] report that this method works for ruling out polysize translations between some of their ordinal preference representation languages. However, we have yet to find any decision problem and pair of goalbase languages with the required properties to make this possible, and hence we do not use this approach for proving any of the strict succinctness results in this chapter.

Probably closest to the contents of this chapter is the work of Wachter and Haenni [2006] on propositional directed acyclic graphs (PDAGs). A PDAG is a rooted DAG consisting of $\triangle$ (AND), $\triangledown$ (OR), and $\diamond$ (NOT) nodes, and leaf nodes $\circ$

labeled with propositional letters or constants ($\top$, $\bot$). In other words, a PDAG is a Boolean circuit composed of AND, OR, and NOT gates. Wachter and Haenni give a succinctness definition nearly the same as the one used by Chevaleyre et al. [2006], and proceed to prove equi- and strict succinctness results for several classes of PDAGs, in some cases relying on knowledge compilation results of Darwiche and Marquis [2002]. Comparing the succinctness of one class of PDAGs to another is almost the same as comparing two classes of goalbases where every goalbase contained in each class is of the form $\{(\varphi, 1)\}$, the difference being that PDAGs may have a succinctness advantage over formulas which contain many copies of the same subformula, as a single subPDAG may have multiple parents. Wachter and Haenni [2006, Definition 2] constructed 15 sublanguages of the full PDAG language by considering PDAGs having combinations of four circuit properties—namely, flatness, decomposability, determinism, and simple-negation. Flatness limits circuit depth, decomposability and determinism concern overlap between subformulas, and simple-negation limits application of negations to subformulas which are atoms. While these properties are useful when considering DAGs, they are not natural properties when working with formulas: The languages we studied are those formed by simple restrictions on formula structure; as it happens, none of these exhibit decomposability or determinism. All of our languages except those based on general formulas have the simple-negation property, but they are all proper sublanguages of the language containing all simply-negated formulas. Which of our languages are flat depends on whether AND and OR gates are permitted to have arbitrary fan-in, or a fan-in of exactly 2, as in propositional formulas. With arbitrary fan-in, all of our cubes and clauses languages are flat; with a fan-in of 2, only languages up to 3-formulas are flat.[1] However, regardless of the fan-in, the set of all flat formulas corresponds to none of the sets of formulas we examined. While it would be interesting to consider flat, decomposable, deterministic, and simply-negated goalbase languages in order to see which results carry over, the complexity of the restrictions on formulas they impose puts them beyond what might be usable by people, say for preference representation in voting or auctions. This is not to say that they might not be usable by computer agents—they might very well be, and deserve some consideration in future work.

Ieong and Shoham [2005] introduce marginal contribution nets (MC-nets) as a way of modeling coalitional games with transferable utility. A coalitional game with transferable utility $\langle N, v\rangle$ is a set of agents N and a valuation function $v\colon S \subseteq N \to \mathbb{R}$ which indicates the value of any coalition S to its members. (The game specifies only how much utility a coalition receives, not how its members should divide it; this is what distinguishes a coalitional game with transferable utility from one without.) An MC-net is a set of rules of the form $\varphi \to w$,

[1] This is because the cube $a \wedge (b \wedge (c \wedge d))$ has a depth of 3. Note also that it makes no difference that this cube could be rebalanced as $(a \wedge b) \wedge (c \wedge d)$ and thereby made flat, since flatness is a property of the language as a whole, not just of the flattest members of each equivalence class.

where φ is a cube and $w \in \mathbb{R}$. A rule $\varphi \to w$ is said to apply to a coalition S iff all of the positive literals in φ are members of S and none of the negative literals in φ are members of S; the value of a coalition S is the sum of weights of all rules which apply to S. It is easy to see that the language of MC-nets is exactly $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma)$ in disguise. Ieong and Shoham [2005, Section 3.1] prove one succinctness result which we prove independently here as part of Theorem 4.4.8, namely that $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma) \succ \mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$; furthermore, they prove that MC-nets are strictly more succinct [Ieong and Shoham, 2005, Propositions 2–3] than the multi-issue representation of coalitional games of Conitzer and Sandholm [2004], and are at least as succinct [Ieong and Shoham, 2005, Proposition 4] as the weighted graphical games of Deng and Papadimitriou [1994]. Elkind et al. [2009] further generalize basic MC-nets to general MC-nets, by additionally permitting arbitrary Boolean connectives in their rules. We discuss two of their succinctness results in Section 4.4.4.

Finally, we note the existence of various absolute succinctness results for the OR/XOR family of languages, which are commonly used as bidding languages in the combinatorial auctions literature. Many of these results exhibit rather precise bounds. For example, Nisan proved that the OR-of-XORs language can express any downward-sloping symmetric valuation on m items using no more than m^2 atomic bids [Nisan, 2000, Lemma 3.4] and that the monochromatic valuation on m items requires at least $2^{m/2+1}$ atomic bids [Nisan, 2000, Theorem 3.5]. These results, and others, are surveyed by Nisan [2006]. We recapitulate two of Nisan's succinctness results much later, in Section 6.3.3, when we discuss the relative succinctness of goalbase languages and the OR/XOR languages.

4.4 Succinctness of Sum Languages

In this section, we give succinctness results for many pairs of sum languages.

4.4.1 Some Basic Succinctness and Equivalence Results

Many succinctness and equivalence results can be arrived at merely by knowing the expressivity of the languages being compared and the basic properties of the succinctness relation contained in Fact 4.2.3.

From Theorem 4.2.4, it follows that all k-languages are pairwise equally succinct. For example, $\mathcal{L}(k\text{-}\mathit{spcubes}, \mathbb{R}, \Sigma) \sim \mathcal{L}(k\text{-}\mathit{forms}, \mathbb{R}^+, \Sigma)$.

Next, we establish the relationships between positive and strictly positive languages:

Lemma 4.4.1. *If Φ is a strictly positive set of formulas, then $\mathcal{L}(\Phi, W, \Sigma) \sim \mathcal{L}(\Phi \cup \{\top\}, W, \Sigma)$.*

Proof. By inclusion, $\mathcal{L}(\Phi, W, \Sigma) \preceq \mathcal{L}(\Phi \cup \{\top\}, W, \Sigma)$. For the converse: Fix $G \in \mathcal{L}(\Phi \cup \{\top\}, W, \Sigma)$. If G does not contain $\top$, then $G \in \mathcal{L}(\Phi, W, \Sigma)$ also. If G contains $\top$, combine all occurrences $(\top, w_1), \ldots, (\top, w_k)$ into a single weighted goal $\left(\top, \sum_{i=1}^{k} w_i\right)$. If $\sum_{i=1}^{k} w_i = 0$, then remove $\top$ to again produce a goalbase in both languages. If instead $\top$ now has nonzero weight, then u_G is not representable in $\mathcal{L}(\Phi, W, \Sigma)$, since u_G is not normalized and by Lemma 3.4.7 only normalized utility functions can be represented using strictly positive formulas (let P be the null property). Therefore, any u representable in both languages has exactly the same representations in both. □

Here we take advantage of unique representations to show that there is no difference in succinctness between the positive and strictly positive versions of several languages:

Theorem 4.4.2.

1. $\mathcal{L}(\mathit{pforms}, W, \Sigma) \sim \mathcal{L}(\mathit{spforms}, W, \Sigma)$,
2. $\mathcal{L}(\mathit{pcubes}, W, \Sigma) \sim \mathcal{L}(\mathit{spcubes}, W', \Sigma)$, *and*
3. $\mathcal{L}(\mathit{pclauses} + \top, W, \Sigma) \sim \mathcal{L}(\mathit{pclauses}, W', \Sigma)$.

Proof. When $W = W'$, the result is a direct consequence of Lemma 4.4.1, giving us the first equivalence. By Theorem 3.4.2, $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$ has unique representations, from which follows that its sublanguages do also, and so any utility function representable in both $\mathcal{L}(\mathit{pcubes}, W, \Sigma)$ and $\mathcal{L}(\mathit{spcubes}, W', \Sigma)$ has the same representation in both, yielding the second equivalence. By Corollary 3.4.4, the same holds for $\mathcal{L}(\mathit{pclauses} + \top, W, \Sigma)$ and $\mathcal{L}(\mathit{pclauses}, W', \Sigma)$, giving the third equivalence. □

The languages $\mathcal{L}(\mathit{spcubes}, \mathbb{R}^+, \Sigma)$ and $\mathcal{L}(\mathit{pclauses}, \mathbb{R}^+, \Sigma)$ are equally succinct due to their limited overlap.

Theorem 4.4.3. $\mathcal{L}(\mathit{spcubes}, \mathbb{R}^+, \Sigma) \sim \mathcal{L}(\mathit{pclauses}, \mathbb{R}^+, \Sigma)$.

Proof. Every utility function expressible in $\mathcal{L}(\mathit{spcubes}, \mathbb{R}^+, \Sigma)$ is supermodular, while every utility function expressible in $\mathcal{L}(\mathit{pclauses}, \mathbb{R}^+, \Sigma)$ is submodular. Let u be such a utility function. The only nonnegative utility functions which are both supermodular and submodular are modular, and so the spcubes representation of u is in 1-spcubes and the pclauses representation is in 1-pclauses. Since 1-spcubes and 1-pclauses are just atoms, u has the same representation in both $\mathcal{L}(\mathit{spcubes}, \mathbb{R}^+, \Sigma)$ and $\mathcal{L}(\mathit{pclauses}, \mathbb{R}^+, \Sigma)$. □

We conclude this section on basic succinctness with an absolute result for $\mathcal{L}(\mathit{atoms}, W, \Sigma)$.

Theorem 4.4.4. *If $G \in \mathcal{L}(atoms, W, \Sigma)$ and contains no duplicate formulas, then there is no other $G' \in \mathcal{L}(forms, \mathbb{R}, \Sigma)$ such that $G \equiv_\Sigma G'$ and* $\text{size}(G') < \text{size}(G)$.

Proof. Fix a $G \in \mathcal{L}(atoms, W, \Sigma)$. Suppose that some $G' \in \mathcal{L}(forms, W', \Sigma)$ distinct from G is such that $G \equiv_\Sigma G'$. Because G contains only atoms and no duplicates, the only way in which G' could be smaller than G is if G' entirely omits some atom p which occurs in G (as (p, w)). Because $u_{G,\Sigma}$ is modular, we have that $u_{G,\Sigma}(\{p\}) = w$, and because G is minimal in $\mathcal{L}(atoms, W, \Sigma)$, $w \neq 0$.

Notice that $M \models \varphi$ is equivalent to $M \cup \{p\} \models \varphi$ when p is not a subformula of φ. Therefore, since p does not occur as a subformula of any formula in G', for every state $p \notin X$ it will be the case that $u_{G',\Sigma}(X) = u_{G',\Sigma}(X \cup \{p\})$. Therefore, $u_{G',\Sigma}(\emptyset) = u_{G',\Sigma}(\{p\})$, while $u_{G,\Sigma}(\emptyset) = 0 \neq w = u_{G,\Sigma}(\{p\})$, which contradicts the hypothesis that $G \equiv_\Sigma G'$. □

In other words, the optimal representations for all nonnegative modular utility functions are the obvious ones in $\mathcal{L}(atoms, W, \Sigma)$.

4.4.2 Equivalence via Goalbase Translation

It is sometimes possible to show that two languages are equally succinct by applying a size-preserving translation to the goalbases in both directions. The following lemma shows that a broad range of languages—those languages which sit between cubes and the union of cubes and clauses, or between clauses and the union of cubes and clauses—are equally succinct. Afterwards, we use this to prove Theorem 4.4.6, which shows that $\mathcal{L}(cubes, \mathbb{R}, \Sigma) \sim \mathcal{L}(clauses, \mathbb{R}, \Sigma)$.

Lemma 4.4.5. *Let Φ and Ψ be sets of formulas. If the following conditions hold,*

- $\Phi \supseteq$ *cubes or* $\Phi \supseteq$ *clauses,*
- $\Psi \supseteq$ *cubes or* $\Psi \supseteq$ *clauses,*
- $\Phi \cup \Psi \subseteq$ *cubes* $\cup$ *clauses,*

then $\mathcal{L}(\Phi, \mathbb{R}, \Sigma) \sim \mathcal{L}(\Psi, \mathbb{R}, \Sigma)$.

Proof. Suppose that $G \in \mathcal{L}(\Phi, \mathbb{R}, \Sigma)$. Enumerate $(\varphi_i, w_i) \in G$. We construct an equivalent goalbase G'. Let

$$G_0 = G$$

$$G_{i+1} = \begin{cases} (G_i \setminus \{(\varphi_i, w_i)\}) \cup \{(\neg\varphi_i, -w_i), (\top, w_i)\} & \text{if } \varphi_i \notin \Psi \\ G_i & \text{otherwise} \end{cases}$$

and let $G' = G_{|G|}$.

The transformation produces an equivalent goalbase: By equivalences (3.5) and (3.6) from Fact 3.4.1, $G_i \equiv G_{i+1}$ for all i, so $G = G_1 \equiv G_2 \equiv \ldots \equiv G_{|G|-1} \equiv G_{|G|} = G'$.

The transformation produces a goalbase in the appropriate language: Suppose that $\varphi \in \Phi$. The set Ψ contains at least every clause or every cube. If φ is a clause, then $\neg\varphi$ is (equivalent to) a cube, and vice versa. Hence at least one of φ and $\neg\varphi$ are in Ψ. $\top$ is both a cube ($\bigwedge \emptyset$) and a clause ($p \vee \neg p$), so $\top \in \Psi$ regardless. Thus $G' \in \mathcal{L}(\Psi, \mathbb{R}, \Sigma)$.

The transformation produces a goalbase as succinct as the original: If φ is a cube, then φ requires the same number of atoms and binary connectives as as $\neg\varphi$ (written as a clause); similarly, if φ is a clause. The only increase in size between G and G' can come from the addition of $\top$, so we have that $|G'| \leq |G| + 1$.

Therefore, $\mathcal{L}(\Phi, \mathbb{R}, \Sigma) \succeq \mathcal{L}(\Psi, \mathbb{R}, \Sigma)$. By the same argument $\mathcal{L}(\Phi, \mathbb{R}, \Sigma) \preceq \mathcal{L}(\Psi, \mathbb{R}, \Sigma)$. So $\mathcal{L}(\Phi, \mathbb{R}, \Sigma) \sim \mathcal{L}(\Psi, \mathbb{R}, \Sigma)$. □

Theorem 4.4.6. $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma) \sim \mathcal{L}(\mathit{clauses}, \mathbb{R}, \Sigma)$.

Proof. Follows immediately from Lemma 4.4.5. □

4.4.3 Strict Succinctness and Incomparability, by Counterexample

The most straightforward method for showing that one language is not more succinct than another is to produce a family of utility functions whose representations grow exponentially in the first language but merely polynomially in the second language. Here we define two families of utility functions which will be used repeatedly for demonstrating strict succinctness and incomparability results.

Definition 4.4.7. Let $u_n^\forall$ and $u_n^\exists$ be the utility functions over $\mathcal{PS}_n$ where

$$u_n^\forall(X) = \begin{cases} 1 & \text{if } X = \mathcal{PS} \\ 0 & \text{otherwise,} \end{cases} \qquad \text{and} \qquad u_n^\exists(X) = \begin{cases} 1 & \text{if } X \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

In all cases in this section where we show that one language is strictly less succinct than another, we rely on the fact that the less succinct language has unique representations in order to rule out representations smaller than the exponential ones we will exhibit.

Theorem 4.4.8.

$$\left.\begin{array}{c} \mathcal{L}(\mathit{pclauses}, \mathbb{R}, \Sigma) \\ \mathcal{L}(\mathit{pclauses} + \top, \mathbb{R}, \Sigma) \end{array}\right\} \prec \mathcal{L}(\mathit{clauses}, \mathbb{R}, \Sigma)$$

$$\left.\begin{array}{c} \mathcal{L}(\mathit{spcubes}, \mathbb{R}, \Sigma) \\ \mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma) \end{array}\right\} \prec \mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma)$$

Proof. $\mathcal{L}(pcubes, \mathbb{R}, \Sigma) \preceq \mathcal{L}(cubes, \mathbb{R}, \Sigma)$ since every pcube is a cube. Consider the family of utility functions $u_n^\exists$, which may be represented in cubes as

$$\left\{(\top, 1), \left(\bigwedge\{\neg p \mid p \in \mathcal{PS}\}, -1\right)\right\}$$

the length of which increases linearly with n. $u_n^\exists$ may be represented in pcubes as

$$\left\{\left(\bigwedge X, w_{\bigwedge X}\right) \;\middle|\; \emptyset \subset X \subseteq \mathcal{PS}\right\} \quad \text{where} \quad w_{\bigwedge X} = \begin{cases} 1 & \text{if } |X| \text{ is odd} \\ -1 & \text{if } |X| \text{ is even.} \end{cases}$$

Every pcube except $\top$ receives a nonzero weight, and and by Theorem 3.4.2 this representation is unique. For any n, $2^n - 1$ pcubes are weighted, so the size of the representation increases exponentially with n.

For $\mathcal{L}(pclauses\ {+}\top, \mathbb{R}, \Sigma) \prec \mathcal{L}(clauses, \mathbb{R}, \Sigma)$, replace "$\exists$", "$\wedge$", "pcubes", "cubes", and "Theorem 3.4.2" in the above proof with "$\forall$", "$\vee$", "pclauses $+\top$", "clauses", and "Theorem 3.4.3", respectively. □

Corollary 4.4.9.

$$\left.\begin{array}{c} \mathcal{L}(spcubes, \mathbb{R}, \Sigma) \\ \mathcal{L}(pclauses, \mathbb{R}, \Sigma) \\ \mathcal{L}(pcubes, \mathbb{R}, \Sigma) \\ \mathcal{L}(pclauses\ {+}\top, \mathbb{R}, \Sigma) \end{array}\right\} \prec \mathcal{L}(forms, \mathbb{R}, \Sigma)$$

Proof. Immediately from Fact 4.2.3.3 and Theorem 4.4.8. □

Next, we exploit the fact that languages based on positive cubes favor $u_n^\forall$ while languages based on positive clauses favor $u_n^\exists$ to show that pcubes and pclauses languages are incomparable.

Theorem 4.4.10.

$$\left.\begin{array}{c} \mathcal{L}(pcubes, \mathbb{R}, \Sigma) \\ \mathcal{L}(spcubes, \mathbb{R}, \Sigma) \end{array}\right\} \perp \left\{\begin{array}{c} \mathcal{L}(pclauses, \mathbb{R}, \Sigma) \\ \mathcal{L}(pclauses\ {+}\top, \mathbb{R}, \Sigma) \end{array}\right.$$

Proof. ($\not\succ$) The family of utility functions $u_n^\exists$ is represented uniquely and linearly as $\{(\bigvee \mathcal{PS}, 1)\}$ in pclauses. When representing $u_n^\exists$ in pcubes, the weights $w_{\bigwedge X} = (-1)^{|X|+1}$, so the unique representation there assigns nonzero weights to $2^n - 1$ distinct pcubes.

($\not\prec$) The family of utility functions $u_n^\forall$ is represented uniquely and linearly as $\{(\bigwedge \mathcal{PS}, 1)\}$ in pcubes, but the representation in pclauses is exponential, as shown in the proof of Theorem 4.4.8. □

Note that from $\mathcal{L}(pcubes, \mathbb{R}, \Sigma) \perp \mathcal{L}(pclauses\ {+}\top, \mathbb{R}, \Sigma)$ we can also conclude, using Fact 4.2.3.4, that $\mathcal{L}(pcubes, \mathbb{R}, \Sigma), \mathcal{L}(pclauses\ {+}\top, \mathbb{R}, \Sigma) \prec \mathcal{L}(pforms, \mathbb{R}, \Sigma)$,

because positive formulas are a superset of the union of positive cubes and positive clauses.

We now make the same comparison as in the previous theorem, but with the weights of one language in each pair restricted to $\mathbb{R}^+$. In doing so, we maintain nonsuccinctness in one direction.

Theorem 4.4.11.

$$\left.\begin{matrix}\mathcal{L}(\mathit{pclauses}, \mathbb{R}, \Sigma) \\ \mathcal{L}(\mathit{pclauses} + \top, \mathbb{R}, \Sigma)\end{matrix}\right\} \not\succeq \begin{Bmatrix}\mathcal{L}(\mathit{spcubes}, \mathbb{R}^+, \Sigma) \\ \mathcal{L}(\mathit{pcubes}, \mathbb{R}^+, \Sigma)\end{Bmatrix}$$

$$\left.\begin{matrix}\mathcal{L}(\mathit{spcubes}, \mathbb{R}, \Sigma) \\ \mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)\end{matrix}\right\} \not\succeq \begin{Bmatrix}\mathcal{L}(\mathit{pclauses}, \mathbb{R}^+, \Sigma) \\ \mathcal{L}(\mathit{pclauses} + \top, \mathbb{R}^+, \Sigma)\end{Bmatrix}$$

Proof. The first part is demonstrated by the $u_n^\forall$ family of functions, the second part by the $u_n^\exists$ family. □

However, it is unknown whether the $\not\preceq$ direction also holds for these languages.

The following theorem shows how even seemingly rather wasteful languages like $\mathcal{L}(\mathit{complete\ cubes}, \mathbb{R}, \Sigma)$ may in some cases have representational advantages over more parsimonious ones:

Theorem 4.4.12. $\mathcal{L}(\mathit{complete\ cubes}, \mathbb{R}, \Sigma) \perp \mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$

Proof. This follows directly from a result of Chevaleyre et al. [2008a, Propositions 4–5], with the utility functions providing the counterexamples found by Chevaleyre et al. [2006, p. 150]. We note that $\mathcal{L}(\mathit{complete\ cubes}, \mathbb{R}, \Sigma)$ has unique representations, due to the fact that no two complete cubes are ever true simultaneously. The necessary counterexamples are $u(M) = |M|$, which is represented in $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$ as $\{(a, 1) \mid a \in \mathcal{PS}\}$ but is the very large

$$\left\{ \left(\bigwedge (M \cup \neg \bar{M}), |M| \right) \;\middle|\; \emptyset \subset M \subseteq \mathcal{PS} \right\}$$

in $\mathcal{L}(\mathit{complete\ cubes}, \mathbb{R}, \Sigma)$; and

$$u(M) = \begin{cases} 1 & \text{if } |M| = 1 \\ 0 & \text{otherwise,} \end{cases}$$

which is the very large

$$\left\{ \left(\bigwedge X, |X| \cdot (-1)^{|X|-1} \right) \;\middle|\; \emptyset \subset X \subseteq \mathcal{PS} \right\}$$

in $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$ but the much smaller

$$\left\{ \left(\bigwedge (M \cup \neg \bar{M}), 1 \right) \;\middle|\; |M| = 1 \right\}$$

in $\mathcal{L}(\mathit{complete\ cubes}, \mathbb{R}, \Sigma)$. □

Furthermore, it follows that $\mathcal{L}(\textit{complete cubes}, \mathbb{R}, \Sigma) \prec \mathcal{L}(\textit{cubes}, \mathbb{R}, \Sigma)$, due to Fact 4.2.3.4, since all positive cubes and complete cubes are cubes.

Finally, a comment about the method used in this section: While it has proved a productive one for us, it has three serious drawbacks. First, this strategy requires us to construct a family of utility functions and their representations in two different languages, but not just any family of utility functions will do. In every case where we have proven strict succinctness, it is not *strong*, i.e., we have shown that there are no utility functions which have significantly larger representations in one language, and some which have significantly smaller representations in the same language (in order to get $\prec$)—but there are also some utility functions which have similarly-sized representations in both languages. In fact, we know of no pair of languages where the smallest representation of *every* utility function in one language is exponentially larger (or even just larger) than the smallest representation in another. Second, proving that a language has unique representations is generally not trivial, as seen in Section 3.4.2. Third, and most seriously, it is easy to demonstrate that many more expressive languages lack the uniqueness property. This proof strategy depends crucially on being able to produce a large representation in the less succinct language while at the same time being certain that no smaller representation exists there. When the less succinct language lacks unique representations, mere possession of a bad representation of a utility function provides us with no assurance that we haven't overlooked a much smaller representation of that utility function in the same language.

4.4.4 Strict Succinctness, Nonconstructively

It is difficult to demonstrate that a language which lacks unique representations is less succinct than another language, because the exhibition of a single exponentially-growing family of utility functions (as above) does not preclude the existence of better representations in the same language. Here, we take a nonconstructive approach to produce the following strict succinctness result, due to Yann Chevaleyre:

Theorem 4.4.13. $\mathcal{L}(\textit{cubes}, \mathbb{R}, \Sigma) \prec \mathcal{L}(\textit{forms}, \mathbb{R}, \Sigma)$.

To prove this theorem, we will introduce the Fourier transform on Boolean domains, using the same notation as Mansour [1994]. Then, to apply the Fourier transform on cubes, we will need two lemmas. The first one will show how the size of cubes relates to their degree. The second lemma will show that a function which approximates parity accurately necessarily has a high degree.

For each $S \subseteq \mathcal{PS}$, the *parity function* $\chi_S \colon 2^{\mathcal{PS}} \to \{-1, 1\}$ is defined as

$$\chi_S(X) = (-1)^{|S \cap X|}.$$

Because these functions form an orthonormal basis for the space of real functions on $2^{\mathcal{PS}}$, any function $f \colon 2^{\mathcal{PS}} \to \mathbb{R}$ can be represented as a linear combination with

respect to this basis. This is known as the *Fourier-Walsh expansion*:

$$f(X) = \sum_{S \subseteq \mathcal{PS}} \hat{f}(S)\chi_S(X),$$

where the $\hat{f}(S) \in \mathbb{R}$ are the Fourier coefficients, which are computed as follows:

$$\hat{f}(S) = \frac{1}{2^n} \sum_{X \subseteq \mathcal{PS}} f(X)\chi_S(X)$$

for all $S \subseteq \mathcal{PS}$. The degree of a function f is the cardinality of the largest subset of S with a nonzero Fourier coefficient: $\deg(f) = \max\{|S| \mid \hat{f}(S) \neq 0\}$.

Lemma 4.4.14. *If $G \in \mathcal{L}(k\text{-cubes}, \mathbb{R}, \Sigma)$, then the degree of u_G will be at most k.*

Proof. Let us first show the lemma under the condition that G contains a single cube of at most k literals. Let $y \in \mathcal{PS}$ be any variable not present in that cube. For all $S \subseteq \mathcal{PS}$ such that $y \in S$, the Fourier coefficients $\hat{u}_G(S)$ are the following:

$$\begin{aligned}\hat{u}_G(S) &= \frac{1}{2^n} \sum_{X \subseteq \mathcal{PS},\, y \notin X} u_G(X)\chi_S(X) + \frac{1}{2^n} \sum_{X \subseteq \mathcal{PS},\, y \in X} u_G(X)\chi_S(X) \\ &= \frac{1}{2^n} \sum_{X \subseteq \mathcal{PS} \setminus \{y\}} \chi_S(X)\big(u_G(X) - u_G(X \cup \{y\})\big) = 0.\end{aligned}$$

Therefore, if $\hat{u}_G(S) \neq 0$ then S contains only variables present in the cube, thus $|S| \leq k$. Thus, the degree of u_G is at most k. Suppose now that G contains more than one cube. Then, u_G can be seen as a linear combination of utilities each generated by single cubes. Because the Fourier transform is linear (in other words, if $f = g + h$ then $\hat{f} = \hat{g} + \hat{h}$), the degree of u_G is also bounded by k. □

The next lemma is familiar from the literature on bounding the complexity of Boolean circuits. The proof is inspired by the lecture notes of Trevisan [2004, Lemma 4].

Lemma 4.4.15. *There are some constants $c > 0$ and $n_0 > 0$ (such constants are completely independent from n) such that, if $n \geq n_0$, then given any function $g\colon 2^{\mathcal{PS}} \to \mathbb{R}$ that agrees with the parity function $\chi_{\mathcal{PS}}$ on at least $\frac{3}{4}$ of $2^{\mathcal{PS}}$, the degree of g will be at least $c\sqrt{n}$.*

Proof. Let $g\colon 2^{\mathcal{PS}} \to \mathbb{R}$ be a function that of $2^{\mathcal{PS}}$. Let t be the degree of g. Let $A = \{X \subseteq \mathcal{PS} \mid g(X) = \chi_{\mathcal{PS}}(X)\}$, which by definition has the property $|A| \geq \frac{3}{4}2^n$ where $n = |\mathcal{PS}|$. Clearly, for any $S \subseteq \mathcal{PS}$ and $X \in A$, we have $\chi_S(X) = \chi_{\mathcal{PS}}(X)\chi_{\mathcal{PS}\setminus S}(X) = g(X)\chi_{\mathcal{PS}\setminus S}(X)$. Note that the function $\chi_S(X)$ has a degree equal to $|S|$, but can be replaced over A by $g(X)\chi_{\mathcal{PS}\setminus S}(X)$, which has a degree of at most $t + n - |S|$. Consequently, any function χ_S over A with $|S| \geq \frac{n}{2}$

can be replaced by its Fourier-Walsh expansion, which is a linear combination over the set of functions $\mathcal{F} = \{\chi_{S'} \mid S' \subseteq \mathcal{PS}, |S'| \leq t + \frac{n}{2}\}$.

The Fourier transform guarantees that any function $f\colon A \to \mathbb{R}$ can be written as a linear combination over $\{\chi_S \mid S \subseteq \mathcal{PS}\}$. But because each of these functions χ_S over A can itself be decomposed over $\mathcal{F}$, $f\colon A \to \mathbb{R}$ can be written as a linear combination over $\mathcal{F}$ as follows:

$$f(X) = \sum_{S \subseteq \mathcal{PS}, |S| \leq t + \frac{n}{2}} \alpha_S \cdot \chi_S(X),$$

with $\alpha_S \in \mathbb{R}$. The number of α_S coefficients is $\sum_{k=0}^{t+\frac{n}{2}} \binom{n}{k}$. However, because the set of functions $f\colon A \to \mathbb{R}$ forms a vector space over the reals of dimension $|A|$, the number of α_S coefficients must be at least $\frac{3}{4}2^n$. This leads to the inequality

$$\sum_{k=\frac{n}{2}}^{t+\frac{n}{2}} \binom{n}{k} \geq \frac{2^n}{4}$$

which, after applying Stirling's approximation and some basic formula manipulation, becomes $t = \Omega(\sqrt{n})$. □

We are now in position to prove Theorem 4.4.13.

Proof. (Theorem 4.4.13.) In the first part of the proof, we will show that the function $\chi_{\mathcal{PS}}$ can be polynomially represented in $\mathcal{L}(\mathit{forms}, \mathbb{R}, \Sigma)$, and in the second part, we will show that this is not the case for $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma)$. Let us prove the first part. It is known that the parity function can be written as a Boolean AND/OR formula φ_{parity} containing at most n^2 literals [Lee, 2006, p. 100]. We can then build the goalbase $G = \{(\top, -1), (\varphi_{\text{parity}}, 2)\}$ which generates $\chi_{\mathcal{PS}}$ with a polynomial number of literals.

Let us now prove the second part. More precisely, we will show that in order to represent $\chi_{\mathcal{PS}}$ in $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma)$, at least $2^{\Omega(\sqrt{n})}$ cubes are required. Consider a goalbase $G = \{(\varphi_i, \alpha_i)\}_i$ where φ_i are cubes (possibly containing negative literals). Let G_{low} be all the pairs (φ_i, α_i) of G such that the number of literals in φ_i is strictly lower than $c\sqrt{n}$, where the constant c is chosen as in Lemma 4.4.15. Let $G_{\text{high}} = G \setminus G_{\text{low}}$. Let $u_{G_{\text{low}}}$ be the utility function generated by G_{low}. Together with Lemma 4.4.14, we can now apply Lemma 4.4.15 which implies that $u_{G_{\text{low}}}$ disagrees with $\chi_{\mathcal{PS}}$ on at least a $\frac{1}{4}$ fraction of $2^{\mathcal{PS}}$. In order for u_G to compute the parity function, the cubes of G_{high} must compensate for the errors made by those of G_{low} on this $\frac{1}{4}$ fraction of $2^{\mathcal{PS}}$, but we will show that this compensation requires a very large number of cubes. Let us thus evaluate the fraction of $2^{\mathcal{PS}}$ which can be affected by the cubes of G_{high}. Because each cube has at least $c\sqrt{n}$ literals, at most $2^{n-c\sqrt{n}}$ interpretations will be affected by each of these cubes. Thus, to affect $\frac{1}{4}$ fraction of $2^{\mathcal{PS}}$, G_{high} will need to have at least $\frac{\frac{2^n}{4}}{2^{n-c\sqrt{n}}} = 2^{c\sqrt{n}-2}$ cubes. □

Corollary 4.4.16. $\mathcal{L}(\mathit{clauses}, \mathbb{R}, \Sigma) \prec \mathcal{L}(\mathit{forms}, \mathbb{R}, \Sigma)$.

Proof. Follows immediately from Theorems 4.4.6 and 4.4.13, Fact 4.2.3.5, and Corollary 3.4.9. □

Elkind et al. [2009, Theorem 3.2] independently derived an entirely different proof of our Theorem 4.4.13, in the context of representing a particular ill-behaved family of coalitional games as basic marginal contribution nets (MC-nets). (For a discussion of MC-nets, see Section 4.3.) Using a probabilistic argument, Elkind et al. determined that the MC-nets generated by this family of coalitional games required $\Omega((\frac{3}{2})^{n/2})$ basic rules.

Furthermore, Elkind et al. [2009, Example 3.1 and Theorem 3.2] give a succinctness result which compares Ieong and Shoham's basic MC-nets with general MC-nets. Because basic MC-nets are effectively goalbases in $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma)$ and general MC-nets are goalbases in $\mathcal{L}(\mathit{forms}, \mathbb{R}, \Sigma)$, we can adapt their proof to arrive at another nonconstructive succinctness result:

Theorem 4.4.17. $\mathcal{L}(\mathit{cubes}, \mathbb{R}^+, \Sigma) \prec \mathcal{L}(\mathit{forms}, \mathbb{R}^+, \Sigma)$.

Proof. $\mathcal{L}(\mathit{cubes}, \mathbb{R}^+, \Sigma) \preceq \mathcal{L}(\mathit{forms}, \mathbb{R}^+, \Sigma)$ by inclusion. For strict succinctness: Enumerate $\mathcal{PS}_{2n} = \{x_1, x_2, \ldots, x_{2n-1}, x_{2n}\}$ and define the family of utility functions

$$u_{2n}(X) = \begin{cases} 1 & \text{if } x_{2i-1} \in X \text{ or } x_{2i} \in X, \text{ for all } 1 \leq i \leq n \\ 0 & \text{otherwise.} \end{cases}$$

The goalbase $\{((x_1 \vee x_2) \wedge \ldots \wedge (x_{2n-1} \vee x_{2n}), 1)\}$ represents $u_{2n} \in \mathcal{L}(\mathit{spforms}, \mathbb{R}^+, \Sigma)$ linearly.

Because u_{2n} is nonnegative, it has a representation $G \in \mathcal{L}(\mathit{cubes}, \mathbb{R}^+, \Sigma)$. If $(\varphi, w) \in G$ and $w > 0$, then for each $1 \leq i \leq n$, φ contains at least one of x_{2i-1} and x_{2i} as a literal: Suppose otherwise, and let X be a state where $X \models \varphi$ but x_{2i-1} and x_{2i} are false. Then $u_{2n}(X) \geq w$ since G contains no negative weights; but $u_{2n}(X) = 0$, and so $w = 0$, contrary to assumption. Next, consider the states X where $u_{2n}(X) = 1$ and for any state $Z \subset X$, $u_{2n}(Z) = 0$. For any two such minimal nonzero states X, Y, they must differ on at least two atoms p, q. (If X and Y differed on only one atom, then $X \subset Y$ or $Y \subset X$, contradicting minimality.) Therefore, every $(\varphi, w) \in G$ such that $X \models \varphi$ contains a literal p but not q and vice versa for every $(\psi, w') \in G$ such that $Y \models \psi$. Since each minimal state has at least one (φ, w) which is true there but in no other minimal state, and there are 2^n such minimal states, $|G| \geq 2^n$, and so $\text{size}(G) \in O(2^{|\mathcal{PS}|})$. □

The bulk of this proof shows that $\mathcal{L}(\mathit{spforms}, \mathbb{R}^+, \Sigma) \not\preceq \mathcal{L}(\mathit{cubes}, \mathbb{R}^+, \Sigma)$. This, combined with the contrapositive of Fact 4.2.3.2, produces many of the $\not\preceq$ results seen in Table 4.1.

	$\mathcal{L}(spcubes, \mathbb{R}^+, \Sigma)$	$\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$	$\mathcal{L}(spforms, \mathbb{R}^+, \Sigma)$	$\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$	$\mathcal{L}(pclauses + \top, \mathbb{R}^+, \Sigma)$	$\mathcal{L}(pforms, \mathbb{R}^+, \Sigma)$	$\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$	$\mathcal{L}(clauses, \mathbb{R}^+, \Sigma)$	$\mathcal{L}(forms, \mathbb{R}^+, \Sigma)$	$\mathcal{L}(spcubes, \mathbb{R}, \Sigma)$	$\mathcal{L}(pclauses, \mathbb{R}, \Sigma)$	$\mathcal{L}(spforms, \mathbb{R}, \Sigma)$	$\mathcal{L}(pcubes, \mathbb{R}, \Sigma)$	$\mathcal{L}(pclauses + \top, \mathbb{R}, \Sigma)$	$\mathcal{L}(pforms, \mathbb{R}, \Sigma)$	$\mathcal{L}(cubes, \mathbb{R}, \Sigma)$	$\mathcal{L}(clauses, \mathbb{R}, \Sigma)$	$\mathcal{L}(forms, \mathbb{R}, \Sigma)$
$\mathcal{L}(forms, \mathbb{R}, \Sigma)$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$
$\mathcal{L}(clauses, \mathbb{R}, \Sigma)$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\simeq$
$\mathcal{L}(cubes, \mathbb{R}, \Sigma)$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\simeq$
$\mathcal{L}(pforms, \mathbb{R}, \Sigma)$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\simeq$	$\simeq$
$\mathcal{L}(pclauses + \top, \mathbb{R}, \Sigma)$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\simeq$	$\simeq$	$\simeq$
$\mathcal{L}(pcubes, \mathbb{R}, \Sigma)$	$\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(spforms, \mathbb{R}, \Sigma)$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(pclauses, \mathbb{R}, \Sigma)$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(spcubes, \mathbb{R}, \Sigma)$	$\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(forms, \mathbb{R}^+, \Sigma)$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(clauses, \mathbb{R}^+, \Sigma)$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(pforms, \mathbb{R}^+, \Sigma)$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(pclauses + \top, \mathbb{R}^+, \Sigma)$	$\simeq$	$\simeq$	$\not\simeq$	$\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$	$\simeq$	$\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(spforms, \mathbb{R}^+, \Sigma)$	$\not\simeq$	$\not\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$	$\simeq$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$
$\mathcal{L}(spcubes, \mathbb{R}^+, \Sigma)$	$\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$	$\not\simeq$

Table 4.1: Summary of succinctness results for sum languages. Entries to be read row first. Empty cells are open questions.

4.4.5 Summary

Our succinctness results for sum languages are summarized in Table 4.1. The table contains many more results than are proved in the text, but in all cases these are straightforward consequences of results which do appear in the text. (E.g., $\mathcal{L}(\mathit{spcubes}, \mathbb{R}, \Sigma) \prec \mathcal{L}(\mathit{clauses}, \mathbb{R}, \Sigma)$ follows immediately from Theorems 4.4.6 and 4.4.8.) There are many open questions (any cell which contains neither $\prec$, $\succ$, nor $\sim$ has something yet to be resolved). All open questions involve at least one language which lacks unique representations. Most cases in which nothing is known involve a language which uses positive formulas or general formulas. We suspect that resolving these questions will require difficult proofs, as the one for Theorem 4.4.13 which shows that $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \Sigma) \prec \mathcal{L}(\mathit{forms}, \mathbb{R}, \Sigma)$.

Finally, it is worth noting that $\sim$ is intransitive, due to the succinctness relation being defined over languages which may differ in expressivity. E.g., $\mathcal{L}(\mathit{atoms}, \mathbb{R}, \Sigma)$ is equally succinct as any other language, so $\mathcal{L}(\mathit{atoms}, \mathbb{R}, \Sigma) \sim \mathcal{L}_1$ and $\mathcal{L}(\mathit{atoms}, \mathbb{R}, \Sigma) \sim \mathcal{L}_2$, but it is still possible that $\mathcal{L}_1 \nsim \mathcal{L}_2$.

4.5 Succinctness of Max Languages

In this section, we turn to the investigation of the succinctness of languages using max as their aggregator. In addition to examining comparative succinctness of max languages, we also address absolute succinctness for $\mathcal{L}(\mathit{pcubes}, W, \max)$ and $\mathcal{L}(\mathit{cubes}, W, \max)$.

4.5.1 Absolute Succinctness

In absolute terms, there is a strong dependency of the size of representations in max languages on the size of the range of the utility function being represented:

Theorem 4.5.1. *For any goalbase G, $|G| \geq |\operatorname{ran} u_{G,\max}|$.*

Proof. By definition, $u_{G,\max}(X) = \max\{w_\varphi \mid X \models \varphi\}$, so for every state X there must be some $w_\varphi = u(X)$. □

While the size of range of a utility function serves as a lower bound on the size of its representation in any max language, there is no such relationship for sum languages: E.g., if $G = \{(a_i, 2^i) \mid a_i \in \mathcal{PS}\}$, then $u_{G,\Sigma}$ has a large range (every value in $0, \ldots, 2^{|\mathcal{PS}|} - 1$) despite that G is itself small (using only $|\mathcal{PS}|$ atoms).

By Theorem 4.5.1 we have that if $|G_n|$ is polynomial then $|\operatorname{ran} u_{G_n}|$ is polynomial. However, the converse does not hold: Let

$$G_n = \left\{ \left(\bigwedge X, 1 \right) \;\middle|\; |X| = \frac{n}{2} \right\} \cup \{(\top, 0)\}.$$

Here, $|\operatorname{ran} u_{G_n}| = 2$ but $|G_n| = \binom{n}{n/2} + 1$ is superpolynomial and G_n is minimal in $\mathcal{L}(pcubes, \mathbb{R}^+, \max)$.

There is a clear connection between superfluity (*cf.* Definition 3.2.3) and goalbase minimality:

Fact 4.5.2. *If G is a minimal goalbase for a utility function $u \in \mathcal{U}(\Phi, W, \max)$, then G contains no superfluous formulas.*

The converse does not hold for $\mathcal{L}(forms, \mathbb{R}, \max)$: $\{(p, 1), (\neg p, 1)\}$ and $\{(\top, 1)\}$ represent the same utility function $u(X) = 1$, yet neither formula in the larger goalbase is superfluous.

Recall from Definition 3.2.4 that a language can have unique representations, meaning that it is sufficiently restrictive as to have exactly one minimal representation of each representable utility function. Several sum languages have unique representations, as discussed in Section 3.4.2. This also occurs for at least one max language:

Theorem 4.5.3. $\mathcal{L}(pcubes, \mathbb{R}, \max)$ *has unique representations.*

Proof. Fix $u \in \mathcal{U}(pcubes, \mathbb{R}, \max)$. Let $G_0 = \emptyset$. While $u_{G_i,\max} \neq u$: Choose a least state X for which $u_{G_i,\max}(X) \neq u(X)$. (By *least*, we mean that $|X|$ is minimal.) Let $G_{i+1} = G_i \cup \{(\bigwedge X, u(X))\}$. Call G the G_i at which the algorithm terminates.

Correctness: $u_{G,\max} = u$ because each iteration ends with one more state correct than in the previous iteration, and there are finitely many states. Setting a weight for $\bigwedge X$ cannot disturb the value of any state $Y \subset X$, as X is the least state where $\bigwedge X$ is true, and cannot prevent us from correctly setting the value of any state $Z \supset X$ during subsequent iterations, because by Theorem 3.5.9 u is monotone. Note also that the order of choice of cubes of the same size makes no difference in the outcome.

Minimality: For any state X, either $\bigwedge X$ receives a weight or not. If $\bigwedge X$ receives a weight, then there is no state $Y \subset X$ for which $(\bigwedge Y, u(Y))$ dominates $(\bigwedge X, u(X))$. (Recall from Definition 3.5.1 that a $(\varphi, w_\varphi) \in G$ is dominated if there exists a $(\psi, w_\psi) \in G$ such that $\varphi \models \psi$ and $w_\varphi < w_\psi$.) Furthermore, there is no state $Z \supset X$ for which $X \models \bigwedge Z$. Hence, if the algorithm assigns a weight to $\bigwedge X$, then this is the sole way in which we can make $u_{G,\max}(X) = u(X)$. If, on the other hand, the algorithm produces a G where $\bigwedge X$ receives no weight, then at some step i in the construction $u_{G_i,\max}(X)$ became correct before we reached state X. If we were to set a weight for $\bigwedge X$, it would be superfluous and so G would not be minimal. In summary: Any smaller G will give an incorrect value for some state, and any different, yet still correct, G will necessarily contain a superfluous formula. □

Note that while this algorithm does show how to construct the minimal representation for any representable utility function in $\mathcal{L}(pcubes, W, \max)$, it is

not an efficient algorithm for finding representations, as it requires us to check exponentially many states in order to set weights for them.

Next, we derive upper and lower bounds for the size of representations in $\mathcal{L}(\mathit{cubes}, W, \max)$, but first we prove several technical lemmas which we will need. For the remainder of this section, we assume that all $u_{G,\max}$ are total. First, recall that *active* formulas in max-aggregated goalbases are those which have a weight equal to the value of some state where they are true, from Definition 3.5.1.

Lemma 4.5.4. *Fix a goalbase $G \in \mathcal{L}(\mathit{cubes}, W, \max)$ and a $(\varphi, w) \in G$. If a state X has an extension $Y \supset X$ such that $u_{G,\max}(Y) < u_{G,\max}(X)$ and (φ, w) is active in X, then φ is not a positive cube.*

Proof. Suppose otherwise. Let (φ, w) be such that $X \models \varphi$, $u_{G,\max}(X) = w$, φ a pcube, and $u_{G,\max}(Y) < u_{G,\max}(X)$. Then for all $Y \supset X$ it follows that $Y \models \varphi$ because φ is a monotone formula. So $u_{G,\max}(Y) \geq w = u_{G,\max}(X)$, contrary to hypothesis. □

In words: If a cube is active in a state which can decline in value when extended, then there must be a negative literal in that cube.

Here we define the $X{\uparrow}$ notation for denoting the set of extensions of X, which is used throughout the remainder of this section:

Definition 4.5.5. If X is a state, then $X{\uparrow} = \{Y \mid X \subseteq Y \subseteq \mathcal{PS}\}$.

The set of states $2^{\mathcal{PS}}$ may be thought of as a Boolean lattice; then $X{\uparrow}$ is the sublattice rooted at X. Alternatively, $X{\uparrow}$ may be thought of as all of the ways of extending X.

Lemma 4.5.6. *Suppose that w is the minimum value of any state in $X{\uparrow}$. Let $(\varphi_1, w), \ldots, (\varphi_k, w) \in G \in \mathcal{L}(\mathit{cubes}, W, \max)$ be the formulas which are true in at least one state in $X{\uparrow}$, false outside of $X{\uparrow}$, and have weight w. Let $G' = G \setminus \{(\varphi_i, w)\}_{1 \leq i \leq k} \cup \{(\bigwedge X, w)\}$. Then:*

1. *$G' \equiv_{\max} G$.*
2. *$\text{size}(G') \leq \text{size}(G)$.*

Proof. For 1, we must show that the changes made to G to get G' result in no states being disturbed from their original values. The formula $\bigwedge X$ is true exactly in $X{\uparrow}$ and nowhere else, so it disturbs no states outside of $X{\uparrow}$. Adding $(\bigwedge X, w)$ disturbs no states in $X{\uparrow}$, since $(\bigwedge X, w)$ is inactive in any Y where $u(Y) > w$, and provides the correct value in the remaining states in $X{\uparrow}$ since w is minimal there. Every $\varphi_i \models \bigwedge X$, so if $M \models \varphi_i$ then $M \models \bigwedge X$ also, which covers all states where a φ_i was active.

For 2: $\bigwedge X$ is not longer than $\varphi_1, \ldots, \varphi_k$: $\bigwedge X$ is the shortest formula which is true only in $X{\uparrow}$. Each φ_i is true only in $X{\uparrow}$ also, so $\text{size}(\varphi_i) \geq \text{size}(\bigwedge X)$, and therefore $\sum_{i=1}^{k} \text{size}(\varphi_i) \geq \text{size}(\bigwedge X)$ (strictly larger if $k > 1$). □

This lemma permits us to reduce iteratively any goalbase in $\mathcal{L}(\mathit{cubes}, W, \max)$: Let $G = G_0$. Apply the reduction to the smallest sublattice $X{\uparrow}$ of G_i to which it has not yet been applied, and let the result be G_{i+1}. (Starting from the smallest sublattice means starting with $\mathcal{PS}$ as the root and working our way downwards to $\emptyset$.) At some stage i, we will reach a fixed point—that is, $G_i = G_\infty$—where no further applications of the reduction will have any effect. (The upper bound for reaching a fixed point happens to be $i = 2^{|\mathcal{PS}|}$, though all we require here is that it happens after finitely many applications of the reduction.) Let $G' = G_\infty$, and call such a G' *pcubes-minimal*.

Since this reduction is never size-increasing, we may make use of it to observe a useful fact about the formulas in minimal goalbases in $\mathcal{L}(\mathit{cubes}, W, \max)$:

Lemma 4.5.7. *For every $G \in \mathcal{L}(\mathit{cubes}, W, \max)$, there is a minimal $G' \equiv_{\max} G$ such that $(\bigwedge X, w) \in G'$ iff $w = \min_{Z \in X{\uparrow}} u(Z)$ and $u(Y) < w$ for every $Y \subset X$.*

Proof. Suppose that $G'' \equiv_{\max} G$ and G'' is minimal. Let G' be the result of exhaustively applying the reduction in Lemma 4.5.6 to G''. Since the reduction is equivalence-preserving and size-reducing, $G' \equiv_{\max} G''$ and $\text{size}(G') \leq \text{size}(G'')$. Since G'' was already minimal, G' cannot be smaller, so $\text{size}(G') = \text{size}(G'')$ and G' is also minimal.

($\Rightarrow$) Suppose that $(\bigwedge X, w) \in G'$. Then $(\bigwedge X, w)$ was not eliminated by the reduction. Since $\bigwedge X$ is a positive cube, it is true exactly in $X{\uparrow}$ and nowhere else. If there were some state $Y \subset X$ for which $u(Y) \geq w$, then the reduction would have eliminated $(\bigwedge X, w)$ and replaced it with $(\bigwedge Y, w)$ instead, so it must be the case that $u(Y) < w$ for all $Y \subset X$. For the other condition, suppose that there is a state $Z \in X{\uparrow}$ such that $u(Z) < w$. Since $Z \models \bigwedge X$ and max is our aggregator, it follows that $u(Z) \geq w$, which is a contradiction.

($\Leftarrow$) Suppose that $w = \min_{Z \in X{\uparrow}} u(Z)$ and $u(Y) < w$ for every $Y \subset X$. Since w is the minimal state value in $X{\uparrow}$, it follows that there is a state $Z \supseteq X$ for which $u(Z) = w$. In order for $u(Z) = w$, we need a formula $(\varphi, w) \in G'$ such that $Z \models \varphi$. Because $u(Y) < w$ for all $Y \subset X$, it must also be the case that $Y \not\models \varphi$ for all $Y \subset X$. The only formula which meets both of these requirements which could have survived the reduction is $(\bigwedge X, w)$, since any longer formula would have been replaced by $(\bigwedge X, w)$ and any shorter formula would either be true in some state $Y \subset X$ or fail to be true in Z. □

Note that w is not necessarily equal to $u(X)$ here—in fact, if u_G is nonmonotone, then w will frequently *not* be $u(X)$. For example, if

$$u(X) = \begin{cases} 2 & \text{if } X = \emptyset \\ 1 & \text{otherwise,} \end{cases}$$

then over $\mathcal{PS} = \{p, q\}$ the goalbase $\{(\top, 1), (\neg p \wedge \neg q, 2)\}$ represents u in the language $\mathcal{L}(\mathit{cubes}, \mathbb{R}, \max)$ and is pcubes-minimal, yet the minimal weight in $\emptyset{\uparrow}$, which is 1, does not equal $u(\emptyset) = 2$.

Lemma 4.5.7 is crucial for the bounds we will derive below, as it tells us exactly which positive cubes we will find in a minimal goalbase which is also pcubes-minimal.

Furthermore, from Lemma 4.5.7, we have the following special case:

Lemma 4.5.8. *For every $G \in \mathcal{L}(\mathit{cubes}, W, \max)$, there is a minimal $G' \equiv_{\max} G$ such that $(\top, \min_{X \in 2^{\mathcal{PS}}} u(X)) \in G'$.*

Proof. $\min_{X \in 2^{\mathcal{PS}}} u(X) = \min_{Z \in \emptyset\uparrow} u(Z)$, and $\emptyset$ has no proper subsets. □

It is not always that case that a G' which results from the reduction under discussion is uniquely minimal. E.g.,

$$u(X) = \begin{cases} 1 & \text{if } a \in X \\ 0 & \text{otherwise} \end{cases}$$

can be represented as either $\{(\top, 0), (a, 1)\}$ or $\{(\neg a, 0), (a, 1)\}$, both of which are the same size.

Now we attempt to calculate bounds on $\text{size}(G)$ when $G \in \mathcal{L}(\mathit{cubes}, W, \max)$.

Lemma 4.5.9. *Let $G \in \mathcal{L}(\mathit{cubes}, W, \max)$ be minimal. Let $G^+ \subseteq G$ be the set of pcubes in G. Then*

$$\text{size}(G^+) = \sum \Big\{ \max(1, |X|) \;\Big|\; \neg\exists Y \subset X \textit{ s.t. } u_{G,\max}(Y) \geq \min_{Z \in X\uparrow} u_{G,\max}(Z) \Big\}.$$

Proof. By Lemma 4.5.7 we may, without loss of generality, assume that

$$G^+ = \Big\{ \Big(\bigwedge X, w \Big) \;\Big|\; \neg\exists Y \subset X \text{ s.t. } u(Y) \geq \min_{Z \in X\uparrow} u(Z) \Big\}.$$

Therefore

$$\text{size}(G^+) = \sum \Big\{ \max(1, |X|) \;\Big|\; \neg\exists Y \subset X \text{ s.t. } u_{G,\max}(Y) \geq \min_{Z \in X\uparrow} u_{G,\max}(Z) \Big\}.$$

Note that we must have $\max(1, |X|)$ instead of simply $|X|$, due to the fact that $|\emptyset| = 0$ but $\text{size}(\bigwedge \emptyset) = \text{size}(\top) = 1$. □

Using this and a result from Chapter 3, we can derive the exact size for every minimal G in the monotone portion of $\mathcal{L}(\mathit{cubes}, W, \max)$:

Theorem 4.5.10. *If $G \in \mathcal{L}(\mathit{cubes}, W, \max)$, G is minimal, and $u_{G,\max}$ is monotone, then*

$$\text{size}(G) = \sum \Big\{ \max(1, |X|) \;\Big|\; \neg\exists Y \subset X \textit{ s.t. } u_{G,\max}(Y) \geq u_{G,\max}(X) \Big\}.$$

Proof. Since $u_{G,\max}$ is monotone, we know by Theorem 3.5.6 that removal of negative literals is equivalence-preserving. Since G is minimal, it follows that the negative literals are already gone, so every formula in G is a positive cube. Hence $G = G^+$, so we have from Lemma 4.5.9 that

$$\text{size}(G) = \sum \Big\{ \max(1, |X|) \;\Big|\; \neg\exists Y \subset X \text{ s.t. } u_{G,\max}(Y) \geq \min_{Z \in X\uparrow} u_{G,\max}(Z) \Big\}.$$

Finally, notice that because $u_{G,\max}$ is monotone, $\min_{Z \in X\uparrow} u_{G,\max}(Z) = u_{G,\max}(X)$, which permits us to simplify the condition. □

Now we turn to the case where $u_{G,\max}$ is nonmonotone. For any such minimal G, we know by Lemma 4.5.9 the precise size of G^+, the positive subset of G, so all that remains is to derive bounds for G^-, the subset of G containing negative literals. First, we derive bounds for the size of G^-.

Lemma 4.5.11. *Let $G \in \mathcal{L}(\textit{cubes}, W, \max)$ be minimal. Let G^- be the set of cubes in G containing negative literals. Then*

$$\text{size}(G^-) \geq \big|\{p \in \mathcal{PS} \mid u_{G,\max}(X) > u_{G,\max}(X \cup \{p\})\}\big|$$

and

$$\text{size}(G^-) \leq |\mathcal{PS}| \cdot \Big|\Big\{ X \;\Big|\; \exists Y \subset X \text{ s.t. } u(Y) \geq \min_{Z \in X\uparrow} u(Z) \Big\}\Big|.$$

Proof. By Lemma 4.5.4, if there are states X and $X \cup \{p\}$ where $u_{G,\max}(X \cup \{p\}) < u_{G,\max}(X)$, then there must be a formula (φ, w) active in state X that contains $\neg p$. These are the formulas which comprise G^-.

The best case is that every such $\neg p$ appears in G^- exactly once, which gives us the lower bound. The worst case is to write a complete cube $\bigwedge X \cup \neg\bar{X}$ to cover each pair of states $X, X \cup \{p\}$ over which there is a decline in value; every such complete cube has length $|\mathcal{PS}|$. □

Now we have all of the pieces necessary for exhibiting upper and lower bounds on the size of goalbases in $\mathcal{L}(\textit{cubes}, W, \max)$ which represent nonmonotone utility functions.

Theorem 4.5.12. *If $G \in \mathcal{L}(\textit{cubes}, W, \max)$, G is minimal, and $u_{G,\max}$ is nonmonotone, then*

$$\text{size}(G) \geq \sum \Big\{ \max(1, |X|) \;\Big|\; \neg\exists Y \subset X \text{ s.t. } u_{G,\max}(Y) \geq \min_{Z \in X\uparrow} u_{G,\max}(Z) \Big\} + \big|\{a \in \mathcal{PS} \mid u_{G,\max}(X) > u_{G,\max}(X \cup \{a\})\}\big|.$$

Proof. Recall that $G = G^+ \cup G^-$. Lemma 4.5.9 gives us the exact size of G^+, while Lemma 4.5.11 gives us a lower bound for the size of G^-. □

Theorem 4.5.13. *If $G \in \mathcal{L}(cubes, W, \max)$, G is minimal, and $u_{G,\max}$ is nonmonotone, then*

$$\text{size}(G) \leq \sum \Big\{ \max(1, |X|) \;\Big|\; \neg\exists Y \subset X \text{ s.t. } u_{G,\max}(Y) \geq \min_{Z \in X\uparrow} u_{G,\max}(Z) \Big\} + |\mathcal{PS}| \cdot \Big| \Big\{ X \;\Big|\; \exists Y \subset X \text{ s.t. } u_{G,\max}(Y) \geq \min_{Z \in X\uparrow} u_{G,\max}(Z) \Big\} \Big|$$

Proof. Recall that $G = G^+ \cup G^-$. Lemma 4.5.9 gives us the exact size of G^+, while Lemma 4.5.11 gives us an upper bound for the size of G^-. □

Note that neither the upper nor lower bounds given here are tight in general, though for each bound we do have an example of a goalbase for which one of the bounds is tight. For the upper bound, consider the utility function

$$u(X) = |X| \bmod 2,$$

the parity function on $\mathcal{PS}$. When $\mathcal{PS} = \{a, b, c\}$, the goalbase

$$\{(\top, 0), (a \wedge b \wedge c, 1), (a \wedge \neg b \wedge \neg c, 1), (\neg a \wedge b \wedge \neg c, 1), (\neg a \wedge \neg b \wedge c, 1)\}$$

is minimal for $u(X)$. The lower bound here is 7, the upper bound is 13, and the actual size is also 13. For the lower bound, consider the utility function

$$u(X) = \begin{cases} 1 & \text{if } a \notin X \\ 0 & \text{otherwise,} \end{cases}$$

for which the goalbase $\{(\top, 0), (\neg a, 1)\}$ is minimal over $\mathcal{PS} = \{a, b, c\}$. In this case, both the actual size and lower bound are 2, while the upper bound is again 13. Finally, the utility function $u(X) = 3 - |X|$ is represented minimally over $\mathcal{PS} = \{a, b, c\}$ by the goalbase

$$\left\{ \begin{array}{c} (\top, 0), \\ (\neg a, 1), (\neg b, 1), (\neg c, 1), \\ (\neg a \wedge \neg b \wedge c, 2), (\neg a \wedge b \wedge \neg c, 2), (a \wedge \neg b \wedge \neg c, 2), \\ (\neg a \wedge \neg b \wedge \neg c, 3) \end{array} \right\}$$

which has size 16, but hits neither the lower nor the upper bound, which are 4 and 22, respectively.

Clearly these bounds could be refined by further analyzing the composition of G^- for various nonmonotone utility functions, but at present what exactly the differences are among the three examples—what causes one to hit the lower bound, one to hit the upper bound, and one to hit neither—is not apparent to us. Furthermore, the bounds themselves are not easy to compute; here again, some additional insight into the structure of such functions might be of use. We leave these issues for future work.

4.5.2 Relative Succinctness

When comparing the succinctness of any two max languages, notice that the available weights play no role in the outcome. If $\mathcal{L}(\Phi, W, \max)$ and $\mathcal{L}(\Psi, W', \max)$ are the languages under comparison, then for any utility function u representable in both languages, $\operatorname{ran} u \subseteq W \cap W'$. Due to this, any weighted formula (φ, w) where $w \notin W \cap W'$ will be superfluous when it occurs in a representation of u in either language. Since only minimal representations are relevant for succinctness, we can disregard all representations of u which use weights outside of $W \cap W'$:

Fact 4.5.14. *For succinctness relations* $\odot \in \{\preceq, \prec, \sim, \perp\}$*:*

$$\mathcal{L}(\Phi, W, \max) \odot \mathcal{L}(\Psi, W', \max) \iff \mathcal{L}(\Phi, W \cap W', \max) \odot \mathcal{L}(\Psi, W \cap W', \max).$$

The same does not hold for arbitrary sum languages, due to the fact that summing weights can produce values for states which lie outside the set of weights. (Consider that if $G = \{(a, 1), (b, 1)\} \in \mathcal{L}(\mathit{atoms}, \{0, 1\}, \Sigma)$, then $u_{G,\Sigma}(\{a, b\}) = 2 \notin \{0, 1\}$.)

Next, we show that all pcubes and cubes languages are equally succinct when using max for our aggregator.

Theorem 4.5.15. *For all* $j, k \in \mathbb{N} \cup \{\omega\}$ *and* $W, W' \subseteq \mathbb{R}$*:*

1. $\mathcal{L}(j\text{-}\mathit{pcubes}, W, \max) \sim \mathcal{L}(k\text{-}\mathit{pcubes}, W', \max)$.
2. $\mathcal{L}(j\text{-}\mathit{pcubes}, W, \max) \sim \mathcal{L}(k\text{-}\mathit{cubes}, W', \max)$.
3. $\mathcal{L}(j\text{-}\mathit{cubes}, W, \max) \sim \mathcal{L}(k\text{-}\mathit{cubes}, W', \max)$.

Proof. When at least one of $j, k \in \mathbb{N}$, all three cases follow immediately from Theorem 4.2.4. We must give a proof when $j = k = \omega$, but here the first and third cases trivialize, so all that remains is to show that $\mathcal{L}(\mathit{pcubes}, W, \max) \sim \mathcal{L}(\mathit{cubes}, W', \max)$. By Fact 4.5.14, we may assume without loss of generality that $W = W'$. $\mathcal{L}(\mathit{pcubes}, W, \max)$ expresses only monotone utility functions, and Theorem 3.5.6 ensures that any representation containing negative literals may be reduced to a shorter, equivalent one by simply deleting the negative literals; the result of such a deletion is in $\mathcal{L}(\mathit{pcubes}, W, \max)$. Hence the minimal representations for any u representable in both $\mathcal{L}(\mathit{pcubes}, W, \max)$ and $\mathcal{L}(\mathit{cubes}, W', \max)$ will be the same, which proves that $\mathcal{L}(\mathit{pcubes}, W, \max) \sim \mathcal{L}(\mathit{cubes}, W', \max)$. □

Recall that Theorem 3.5.4 shows that disjunctions as main connectives do not affect succinctness, because the translation required to eliminate disjunction does not affect the size of a goalbase. However, the same is not necessarily true for disjunctions which occur within the scope of other connectives. Therefore, for our analysis of expressivity of max languages in Chapter 3 we could safely ignore disjunction, but for succinctness we cannot, as the next result demonstrates.

Theorem 4.5.16. $\mathcal{L}(pcubes, \mathbb{R}^+, \max) \prec \mathcal{L}(pforms, \mathbb{R}^+, \max)$.

Proof. We have that $\mathcal{L}(pcubes, \mathbb{R}^+, \max) \preceq \mathcal{L}(pforms, \mathbb{R}^+, \max)$ because every pcube is a positive formula. For strict succinctness: The family of utility functions represented by

$$\{((p_1 \vee p_2) \wedge (p_3 \vee p_4) \wedge \ldots \wedge (p_{n-1} \vee p_n), 1)\}$$

in $\mathcal{L}(pforms, \mathbb{R}^+, \max)$ grows linearly with n, while the minimal representation in $\mathcal{L}(pcubes, \mathbb{R}^+, \max)$ is

$$\left\{ \left(\bigwedge_{k=1}^{n/2} p_{i_k}, 1 \right) \;\middle|\; i_1, \ldots, i_{n/2} \in \{1,2\} \times \{3,4\} \times \ldots \times \{n-1, n\} \right\}$$

which has size 2^{n-1} for any (even) n. □

More generally, the same argument shows that for any intersecting sets of weights W, W', $\mathcal{L}(pcubes, W, \max) \prec \mathcal{L}(pforms, W', \max)$—so long as there is some $w \in W \cap W'$, we may use that for the formula weights instead of using 1 as we do in the proof—and also that $\mathcal{L}(pcubes, W, \max) \prec \mathcal{L}(forms, W, \max)$ and $\mathcal{L}(cubes, W, \max) \prec \mathcal{L}(pforms, W, \max)$, by virtue of Theorem 3.5.6.

When dealing with negation-containing goalbases for monotone utility functions, we might wish to put all nonpositive formulas in a standard form in order to simplify working with them. Negation normal form is a way of doing this.

Definition 4.5.17 (Negation Normal Form)**.** A formula φ is in *negation normal form* (NNF) if all occurrences of negation apply to atoms only.

Any formula may be rewritten to an equivalent formula in NNF without an increase in size, by recursive application of these rewrite rules to its subformulas:

$$\begin{aligned} \neg\neg\varphi &\mapsto \varphi \\ \neg(\varphi \wedge \psi) &\mapsto \neg\varphi \vee \neg\psi \\ \neg(\varphi \vee \psi) &\mapsto \neg\varphi \wedge \neg\psi \end{aligned}$$

For example, $\neg((p \wedge q) \vee (r \wedge \neg(s \vee t)))$ is not in NNF, while the equivalent formula $(\neg p \vee \neg q) \wedge (\neg r \vee (s \vee t))$ is in NNF. Once we have all formulas in a monotone max goalbase translated to NNF, we may apply the following equivalence in order to remove the negative literals altogether:

Lemma 4.5.18. *If $u_{G,\max}$ is monotone and every formula in G is in NNF, then $G[\top/\neg p_1, \ldots, \top/\neg p_n] \equiv_{\max} G$.*

Proof. To show that $G[\top/\neg p_1, \ldots, \top/\neg p_n] \equiv_{\max} G$, it suffices to show for a single $(\varphi, w) \in G$ that $(G \setminus \{(\varphi, w)\}) \cup \{(\varphi[\top/\neg p], w)\} \equiv_{\max} G$, as we can then repeat the process for each other formula and each other $p \in \mathcal{PS}$.

Fix a $(\varphi, w_\varphi) \in G$ which has $\neg p$ as a subformula. If there is no model M such that $p \notin M$, $M \models \varphi$, and $M \cup \{p\} \not\models \varphi$, then we immediately have that $\models \varphi[\top/\neg p] \leftrightarrow \varphi$ and we are done. Otherwise, let M be such a model. Since φ is in NNF, $\varphi[\top/\neg p]$ will remain true in every state where φ was true; but there is additionally the possibility that $M \cup \{p\} \models \varphi[\top/\neg p]$ when $M \not\models \varphi$. However, since $u_{G,\max}$ is monotone and $M \models \varphi$, we know that $u_{G,\max}(M \cup \{p\}) \geq w_\varphi$, and so there must already be some $(\psi, w_\psi) \in G$ such that $M \cup \{p\} \models \psi$ and $w_\psi = u_{G,\max}(M \cup \{p\})$. Therefore, making it so that $M \cup \{p\} \models \varphi[\top/\neg p]$ will not disturb the value of the utility function there (or in any other state). Hence, $(G \setminus \{(\varphi, w)\}) \cup \{(\varphi[\top/\neg p], w)\} \equiv_{\max} G$. □

With this lemma in hand, we now improve on Theorem 3.5.6 and show that there is no succinctness gain from using negation in arbitrary formulas, not just cubes, when representing monotone utility functions in max languages.

Theorem 4.5.19. *If Φ is closed under transformation to NNF and for every $G \in \mathcal{L}(\Phi, W, \max)$ which is in NNF, there exists a $G[\top/\neg p \mid p \in \mathcal{PS}] \in \mathcal{L}(\Psi, W', \max)$, then $\mathcal{L}(\Phi, W, \max) \preceq \mathcal{L}(\Psi, W', \max)$.*

Proof. By Lemma 4.5.18, if G is in NNF, then $G \equiv_{\max} G[\top/\neg p_1, \ldots, \top/\neg p_n]$. Since transformation to NNF and substitution of $\top$ for all negative literals are both size-preserving, $\text{size}(G) = \text{size}(G[\top/\neg p_1, \ldots, \top/\neg p_n])$, so there is always a space-efficient translation from $\mathcal{L}(\Phi, W, \max)$ to $\mathcal{L}(\Psi, W', \max)$. □

Note that W' cannot be completely arbitrary here: The condition requiring that $G[\top/\neg p \mid p \in \mathcal{PS}] \in \mathcal{L}(\Psi, W', \max)$ implies that $W' \supseteq W$, though it is not stated in the theorem.

Corollary 4.5.20. *$\mathcal{L}(\mathit{forms}, W, \max) \sim \mathcal{L}(\mathit{pforms}, W', \max)$, for all $W, W' \subseteq \mathbb{R}$.*

Proof. By Fact 4.5.14, we may assume without loss of generality that $W = W'$. Then $\mathcal{L}(\mathit{forms}, W, \max) \succeq \mathcal{L}(\mathit{pforms}, W', \max)$ follows by inclusion, and $\mathcal{L}(\mathit{forms}, W, \max) \preceq \mathcal{L}(\mathit{pforms}, W', \max)$ follows from Theorem 4.5.19. □

4.5.3 Summary

Our relative succinctness results for max languages are summarized in Table 4.2. The table contains many more results than are proved in the text, but in all cases these are straightforward consequences of results which do appear in the text. In particular, most relations in the table are due to a combination of Fact 4.5.14, which lets us consider the intersection of the sets of weights; Theorem 4.5.15, which gives us equal succinctness between any pair of cubes and pcubes languages; and

	$\mathcal{L}(pcubes, \mathbb{R}^+, \max)$	$\mathcal{L}(atoms +\top, \mathbb{R}^+, \max)$	$\mathcal{L}(pforms, \mathbb{R}^+, \max)$	$\mathcal{L}(cubes, \mathbb{R}^+, \max)$	$\mathcal{L}(literals, \mathbb{R}^+, \max)$	$\mathcal{L}(forms, \mathbb{R}^+, \max)$	$\mathcal{L}(pcubes, \mathbb{R}, \max)$	$\mathcal{L}(atoms +\top, \mathbb{R}, \max)$	$\mathcal{L}(pforms, \mathbb{R}, \max)$	$\mathcal{L}(cubes, \mathbb{R}, \max)$	$\mathcal{L}(literals, \mathbb{R}, \max)$	$\mathcal{L}(forms, \mathbb{R}, \max)$
$\mathcal{L}(forms, \mathbb{R}, \max)$	$\succ$	$\sim$	$\sim$	$\succ$	$\sim$	$\sim$	$\succ$	$\sim$	$\sim$	$\succ$	$\sim$	$\sim$
$\mathcal{L}(literals, \mathbb{R}, \max)$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	
$\mathcal{L}(cubes, \mathbb{R}, \max)$	$\sim$	$\sim$	$\prec$	$\sim$	$\sim$	$\prec$	$\sim$	$\sim$	$\prec$	$\sim$		
$\mathcal{L}(pforms, \mathbb{R}, \max)$	$\succ$	$\sim$	$\sim$	$\succ$	$\sim$	$\sim$	$\succ$	$\sim$	$\sim$			
$\mathcal{L}(atoms +\top, \mathbb{R}, \max)$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$				
$\mathcal{L}(pcubes, \mathbb{R}, \max)$	$\sim$	$\sim$	$\prec$	$\sim$	$\sim$	$\prec$	$\sim$					
$\mathcal{L}(forms, \mathbb{R}^+, \max)$	$\succ$	$\sim$	$\sim$	$\succ$	$\sim$	$\sim$						
$\mathcal{L}(literals, \mathbb{R}^+, \max)$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$							
$\mathcal{L}(cubes, \mathbb{R}^+, \max)$	$\sim$	$\sim$	$\prec$	$\sim$								
$\mathcal{L}(pforms, \mathbb{R}^+, \max)$	$\succ$	$\sim$	$\sim$									
$\mathcal{L}(atoms +\top, \mathbb{R}^+, \max)$	$\sim$	$\sim$										
$\mathcal{L}(pcubes, \mathbb{R}^+, \max)$	$\sim$											

Table 4.2: Summary of succinctness results for max languages. Entries to be read row first.

Corollary 3.5.5, which reduces the clauses languages to simpler cubes languages. Because these three results are much more powerful than comparable results we proved for sum languages, the table for max languages is complete. The most common result by an overwhelming margin is for two max languages to be equally succinct. All differences in succinctness shown in the table are due to Theorem 4.5.16.

Our absolute succinctness results for max languages focus primarily on $\mathcal{L}(\mathit{cubes}, W, \max)$, for which we calculate the exact size of the minimal representations of monotone utility functions (Theorem 4.5.10), and upper and lower bounds on the size of representations of nonmonotone utility functions (Theorems 4.5.13 and 4.5.12). These results apply also to any sublanguage of $\mathcal{L}(\mathit{cubes}, W, \max)$, which includes all max languages we have examined, save $\mathcal{L}(\mathit{pforms}, W, \max)$ and its superlanguages.

4.6 Cross-Aggregator Succinctness

Here we examine the succinctness of some max languages with respect to some sum languages. These results indicate that each aggregator favors short representations for certain kinds of utility functions; whether max or sum is better for a particular application will depend on what utility functions agents are likely to have. Additionally, the results in this section highlight why it was necessary to index our succinctness relation to a particular class of comparison; a discussion of this appears at the end of the section.

First, we use $u_n^{\exists}$ (see Definition 4.4.7) and a utility function with maximal range to compare $\mathcal{L}(\mathit{pcubes}, \mathbb{R}^+, \max)$ with $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$:

Theorem 4.6.1. $\mathcal{L}(\mathit{pcubes}, \mathbb{R}^+, \max) \perp \mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$.

Proof. ($\not\preceq$) Let $u_n(X) = \sum_{a_i \in X} 2^i$. Then $|\operatorname{ran} u_n| = 2^n$, and so if $G_{\max}$ represents u_n, then by Theorem 4.5.1, $|G_{\max}| \geq 2^n$. In $\mathcal{L}(\mathit{atoms}, \mathbb{R}^+, \Sigma)$, we have $G_\Sigma = \{(a_i, 2^i) \mid 0 \leq i < n\}$ which represents u_n. Hence $\mathcal{L}(\mathit{pcubes}, \mathbb{R}^+, \max) \not\preceq \mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$.

($\not\succeq$) Recall from Definition 4.4.7 that

$$u_n^{\exists}(X) = \begin{cases} 1 & \text{if } X \neq \emptyset \\ 0 & \text{otherwise,} \end{cases}$$

and that $u_n^{\exists}$ was shown in the proof of Theorem 4.4.8 to have a large representation in $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$. In $\mathcal{L}(\mathit{pcubes}, \mathbb{R}^+, \max)$, $u_n^{\exists}$ is represented by $G = \{(p, 1) \mid p \in \mathcal{PS}\}$. Therefore $\mathcal{L}(\mathit{pcubes}, \mathbb{R}^+, \max) \not\succeq \mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$. □

Next, we use $u_n^{\forall}$ to arrive at a similar result for $\mathcal{L}(\mathit{pcubes}, \mathbb{R}^+, \max)$ and $\mathcal{L}(\mathit{pclauses}, \mathbb{R}, \Sigma)$:

Theorem 4.6.2. $\mathcal{L}(\textit{pcubes}, \mathbb{R}^+, \max) \perp \mathcal{L}(\textit{pclauses}, \mathbb{R}, \Sigma)$.

Proof. ($\not\leq$) Recall from Definition 4.4.7 that

$$u_n^\forall(X) = \begin{cases} 1 & \text{if } X = \mathcal{PS} \\ 0 & \text{otherwise,} \end{cases}$$

and that $u_n^\forall$ was shown in the proof of Theorem 4.4.8 to be large in the language $\mathcal{L}(\textit{pclauses}, \mathbb{R}, \Sigma)$. In $\mathcal{L}(\textit{pcubes}, \mathbb{R}^+, \max)$, $u_n^\forall$ is represented by $\{(\bigwedge \mathcal{PS}, 1)\}$.

($\not\geq$) Same argument as for $\mathcal{L}(\textit{pcubes}, \mathbb{R}, \Sigma)$. □

Next we have an example of how cross-aggregator succinctness may be surprising, as we show that the austere language $\mathcal{L}(\textit{atoms}, \mathbb{R}, \max)$ is strictly more succinct than the seemingly-richer $\mathcal{L}(\textit{pcubes}, \mathbb{R}, \Sigma)$.

Theorem 4.6.3. $\mathcal{L}(\textit{pcubes}, \mathbb{R}, \Sigma) \prec \mathcal{L}(\textit{atoms}, \mathbb{R}, \max)$.

Proof. $\mathcal{U}(\textit{atoms}, \mathbb{R}, \max)$ corresponds to the class of unit-demand utility functions. Every unit-demand utility function u is expressible linearly in the language $\mathcal{L}(\textit{atoms}, \mathbb{R}, \max)$ as $\{(a, u(a))\}_{a \in \mathcal{PS}}$. In $\mathcal{L}(\textit{pcubes}, \mathbb{R}, \Sigma)$ (which is fully expressive and has uniqueness, *cf.* Theorem 3.4.2 and Corollary 3.4.9), u is represented by $\{(\bigwedge X, w_{\bigwedge X}) \mid X \subseteq \mathcal{PS}\}$ where

$$w_{\bigwedge X} = (-1)^{|X|+1} \min_{a \in X} u(a).$$

For any unit-demand u which is also single-minded (i.e., exactly one $a \in \mathcal{PS}$ is such that $u(a) \neq 0$), $G = \{(a, u(a))\}$, the same as in $\mathcal{L}(\textit{atoms}, \mathbb{R}, \max)$. For non-single-minded u, let $X \subseteq \mathcal{PS}$ be the items which u assigns nonzero value. Then G will contain a nonzero weight for $w_{\bigwedge Y}$ whenever $Y \subseteq X$. So, for the family $u_n^\exists$, which is the family of simple unit-demand utility functions,

$$u_n^\exists(X) = \begin{cases} 1 & \text{if } X \neq \emptyset \\ 0 & \text{otherwise,} \end{cases}$$

we have that representations in $\mathcal{L}(\textit{pcubes}, \mathbb{R}, \Sigma)$ are exponential in $|\mathcal{PS}|$. □

Now we compare $\mathcal{L}(\textit{pclauses}, \mathbb{R}, \Sigma)$ and $\mathcal{L}(\textit{atoms}, \mathbb{R}, \max)$. A feature of interest here is that we establish a succinctness gap which falls below the discriminating power of our succinctness relation, since complex unit-demand valuations are linearly representable in $\mathcal{L}(\textit{atoms}, \mathbb{R}, \max)$, but the best representations of the same are quadratic in $\mathcal{L}(\textit{pclauses}, \mathbb{R}, \Sigma)$.

Theorem 4.6.4. $\mathcal{L}(\textit{pclauses}, \mathbb{R}, \Sigma) \sim \mathcal{L}(\textit{atoms}, \mathbb{R}, \max)$.

Proof. Recall that $\mathcal{U}(atoms, \mathbb{R}, \max)$ is the class of unit-demand utility functions. Simple unit-demand utility functions are expressible linearly in $\mathcal{L}(pclauses, \mathbb{R}, \Sigma)$ as $\{(\bigvee \mathcal{PS}, 1)\}$. Consider complex unit-demand utility functions u (where items may differ in value but the value of a bundle is the value of the best item contained in it) expressed in $\mathcal{L}(pclauses, \mathbb{R}, \Sigma)$. Without loss of generality, suppose that the items are ordered $a_1 \leq \ldots \leq a_n$ in value. Then

$$\left\{\left(\bigvee \mathcal{PS} \setminus \{a_1, \ldots, a_{i-1}\}, u(a_i) - u(a_{i-1})\right)\right\}_{1 \leq i \leq n}$$

is the unique minimal representative of u in $\mathcal{L}(pclauses, \mathbb{R}, \Sigma)$, containing $\frac{n(n-1)}{2}$ atoms. □

Finally, we present a result similar to Theorem 4.6.3, but with the aggregators reversed. It may at first seem surprising to find that there is a language with a natural definition which is strictly less succinct than the extremely limited $\mathcal{L}(atoms, \mathbb{R}^+, \Sigma)$, but we get this result because max languages are poor at representing modular utility functions.

Theorem 4.6.5. $\mathcal{L}(cubes, \mathbb{R}^+, \max) \prec \mathcal{L}(atoms, \mathbb{R}^+, \Sigma)$.

Proof. ($\preceq$) From Corollary 3.4.10 we have that $\mathcal{U}(atoms, \mathbb{R}^+, \Sigma)$ is the class of normalized nonnegative modular utility functions. From Theorem 3.5.9, $\mathcal{U}(cubes, \mathbb{R}^+, \max)$ is the class of nonnegative monotone utility functions, so $\mathcal{L}(atoms, \mathbb{R}^+, \Sigma)$ is the expressive intersection of the two languages. Every representation in $\mathcal{L}(atoms, \mathbb{R}^+, \Sigma)$ is linear in $|\mathcal{PS}|$. If $u(\{a\}) \geq 0$, then the atom a will appear somewhere in any representation of u in $\mathcal{L}(cubes, \mathbb{R}^+, \max)$, so no representations which grow logarithmically in $|\mathcal{PS}|$ are possible there.

($\not\succeq$) Consider the family of utility functions $u(X) = |X|$. In $\mathcal{L}(atoms, \mathbb{R}^+, \Sigma)$, u is represented by $\{(a, 1) \mid X \in \mathcal{PS}\}$, which is linear in $|\mathcal{PS}|$, while the unique representation in $\mathcal{L}(cubes, \mathbb{R}^+, \max)$ is $\{(\bigwedge X, |X|) \mid X \subseteq \mathcal{PS}\}$, which is exponential in $|\mathcal{PS}|$. □

We are now in position to demonstrate why the more general definition of succinctness (subscripted by the comparison class) is needed: In Theorem 4.6.3 we showed that $\mathcal{L}(pcubes, \mathbb{R}, \Sigma) \prec \mathcal{L}(atoms, \mathbb{R}, \max)$, while in Theorem 4.6.5, we showed that $\mathcal{L}(cubes, \mathbb{R}^+, \max) \prec \mathcal{L}(atoms, \mathbb{R}^+, \Sigma)$. Furthermore, it is obvious that $\mathcal{L}(pcubes, \mathbb{R}, \Sigma) \sim \mathcal{L}(atoms, \mathbb{R}^+, \Sigma)$, since the expressive intersection of the two languages is the whole of the latter, and in the latter every representation is small. Finally, $\mathcal{L}(atoms, \mathbb{R}, \max) \sim \mathcal{L}(cubes, \mathbb{R}^+, \max)$ because their expressive intersection is the class of nonnegative unit-demand utility functions, which have small representations in both languages. We now have the following situation:

$$\begin{array}{ccc} \mathcal{L}(pcubes, \mathbb{R}, \Sigma) & \prec & \mathcal{L}(atoms, \mathbb{R}, \max) \\ \wr & & \wr \\ \mathcal{L}(atoms, \mathbb{R}^+, \Sigma) & \succ & \mathcal{L}(cubes, \mathbb{R}^+, \max) \end{array}$$

From this it can be seen that the *unsubscripted* succinctness relation is not transitive. This comes about because no two pairs of these four languages have the same expressive intersection. Because the expressive intersection of the languages can shift from comparison to comparison, we must make explicit the class of utility functions over which the comparison is being made if we wish to do more than pairwise comparison.

4.7 Conclusion

In this chapter we have defined a notion of succinctness of representation for goalbase languages and presented numerous pairwise relative succinctness results. We attempted to systematically resolve all questions of pairwise succinctness among sum languages and among max languages, and examined the succinctness of selected pairs of languages where one is a sum language and the other a max language. For a summary of results for pairs of sum languages, see Table 4.1; for pairs of max languages, see Table 4.2; for selected max-sum pairs, see Section 4.6.

The results for max languages are complete over the languages we considered. Determining the relative succinctness of languages which use max as their aggregator is dramatically simpler than for languages which use sum, due to the close connection between a utility function's range and the size of its representation in any max language (*cf.* Sections 4.4.4 and 4.5.2).

There are still numerous open questions regarding the relative succinctness of pairs of sum languages. In order to produce succinctness results involving languages without unique representations, we need some way of addressing the size of *all* representations of a particular utility function, not just one representation which we happen to have constructed. It is not obvious how to do this generally. The two results of which we are aware which show that a language without unique representations is strictly less succinct than another language are the ones in Section 4.4.4, along with an alternative proof of Theorem 4.4.13 by Elkind et al. [2009, Theorem 3.3]. Unfortunately, we see no straightforward way to generalize these results to apply to other languages, so we suspect that the remaining open questions will require novel proofs to settle them. Among these, many revolve around how unlimited nesting of $\wedge$ and $\vee$ (as with positive formulas) compares with the power of $\neg$ applied to atoms only (as with cubes and clauses). Many of the empty cells in Table 4.1 are versions of this question.

Chapter 5

Complexity

5.1 Introduction

In this chapter, we analyze the effect that restrictions on goalbases have on the complexity of answering questions about the utility functions they represent, focusing specifically on the decision problems MAX-UTIL, MIN-UTIL, and MAX-CUF, which are, respectively, the problem of deciding whether there is a model producing at least a given amount of utility for an individual, the problem of deciding whether every model produces at least a given amount of utility for an individual, and the problem of deciding whether there is an allocation producing at least a given amount of utility for a group.

We begin in Section 5.2 with the background necessary for the complexity theory we use in this chapter. Readers already familiar with complexity theory should feel free to skip ahead to Sections 5.3 and 5.4 where we define our decision problems and discuss related work. The remaining sections contain our results for MAX-UTIL and MIN-UTIL (Section 5.5) and MAX-CUF (Section 5.6), followed by an exploration of an alternative version of MAX-UTIL (Section 5.7).

5.2 Background

Every result in this chapter is a complexity-theoretic one. We present just enough complexity theory in this section for someone unfamiliar with complexity theory to have a barely-adequate understanding of the rest of the chapter. Anyone wanting a more thorough grounding in complexity theory may wish to consult [Sipser, 1997, Part Three] for a gentle introduction, or [Papadimitriou, 1994a] for the full-on treatment.

Definition 5.2.1 (Decision Problem)**.** A decision problem is a subset of the set of all finite binary strings $\{0,1\}^*$.

By convention, we write names of decision problems in small caps. E.g., the (made-up) problem WIDGET FROBNICATION can be recognized as decision problem in this way. An *instance* of a decision problem is an object for which we might ask whether it is a member of the decision problem. For example, we might ask whether a particular formula φ is a member of SAT (that is, whether it is a satisfiable formula). If an instance is a member, then we say that it is a *positive* or *accepting* instance, and if not a member, then a *negative* or *rejecting* instance. We generally do not speak of members of a decision problem as binary strings, but rather as structures which could be represented as binary strings if we wanted to go through the trouble of doing so, since binary is too low-level a description to be handy for our proofs. When speaking of instances which are tuples, we will often write them as $\langle X_1, \ldots, X_k \rangle$ to make their structure apparent. (For all instances of this sort, a bijection with binary strings may be found simply by enumerating all characters we intend to use for our alphabet and then replacing them with the binary sequences corresponding to their indices.)

Because decision problems are sets, every decision problem has a complementary decision problem, where the accepting and rejecting instances are reversed. We overline decision problems to indicate their complementary problem. E.g., $\overline{\text{WIDGET FROBNICATION}}$ is the set of all instances which are not members of WIDGET FROBNICATION. In the following, we also speak of decision problems as *languages* for the reason that they are sets of strings.

Now we turn to the classification of decision problems according to the difficulty of deciding arbitrary instances:

Definition 5.2.2 (Big-O Notation)**.** Let $f, g \colon \mathbb{N} \to \mathbb{N}$ be arbitrary functions. Then we say that $f(n) = O(g(n))$ iff there are $c, n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.

Informally, saying that a function f is $O(g)$ means that f grows no faster than g, within a constant factor. (For example, $2n^2 + 1$ is an $O(n^2)$ function.)

Complexity classes are sets of languages, usually defined by bounds on the resources which are available for deciding those languages. One way of giving such bounds is by the runtime of optimal decision algorithms. By convention, we write the names of complexity classes in sans-serif for easy identification.

Definition 5.2.3 (Time Complexity)**.**

- $\mathsf{TIME}(t(n))$ is the class of languages decidable by a deterministic $O(t(n))$-time Turing machine.
- $\mathsf{NTIME}(t(n))$ is the class of languages decidable by a nondeterministic $O(t(n))$-time Turing machine.
- $\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(n^k)$.

- $\mathsf{NP} = \bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(n^k)$.

In other words, P is the class of languages for which there are deterministic polynomial-time algorithms to decide them, and NP is the class of languages for which there are nondeterministic polynomial-time algorithms to decide them. An alternative characterization of NP, one of which we shall frequently make use, is that NP is the class of languages for which examples are polynomially verifiable. That is, if we are given a purported proof of the membership of an instance, then if the original language is a member of NP, there will be a polynomial-time algorithm for verifying that proof of membership.

Like decision problems, complexity classes have complements, though their complements are not formed in the same way. (Consider that set-theoretic complementation would not be useful here, since, e.g., the set-theoretic complement of P would be all languages for which there is no polynomial algorithm to decide them, and this would include not just languages for which the best algorithms are exponential, but also languages which are not even decidable.)

Definition 5.2.4 (Complementary Complexity Classes)**.** For a given complexity class $\mathcal{C}$, $\mathsf{co}\mathcal{C} = \{\overline{L} \mid L \in \mathcal{C}\}$ is its complementary complexity class.

In particular, we are interested coNP, which is the class of languages for which counterexamples are polynomially verifiable. All deterministic complexity classes are closed under complement, so in particular we will never write coP because $\mathsf{coP} = \mathsf{P}$.

While it is clear that $\mathsf{P} \subseteq \mathsf{NP}, \mathsf{coNP}$, nothing further is known, though it is strongly suspected that the inclusion is strict and also that NP is distinct from coNP. A selection of complexity classes may be seen in Figure 5.1.[1]

In order to show that a decision problem is a member of some complexity class, it suffices to exhibit an algorithm to decide the problem which respects the resource bounds of the target class. This is an upper bound on the complexity of the decision problem. (For example, if we can give a polynomial algorithm which decides WIDGET FROBNICATION, then we know that it is no harder than polynomial.) What this does not tell us is whether we can do better. The following definitions are needed for describing lower bounds on the complexity of decision problems.

As we sometimes speak of polynomial-time or logarithmic-space computable functions, we now give a definition:

Definition 5.2.5 (Bounded Time- or Space-Computable Functions)**.** A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is *computable in $O(g(n))$-time* if there is a Turing machine which, for each input $x \in \{0,1\}^*$, halts with $f(x)$ on its tape after no more than

[1] The classes shown here are a minuscule sampling of those which have been defined. For those wishing to see (hugely) more, the Complexity Zoo [2009] strives to be something of a *Jane's All the World's Complexity Classes*.

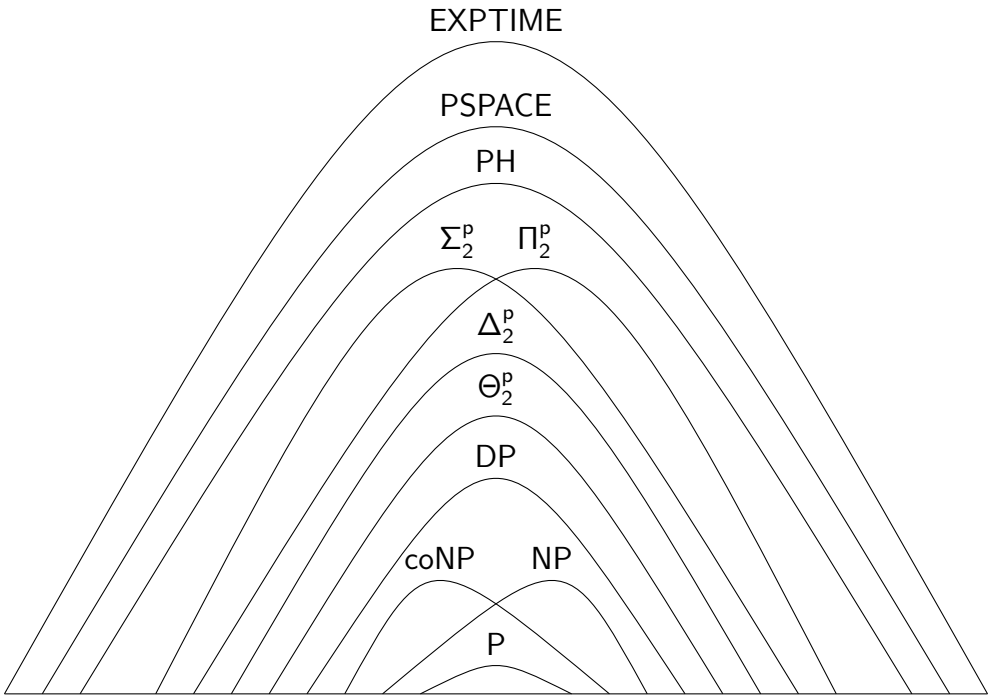


Figure 5.1: Some complexity classes for decision problems.

$O(g(n))$ steps. Similarly, f is *computable in $O(g(n))$-space* if there is a Turing machine which, for each input $x \in \{0,1\}^*$, halts with $f(x)$ on its tape after using no more than the first $O(g(n))$ cells on the tape.

It is often possible to convert one decision problem into another without expending much computation on the conversion. We formalize that notion here:

Definition 5.2.6 (Many-One Reductions)**.** A language A is *many-one reducible* to a language B if there is a (total) computable function f such that for every $n \in \{0,1\}^*$, $n \in A$ iff $f(n) \in B$.

Intuitively, if the decision problem A is reducible to the decision problem B it means that we can convert instances of A into instances of B, solve B, and then recover a solution to A from the solution to B. Therefore, we can say that deciding A is no harder than doing the reduction and then deciding B. We may combine the notion of boundedly-computable functions with that of reductions to limit the difficulty of reductions. Almost all reductions we use in this chapter are polynomial-time many-one reductions, which means that the reduction itself must be carried out in an amount of time bounded by some polynomial. Note that the notion of polynomial-time many-one reduction only makes sense for problems which are known (or at least thought) to be outside of P, because within P the difficulty of the reduction itself might swamp the difficulty of two problems involved in the reduction. Within P, more restrictive kinds of reductions are needed: In Section 5.7.3, we use logarithmic-space reductions in order to work with problems known to be polynomial.

If there were a decision problem to which every problem in a class $\mathcal{C}$ could be reduced, we might say that this problem was at least as hard as every problem in $\mathcal{C}$.

Definition 5.2.7 (Hardness)**.** A language A is *hard* for a complexity class $\mathcal{C}$ if every language $B \in \mathcal{C}$ reduces to A.

A problem which is hard for $\mathcal{C}$ we call $\mathcal{C}$-hard. For example, SAT is NP-hard: There is a way to polynomially reduce every problem in NP to SAT. Hardness is a lower bound on complexity. The method we use in this chapter for showing $\mathcal{C}$-hardness is not the direct method suggested by the definition—i.e., demonstrating directly that arbitrary languages in $\mathcal{C}$ may be reduced to our target language. Polynomial-time many-one reductions are transitive, in the sense that if A reduces to B and B reduces to C, then A reduces to C also. We take advantage of this fact in order to avoid doing direct hardness proofs. If A is a $\mathcal{C}$-hard problem, then by definition *every* problem in $\mathcal{C}$ reduces to it; therefore, if we can reduce A to B, then by transitivity every problem in $\mathcal{C}$ reduced to B also, and hence we have shown B to be $\mathcal{C}$-hard as well. Using this approach, we only need one direct proof to prime the pump; after that, it is much more expedient to rely on reductions from hard problems. Cook [1971] gave the first direct proof of NP-hardness by

proving SAT to be NP-hard. Since then, a bewildering number of other problems have been shown to be NP-hard, and we are free to reduce from whichever is most convenient when showing NP-hardness.

We can exactly characterize the complexity of a decision problem by showing that it is both a member of and hard for the same complexity class. When this happens, we say that a problem is complete for that class.

Definition 5.2.8 (Completeness)**.** A language L is *complete* for a complexity class $\mathcal{C}$ if both $L \in \mathcal{C}$ and L is $\mathcal{C}$-hard.

A $\mathcal{C}$-complete problem may be thought of as one of the most difficult problems in class $\mathcal{C}$, since it can be used to solve all problems in $\mathcal{C}$, but yet is still a member of $\mathcal{C}$. The problem SAT is a typical example of an NP-complete problem.

5.3 The Decision Problems MAX-UTIL, MIN-UTIL, and MAX-CUF

Here we define the three decision problems which we analyze in this chapter. Motivation for these problems appears in subsequent sections.

The decision problem MAX-UTIL is the problem of determining whether an agent, given his preferences, can attain at least some specified amount of utility.

Definition 5.3.1 (The Decision Problem MAX-UTIL)**.** The decision problem MAX-UTIL(Φ, W, F) is defined as: Given a goalbase $G \in \mathcal{L}(\Phi, W, F)$ and an integer K, is there a model $M \in 2^{\mathcal{PS}}$ where $u_G(M) \geq K$?

The decision problem MIN-UTIL is the pessimal version of MAX-UTIL, asking whether an agent will always obtain at least some specified amount of utility, no matter what state he finds himself in.

Definition 5.3.2 (The Decision Problem MIN-UTIL)**.** The decision problem MIN-UTIL(Φ, W, F) is defined as: Given a goalbase $G \in \mathcal{L}(\Phi, W, F)$ and an integer K, are all models $M \in 2^{\mathcal{PS}}$ such that $u_G(M) \geq K$?

In addition to considering individual utility, we might also consider the utility of groups of agents.

Definition 5.3.3 (Collective Utility Functions)**.** A *collective utility function* (CUF) is a mapping $\sigma\colon \mathbb{R}^* \to \mathbb{R}$.

A collective utility function takes a tuple of individual utilities as its input, and aggregates them into a single group utility.[2] The decision problem MAX-CUF is like MAX-UTIL, but lifted from an individual agent to a group of agents.

[2]Note that all of the collective utility functions we consider are associative and commutative, so aggregating a tuple of individual utilities is the same as aggregating a multiset of individual utilities. Functions which are nonassociative or noncommutative tend to be less interesting as CUFs, because they fail to treat all agents equally. See also p. 12, footnote 3.

Definition 5.3.4 (The Decison Problem MAX-CUF)**.** The decision problem MAX-CUF(Φ, W, F, σ) for n agents is defined as: Given goalbases $G_1, \ldots, G_n \in \mathcal{L}(\Phi, W, F)$, a collective utility function σ, and an integer K, is there a partition $\langle M_1, \ldots, M_n \rangle$ of $\mathcal{PS}$ such that $\sigma(u_{G_1,F}(M_1), \ldots, u_{G_n,F}(M_n)) \geq K$?

5.4 Related Work

The Winner Determination Problem (WDP) for combinatorial auctions is the problem of dividing goods among bidders in such a way as to maximize revenue. These goods may display synergies for some bidders, and usually the bidders will have some way of expressing these synergies in their bids. (For further discussion of combinatorial auctions and the WDP, see Sections 6.2 and 6.4, and also [Cramton, Shoham, and Steinberg, 2006].) The complexity of the Winner Determination Problem for combinatorial auctions has been studied extensively for the OR/XOR family of bidding languages [Fujishima, Leyton-Brown, and Shoham, 1999; Nisan, 2000; Müller, 2006], as well as the effects of restricting bids to certain bundles due to their structure [Rothkopf, Pekeč, and Harstad, 1998]. For certain restricted goalbase languages—in particular, those into which the XOR language may be embedded—our decision problem MAX-UTIL is a degenerate case of the Winner Determination Problem, where only a single bidder shows up for the auction. On the other hand, the Winner Determination Problem is itself a special case of our MAX-CUF decision problem where we restrict ourselves to using summation for both the individual and collective aggregators.

Bouveret [2007, Section 4.2] takes up a version of MAX-CUF similar to the one we discuss here. His MAX-CUF is both more and less general than ours. Bouveret's MAX-CUF is less general than ours in that only positive formulas, no negations, are used in his language for specifying agent's preferences. Because we have languages which permit negation, not just $\wedge$, $\vee$, and propositional variables, we have some languages which are expressively different from his. Bouveret's MAX-CUF is more general in terms of the exogenous constraints which may be imposed on outcomes. The constraint Bouveret calls *preemption* is built into our MAX-CUF, but is a parameter for his MAX-CUF. We treat items as rivalrous goods—agent 1's possession of item a preempts any other agent from possessing item a at the same time—but there are some cases where we might wish to allow joint possession of outcomes and their costs. One example of this is the positioning of a satellite paid for by multiple parties. The satellite's position is a shared good, so it makes sense to divide the cost of positioning the satellite among all parties who wanted it in that position, not just the one who was willing to pay the most. Other constraints which Bouveret permits are *exclusion* and *volume*. Exclusion constraints prevent certain sets of object from being allocated simultaneously (e.g., perhaps we cannot fire our thrusters and take a photo at the same time, so if one of these "goods" is allocated, then the other must not be), while volume constraints place upper

bounds on the number of goods which may be allocated at one time (e.g., perhaps there is insufficient power to run more than five of our satellite's sensors at once). Our MAX-CUF lacks the ability to handle exclusion and volume constraints directly, though with an expressive enough language it will be possible to simulate these using formulas with specially devised weights.[3] Finally, Bouveret considers two aggregators which we do not: in particular, he presents results for some versions of MAX-CUF using min or leximin as the individual aggregator, and some using leximin as the collective aggregator. Where there is overlap between Bouveret's MAX-CUF and ours, we make use of his results showing NP-completeness.

Lang [2004] discusses combinatorial voting, where the candidates to be voted on have a combinatorial structure but where voting on the underlying variables individually is made difficult by dependencies among them. The significant difference between our MAX-CUF and the decision problems studied by Lang—COMPARISON, NON-DOMINANCE, and CAND-OPT-SAT—is that these problems are not partitioning problems, while MAX-CUF is. In the combinatorial vote setting, the fundamental problem is to select the shared result which is socially optimal; in contrast, MAX-CUF is a multiagent resource allocation problem. Concretely, the result of a combinatorial vote might be that the group will have a mushroom risotto with fish and white wine, while a resource allocation problem over the same domain might give one agent the risotto, another the fish, and the bottle of wine to a third. As we shall see in Chapter 7, the two settings are strongly connected; however, we do not exploit that connection here.

At the collective level, there are other interesting problems besides finding allocations which maximize social welfare. For example, given an allocation, we may wish to determine whether it is Pareto-optimal, or whether there can be a Pareto improvement through a series of (possibly individually rational) trades among the agents. We might also wish to consider how satisfied agents are with the bundles they receive. An allocation is *envy-free* if no agent would prefer the bundle received by another agent to his own bundle. While it is trivial to achieve an envy-free allocation—simply burn all of the items and give every agent nothing—it is frequently quite difficult to determine whether there is an allocation which is both efficient and envy-free at the same time. This problem is taken up by Bouveret [2007, Section 4.1] under the name EEF EXISTENCE; under various assumptions about agents' preferences, EEF EXISTENCE may range from being a member of P or being *merely* NP-complete to being complete for classes in the Boolean and polynomial hierarchies, such as coBH_2, Θ_2^{p}, Δ_2^{p}, and Σ_2^{p}. While we present no results about EEF EXISTENCE ourselves, some of Bouveret's results should be applicable to our framework, as should some results of Bouveret and Lang [2008] which additionally cover languages having negation as a connective.

[3]For example, the exclusion constraint which says that we cannot fire our thrusters and take a photo simultaneously might be written as $(t \wedge p, -\infty)$ and added to every agent's goalbase. Note that we can always replace $-\infty$ by some suitably large negative finite value. See the alternative proofs of Theorems 5.5.6 and 5.5.7 and their accompanying Figures 5.2 and 5.3 for an example of how this can be done.

5.5 The Complexity of MAX-UTIL and MIN-UTIL

Who (if anyone) needs to solve MAX-UTIL depends on the context in which our preference representation languages are being applied. Take auctions, for example: Whether MAX-UTIL needs to be solved by the center (e.g., the auctioneer) immediately in order to determine the winner depends on his concrete algorithm; the center *does* solve MAX-UTIL if the resources are shareable. Specifically, MAX-UTIL *is* the Winner Determination Problem for combinatorial auctions where the auctioneer has free disposal, the bidders do not have free disposal, and allocated goods are shared among all bidders. This might at first sound like a strange sort of auction, one where all bidders receive every good won by any bidder—but this is precisely what an election is. The candidates are the goods, and everyone shares whatever good (or goods, in the case of a multi-winner election) is allocated. Solving MAX-UTIL over the admissible models, i.e., the ones which elect the correct number of candidates, tells you who has won the election. Many popular voting methods have analogues in this framework. (For further discussion, see Chapter 7.)

Even in cases where it is not necessary to solve MAX-UTIL in order to solve the Winner Determination Problem, the complexity of MAX-UTIL provides a lower bound on how complex the Winner Determination Problem can be: Observe that in the (degenerate) single-bidder case, the two problems coincide. If only one bidder shows up to the auction, then determining which items he wins is precisely the same as finding his optimal state. Therefore, the Winner Determination Problem can never be easier than MAX-UTIL, as it contains MAX-UTIL as a subproblem.

Finally, for an agent herself it is useful to solve MAX-UTIL if she builds her bids not directly from an explicitly represented utility function, but instead from constraints or through elicitation. In that case, the agent may only be able to figure out her optimal state by solving MAX-UTIL. Here, all value is measured along a single axis, utility. Were we to consider an extension of weighted formulas to encompass multiple, incommensurable measures, as in multi-criteria decision making, it would be even less likely that an agent would be aware of her optimal states, and hence solving MAX-UTIL becomes even more important in that setting.

Our strategy in each of the following subsections is as follows: Where MAX-UTIL is polynomial, we show that for the most expansive languages for which we know it holds. Where MAX-UTIL is NP-hard, we show that for the most restrictive languages we can. The rationale here is that hardness passes upwards to superlanguages, while membership passes downwards to sublanguages, so hardness results for small languages and easiness results for large languages permit us to cover the ground most economically.

Note that if we permit goalbases to contain unsatisfiable formulas, then MAX-UTIL trivializes to the prototypical NP-complete problem SAT, since deciding MAX-UTIL in the general case will involve determining whether any particular formula in a given goalbase is even satisfiable. Therefore, in the cases where we show NP-completeness, we do so *even in the case where goalbases contain only*

satisfiable formulas. In contrast, in the cases where we show that MAX-UTIL is in P, we do so without this restriction. Furthermore, we consider only cases where the set of weights W is a subset of $\mathbb{Q}$, in order to avoid issues of how to represent irrational weights.[4]

5.5.1 Hardness Results for MAX-UTIL

First, we provide an upper bound on the complexity of MAX-UTIL for languages with reasonable aggregation functions:

Lemma 5.5.1. *For any set of formulas* Φ *and polynomially-computable aggregation function* F, MAX-UTIL$(\Phi, \mathbb{Q}, F) \in$ NP.

Proof. Any purported example—that is, a model M for which $u_{G,F}(M) \geq K$—is polynomially checkable, since determining which $(\varphi, w) \in G$ are true in M can be done in polynomial time and by assumption applying F to the weights of true formulas can also be done polynomially. □

All of our NP-completeness results for MAX-UTIL implicitly rely on this lemma for the NP membership part of their proofs; as such, we will not mention it each time it is invoked.

Next, we give a straightforward reduction from the decision problem MAXSAT to MAX-UTIL$(\mathit{forms}, \mathbb{Q}, \Sigma)$ in order to show that MAX-UTIL is NP-complete for the unrestricted language.

Theorem 5.5.2. MAX-UTIL$(\mathit{forms}, \mathbb{Q}, \Sigma)$ *is* NP*-complete.*

Proof. By reduction from the well-known NP-hard problem MAXSAT [Garey and Johnson, 1979]: Convert a MAXSAT instance containing the formulas $\varphi_1, \ldots, \varphi_n$ into the goalbase $\{(\varphi_1, 1), \ldots, (\varphi_n, 1)\}$ and solve MAX-UTIL for that goalbase, using the same integer K from the MAXSAT instance. □

Next, we consider the difficulty of MAX-UTIL for the apparently-simpler k-cubes family of languages, and see that we still do not avoid NP-completeness even after this dramatic reduction in our stock of formulas.

Theorem 5.5.3. MAX-UTIL$(k\text{-}\mathit{cubes}, \mathbb{Q}^+, \Sigma)$ *is* NP*-complete for* $k \geq 2$*, even if goalbases contain only satisfiable formulas.*

Proof. The decision problem MAX k-CONSTRAINT SAT is defined as: Given a set C of k-cubes in $\mathcal{PS}$ and an integer K, check whether there is a model $M \in 2^{\mathcal{PS}}$ which satisfies at least K of the k-cubes in C. MAX-UTIL$(k\text{-}\mathit{cubes}, \mathbb{Q}^+, \Sigma)$ is a weighted version of MAX k-CONSTRAINT SAT, which is NP-complete for $k \geq 2$

[4] As users of goalbase languages are unlikely to want to specify irrational weights, we do not view this as a significant limitation. For applications where $(p \wedge q, \pi^e + \sqrt{2})$ is required, users will find that that $\mathbb{Q}$ affords them arbitrarily close rational approximations.

[Ausiello, Crescenzi, Gambosi, Kann, Marchetti-Spaccamela, and Protasi, 1999, LO12, Appendix B]. □

Note that we are able to prove the stronger result, namely that we have NP-completeness even when we know that all formulas in our goalbases are satisfiable formulas, due to the fact that UNSAT is polynomial for cubes. (Algorithm: Sort the literals within the cube. Check whether there are adjacent p and $\neg p$ in the sorted cube.) Because MAX-UTIL(2-*cubes*, $\mathbb{Q}^+, \Sigma$) is a subproblem of MAX-UTIL(*forms*, $\mathbb{Q}, \Sigma$), we may state as a corollary the following even stronger result for the unrestricted language:

Corollary 5.5.4. MAX-UTIL(*forms*, $\mathbb{Q}, \Sigma$) *is* NP*-complete, even if goalbases contain only satisfiable formulas.*

In the remaining NP-completeness results for sum languages in this subsection, we do not state the requirement that goalbases contain only satisfiable formulas, as this requirement is vacuous for languages (such as clauses and strictly positive cubes) which contain no unsatisfiable formulas.

Theorem 5.5.5. MAX-UTIL(k-*clauses*, $\mathbb{Q}^+, \Sigma$) *is* NP*-complete for* $k \geq 2$.

Proof. MAX-UTIL(2-*clauses*, $\mathbb{Q}^+, \Sigma$) is a weighted version of the well-known NP-complete problem MAX 2-SAT [Garey and Johnson, 1979]. Furthermore, MAX-UTIL(k-*clauses*, $\mathbb{Q}^+, \Sigma$) contains MAX-UTIL(k-*clauses*, $\mathbb{Q}^+, \Sigma$) for $k \geq 2$. □

We have now seen that neither short cubes nor short clauses will keep MAX-UTIL from being NP-hard. We might instead try to trade negation in our formulas for negative weights. However, this fails for strictly positive cubes, as shown by the following theorem:

Theorem 5.5.6. MAX-UTIL(k-*spcubes*, $\mathbb{Q}, \Sigma$) *is* NP*-complete for* $k \geq 2$.

Proof. We show NP-hardness for $k = 2$ by reduction from MAX 2-SAT [Garey and Johnson, 1979], using a construction previously employed by Chevaleyre et al. [2008a] to show NP-hardness of the Winner Determination Problem in combinatorial auctions when bids are encoded using k-additive functions. Let S be a set of 2-clauses and let $K \leq |S|$. MAX 2-SAT asks whether there exists a subset S' of S with $|S'| \geq K$ that is satisfiable. We construct a goalbase G as follows:

- For any literal $\ell \in S$, add $(\ell, 1)$ to G.
- For any clause $p \vee q \in S$, add $(p, 1)$, $(q, 1)$, and $(p \wedge q, -1)$ to G.
- For any clause $p \vee \neg q \in S$, add $(\top, 1)$, $(q, -1)$, and $(p \wedge q, 1)$ to G.
- For any clause $\neg p \vee \neg q \in S$, add $(\top, 1)$ and $(p \wedge q, -1)$ to G.

Clearly, there exists a satisfiable $S' \subseteq S$ with $|S'| \geq K$ iff there exists a model M such that $u_G(M) \geq K$. We are not yet done, because G is not a goalbase in *strictly* positive cubes. Let G' be the result of removing all occurrences of $(\top, 1)$ from G. If d is the number of nonpositive clauses in S, then $u_{G'}(M) = u_G(M) - d$ for any model M. Hence, MAX 2-SAT for S will succeed iff there exists a model M such that $u_{G'}(M) \geq K - d$. Therefore, MAX-UTIL(2-*spcubes*, $\mathbb{Q}, \Sigma$) must be at least as hard as MAX 2-SAT. □

Similarly, we cannot avoid NP-completeness by using short positive clauses if we want also to have negative weights.

Theorem 5.5.7. MAX-UTIL(k-*pclauses*, $\mathbb{Q}, \Sigma$) *is* NP-*complete for* $k \geq 2$.

Proof. The proof works by reduction from MAX 2-SAT, just as for Theorem 5.5.6, except that now we construct G as follows:

- For any literal $\ell \in S$, add $(\ell, 1)$ to G.
- For any clause $p \vee q \in S$, add $(p \vee q, 1)$ to G.
- For any clause $p \vee \neg q \in S$, add $(\top, 1)$, $(p, 1)$, and $(p \vee q, -1)$ to G.
- For any clause $\neg p \vee \neg q \in S$, add $(\top, 1)$, $(p, -1)$, $(q, -1)$, and $(p \vee q, 1)$ to G.

As $\top$ is not a positive clause, we must eliminate all occurrences of $(\top, 1)$ in the same way as we did in the proof of Theorem 5.5.6. □

We have already seen that restricting ourselves to short clauses and positive weights is insufficient to keep MAX-UTIL polynomial (supposing that P ≠ NP). We might try to avoid NP-completeness by imposing an additional constraint on our clauses, namely we could force them to be Horn. (Recall that a Horn clause is a clause with at most one positive literal.) Certain problems are known to be easier with Horn clauses than with general formulas, e.g., HORNSAT is only P-complete, while SAT is NP-complete [Papadimitriou, 1994a, Corollary, p. 176]. Unfortunately, the restriction to Horn clauses is still not severe enough:

Theorem 5.5.8. MAX-UTIL(k-*Horn*, $\mathbb{Q}^+, \Sigma$) *is* NP-*complete for* $k \geq 2$.

Proof. The problem MAX HORN 2-SAT is NP-complete [Jaumard and Simeone, 1987, Proposition 3.1]. We exhibit a polynomial reduction from MAX HORN 2-SAT to MAX-UTIL(2-*Horn*, $\mathbb{Q}^+, \Sigma$): Given a set C of 2-Horn formulas and an integer K, construct a goal base $G = \{(c, 1) \mid c \in C\}$. Then there is a model M satisfying at least K 2-Horn formulas in C iff there is a model M (actually, the same M) for which $u_G(M) \geq K$. MAX-UTIL(2-*Horn*, $\mathbb{Q}^+, \Sigma$) is contained in MAX-UTIL(k-*Horn*, $\mathbb{Q}^+, \Sigma$) for $k \geq 2$, so MAX-UTIL(k-*Horn*, $\mathbb{Q}^+, \Sigma$) is NP-complete for $k \geq 2$. □

1. For each $(\bigwedge X, w) \in G$:
   (a) Let $X' = \{x \mid x \in X\} \cup \{\bar{x} \mid \neg x \in X\}$.
   (b) Put $(\bigwedge X', w) \in G'_0$.
2. Normalize G'_0 to $[-1, 1]$.
3. Let $\delta = \sum_{(\varphi,w)\in G'_0} |w|$.
4. Let $\alpha = \delta + 1$, and $\beta = -3\delta - 3$.
5. For each $x \in \mathcal{PS}$, put $(x, \alpha), (\bar{x}, \alpha), (x \wedge \bar{x}, \beta) \in G'_1$.
6. Let $G' = G'_0 \oplus G'_1$.

Figure 5.2: Translation from $\mathcal{L}(k\text{-}cubes, \mathbb{Q}, \Sigma)$ to $\mathcal{L}(k\text{-}pcubes, \mathbb{Q}, \Sigma)$.

The nature of NP-completeness guarantees that there is an abundance of different proofs reducing one NP-complete problem to another. In principle, any two NP-complete problems, no matter how different they are on the surface, are interreducible. How straightforward or baroque such reductions will be depends on how structurally similar the problems involved are. (The observant reader will notice that we have made use only of logic-based problems to this point. Having formulas on both sides narrows the conceptual gap between the source and target language for us, so is helpful, though not necessary. We could as well have done all of our NP-hardness reductions from TRAVELING SALESMAN [Garey and Johnson, 1979] or even MINESWEEPER [Kaye, 2000], had we wanted to produce a chapter full of gratuitous, difficult, gnarly reductions.) That said, we offer alternative proofs of Theorems 5.5.6 and 5.5.7, because the reductions used there are of particular interest. In reducing MAX-UTIL(k-*cubes*, $\mathbb{Q}, \Sigma$) to MAX-UTIL(k-*pcubes*, $\mathbb{Q}, \Sigma$), we exhibit a technique for simulating general cubes as positive cubes by converting negative literals into new propositional variables while maintaining the correct logical relationships through the addition of carefully selected penalty weights.

First, the alternative proof of Theorem 5.5.6:

Proof. MAX-UTIL(k-*cubes*, $\mathbb{Q}^+, \Sigma$) is NP-complete for $k \geq 2$ by Theorem 5.5.3; MAX-UTIL(k-*cubes*, $\mathbb{Q}, \Sigma$) contains it for any fixed k, so is NP-complete also. Now we exhibit a polynomial reduction from MAX-UTIL(k-*cubes*, $\mathbb{Q}, \Sigma$) to MAX-UTIL(k-*pcubes*, $\mathbb{Q}, \Sigma$). Given a goalbase $G \in \mathcal{L}(k\text{-}cubes, \mathbb{Q}, \Sigma)$, construct G' as in Figure 5.2.

Let $\overline{\mathcal{PS}} = \{\bar{p} \mid p \in \mathcal{PS}\}$. That is, $\overline{\mathcal{PS}}$ is purely syntactic, and contains new atoms which differ from the old atoms by virtue of the bar drawn over them.

Lemma 5.5.9. *Fix $A \subseteq \mathcal{PS} \cup \overline{\mathcal{PS}}$ such that $x, \bar{x} \notin A$. Then $u_{G'}(A \cup \{x, \bar{x}\}) < u_{G'}(A) < u_{G'}(A \cup \{x\}), u_{G'}(A \cup \{\bar{x}\})$.*

Proof. Note that for any two models M, N we have that $|u_{G'_0}(M) - u_{G'_0}(N)| \leq \delta$. δ is a (not necessarily tight) upper bound on the utility change in G'_0 between arbitrary models. This fact is used below to bound away the terms $u_{G'_0}(A \cup \{x\})$ and $u_{G'_0}(A \cup \{x, \bar{x}\})$:

$$\begin{aligned}
u_{G'}(A \cup \{x\}) &= u_{G'_0}(A \cup \{x\}) + u_{G'_1}(A \cup \{x\}) \\
&= u_{G'_0}(A \cup \{x\}) + u_{G'_1}(A) + w_x^{G'_1} \\
&= u_{G'_0}(A \cup \{x\}) + u_{G'_1}(A) + \delta + 1 \\
&\geq u_{G'_0}(A) - \delta + u_{G'_1}(A) + \delta + 1 \\
&= u_{G'_0}(A) + u_{G'_1}(A) + 1 \\
&> u_{G'_0}(A) + u_{G'_1}(A) = u_{G'}(A)
\end{aligned}$$

Similarly, $u_{G'}(A \cup \{\bar{x}\}) > u_{G'}(A)$. Finally,

$$\begin{aligned}
u_{G'}(A \cup \{x, \bar{x}\}) &= u_{G'_0}(A \cup \{x, \bar{x}\}) + u_{G'_1}(A \cup \{x, \bar{x}\}) \\
&= u_{G'_0}(A \cup \{x, \bar{x}\}) + u_{G'_1}(A) + w_x^{G'_1} + w_{\bar{x}}^{G'_1} + w_{x \wedge \bar{x}}^{G'_1} \\
&= u_{G'_0}(A \cup \{x, \bar{x}\}) + u_{G'_1}(A) - \delta - 1 \\
&\leq u_{G'_0}(A) + \delta + u_{G'_1}(A) - \delta - 1 \\
&= u_{G'_0}(A) + u_{G'_1}(A) - 1 \\
&< u_{G'_0}(A) + u_{G'_1}(A) = u_{G'}(A)
\end{aligned}$$

(Lemma 5.5.9) □

If M' is a model in $\mathcal{PS} \cup \overline{\mathcal{PS}}$, let $M = M' \setminus \overline{\mathcal{PS}}$. By Lemma 5.5.9, we have that every model optimal for $u_{G'}$ will contain exactly one of x and $\bar{x}$ for all $x \in \mathcal{PS}$. (If M' contains both x and $\bar{x}$, we could gain at least 1 utility by removing both; if M' has neither, we could gain at least 1 utility by adding one.) Call a model M' in $\mathcal{PS} \cup \overline{\mathcal{PS}}$ *full* if for every $x \in \mathcal{PS}$ either $x \in M'$ or $\bar{x} \in M'$, and *bivalent* if for every $x \in \mathcal{PS}$ either $x \notin M'$ or $\bar{x} \notin M'$. Whenever M' is full and bivalent, M will be a model in $\mathcal{PS}$. An operation which converts a goalbase G to another goalbase G' is *order-preserving* over models if for all $M, M' \subseteq \mathcal{PS}$, $u_G(M) < u_G(M')$ iff $u_{G'}(M) < u_{G'}(M')$.

All of the operations applied in generating G' from G are order-preserving over full, bivalent models: Consider G'_0 prior to normalization. $u_{G'_0}(X') = u_G(X)$ for all models X. Normalization is order-preserving. Every full, bivalent model is optimal for $u_{G'_1}$, since all full, bivalent models have the same value ($w_x = w_{\bar{y}}$ and $w_{x \wedge \bar{x}} = w_{y \wedge \bar{y}}$ for all $x, y \in \mathcal{PS}$) and by Lemma 5.5.9 all nonfull or nonbivalent models are strictly dominated. Adding G'_1 to G'_0 increases every atomic weight

1. For each $(\bigvee X, w) \in G$:
   (a) Let $X' = \{x \mid x \in X\} \cup \{\bar{x} \mid \neg x \in X\}$.
   (b) Put $(\bigvee X', w) \in G'_0$.
2. Normalize G'_0 to $[-1, 1]$.
3. Let $\delta = \sum_{(\varphi,w)\in G'_0} |w|$.
4. Let $\alpha = -2\delta - 2$, and $\beta = 3\delta + 3$.
5. For each $x \in \mathcal{PS}$, put $(x, \alpha), (\bar{x}, \alpha), (x \vee \bar{x}, \beta) \in G'_1$.
6. Let $G' = G'_0 \oplus G'_1$.

Figure 5.3: Translation from $\mathcal{L}(k\text{-}clauses, \mathbb{Q}, \Sigma)$ to $\mathcal{L}(k\text{-}pclauses, \mathbb{Q}, \Sigma)$.

by α, which is order-preserving; and increases $w_{x\wedge\bar{x}}$ by β, which has no effect at all since $x \wedge \bar{x}$ is false on every bivalent model. Therefore, if $u_{G'}(X') < u_{G'}(Y')$ where X' and Y' are full and bivalent, then $u_G(X) < u_G(Y)$.

Suppose that M' is optimal for $u_{G'}$. It follows from the Lemma that M' is full and bivalent, so it follows from the above that M is optimal for u_G. This completes the reduction of MAX-UTIL$(k\text{-}cubes, \mathbb{Q}, \Sigma)$ to MAX-UTIL$(k\text{-}pcubes, \mathbb{Q}, \Sigma)$. Generating G' from G and recovering M from M' are operations linear in the size of G and $\mathcal{PS}$, respectively, so the reduction is polynomial. Hence MAX-UTIL$(k\text{-}pcubes, \mathbb{Q}, \Sigma)$ is NP-complete. □

This method can easily be adapted to obtain the analogous NP-completeness result for positive clauses, giving an alternative proof of Theorem 5.5.7:

Proof. Similar to the proof for the NP-completeness of MAX-UTIL$(k\text{-}pcubes, \mathbb{Q}, \Sigma)$. Given a goalbase $G \in \mathcal{U}(k\text{-}clauses, \mathbb{Q}, \Sigma)$, construct G' as in Figure 5.3.

As in the previous proof, construction of G' from G is order-preserving over full, bivalent models. ($x \vee \bar{x}$ is true in every full, bivalent model and hence the disjunctive weights do not disturb the ordering.) Hence by the same argument, MAX-UTIL$(k\text{-}clauses, \mathbb{Q}, \Sigma)$ reduces polynomially to MAX-UTIL$(k\text{-}pclauses, \mathbb{Q}, \Sigma)$, and hence MAX-UTIL$(k\text{-}pclauses, \mathbb{Q}, \Sigma)$ is NP-complete. □

5.5.2 Easiness Results for MAX-UTIL

The previous subsection might leave us wondering whether we can *ever* avoid NP-completeness for MAX-UTIL for sum languages, as we found NP-completeness

wherever we looked. For two sum languages (and all their sublanguages), however, we *can* obtain easiness results. The first of these is MAX-UTIL$(\mathit{pforms}, \mathbb{Q}^+, \Sigma)$, where we take advantage of the fact that the largest optimal state is easy to construct.

Theorem 5.5.10. MAX-UTIL$(\mathit{pforms}, \mathbb{Q}^+, \Sigma) \in \mathsf{P}$.

Proof. Since all weights are positive, whichever state makes the most formulas true is optimal. Because all formulas in the language are positive, we are guaranteed that every formula we encounter is satisfiable. In particular, the state $\mathcal{PS}$, in which all atoms are true, makes every positive formula true, and hence $\mathcal{PS}$ is always an optimal state. (In fact, $\mathcal{PS}$ is the maximal optimal state. There might also be optimal states making fewer atoms true.) This means that the algorithm which checks whether $u(\mathcal{PS}) \geq K$ decides every instance of MAX-UTIL$(\mathit{pforms}, \mathbb{Q}^+, \Sigma)$; furthermore, finding the value of any single state is linear. □

Next, we give a constructive proof for literals. Here, we use the fact that all utility functions in $\mathcal{U}(\mathit{literals}, \mathbb{Q}, \Sigma)$ are modular to decide for each item whether it should be in or out of an optimal model.

Theorem 5.5.11. MAX-UTIL$(\mathit{literals}, \mathbb{Q}, \Sigma) \in \mathsf{P}$.

Proof. A simple polynomial algorithm: Fix a goalbase $G \in \mathcal{L}(\mathit{literals}, \mathbb{Q}, \Sigma)$. Keep for each atom p a number δ_p, the difference between the sum of p's positive occurrences and sum of p's negative occurrences seen so far. (Initially $\delta_p = 0$.) Iterate over the formulas in G, updating the deltas as we go. (Thus, on seeing $(\neg p, 5)$, we subtract 5 from δ_p.) On reaching the end of the goalbase, define a model $M = \{p \mid \delta_p > 0\}$. M will be the minimal optimal model. (The maximal optimal model is $\{p \mid \delta_p \geq 0\}$.) This algorithm is $O(n \log n)$, since for each literal in G, we have to retrieve the corresponding δ_p. □

In contrast to the hardness results we have for most sum languages, solving MAX-UTIL for *any* max language is trivial:

Theorem 5.5.12. MAX-UTIL$(\Phi, W, \max)$ *is linear in the size of the goalbase, for any* $\Phi \subseteq \mathcal{L}_{\mathcal{PS}}$ *and any* $W \subseteq \mathbb{Q}$*, so long as goalbases contain only satisfiable formulas.*

Proof. An algorithm solving MAX-UTIL for any max language simply has to iterate over the formulas in the goalbase, answer affirmatively as soon as it encounters a (φ, w) for which $w \geq K$, and answer negatively otherwise. □

This complexity result requires some discussion. First, without the restriction to satisfiable formulas, MAX-UTIL$(\Phi, W, \max)$ is NP-complete, as lifting this restriction imposes the additional requirement of checking whether φ is satisfiable whenever $w \geq K$. Second (assuming that we retain the satisfiability condition),

we must be careful about how we interpret the low complexity of MAX-UTIL. Note that our algorithm does not *compute* the actual model M yielding the desired level of utility; it only checks whether such an M *exists*. If we also require M itself, then we still need to extract a satisfying model M from some goal (φ, w) where $w \geq K$.

The problem of finding a satisfying assignment for an arbitrary formula that is already known to be satisfiable is probably still intractable: FSAT, which is the function problem version of SAT, is complete for FNP, the extension of NP to function problems. Given a formula φ, FSAT will return either a satisfying model M, or "no" if there is no satisfying model. If we somehow know already that φ is satisfiable, then we know that FSAT will always give us a model instead of answering "no". Call this subproblem of FSAT where the input formulas are guaranteed to be satisfiable TFSAT (for "total" FSAT). TFSAT is a member of the class TFNP, which is the subset of FNP where all problems are total—that is to say, these problems never return "no" as an answer. Clearly, $\mathsf{FP} \subseteq \mathsf{TFNP} \subseteq \mathsf{FNP}$, but no more beyond that is known. If $\mathsf{FP} = \mathsf{TFNP}$, this would imply that $\mathsf{P} = \mathsf{NP} \cap \mathsf{coNP}$, which is considered unlikely [Papadimitriou, 1994b]. Hence, it is likely that there is no polynomial algorithm for finding a satisfying assignment for an arbitrary known-satisfiable formula, so in general, the low complexity of MAX-UTIL for max languages does not imply low complexity of the corresponding function problem which finds a witness.

In contrast to this observation, for sum languages, we are not aware of any case where the complexity of checking existence of an alternative giving at least K utility and computing that alternative differ, so long as we restrict ourselves to languages closed under substitution of logical constants.[5] For languages with an NP-complete MAX-UTIL this is a non-issue; for all sum languages with polynomial MAX-UTIL the proofs are constructive and directly show the computation of the top alternative to be polynomial.

Finally, we stress that both of these limitations of Theorem 5.5.12—the assumption that goals are satisfiable, and the difference for $\mathcal{L}(\mathit{forms}, W, \max)$ between solving MAX-UTIL and actually computing the best alternative—vanish for the max languages considered in this chapter which do not permit arbitrary formulas. For both cubes and clauses (and any of their sublanguages) determining the satisfiability of single formulas and finding a model for a single satisfiable formula are trivial tasks.

5.5.3 The Complexity of MIN-UTIL

So far, we have considered optimal states, but what of pessimal states? Just as an agent may wish to know how well he can do, he may wish to know how poorly,

[5]For languages which are *not* closed under substitution of logical constants, it is not always the case that the decision problem can be used to solve the function problem. For a discussion of this, see Section 5.7.1.

as well. MIN-UTIL can be seen as the pessimistic dual of the optimistic MAX-UTIL, in the sense that it checks lower bounds instead of upper bounds. (Note that MIN-UTIL is *not* the complement of MAX-UTIL: This can easily be seen from the problem instance $\langle\{(\top, 1)\}, 1\rangle$, which is a member of both decision problems, for many different languages and choices of aggregators.)

First, we give an upper bound on the complexity of MIN-UTIL for sum languages:

Lemma 5.5.13. *For any* Φ, $\text{MIN-UTIL}(\Phi, \mathbb{Q}, \Sigma) \in \mathsf{coNP}$.

Proof. Any purported counterexample—that is, a model M for which $u_G(M) < K$—is polynomially checkable. □

With an upper bound in hand, we are in position to give a direct proof of the complexity of MIN-UTIL for the full language, using a straightforward reduction from the problem UNSAT.

Theorem 5.5.14. $\text{MIN-UTIL}(\mathit{forms}, \mathbb{Q}, \Sigma)$ *is* coNP*-complete.*

Proof. coNP membership follows from Lemma 5.5.13. For coNP-hardness: Let φ be an instance of UNSAT, and $\langle\{(\neg\varphi, 1)\}, 1\rangle$ an instance of $\text{MIN-UTIL}(\mathit{forms}, \mathbb{Q}, \Sigma)$. It is easy to see that if φ is not satisfiable, then $u_{\{(\neg\varphi,1)\}}(M) = 1$ for all models M, and vice versa. Hence UNSAT reduces to $\text{MIN-UTIL}(\mathit{forms}, \mathbb{Q}, \Sigma)$. UNSAT is a well-known coNP-hard problem. □

While we could proceed by giving an independent proof demonstrating the complexity of MIN-UTIL for each of the other sum languages, we do not do so; instead, we prove the following lemma to exploit a connection between the complexity of MAX-UTIL and MIN-UTIL for sum languages:

Lemma 5.5.15. *Let* $\mathcal{C}$ *be a complexity class closed under polynomial-time many-one reductions,* W *be a set of weights, and* $-W = \{-w \mid w \in W\}$. *Then:*

1. $\text{MAX-UTIL}(\Phi, W, \Sigma) \in \mathcal{C}$ *iff* $\text{MIN-UTIL}(\Phi, -W, \Sigma) \in \mathsf{co}\,\mathcal{C}$.
2. $\text{MAX-UTIL}(\Phi, W, \Sigma)$ *is* $\mathcal{C}$*-hard iff* $\text{MIN-UTIL}(\Phi, -W, \Sigma)$ *is* $\mathsf{co}\,\mathcal{C}$*-hard.*

Proof. Recall that

$$\overline{\text{MAX-UTIL}(\Phi, W, \Sigma)} = \{\langle G, K\rangle \mid \langle G, K\rangle \notin \text{MAX-UTIL}(\Phi, W, \Sigma)\}.$$

Since $\overline{\text{MAX-UTIL}}$ is the complementary problem to MAX-UTIL we have by definition that $\overline{\text{MAX-UTIL}} \in \mathsf{co}\,\mathcal{C}$ iff MAX-UTIL $\in \mathcal{C}$. If G is a goalbase, let $-G = \{(\varphi, -w) \mid (\varphi, w) \in G\}$. Clearly we have that

$$\langle G, K\rangle \in \overline{\text{MAX-UTIL}(\Phi, W, \Sigma)} \iff \langle -G, -K\rangle \in \text{MIN-UTIL}(\Phi, -W, \Sigma)$$

which shows that $\overline{\text{MAX-UTIL}(\Phi, W, \Sigma)}$ and $\text{MIN-UTIL}(\Phi, -W, \Sigma)$ are interreducible, and hence that $\text{MIN-UTIL}(\Phi, -W, \Sigma) \in \mathsf{co}\,\mathcal{C}$ also. Similarly, if MAX-UTIL is $\mathcal{C}$-hard, then $\overline{\text{MAX-UTIL}}$ is $\mathsf{co}\,\mathcal{C}$-hard by definition, and due to the interreducibility of $\overline{\text{MAX-UTIL}(\Phi, W, \Sigma)}$ and $\text{MIN-UTIL}(\Phi, -W, \Sigma)$, we have that $\text{MIN-UTIL}(\Phi, -W, \Sigma)$ is $\mathsf{co}\,\mathcal{C}$-hard as well. □

This lemma permits us to immediately derive complexity results for MIN-UTIL corresponding to some of those for MAX-UTIL above:

Theorem 5.5.16. MIN-UTIL(k-*spcubes*, $\mathbb{Q}, \Sigma$) *is* coNP-*complete for* $k \geq 2$.

Theorem 5.5.17. MIN-UTIL(k-*pclauses*, $\mathbb{Q}, \Sigma$) *is* coNP-*complete for* $k \geq 2$.

Theorem 5.5.18. MIN-UTIL(*literals*, $\mathbb{Q}, \Sigma$) $\in$ P.

We may apply Lemma 5.5.15 in these cases because the rationals are closed under negation (that is, $\mathbb{Q} = -\mathbb{Q}$). Theorem 5.5.18 relies on the fact that deterministic complexity classes are closed under complementation, so we may say P there instead of coP [Papadimitriou, 1994a, p. 142]. (Note also that we could have used Lemma 5.5.15 to immediately derive Theorem 5.5.14; we chose to give the reduction from UNSAT instead to show how a direct reduction would look, and because the construction there is reused later in Theorem 5.5.22. Additionally, the algorithm given for deciding MAX-UTIL(*literals*, $\mathbb{Q}, \Sigma$) in the proof of Theorem 5.5.11 may be used to construct a pessimal model by taking the complement of the maximal optimal model, so MIN-UTIL(*literals*, $\mathbb{Q}, \Sigma$) is $O(n \log n)$.)

For MIN-UTIL(*pforms*, $\mathbb{Q}^+, \Sigma$), we give a direct proof because its set of weights is not closed under negation, so Lemma 5.5.15 is not applicable:

Theorem 5.5.19. MIN-UTIL(*pforms*, $\mathbb{Q}^+, \Sigma$) $\in$ P.

Proof. For any instance $\langle G, K \rangle$, we know that u_G is monotone. Hence, the worst state is $\emptyset$. Therefore, $\langle G, K \rangle \in$ MIN-UTIL(*pforms*, $\mathbb{Q}^+, \Sigma$) iff $u_G(\emptyset) \geq K$, which can be verified polynomially. □

We characterize the complexity of MIN-UTIL for the remaining sum languages using a reduction from the complement of MIN 2-SAT:

Theorem 5.5.20. MIN-UTIL(k-*clauses*, $\mathbb{Q}^+, \Sigma$) *is* coNP-*complete for* $k \geq 2$.

Proof. coNP membership follows from Lemma 5.5.13. For coNP-hardness, we give a reduction from (the complement of) MIN 2-SAT, which is NP-complete [Garey and Johnson, 1979]. An instance of MIN 2-SAT is $\langle C, K \rangle$, where C is a set of 2-clauses and K an integer, and $\langle C, K \rangle \in$ MIN 2-SAT iff there is a model M which satisfies no more than K of the clauses. Given a MIN 2-SAT instance $\langle C, K \rangle$, construct the MIN-UTIL(2-*clauses*, $\mathbb{Q}^+, \Sigma$) instance $\langle \{(\varphi, 1) \mid \varphi \in C\}, K + 1 \rangle$. If at least $K + 1$ clauses are true regardless of the model, then it is false that there is a state where at most K clauses are true, and vice versa. Hence, $\langle \{(\varphi, 1) \mid \varphi \in C\}, K + 1 \rangle \in$ MIN-UTIL(2-*clauses*, $\mathbb{Q}^+, \Sigma$) iff $\langle C, K \rangle \notin$ MIN 2-SAT. Hence MIN-UTIL(2-*clauses*, $\mathbb{Q}^+, \Sigma$) is coNP-complete. □

Theorem 5.5.21. MIN-UTIL(k-*cubes*, $\mathbb{Q}^+, \Sigma$) *is* coNP-*complete for* $k \geq 2$, *even if goalbases contain only satisfiable formulas.*

Proof. coNP membership follows from Lemma 5.5.13. For coNP-hardness: We first note that MIN 2-CONSTRAINT SAT, which is the minimization analog of MAX 2-CONSTRAINT SAT, is NP-complete: Let $\langle C, K\rangle$ be a MAX 2-SAT instance. Let $C' = \{\neg\varphi \mid \varphi \in C\}$. (Since C is a set of 2-clauses, by De Morgan's Law C' is a set of 2-cubes.) Then $\langle C', |C| - K\rangle \in$ MIN 2-CONSTRAINT SAT iff $\langle C, K\rangle \in$ MAX 2-SAT, since every false member of C is a true member of C'. Having established that MIN 2-CONSTRAINT SAT is NP-hard, its complement may be reduced to MIN-UTIL(k-*cubes*, $\mathbb{Q}^+, \Sigma$) using the same construction as in Theorem 5.5.20. □

We have seen that for sum languages, MIN-UTIL behaves similarly to MAX-UTIL. However, this is not the case for max languages:

Theorem 5.5.22. MIN-UTIL(*forms*, $\mathbb{Q}$, max) *is* coNP*-complete.*

Proof. For coNP membership: Any purported counterexample state M is polynomially checkable, simply evaluating $u_{G,\max}(M)$ to see if it is less than K.

For coNP-hardness: The reduction from UNSAT to MIN-UTIL(*forms*, $\mathbb{Q}, \Sigma$) in Theorem 5.5.14 relies on constructing a single-formula goalbase. For singleton goalbases, max and sum have the same behavior, so the construction used there reduces UNSAT to MIN-UTIL(*forms*, $\mathbb{Q}$, max) as well. □

If we restrict the goalbases in our inputs to those containing no superfluous formulas, however, we get a more favorable result for MIN-UTIL(*forms*, $\mathbb{Q}$, max):

Theorem 5.5.23. MIN-UTIL(*forms*, $\mathbb{Q}$, max) $\in$ P*, when restricted to goalbases containing no superfluous formulas.*

Proof. Since no $(\varphi, w) \in G$ is superfluous, any such φ will determine the value of $u_G(M)$ for some model M. Hence, the value of the worst state may be found simply by finding the (φ, w) with the least w. If that $w \geq K$, the MIN-UTIL instance is positive, and negative otherwise. □

As with the sum languages over the same sets of formulas, there are some max languages for which MIN-UTIL remains polynomial in the absence of any further restrictions. This may be seen in the following three theorems.

Theorem 5.5.24. MIN-UTIL(*pforms*, $\mathbb{Q}$, max) $\in$ P.

Proof. Same proof as for MIN-UTIL(*pforms*, $\mathbb{Q}, \Sigma$) in Theorem 5.5.19. The least-valued state is always $\emptyset$. Check whether $u_{G,\max}(\emptyset) \geq K$. □

Theorem 5.5.25. MIN-UTIL(*literals*, $\mathbb{Q}$, max) $\in$ P.

Proof. We present a polynomial-time algorithm: Find δ_p for each atom, and construct the maximal optimal model $M = \{p \mid \delta_p \geq 0\}$ as in the proof of Theorem 5.5.18. Then $\mathcal{PS} \setminus M$ will be the minimal pessimal model (the smallest worst-case model). Check whether $u_{G,\max}(\mathcal{PS} \setminus M) \geq K$. □

Theorem 5.5.26. MIN-UTIL$(\mathit{cubes}, \mathbb{Q}, \max) \in \mathsf{P}$.

Proof. We argue that it is easy to identify and remove superfluous formulas from goalbases in $\mathcal{L}(\mathit{cubes}, \mathbb{Q}, \max)$; once G contains no superfluous formulas, we may invoke Theorem 5.5.23 to show that MIN-UTIL$(\mathit{cubes}, \mathbb{Q}, \max) \in \mathsf{P}$.

First, observe that when $X, Y, X', Y' \subseteq \mathcal{PS}$,

$$\models \left(\bigwedge X \wedge \bigwedge \neg Y\right) \rightarrow \left(\bigwedge X' \wedge \bigwedge \neg Y'\right)$$

is equivalent to

$$X' \subseteq X \text{ and } Y' \subseteq Y, \text{ or } X \cap Y \neq \emptyset.$$

That is to say, testing whether one cube implies another involves only checking whether some sets intersect or are supersets of some other sets, all of which are $O(n \log n)$ operations. This means we can find and remove superfluous cubes from any $G \in \mathcal{L}(\mathit{cubes}, \mathbb{Q}, \max)$ like so:

For each pair of cubes $(\bigwedge X \wedge \bigwedge \neg Y, w), (\bigwedge X' \wedge \bigwedge \neg Y', w') \in G$, if $w' > w$ and either $X' \subseteq X$ and $Y' \subseteq Y$ or $X \cap Y \neq \emptyset$, then $(\bigwedge X \wedge \bigwedge \neg Y, w)$ is superfluous; remove it from G.

This algorithm is quadratic in $|G|$. Once G contains no superfluous formulas, the least remaining weight w may be found and checked for whether $w \geq K$. □

It is worth noting the dramatic difference the choice of aggregator makes for MIN-UTIL over cubes languages: From Theorem 5.5.21, we have that MIN-UTIL is already coNP-complete for positively-weighted 2-cubes using summation, while here we have shown that MIN-UTIL for arbitrarily-weighted cubes of any length remains polynomial when aggregating with max.

5.5.4 Summary

Theorems 5.5.3, 5.5.5, 5.5.6, and 5.5.7 show that MAX-UTIL is NP-complete for every language which contains any of $\mathcal{L}(2\text{-}\mathit{pclauses}, \mathbb{Q}, \Sigma)$, $\mathcal{L}(2\text{-}\mathit{spcubes}, \mathbb{Q}, \Sigma)$, $\mathcal{L}(2\text{-}\mathit{cubes}, \mathbb{Q}^+, \Sigma)$, or $\mathcal{L}(2\text{-}\mathit{clauses}, \mathbb{Q}^+, \Sigma)$. This covers every sum language mentioned in Chapter 3 except $\mathcal{L}(\mathit{pforms}, \mathbb{Q}^+, \Sigma)$ and $\mathcal{L}(\mathit{literals}, \mathbb{Q}, \Sigma)$ and their sublanguages, for which MAX-UTIL is in P. Theorem 5.5.12 shows that MAX-UTIL is in P for all max languages.

For sum languages, MIN-UTIL is like MAX-UTIL but reflected into complementary complexity classes—NP into coNP, P = coP into itself. For max languages, the general case of MIN-UTIL is surprisingly hard, being coNP-complete. The full language at least, is perhaps more suitable for optimists interested in how much utility they may hope to achieve, rather than pessimists interested in how much utility they are guaranteed. On the other hand, as there is no difference in expressivity between $\mathcal{L}(\mathit{cubes}, \mathbb{Q}, \max)$ and $\mathcal{L}(\mathit{forms}, \mathbb{Q}, \max)$ (see Corollary 3.5.5),

Decision Problem				Complexity
MAX-UTIL	2-pclauses	$\mathbb{Q}$	Σ	NP-complete
MAX-UTIL	2-spcubes	$\mathbb{Q}$	Σ	NP-complete
MAX-UTIL	2-clauses	$\mathbb{Q}^+$	Σ	NP-complete
MAX-UTIL	2-cubes	$\mathbb{Q}^+$	Σ	NP-complete
MAX-UTIL	pforms	$\mathbb{Q}^+$	Σ	$O(n)$
MAX-UTIL	literals	$\mathbb{Q}$	Σ	$O(n \log n)$
MAX-UTIL	formulas	$\mathbb{Q}$	max	$O(n)$
MIN-UTIL	2-pclauses	$\mathbb{Q}$	Σ	coNP-complete
MIN-UTIL	2-spcubes	$\mathbb{Q}$	Σ	coNP-complete
MIN-UTIL	2-clauses	$\mathbb{Q}^+$	Σ	coNP-complete
MIN-UTIL	2-cubes	$\mathbb{Q}^+$	Σ	coNP-complete
MIN-UTIL	pforms	$\mathbb{Q}^+$	Σ	$O(n)$
MIN-UTIL	literals	$\mathbb{Q}$	Σ	$O(n \log n)$
MIN-UTIL	pforms	$\mathbb{Q}$	max	$O(n)$
MIN-UTIL	literals	$\mathbb{Q}$	max	$O(n \log n)$
MIN-UTIL	cubes	$\mathbb{Q}$	max	$O(n^2)$
MIN-UTIL	formulas	$\mathbb{Q}$	max	coNP-complete

Table 5.1: Summary of complexity results for MAX-UTIL and MIN-UTIL.

nothing compels us to use the additional formulas we gain by permitting disjunction; and in fact it seems that we are punished with additional complexity for using disjunction in this case.

See Table 5.1 for a complete summary of results for MAX-UTIL and MIN-UTIL.

5.6 The Complexity of Collective Utility Maximization

When there are several agents, each with a utility function encoded using the same language, then the *collective utility maximization problem* (MAX-CUF), the problem of finding a solution maximizing collective utility, is of interest. By "solution" we mean a partition of the set of propositional variables among the agents, thereby fixing a model for each of them. This definition is natural, for instance, if we think of variables as goods.[6]

There are a number of ways in which to define *collective utility* [Moulin, 1988]; the four which are commonly encountered in the literature are egalitarian, utilitarian, elitist, and Nash product:

[6]Other types of solutions, such as *finding* a single model which maximizes collective utility, are also of interest, but shall not be considered here. The combinatorial vote problem of Lang [2004] is exactly this problem, in the context of voting.

Definition 5.6.1 (Common Collective Utility Functions)**.**

- $\sigma = \max$ is the *elitist* collective utility function.
- $\sigma = \min$ is the *egalitarian* collective utility function.
- $\sigma = \Sigma$ is the *utilitarian* collective utility function.
- $\sigma = \Pi$ is the *Nash product* collective utility function.

The *utilitarian* collective utility of an alternative is the sum of the individual utilities. Optimizing with respect to utilitarian collective utility is equivalent to the Winner Determination Problem in combinatorial auctions, where it is interpreted as finding an allocation of goods to bidders that would maximize the sum of the prices offered [Lehmann, Müller, and Sandholm, 2006b]. The *egalitarian* collective utility is the utility of the agent worst off. A finer-grained version of egalitarian collective utility, the *leximin ordering* was advocated by Rawls [1971]. Other options include maximizing the median of the set of individual utilities generated by an alternative (*median-rank dictator*) and maximizing the utility of the agent that is best off (*elitist collective utility*). Finally, the *Nash product*, the product of individual utilities, attempts to strike a balance between fairness and total utility.

First, we state two lemmas bounding the complexity of MAX-CUF:

Lemma 5.6.2. MAX-CUF$(\Phi, W, F, \sigma) \in \mathsf{NP}$ *whenever* F *and* σ *are polynomially-computable functions.*

This holds because whenever F and σ are polynomially-computable functions, we can in all cases easily check whether a given allocation does in fact produce at least K utility.

Before proceeding to our next lemma, we need to define a reasonableness notion for individual and collective utility functions.

Definition 5.6.3 (Singleton Consistency)**.** A function $f\colon \mathbb{R}^* \to \mathbb{R}$ without fixed arity is *singleton consistent* if $f(\alpha) = \alpha$ for all $\alpha \in \mathbb{R}$.

Any reasonable individual aggregator or collective utility function will be singleton consistent. For individual aggregators, singleton consistency means that an agent having exactly one satisfied weighted formula (φ, w) has utility w. For collective utility functions, singleton consistency means that a single-agent society has the same utility as its sole member. Singleton consistent functions give the intuitively right answers for the utility of single agents stranded on desert islands. All of the functions we consider here—min, max, sum, and product—are singleton consistent.

Lemma 5.6.4. MAX-CUF(Φ, W, F, σ) *is at least as hard as* MAX-UTIL(Φ, W, F) *for any singleton-consistent* σ*.*

Here, singleton consistency ensures that σ behaves as the identity function when only a single agent is involved, and hence for such σ MAX-CUF and MAX-UTIL coincide. The preceding two lemmas together imply that MAX-CUF is NP-complete for most languages we have considered; in particular:

Theorem 5.6.5. *The following problems are* NP*-complete for all polynomially-computable, singleton consistent collective utility functions σ:*

1. MAX-CUF$(2\text{-}pclauses, \mathbb{Q}, \Sigma, \sigma)$,
2. MAX-CUF$(2\text{-}spcubes, \mathbb{Q}, \Sigma, \sigma)$,
3. MAX-CUF$(2\text{-}clauses, \mathbb{Q}^+, \Sigma, \sigma)$,
4. MAX-CUF$(2\text{-}cubes, \mathbb{Q}^+, \Sigma, \sigma)$.

Therefore, the interesting cases to investigate are languages which give rise to an easy MAX-UTIL problem, to see if they remain easy under MAX-CUF—$\mathcal{L}(pforms, \mathbb{Q}^+, \Sigma)$, $\mathcal{L}(literals, \mathbb{Q}, \Sigma)$, and $\mathcal{L}(forms, \mathbb{Q}, \max)$—and more restrictive sublanguages of the ones in Theorem 5.6.5 to see if they remain hard.

First, we show that elitist collective utility remains easy for $\mathcal{L}(pforms, \mathbb{Q}^+, \Sigma)$ and $\mathcal{L}(literals, \mathbb{Q}, \Sigma)$:

Theorem 5.6.6.

1. MAX-CUF$(pforms, \mathbb{Q}^+, \Sigma, \max) \in$ P.
2. MAX-CUF$(literals, \mathbb{Q}, \Sigma, \max) \in$ P.

Proof. In both cases: Decide whether $\langle G_i, K\rangle \in$ MAX-UTIL for each agent i. If any $\langle G_i, K\rangle \in$ MAX-UTIL, answer affirmatively; otherwise answer negatively. □

Maximizing utilitarian collective utility is also easy for $\mathcal{L}(literals, \mathbb{Q}, \Sigma)$:

Theorem 5.6.7. MAX-CUF$(literals, \mathbb{Q}, \Sigma, \Sigma) \in$ P.

Proof. For each $a \in \mathcal{PS}$, allocate a to the agent i who maximizes

$$u_{G_i,\Sigma}(\{a\}) + \sum_{\substack{a \in \mathcal{PS} \\ j \neq i}} u_{G_j,\Sigma}(\emptyset).$$

Because all representable utility functions in this language are modular, the allocation built this way will be optimal, and all that remains is to check its value. □

MAX-CUF is easy for max languages when using the elitist collective utility function, for the same reasons as those stated in support of Theorem 5.5.12.

Fact 5.6.8. MAX-CUF$(\Phi, W, \max, \max)$ *is linear in the combined size of the goalbases, for any* $\Phi \subseteq \mathcal{L}_{\mathcal{PS}}$ *and any* $W \subseteq \mathbb{Q}$*, so long as goalbases contain only satisfiable formulas.*

Now we turn to languages more restrictive than (or differently restrictive from) those in Theorem 5.6.5, which nonetheless remain NP-complete. Recall that both MAX-CUF(2-*cubes*, $\mathbb{Q}^+, \Sigma, \Sigma$) and MAX-CUF(2-*spcubes*, $\mathbb{Q}, \Sigma, \Sigma$) are NP-complete. We might ask whether their "intersection", MAX-CUF(2-*spcubes*, $\mathbb{Q}^+, \Sigma, \Sigma$) is also NP-complete. Let us approach from a different direction. We can say that the MAX-CUF(*spcubes*, $\mathbb{Q}^+, \Sigma, \Sigma$) problem is identical to the single-minded bidder allocation problem, proved to be NP-complete by Blumrosen and Nisan [2007, Definition 1.4, Proposition 1.5] via a reduction from the INDEPENDENT SET problem. We can improve this result by showing it not just for spcubes, but for 3-spcubes:

Theorem 5.6.9. MAX-CUF(3-*spcubes*, $\mathbb{Q}^+, \Sigma, \Sigma$) *is* NP*-complete.*

Proof. The problem SET PACKING asks whether given a collection $\mathcal{C}$ of sets and an integer K, there are at least K mutually disjoint sets in $\mathcal{C}$. SET PACKING is NP-complete, even when all $C \in \mathcal{C}$ have $|C| = 3$ [Garey and Johnson, 1979].

We reduce SET PACKING for sets of size 3 to MAX-CUF(3-*spcubes*, $\mathbb{Q}^+, \Sigma, \Sigma$). For each $C \in \mathcal{C}$, construct a goalbase $G_C = \{(\bigwedge C, 1)\}$. Because items cannot be shared, there is no allocation of items to agents which will result in $(\bigwedge C, 1)$ and $(\bigwedge C', 1)$ being satisfied at the same time if $C \cap C' \neq \emptyset$. This enforces that every allocation corresponds to a set packing; and if there is an allocation with at least K utility, then that allocation corresponds to a set packing of at least size K, and vice versa. Therefore $\langle \{G_C\}_{C \in \mathcal{C}}, K \rangle \in$ MAX-CUF(3-*spcubes*, $\mathbb{Q}^+, \Sigma, \Sigma$) iff $\langle \mathcal{C}, K \rangle \in$ SET PACKING restricted to size-3 sets. □

Note that this reduction does not go through for 2-spcubes, since SET PACKING is in P when all $C \in \mathcal{C}$ have $|C| \leq 2$. This is tantalizingly close to the result we were seeking, but whether MAX-CUF(2-*spcubes*, $\mathbb{Q}^+, \Sigma, \Sigma$) is NP-complete remains open.

Several of the languages mentioned in Theorem 5.6.5 may be restricted quite severely and yet MAX-CUF remains NP-complete for them over a variety of aggregators.

Theorem 5.6.10. *All of the following problems are* NP*-complete:*

1. MAX-CUF(2-*spcubes*, $\{0, 1\}, \max, \Sigma$)
2. MAX-CUF(2-*spcubes*, $\{0, 1\}, F, \sigma$)*, where* $F \in \{\max, \Sigma\}$ *and* $\sigma \in \{\min, \Pi\}$*.*

Proof. For the first case, we follow van Hoesel and Müller [2001, Theorem 2] by reducing the known NP-complete problem TRIPARTITE MATCHING [Karp, 1972] to MAX-CUF(2-*spcubes*, $\{0, 1\}, \max, \Sigma$). Instances of TRIPARTITE MATCHING

are $\langle X, Y, Z, T\rangle$, where the sets X, Y, Z are such that $|X| = |Y| = |Z|$ and $T \subseteq X \times Y \times Z$. An instance $\langle X, Y, Z, T\rangle$ is a member iff there is an $M \subseteq T$ which is a perfect matching (i.e., each $x \in X$, $y \in Y$, and $z \in Z$ appears in exactly one triple in M).

For the reduction, we interpret the set X as bidders and the sets Y and Z as goods appearing in 2-spcubes. For each $(x, y, z) \in T$, put $(y \wedge z, 1) \in G_x$. Let $K = |X|$. The only way to achieve K utility by allocating $Y \cup Z$ to the bidders in X is to ensure that at least one bid is satisfied from each G_x; conversely, satisfying more than one bid in any G_x does not increase collective utility, since the individual aggregator is max. Hence $\langle X, Y, Z, T\rangle \in$ TRIPARTITE MATCHING iff $\langle \{G_x\}_{x\in X}, |X|\rangle \in$ MAX-CUF(2-*spcubes*, $\{0, 1\}$, max, Σ).

In the other cases, use the same reduction from TRIPARTITE MATCHING but let $K = 1$. □

This result subsumes parts of the NP-completeness results of Bouveret, Fargier, Lang, and Lemaître [2005, Figure 1] and Bouveret [2007, Proposition 4.22] for $\sigma = \min$. As they do not consider negation, their results (by a reduction from SET PACKING) apply to MAX-CUF(*pforms*, $\mathbb{Q}$, max, min), which contains the problem MAX-CUF(2-*spcubes*, $\{0, 1\}$, max, min) that we have just proved to be NP-complete.

Finally, we give a slightly different result for MAX-CUF using the Nash product as the collective utility function. For the Nash product to be a meaningful metric of social welfare we must restrict ourselves to positive utilities; hence, in this context we assume that all weights are positive and that only goalbases specifying fully defined utility functions are used (e.g., by including $(\top, 0)$ in all max goalbases). To the best of our knowledge, the complexity of this variant of MAX-CUF has not been studied before. In the case of max languages, it is possible to give a simple reduction from the utilitarian case to this one.

Theorem 5.6.11. MAX-CUF(2-*pcubes*, $\{1, 2\}$, max, Π) *is* NP-*complete.*

Proof. We first exhibit a reduction from MAX-CUF(2-*pcubes*, $\mathbb{Q}^+$, max, Σ) to MAX-CUF(2-*pcubes*, $\mathbb{Q}^+$, max, Π). Suppose we are given an instance of the former, with goalbases G_i and bound K. We construct new goalbases G'_i by replacing each weight w in G with 2^w.

Now consider the instance of MAX-CUF(2-*pcubes*, $\mathbb{Q}^+$, max, Π) with the new goalbases G'_i and bound 2^K. Note that $w_1 + \ldots + w_n \geq K$ iff $2^{w_1} \times \ldots \times 2^{w_n} \geq 2^K$. Hence, a model M achieves utilitarian collective utility $\geq K$ with respect to goalbases G_i iff M achieves Nash collective utility $\geq 2^K$ with respect to goalbases G'_i. So the Nash MAX-CUF problem must be at least as hard as the utilitarian MAX-CUF problem.

Then, observe that the same reduction works for the restricted case of MAX-CUF(2-*pcubes*, $\{0, 1\}$, max, Σ) to MAX-CUF(2-*pcubes*, $\{1, 2\}$, max, Π), the former of which was proven to be NP-complete in Theorem 5.6.10. □

Decision Problem					Complexity
MAX-CUF	pforms	$\mathbb{Q}^+$	Σ	max	P
MAX-CUF	literals	$\mathbb{Q}$	Σ	max	P
MAX-CUF	literals	$\mathbb{Q}$	Σ	Σ	P
MAX-CUF	satisfiable φ	$\mathbb{Q}$	max	max	P
MAX-CUF	2-cubes	$\mathbb{Q}^+$	Σ	reasonable[7]	NP-complete
MAX-CUF	2-clauses	$\mathbb{Q}^+$	Σ	reasonable	NP-complete
MAX-CUF	2-pclauses	$\mathbb{Q}$	Σ	reasonable	NP-complete
MAX-CUF	2-spcubes	$\mathbb{Q}$	Σ	reasonable	NP-complete
MAX-CUF	3-spcubes	$\mathbb{Q}^+$	Σ	Σ	NP-complete
MAX-CUF	2-spcubes	$\{0,1\}$	max	Σ	NP-complete
MAX-CUF	2-spcubes	$\{0,1\}$	max	min	NP-complete
MAX-CUF	2-spcubes	$\{0,1\}$	max	Π	NP-complete
MAX-CUF	2-spcubes	$\{0,1\}$	Σ	min	NP-complete
MAX-CUF	2-spcubes	$\{0,1\}$	Σ	Π	NP-complete
MAX-CUF	2-pcubes	$\{1,2\}$	max	Π	NP-complete

Table 5.2: Summary of complexity results for MAX-CUF.

The simple reduction in the proof is possible because we are working with max languages. In this setting, the utility of any model M will always be equal to one of the weights in the goalbase.

5.6.1 Summary

MAX-CUF is significantly harder than MAX-UTIL. There are numerous languages for which MAX-UTIL is polynomial, but where MAX-CUF is NP-complete for some collective utility function. For example, Theorems 5.6.9, 5.6.10, and 5.6.11 each show that a tiny fragment of a language with positive formulas and positive weights already has an NP-complete MAX-CUF problem, despite that MAX-UTIL is trivial over the same languages.

Still open are the complexities of the problems MAX-CUF($\mathit{literals}, \mathbb{Q}, \Sigma, \min$) and MAX-CUF($\mathit{literals}, \mathbb{Q}, \Sigma, \Pi$). While MAX-UTIL is easy for both of the underlying languages, we have neither a polynomial algorithm for solving these nor a reduction from any known NP-complete problem to either one. Additionally, the complexity of MAX-CUF($2\text{-}\mathit{spcubes}, \{0,1\}, \Sigma, \Sigma$) remains unknown.

See Table 5.2 for a summary of results for MAX-CUF.

[7] *Reasonable* here means that the collective utility function is polynomially computable and singleton consistent.

5.7 An Alternate Formulation of MAX-UTIL

MAX-UTIL for sum languages consists in finding an assignment which maximizes the sum of those weights which are associated with satisfied formulas. The complexity of a decision problem version of MAX-UTIL is considered in Section 5.5. The picture which emerges is bipolar: Every language for which a positive result is presented has either a trivial decision problem or an NP-complete one. This naturally led us to wonder whether there are any preference representation languages which occupy the (previously unexplored) middle ground. As we will demonstrate in this section, the fact that we found no natural goalbase languages for which MAX-UTIL has intermediate complexity is a consequence of the design decisions we made when we defined MAX-UTIL. If we consider an alternative form of MAX-UTIL—which we will call MAX-UTIL*—that asks whether particular atoms are true in an optimal model, then we find some languages which occupy that middle ground, where MAX-UTIL* is P-complete.

5.7.1 Revising the MAX-UTIL Decision Problem

Here is the function problem version of MAX-UTIL, corresponding to the decision problem given in Definition 5.3.1.

Definition 5.7.1 (The Function Problem MAX-UTIL)**.** The function problem MAX-UTIL(Φ, W, F) is defined as: Given a goalbase $G \in \mathcal{L}(\Phi, W, F)$, find a model $M \in 2^{\mathcal{PS}}$ such that $u_G(M)$ is maximized.

The relationship between the two should be apparent, the surface differences being that there is no integer K in the input and the answer returned is a model rather than a decision on whether K utility can be met.

In terms of computational complexity, we have already shown the decision problem MAX-UTIL$(\mathit{forms}, \mathbb{Q}, \Sigma)$ to be NP-complete (see Theorem 5.5.2); and it is clear from the definition that the corresponding function problem is in TFNP, which is the class of function problems on polytime-decidable predicates for which there is guaranteed to be a witness. (Megiddo and Papadimitriou [1991] provide a thorough discussion of complexity classes associated with function problems.)

Often, decision problems are used to simplify the formulation of some function problem to a yes/no question, and hence it is desirable that the complexity of finding a solution should be preserved in the transformation from a function problem into a decision problem. If a function problem and its corresponding decision problem are related in this sense, then solving one enables one to solve the other one easily [Papadimitriou, 1994a].

We can give a general method for solving the function problem MAX-UTIL using $O(|\mathcal{PS}|)$ calls to a decision problem MAX-UTIL oracle, by combining the methods given for SAT and TSP by Papadimitriou [1994a, Examples 10.3, 10.4]:

1. *Find the value of an optimal state:* The value of any state for a goalbase in a sum language may never be less than

$$\sum_{\substack{(\varphi,w)\in G\\ w<0}} w,$$

nor more than

$$\sum_{\substack{(\varphi,w)\in G\\ w>0}} w,$$

so it follows that Ω, the value of an optimal state for G, lies in this range. Without loss of generality, multiply out the all fractions which appear as weights in G. Call the resulting (integer) range of state values $[\ell, h]$. Let $n \in \mathbb{N}$ be the least such that $2^n > h - \ell$, and then set $h = 2^n + \ell$, which ensures that $\Omega \in [\ell, h)$. Now we can use the MAX-UTIL oracle to do a binary search for Ω. Ask the oracle whether $\langle G, \ell + 2^{n-1}\rangle \in$ MAX-UTIL. If so, then let $\ell := \ell + 2^{n-1}$; if not, let $h := \ell + 2^{n-1}$. Decrement n and repeat while $n \geq 0$. On termination, $\ell = \Omega$ and $h = \Omega + 1$. Hence, we find Ω in $O(\log(h - \ell))$ steps. Now recover the original (unmultiplied) Ω by dividing by whatever factor we multiplied by to eliminate fractions.

2. *Find an optimal state:* Recall that $\varphi[\nu/\omega]$ is the formula φ with all occurrences of ω replaced by ν, and $G[\nu/\omega]$ the goalbase with the same substitution applied to all of its formulas. For each item $p \in \mathcal{PS}$: Use the MAX-UTIL oracle to decide whether $G[\top/p]$ can yield Ω utility. If so, then set $G := G[\top/p]$; otherwise, set $G := G[\bot/p]$. Read an optimal model from the substitution instance created once all items are assigned.

It is much more straightforward—though not much different from the point of view of complexity theory—to solve the decision problem MAX-UTIL by making a single call to a function problem MAX-UTIL oracle and then checking whether the optimal model so returned has a value of at least K.

An objection which might be made at this point is that the procedure we have described for solving the function problem using the decision problem works only for languages which are closed under substitution of (formulas equivalent to) constants. In particular: Suppose that $(\varphi, w) \in G \in \mathcal{L}(\Phi, W, \Sigma)$, $p \in \mathcal{PS}$, and p occurs in φ, but that $\varphi[\top/p] \notin \Phi$. In this case, we cannot query a MAX-UTIL(Φ, W, Σ) oracle about $\langle G[\top/p], K\rangle$, because $G[\top/p] \notin \mathcal{L}(\Phi, W, \Sigma)$; we have substituted ourselves into a language which our oracle does not speak. There is a similar problem if $\varphi[\bot/p] \notin \Phi$: If $\langle G[\top/p], K\rangle \notin$ MAX-UTIL(Φ, W, Σ), then by the next time we query the oracle we will have already substituted $\bot$ for p, again carrying us outside of the language.

Many of the languages we have considered *are* closed under substitution of constants. All cubes languages have this property, since $\top$ is the unique 0-pcube

(the empty conjunction), and $p \wedge \neg p$ is equivalent to $\bot$. Similarly, we can simulate the effects of $\bot$ for positive cubes by just eliminating any positive cube into which $\bot$ would be substituted. For clauses, we may write $\top$ as $p \vee \neg p$ and $\bot$ as either the unique 0-clause (the empty disjunction) or by deleting any disjuncts into which $\bot$ would be substituted. For positive clauses, we may also handle $\bot$ by deletion of disjuncts, but we also lack a positive clause which is equivalent to $\top$. There we can remove (φ, w) from G instead of substituting $\top$ into it, and reset K to $K - w$ to account for having satisfied (φ, w). In other words, for such languages, MAX-UTIL is self-reducible.

There are some languages, however, where no such tricks are available. For example, consider the following class $\mathcal{C}$ of goalbases:

$$\mathcal{C} = \Big\{ \{(\varphi_i, w_i)\}_i \;\Big|\; \bigwedge \varphi_i \text{ is satisfiable and all } w_i \geq 0 \Big\}.$$

Here, it is possible that we might leave the class $\mathcal{C}$ on our first substitution. The goalbase $G = \{(p \rightarrow q, 1), (\neg q, 1)\} \in \mathcal{C}$, but $G[\top/p]$ clearly is not since $\{\top \rightarrow q, \neg q\}$ is not a satisfiable set. While the decision problem for $\mathcal{C}$ is easily solved—just sum the weights and check whether the sum exceeds the given K—it gives no guidance as to the solution of the function problem.

Hence, for languages which are not closed under substitution of constants, we need a different decision problem. Here we propose an alternative version of MAX-UTIL, one which focuses on true atoms in an optimal model rather than the existence of models of at least a given value.

Definition 5.7.2 (The Decision Problem MAX-UTIL*)**.** Given a goalbase $G \in \mathcal{L}$ and an atom $p \in \mathcal{PS}$, is p true under the maximizing assignment (fix an arbitrary one if not unique)?

To see why it is necessary to fix one maximizing assignment in case there are several, consider the goalbase $\{(p \wedge \neg q, 1), (\neg p \wedge q, 1)\}$) and note that both p and q are true under *some* maximizing assignment, but both taken together do *not* maximize the utility.

It may seem ugly to fix an arbitrary assignment, and indeed one could, for example, require the least assignment with respect to some ordering; however by doing so the complexity of the problem may actually increase. This becomes evident with the PLP goalbase class presented in Section 5.7.3.

By executing an algorithm to decide the MAX-UTIL* problem $|\mathcal{PS}|$ times, one can construct a solution to the original function problem. Conversely, solving the function problem obviously enables one to solve the revised decision problem. Hence, the revised decision problem given in Definition 5.7.2 is related to the function problem in the proper way for all languages, not just languages closed under substitution of constants.

5.7.2 Horn Clauses, Propositional Logic Programming, and HORNSAT

In the following subsections, we frequently refer to Horn clauses, propositional logic programming, and the decision problem HORNSAT, all of which we now define.

We make use of the following notions and results from propositional logic programming (PLP), surveyed by Dantsin, Eiter, Gottlob, and Voronkov [2001].

Definition 5.7.3 (Horn Clauses and Least Models)**.** A *strict Horn clause* is a nonempty disjunction of exactly one atom and zero or more negated atoms. A *general Horn clause* is a nonempty disjunction of at most one atom and zero or more negated atoms.

For a set S of strict Horn clauses, a *least model* $\mathrm{LM}(S)$ of S is a smallest set $M \subseteq \mathcal{PS}$ such that $M \models S$, that is, $M \models \varphi$ for all $\varphi \in S$.

Fact 5.7.4. *Any set S of strict Horn clauses has a unique least model.*

Definition 5.7.5. The PLP *decision problem* is as follows: Given a set S of strict Horn clauses and some $p \in \mathcal{PS}$, is $p \in \mathrm{LM}(S)$?

Fact 5.7.6 (Dantsin et al. [2001, p. 385])**.** *The* PLP *decision problem is* P*-complete.*

Finally, we will use the well-known decision problem HORNSAT, along with its associated complexity result [Greenlaw, Hoover, and Ruzzo, 1992].

Definition 5.7.7. The HORNSAT *decision problem* is as follows: Given a set, S of general Horn clauses, is S satisfiable?

Fact 5.7.8. *The* HORNSAT *decision problem is* P*-complete.*

Logically speaking, Horn clauses express facts and dependencies in the following ways:

- Strict Horn clauses with no negated atoms, i.e., consisting only of one atom, represent plain *facts*. In the context of auctions, these are statements about single goods, in voting, single candidates: "I'll pay $50 for the Elvis statue.", "I cast a vote for Obama."

- Strict Horn clauses containing negated atoms correspond to *implications*. In our context they can be viewed as statements conditioned on several goods with one good as consequence: "If you don't eat your meat, you can't have any pudding." Additionally, strict horn clauses with negated atoms lend themselves to describing situations in which both *goods* and *bads* must be divided: "For $1, either I get the last piece of cake, or I don't have to clean the bathroom."

- Non-strict Horn clauses, i.e., disjunctions containing only negated atoms, correspond to *negated conjunctions*; we can think of them as "negative synergies", or exclusions of certain combinations of goods: "A committee with *both* Alice *and* Bob on it would be a disaster.", "I would appreciate not having *both* my defense *and* my job interview today.", "If I have to change planes in London, it's worth $50 to me to avoid doing it at Heathrow."

These ways of interpreting Horn clauses have proved their usefulness in the area of logic programming. We believe that this also makes them a versatile and powerful base for preference representation languages.

5.7.3 Finding P-Complete Goalbase Languages

Recall that, in the general case, MAX-UTIL is NP-complete; attempts to find tractable subclasses in Section 5.5 consisted in putting natural restrictions on the formulas and weights, e.g., by allowing only conjunctions of (negated) atoms and positive weights. As seen in Table 5.1, for the resulting classes MAX-UTIL either remained intractable or became quite easy (either $O(n)$ or $O(n \log n)$). This raises the question: Are there goalbase languages which are tractable, but nontrivial, for MAX-UTIL?

In order to seek out such languages, we now propose reversing our previous approach: Instead of putting restrictions on the goalbases and then examining the complexity of MAX-UTIL* for the resulting languages, we take a problem which has the desired complexity and find a class of goalbases whose MAX-UTIL* problem corresponds to it.

Intuitively, it is evident that Horn clauses are more versatile and expressive than some of the above-mentioned restrictions. For example, $(\neg a \vee b, 1)$ translates into positive cubes as $\{(\top, 1), (a, -1), (a \wedge b, 1)\}$, while $(\neg a \vee \neg b, 1)$ becomes $\{(\top, 1), (a \wedge b, -1)\}$. While these are not cumbersome on their own, it can become so when several Horn clauses are translated together, since in translation the weight of each Horn clause is distributed over multiple positive cubes. Translation into another simple language, positive clauses with positive weights, will not typically be possible, as general Horn clauses are not monotone formulas and so require a language which offers either negation as a connective or permits negative weights.

Furthermore, there are various P-completeness results involving Horn clauses, two of which we stated in the previous subsection. For these reasons, in the following we will apply our approach to find two P-complete goalbase classes related to Horn clauses.

PLP Goalbases

Definition 5.7.9. The language $\mathcal{L}_{\text{PLP}}$ of *PLP goalbases* consists of all goalbases

$$G = \{(\varphi_i, w_i)\}_i \cup \left\{ \left(p, -\frac{m}{|\mathcal{PS}| + 1} \right) \right\}_{p \in \mathcal{PS}}$$

where

- the φ_i are strict Horn clauses,
- the w_i are positive, and
- $m = \min_i\{w_i\}$.

$\text{LP}(G) = \{\varphi_i\}_i$ is the *underlying logic program* consisting of all positively weighted formulas. The remaining terms are *penalty terms*.

The penalty terms are needed for technical reasons, and we will return to them in the discussion which follows.

Fact 5.7.10. *The weights of the penalty terms sum to an absolute value less than any of the w_i. That is, for all i,*

$$w_i > \sum_{p \in \mathcal{PS}} \frac{m}{|\mathcal{PS}| + 1}.$$

Corollary 5.7.11. *The (unique) maximizing valuation of any $G \in \mathcal{L}_{\text{PLP}}$ is the least model of the underlying logic program, i.e.,* $\text{LM}(\text{LP}(G))$.

Proof. $\text{LM}(\text{LP}(G))$ obviously satisfies all formulas of G that have positive weights. Since it is a *least* model, due to Fact 5.7.10, none of its subsets get a higher value; due to the penalty terms, none of its supersets get a higher value; and due to Fact 5.7.4, it is unique. □

Lemma 5.7.12. *The* MAX-UTIL* *decision problem for PLP goalbases is in* P.

Proof. Given $G \in \mathcal{L}_{\text{PLP}}$ and $p \in \mathcal{PS}$, $\text{LP}(G)$ can be computed in linear time, and then $p \in \text{LM}(\text{LP}(G))$ is decidable in polynomial time due to Fact 5.7.6. Due to Corollary 5.7.11, this yields the answer to the MAX-UTIL* decision problem. □

Lemma 5.7.13. *PLP can be reduced in logarithmic space to the* MAX-UTIL* *decision problem for PLP goalbases.*

Proof. Given a logic program $S = \{\varphi_i\}_i$ and $p \in \mathcal{PS}$, let

$$G = \bigcup_{i=1}^{n} \{(\varphi_i, 1)\} \cup \bigcup_{p \in \mathcal{PS}} \left\{ \left(p, -\frac{1}{|\mathcal{PS}| + 1} \right) \right\}.$$

Obviously, $G \in \mathcal{L}_{\text{PLP}}$, and due to Corollary 5.7.11, solving the MAX-UTIL* decision problem instance $\langle G, p \rangle$ yields the answer to the PLP decision problem instance (S, p). □

Corollary 5.7.14. *The* MAX-UTIL* *decision problem for PLP goalbases is* P*-complete.*

Proof. Follows immediately from Lemmas 5.7.12 and 5.7.13. □

HS Goalbases

Definition 5.7.15. The language $\mathcal{L}_{\text{HS}}$ of HORNSAT *goalbases* consists of all sets G of weighted general Horn clauses with positive weights, subject to the following condition:

Let w_i denote the weights of the strict Horn clauses in G and w'_j denote the remaining weights. Then we require that

$$\sum_j w'_j < \min_i \{w_i\}.$$

That is, the sum of weights of non-strict clauses (i.e., those containing no positive atom) is less than the weight associated with any strict clause.

This condition does not appear to be very intuitive, and we will return to it in the discussion. For the time being, note that it is only needed to ensure that the complexity stays within P (Lemma 5.7.16); it may be possible to find a more intuitive condition to this effect.

Lemma 5.7.16. *The* MAX-UTIL* *decision problem for HS goalbases is in* P*.*

Proof. Given $G \in \mathcal{L}_{\text{HS}}$, use, e.g., unit propagation [Zhang and Stickel, 1996] to find a satisfying assignment if one exists. If it does exist, this is the maximizing assignment since all weights are positive. If it does not exist, let $G' \subset G$ be the subset containing all *strict* Horn clauses. Due to the condition in Definition 5.7.15, $\text{LM}(G')$ is a maximizing assignment for G, since

- it satisfies all strict Horn clauses, and
- among all such assignments which satisfy all strict Horn clauses, it satisfies the most non-strict Horn clauses.

The second item holds due to the fact that we have a *least* model of G', that is, one that satisfies the *greatest* set of negated atoms, and non-strict Horn clauses are just disjunctions of those. □

Lemma 5.7.17. HORNSAT *can be reduced in logarithmic space to* MAX-UTIL* *for HS goalbases.*

Proof. Given a set $S = \{\varphi_1, \ldots, \varphi_n, \varphi'_1, \ldots, \varphi'_m\}$ of strict (φ_i) and non-strict (φ'_i) Horn clauses, build the HS goalbase

$$G = \bigcup_{i=1}^{n} \{(\varphi_i, 1)\} \cup \bigcup_{i=1}^{m} \left\{ \left(\varphi'_i, \frac{1}{m+1} \right) \right\},$$

obtain the maximizing assignment by solving MAX-UTIL* for G and each $p \in \mathcal{PS}$, and check whether it satisfies all formulas in G. Since the assignment is maximizing and all weights are positive, it will do so iff G is satisfiable. □

Corollary 5.7.18. *The* MAX-UTIL* *decision problem for HS goalbases is* P*-complete.*

Proof. Follows from Lemmas 5.7.16 and 5.7.17. □

5.7.4 Discussion

As mentioned earlier, we believe that Horn clauses form a versatile and powerful base for preference representation languages, since their form is restricted in a clear way, but they retain the ability to express natural forms of dependency. The existence of various P-completeness results involving Horn clauses suggests that they lend themselves to our approach. We therefore focused on these, without meaning to suggest that other classes of formulas might not be worth considering. There are certainly other P-complete fragments of the full weighted formula language which are induced by other P-complete problems and embody different kinds of synergies from those examined here. We leave these for future investigation.

While some of our examples focused on auctions, an issue to which we will return in Chapter 6, Horn clauses also have useful interpretations in multi-winner voting. They can express dependencies among candidates, e.g., to say that Alice should be on a committee whenever Bob is, or that Alice should *not* be on a committee if Bob is. We return to this issue in detail in Chapter 7.

The goalbase classes we presented may at first glance seem artificial and unnatural, and they may then simply be viewed as proof of concept for our approach, and proof of existence for logic-based preference representation languages of intermediate complexity.

However, the penalty terms which occur in PLP do reflect an intuitively justifiable desideratum, since they make, ceteris paribus, assigning fewer items favorable. For example, in an auction, if no one benefits from obtaining some additional item, why should the auctioneer give that item away for nothing instead of keeping it for some later auction where someone might benefit from having it? In that sense, it might even be desirable to require a *least* maximizing assignment in the definition of the MAX-UTIL* problem itself. With such an alternative definition, one could remove the penalty terms from PLP goalbases and obtain a quite natural

P-complete goalbase class. This also shows that, as noted under Definition 5.7.2, requiring the *least* (instead of an arbitrary) maximizing assignment has an effect on the complexity of MAX-UTIL*: With such a requirement, it would be P-complete for PLP goalbases without penalty terms, while as it stands, it is trivially solved by making all atoms true.

As for HS goalbases, as mentioned above, the unintuitive condition in Definition 5.7.15 is only used to prove Lemma 5.7.16, and it may be possible to find a more intuitive condition to that effect. However, this condition might even be acceptable if bids or preferences can be described "lexicographically" on two levels: Strict Horn clauses (facts and implications) describe the primary bid in form of a logic program. Then, non-strict Horn clauses (exclusions of certain combinations) can be added for fine-tuning and favoring certain models of the logic program over others. Note that this secondary bid matters, since HS goalbases, unlike PLP goalbases, do not enforce *least* models.

5.8 Conclusion

In this chapter we have proposed, motivated, and characterized the complexity of three decision problems, MAX-UTIL, MIN-UTIL, and MAX-CUF, over goalbase languages. MAX-UTIL, the problem of deciding whether a certain level of utility is attainable for an individual agent, tends to be NP-complete for sum languages which permit formulas of at least length 2 and either negation or negative weights, but polynomial otherwise. In contrast, MAX-UTIL is easy for all languages using the max aggregator. MIN-UTIL, the pessimal version of MAX-UTIL, is coNP-complete for all sum languages where MAX-UTIL is NP-complete, but surprisingly is also coNP-complete for the full max language. MAX-CUF, the problem of deciding whether a certain level of collective utility is attainable, is NP-complete even for some languages where MAX-UTIL is easy.

We proposed an alternative version of MAX-UTIL the decision problem, called MAX-UTIL*, which does not require languages to be closed under substitution of logical constants in order to use it for solving MAX-UTIL the function problem. We proceeded to find two languages for which for which MAX-UTIL* is P-complete.

We might have considered other problems. At the individual level, the problem of COMPARISON is a useful problem to solve—given states A and B, does an agent prefer A to B?—but there is nothing interesting to say about it in our setting: COMPARISON will be polynomial whenever the individual aggregator used is polynomial, and both aggregators we consider, sum and max, are polynomially computable. In other words, COMPARISON would become theoretically interesting only by becoming practically useless; hence we do not consider it here. One could imagine other variants on MIN-UTIL and MAX-UTIL, such as MEAN-UTIL and MEDIAN-UTIL. We note that MEAN-UTIL and MEDIAN-UTIL would appear to be harder than MIN-UTIL and MAX-UTIL since the mean and median depend on

the whole set of states; however, we do not examine these problems at present because they seem less compelling than MIN-UTIL and MAX-UTIL.

If we interpret goalbases as representing coalitional games with transferable utility as do Ieong and Shoham [2005] and Elkind et al. [2009], rather than as representing the utility functions of individuals, then there are various complexity problems from coalitional game theory which can be explored. The decision problems CORE-MEMBERSHIP and CORE-NON-EMPTINESS, which ask questions about the core of the coalitional game being represented, and computing the Shapley value and Banzhaf index from a given representation are problems are of particular interest; it would be worth investigating whether the difficulty of these problems depends on the goalbase language used, as we have done here for MAX-UTIL, MIN-UTIL, and MAX-CUF.

Finally, we point out two directions which we do not pursue here, namely approximation and truthfulness. For each NP-completeness result we give in this chapter, there are corresponding approximation results to be found. For example, Lipton, Markakis, Mossel, and Saberi [2004] do this for envy-freeness. It would be interesting to see how accurately MAX-UTIL and MAX-CUF can be approximated in the cases where they are NP-complete. When we consider MAX-CUF, we assume that the goalbases we are given are truthful, in the sense that they are not a willful misrepresentation of an agent's preferences. There are a whole host of issues around the issue of truthfulness, such as strategyproofness and incentive compatibility. As should be obvious from the discussion here, there were many directions to pursue; many which we did not pursue would make for interesting future work.

Part II

Applications

Chapter 6

Combinatorial Auctions

6.1 Introduction

In this chapter we move from the purely theoretical considerations of previous chapters to one of the intended applications of goalbase languages: as bidding languages for combinatorial auctions. In Section 6.2, we give some background on types of auctions and an extended example which shows the advantages of combinatorial auctions over standard, sequential auctions. In Section 6.3, we present the OR/XOR/OR* family of bidding languages, the most commonly-seen bidding languages for combinatorial auctions, and compare them with our goalbase languages. (In particular, Section 6.3.3 contains relative succinctness results for some OR/XOR/OR* and goalbase languages.) In Section 6.4, we take up the Winner Determination Problem (WDP) for combinatorial auctions, and present two methods for solving it—integer programming (IP) and branch-and-bound—when using goalbases as bids. Section 6.5 gives examples of branch-and-bound heuristics tailored for use with specific goalbase languages. The final two sections, Section 6.6 and 6.7, present the setup for and results of experiments we conducted to test the feasibility of solving the WDP for goalbase languages using our IP formulation and branch-and-bound solver.

6.2 Auctions

In the most general sense, an auction is a mechanism for allocating items and costs among bidders. A wide variety of auctions have been studied by economists and, increasingly, by computer scientists, which has resulted in the naming of many features of auctions: A *sealed-bid* auction is one in which bidders submit their bids privately to the auctioneer; an *open-bid* auction is one in which bidders publicly announce their bids. An open-bid auction is *ascending-price* if the price for an item increases until no bidder is willing to pay more, and is *descending-price*

if the price of an item decreases until some bidder announces he is willing to pay it. An auction is *first-price* if the price of an item is the highest bid, and *second-price* if the second-highest bid.

The traditional types of auctions are English (ascending-price open-bid), Dutch (descending-price open-bid), and first-price sealed-bid. English auctions are commonly used for the sale of antiques, art, wine, livestock, and land. Those having lived in an agricultural area in the United States might recognize the English auction as a *farm sale*. Dutch auctions have historically been used for the sale of perishable products, particularly in the sale of large lots of cut flowers in Holland. First-price sealed-bid auctions are used by some governments for selling treasury bonds, and are also the most common method of selling houses in Scotland. The first-price sealed-bid auction is strategically equivalent to (i.e., bidding strategy and results are the same as) the Dutch auction. A fourth type, the Vickrey auction, a second-price sealed bid auction, is a 20th-century invention and is strategically equivalent to the English auction [Vickrey, 1961]. McAfee and McMillan [1987] and Milgrom [2004] provide wide-ranging overviews of auction theory for those wishing to learn more.

All four of the traditional types of auction are sequential, meaning that for each item being sold, the winner of the nth is determined before any bids are received for the $(n+1)$th. Selling items sequentially may be problematic when bidders do not value items independently—that is, when bidders do not have modular utility functions. The strong synergy between the shoes in a pair is a clear example of this. Suppose that shoes are being sold in a sequential auction individually, rather than bundled as pairs. From the perspective of the typical bidder—someone with two feet but not unlimited wealth—an auction for single shoes poses a difficult strategic problem. A typical bidder places much value on a matched pair, but presumably would prefer to pay nothing and win no shoes instead of paying to win a single shoe. If the bidder bids too high on the first shoe, he may win it, but then risks having too little money left to win the second shoe. If the bidder bids too low on the first shoe, he might not win it, in which case there is little point in bidding on the now-worthless second shoe. Sequential auctions force bidders with nonmodular preferences to bid more conservatively than they would like, or undertake more risk than they would like.

Sequential auctions for synergistic items are not just problematic for the bidders, but may also be suboptimal for the auctioneer.[1] If bidders bid conservatively as a hedge against failing to win certain combinations of items, then the auctioneer will collect less revenue from the auction than if the bidders felt comfortable bidding more aggressively. Suppose, for example, that we are conducting an English auction for a left (ℓ) and a right (r) shoe, and we have two bidders with the

[1]The auctioneer is treated as the owner of the items in the auctions literature, which we follow. Readers familiar with real-world auctions will note that this differs from the usage there, where the auctioneer is a third party (e.g., Sotheby's, eBay, or a fast-talking man with a voice that carries well) who collects a fee for organizing and conducting the auction.

following utility functions:

$$u_1(X) = \begin{cases} \$40 & \text{if } X = \{\ell, r\} \\ \$0 & \text{otherwise,} \end{cases} \qquad u_2(X) = \begin{cases} \$20 & \text{if } X = \{\ell, r\} \\ \$10 & \text{if } X = \{\ell\} \text{ or } X = \{r\} \\ \$0 & \text{otherwise.} \end{cases}$$

The overall utility for a bidder is the utility of the bundle he wins less the price he paid to win it. If we auction the left shoe first, then by bidding anything at all the first bidder risks having negative utility, in the case where he wins the left shoe but fails to win the right shoe. If the first bidder knew the second bidder's utility function then it would be clear to him that could ensure that he would win the pair by bidding \$11 for each shoe and take away a handy surplus of \$18—but since he does not know the other bidder's utility function, he cannot be certain how to bid. For all he knows, the second bidder might value the right shoe higher than he values the whole pair—this could happen if the second bidder were a left-leg amputee, for example. If the first bidder is unwilling to assume any risk, then he will simply not bid. Since the second bidder has a modular utility function, he is able to avoid the question of risk entirely and is free to bid up to his valuation for each shoe. As a result, the second bidder will pay \$2 (assuming that only whole-number bids are possible) and walk away with both shoes. Neither the auctioneer nor the first bidder should be happy with this outcome, as the auctioneer's aim is to maximize revenue and the first bidder would have been willing to pay significantly more than \$2 for the shoes as a pair.

The obvious solution to this particular problem is the one already adopted by shoe sellers worldwide, namely to sell shoes in pairs only. However, this merely disadvantages a different group of buyers—amputees, people who have a cast on one leg, people who have lost one shoe along the roadside—and so is not a general solution.[2] Furthermore, this example has as a peculiar feature that we know beforehand something about the structure of typical bidders' preferences: most bidders will want matched pairs. Leaving shoes aside, there are other domains where the synergies between goods vary greatly from bidder to bidder and cannot be predicted beforehand by the auctioneer. In such domains, imposing constraints on how bidders may express their preferences may lead to unexpected revenue loss by making bidding risky for some bidders.

This is the motivation behind *combinatorial* auctions, which sell items simultaneously rather than sequentially by permitting bids on bundles rather than just single items. Returning to our example of the shoe auction, it is easy to see that revenue for the auctioneer would improve if the first bidder were able to place a bid of more than \$20 for the left and right shoe together, but no bids (or, equivalently bids of \$0) for each shoe alone. To do this, we must have a *bidding*

[2]Incidentally, this problem has been addressed in the United States since 1943 by the National Odd Shoe Exchange, which serves as an intermediary for people who have purchased pairs of shoes but need only one of them [National Odd Shoe Exchange, 2009].

language in which to specify bids for bundles, which we discuss in Section 6.3. We anticipate the application of our goalbase languages to the problem (see Section 6.3.2) by writing the first bidder's bid as $\{(\ell \wedge r, \$40)\}$ and the second bidder's as $\{(\ell, \$10), (r, \$10)\}$. The problem now faced by the auctioneer is to select which bids to accept. This is trivial in sequential auctions—in first-price auctions it amounts to recognizing the largest bid, and for second-price auctions additionally spotting the second-largest bid—but finding a revenue-maximizing set of bids in a combinatorial auction can be quite difficult, an issue we take up in Section 6.4.

In anticipation of needing them later, we define a few more terms: A bidder is said to have *free disposal* if he is able to accept more items without losing utility—the intuition being that a bidder who does not want the additional items may simply discard them. In other words, a bidder with free disposal has a monotone utility function. Free disposal may also apply to the auctioneer, however. The auctioneer has free disposal if he may refrain from allocating some items (possibly holding them back for the next auction, or discarding them). It will usually be the case when allocating goods that bidders have free disposal; in the event that we are allocating bads (e.g., tasks which the bidders wish to pay to avoid, or toxic waste which the bidders must store) we must assume that bidders lack free disposal if we want to ensure a sensible outcome. Similarly, in not all situations does it make sense to assume that the auctioneer has free disposal. When auctioning perishable goods, for example, it does not make sense for the auctioneer to have free disposal, as there may be a real cost to disposing of spoiled items.

There has recently been a great deal of work on combinatorial auctions in economics, operations research, and computer science. Cramton et al. [2006] provide a thorough overview of the field; of particular relevance for us are the chapters on bidding languages [Nisan, 2006], and the complexity of the Winner Determination Problem [Lehmann et al., 2006b; Müller, 2006] and heuristics for solving it [Sandholm, 2006]. We discuss other related work throughout the chapter at the point where it becomes relevant.

6.3 Bidding Languages

The XOR/OR/OR* family of languages contains the typical bidding languages found in the literature on combinatorial auctions. Each language in this family has at least one undesirable characteristic—the XOR language is extremely verbose, determining the value of a bundle given an OR (or OR*) bid is NP-complete—characteristics which a combinatorial auction designer might wish to avoid. In this section, we compare these established bidding languages with our goalbase languages, each exhibiting different assortments of characteristics desirable and undesirable, with the aim of affording the combinatorial auction designer a wider array of poisons from which to pick.

6.3.1 The XOR, OR, and OR* Languages

Here we define the XOR, OR, and OR* languages, which are the most commonly used bidding languages in the combinatorial auctions literature [Fujishima et al., 1999; Sandholm, 2002; Nisan, 2006].

Definition 6.3.1 (XOR, OR, and OR*)**.** Let $\mathcal{PS}$ be a fixed set of items. An *atomic bid* is an ordered pair $\langle X, w\rangle$, where $X \subseteq \mathcal{PS}$ (the bundle) and $w \in \mathbb{R}^+$ (the price). A bid in the XOR language is a finite list of atomic bids

$$\langle X_1, w_1\rangle \text{ XOR} \ldots \text{XOR } \langle X_n, w_n\rangle,$$

which generates the utility function

$$u(X) = \max_{X_i \subseteq X} w_i.$$

A bid in the OR language consists of a finite list of atomic bids:

$$\langle X_1, w_1\rangle \text{ OR} \ldots \text{OR } \langle X_n, w_n\rangle.$$

A set of atomic bids $\mathcal{F}$ is *feasible* if the atomic bids in $\mathcal{F}$ contain pairwise disjoint bundles. A bid $\mathcal{B}$ in the OR language generates the utility function

$$u(X) = \max_{\substack{\mathcal{F} \subseteq \mathcal{B} \\ \mathcal{F} \text{ feasible}}} \sum_{\langle X_i, w_i\rangle \in \mathcal{F}} w_i.$$

A bid in the OR* language is an OR bid where the bundle in each atomic bid is permitted to contain zero or more dummy items $d_i \notin \mathcal{PS}$.

The purpose of dummy items is to enforce overlap between bundles which would not normally overlap. E.g., $\langle\{a\}, 1\rangle$ OR $\langle\{b\}, 2\rangle$ gives value 3 for the bundle $\{a, b\}$. If the bidder wants instead to express that he is willing to buy at most one of a and b, then the bundles which should be mutually exclusive may be tagged with a dummy item d, like so: $\langle\{a, d\}, 1\rangle$ OR $\langle\{b, d\}, 2\rangle$.

6.3.2 Goalbase Bidding Languages

Goalbases languages may be used as bidding languages for combinatorial auctions. Each item in an auction is associated with exactly one propositional variable in $\mathcal{PS}$. (For example, Lead Belly's guitar might be assigned a and the ship's bell from the *RMS Titanic* might get b.) We consider a propositional variable true for a bidder when that bidder is given the associated item. Bidders express their valuations for items by stating propositional formulas containing the items' propositional variables, along with what they would be willing to pay were those formulas made true. (A bid of $(a, \$500{,}000)$ means that I am willing to pay \$500,000 for Lead Belly's guitar, and a bid of $(a \wedge b, \$1{,}000{,}000)$ means that I am willing to pay a

million dollars for the guitar and the ship's bell together. Were I to place both bids, that would mean that I am willing to pay $500,000 for the guitar, and an *additional* one million for the guitar and bell together, i.e., a total of $1.5 million for both.)

When used as a bidding language, $\mathcal{L}(pcubes, \mathbb{R}^+, \max)$ is the same as the XOR language. This is easily seen by noting that any atomic bid $\langle X, w\rangle$ is satisfied in the same states as the weighted formula $(\bigwedge X, w)$, and that the max aggregator forces an agent's utility to equal that of the highest weight of any satisfied formula, the same as the XOR operator does.

6.3.3 Succinctness

When bids in one language are significantly more verbose than the same bids in another language, that is a reason for preferring the more compact language over the less compact one. The definition of succinctness we used in Chapter 4 to compare goalbase languages is not sufficiently general to permit comparison between goalbase languages and the XOR/OR/OR* family of languages. However, it is not difficult to extend it further so that we can compare any two languages in which utility functions over sets can be represented—so long as we are able to measure the size of representations in bits. This ensures that we can make meaningful comparisons between such dissimilar languages as those in the XOR/OR/OR* family and the goalbase languages we consider here.

A *set-based utility function representation language* $\mathcal{L}$ is a class of strings such that each string corresponds to a single utility function over sets. We say of a string $r \in \mathcal{L}$ that it *represents* a utility function u, and that u is expressible in $\mathcal{L}$ because $r \in \mathcal{L}$. The utility function u_r is the utility function generated by r. The size of a representation, size(r), is the number of bits used by r.

Definition 6.3.2 (Succinctness, Extended)**.** Let $\mathcal{L}$ and $\mathcal{L}'$ be utility function representation languages, and $\mathcal{U}$ a class of utility functions such that every member of $\mathcal{U}$ is representable in both languages. Then $\mathcal{L} \preceq_{\mathcal{U}} \mathcal{L}'$ ($\mathcal{L}'$ is at least as succinct as $\mathcal{L}$ with respect to $\mathcal{U}$) iff there exists a function $f\colon \mathcal{L} \to \mathcal{L}'$ and a polynomial p such that for all representations $r \in \mathcal{L}$, if $u_r \in \mathcal{U}$ then $u_r = u_{f(r)}$ and $\text{size}(f(r)) \leq p(\text{size}(r))$.

While succinctness is frequently beneficial, it is worth noting that succinctness is not always a blessing. Fix some class of (rational-valued) utility functions $\mathcal{U}$ and enumerate them. (That is, the representations of $u_0, u_1, \ldots$ are just the numbers $0, 1, 2, \ldots \in \mathbb{N}$.) The language formed by this enumeration is *maximally* succinct for the class $\mathcal{U}$. This language uses exactly as many bits as are needed to name each utility function, so there is no other representation language with the same expressivity which can beat it in terms of succinctness. No bits go to waste here, but a direct consequence of this miserliness is that recovering a utility function

from its representation may involve difficult computations (or a very large lookup table).

In terms of relative succinctness, the XOR language is inferior to both the OR and the OR* languages. This follows from results given by Nisan [2006].

Theorem 6.3.3. XOR $\prec$ OR.

Proof. Nisan [2006, Proposition 9.3] shows XOR $\not\succeq$ OR by observing that the function $u_n(X) = |X|$ has a succinct representation in OR, but requires an exponential number of atomic bids in XOR. It remains to be shown that XOR $\preceq$ OR; that is, we need to show that any XOR representation of a utility function representable in both languages can be translated into an equivalent OR representation, without a superpolynomial blowup in size. Nisan [2006, Proposition 9.1] shows that the OR language corresponds to the class of superadditive utility functions. (Recall from Definition 3.2.1 that a utility function u is superadditive if $u(X \cup Y) \geq u(X) + u(Y)$ for all disjoint X, Y.) Therefore, it suffices to point out that whenever $\langle S_1, p_1\rangle$ XOR ... XOR $\langle S_n, p_n\rangle$ represents a superadditive function, then $\langle S_1, p_1\rangle$ OR ... OR $\langle S_n, p_n\rangle$ represents that very same function. □

Theorem 6.3.4. XOR $\prec$ OR*.

Proof. Nisan [2006, Theorem 9.3] shows that any bid using a combination of OR- and XOR-operators (and thereby certainly any pure XOR-bid) can be translated into an OR*-bid with the same number of atomic bids, introducing at most a quadratic number of dummy items. Hence, XOR $\preceq$ OR*. The same function $u_n(X) = |X|$ as above demonstrates that the succinctness relation is strict. □

Note that, because the OR-language is not fully expressive, XOR $\prec$ OR does not entail that XOR $\prec$ OR* (even though OR* subsumes OR), so the preceding two theorems are independently interesting.

Now we present three results comparing the succinctness of the XOR, OR, and OR* languages with some of our goalbase languages.

Theorem 6.3.5. OR $\perp \mathcal{L}(pcubes, \mathbb{R}, \Sigma)$.

Proof. ($\not\succeq$) Consider the family of utility functions $u_n(X) = \binom{|X|}{2}$. The goalbase $\{(a \wedge b, 1) \mid a, b \in \mathcal{PS} \text{ and } a \neq b\}$ represents u_n in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$, and is quadratic in $|\mathcal{PS}|$. The sole representation of u_n in the OR language,

$$\bigOR_{\substack{X \subseteq \mathcal{PS} \\ |X| \geq 2}} \left\langle X, \binom{|X|}{2} \right\rangle,$$

has size exponential in $|\mathcal{PS}|$. We prove by induction that this is the only correct representation: No atomic bids for $\emptyset$ or any singleton occur in an OR bid corresponding to u_n, since $u_n(\emptyset) = u_n(\{a\}) = 0$, for all $a \in \mathcal{PS}$. For every distinct

$a, b \in \mathcal{PS}$, the atomic bid $\langle\{a, b\}, 1\rangle$ occurs in the OR bid, since there are no bids for smaller bundles, and no other bid can be accepted in state $\{a, b\}$. Suppose that every bundle X where $|X| = k \geq 2$ has an atomic bid corresponding to it. Fix some $p \notin X$. Let $x_1, \ldots, x_n$ be the sizes of some combination of $n \geq 2$ bundles which partition $X \cup \{p\}$. Then $\sum_{i=1}^{n} x_i = |X \cup \{p\}|$. The total value of these smaller bundles according to their associated atomic bids is

$$\sum_{i=1}^{n} \binom{x_i}{2} = \sum_{i=1}^{n} \frac{x_i(x_i - 1)}{2} = \frac{\sum_{i=1}^{n} {x_i}^2 - \sum_{i=1}^{n} x_i}{2},$$

while the value of $X \cup \{p\}$ according to u_n is

$$\binom{|X \cup \{p\}|}{2} = \binom{\sum_{i=1}^{n} x_i}{2} = \frac{\left(\sum_{i=1}^{n} x_i\right)^2 - \sum_{i=1}^{n} x_i}{2}.$$

Observe that $\sum_{i=1}^{n} {x_i}^2 < \left(\sum_{i=1}^{n} x_i\right)^2$, and hence the value of $X \cup \{p\}$ is always greater than the sum of the values of the bids for any partition thereof. Furthermore, no atomic bids for other bundles of size $\geq k + 1$ can be accepted in this state, so we must include $\left\langle X \cup \{p\}, \binom{|X \cup \{p\}|}{2} \right\rangle$ as an atomic bid. Therefore, every bundle of size $k + 1$ also has an atomic bid in the OR representation.

($\not\in$) The family of utility functions $u_n(X) = \left\lfloor \frac{|X|}{2} \right\rfloor$ is quadratically representable in the OR language as

$$\bigOR_{\substack{a,b \in \mathcal{PS} \\ a \neq b}} \langle\{a, b\}, 1\rangle$$

but in $\mathcal{L}(pcubes, \mathbb{R}, \Sigma)$ the sole representation is

$$\left\{ \left(\bigwedge X, (-2)^{|X|-2} \right) \;\middle|\; X \subseteq \mathcal{PS} \text{ and } |X| \geq 2 \right\},$$

which is exponential in $|\mathcal{PS}|$.

That this is the representation in $\mathcal{L}(pcubes, \mathbb{R}, \Sigma)$ is shown inductively. $w_\emptyset = w_a = 0$, since $u(\emptyset) = u(\{a\}) = 0$, and similarly $w_{a \wedge b} = 1 = (-2)^{|\{a,b\}|-2}$. For the inductive case, suppose that $w_{\bigwedge X} = (-2)^{|X|-2}$. We have that

$$w_{\bigwedge X \cup \{a\}} = \sum_{Y \subseteq X \cup \{a\}} (-1)^{|(X \cup \{a\}) \setminus Y|} \left\lfloor \frac{|Y|}{2} \right\rfloor$$

by the Möbius inversion (see p. 36 and [Rota, 1964]), and continuing with that equality, we have

$$\begin{aligned}
&= \sum_{Y \subseteq X} (-1)^{|X \setminus Y|+1} \left\lfloor \frac{|Y|}{2} \right\rfloor + \sum_{Y \subseteq X} (-1)^{|(X \cup \{a\}) \setminus (Y \cup \{a\})|} \left\lfloor \frac{|Y \cup \{a\}|}{2} \right\rfloor \\
&= \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} \left(- \left\lfloor \frac{|Y|}{2} \right\rfloor + \left\lfloor \frac{|Y|+1}{2} \right\rfloor \right) \\
&= \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} (|Y| \bmod 2) = \sum_{\substack{Y \subseteq X \\ |Y| \text{ odd}}} (-1)^{|X \setminus Y|} \\
&= (-1)^{|X|+1} \sum_{\substack{Y \subseteq X \\ |Y| \text{ odd}}} 1 = (-2)^{|X|-1} = (-2)^{|X \cup \{a\}|-2},
\end{aligned}$$

which completes the induction. □

Next, we turn to the OR* language:

Theorem 6.3.6. OR* $\perp \mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$.

Proof. ($\not\geq$) By the same argument found in the $\not\geq$ direction of Theorem 6.3.5. Adding dummy variables to the OR language cannot make the OR representation smaller—the OR representation given already has too much overlap between atomic bids, and dummy variables can only increase these conflicts.

($\not\leq$) The family of utility functions

$$u_n^{\exists}(X) = \begin{cases} 1 & \text{if } X \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

is representable linearly in the OR* language (with dummy item d) as

$$\bigOR_{a \in \mathcal{PS}} \langle \{a, d\}, 1 \rangle$$

but the unique representation in $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$,

$$\left\{ \left(\bigwedge X, (-1)^{|X|-1} \right) \;\middle|\; \emptyset \subset X \subseteq \mathcal{PS} \right\},$$

is exponential in $|\mathcal{PS}|$, as shown in the proof of Theorem 4.4.8. □

In the proofs of Theorems 6.3.5 and 6.3.6, we make use of the fact that $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$ has unique representations, meaning that any utility function representable in the language is representable in exactly one way (see Theorem 3.4.2 for a proof of this fact).

Theorem 6.3.7. XOR $\prec \mathcal{L}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$.

Proof. ($\preceq$) Let $\langle X_1, w_1\rangle$ XOR ... XOR $\langle X_n, w_n\rangle$ be an XOR bid such that $w_1 \leq \ldots \leq w_n$, and for convenience let $w_0 = 0$. Then the goalbase

$$\left\{ \left(\bigvee_{j=i}^{n} \bigwedge X_j, w_i - w_{i-1} \right) \;\middle|\; 1 \leq i \leq n \right\}$$

in $\mathcal{L}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$ is equivalent to, and quadratic in the size of, the given XOR bid. Therefore, XOR $\preceq \mathcal{L}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$.

($\not\succeq$) The family of utility functions $u_n(X) = |X|$ is representable linearly in $\mathcal{L}(\mathit{atoms}, \mathbb{R}^+, \Sigma)$, a sublanguage of $\mathcal{L}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$, as $\{(a, 1) \mid a \in \mathcal{PS}\}$, while the sole XOR representation is

$$\bigXOR_{X \subseteq \mathcal{PS}} \langle X, |X|\rangle,$$

which is exponential in $|\mathcal{PS}|$. □

Theorems 6.3.5 and 6.3.6 show that some bids which are very large in OR or OR* will be small in $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$, and vice versa, and Theorem 6.3.7 shows the advantage of $\mathcal{L}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$ over XOR. Hoos and Boutilier [2000], Boutilier and Hoos [2001], and Boutilier [2002] have advocated for logic-based bidding languages before, but this is the first formal argument we have seen regarding their succinctness with respect to the XOR/OR/OR* family of languages. These three results demonstrate that logic-based languages can provide more efficient representations of some bids than the XOR/OR/OR* family of languages, and thus merit consideration when designing combinatorial auctions.

Finally, we offer some general comments on translation of OR bids: Consider the bundle evaluation problem, in which we are asked to decide if a given bundle is worth at least a given amount of utility.

Definition 6.3.8 (The Decision Problem $\mathcal{L}$-EVAL)**.** The decision problem $\mathcal{L}$-EVAL is defined as: Given a utility function representation $R \in \mathcal{L}$, a model M, and an integer K, is the value of M at least K?

It is clear that so long as we have chosen a polynomially-computable aggregator, then for any goalbase language $\mathcal{L}$, the bundle evaluation problem $\mathcal{L}$-EVAL is in P. This is because bundle evaluation for a goalbase language amounts to model checking for propositional formulas, plus the application of the aggregator. The former is always polynomial, so having a polynomially-computable aggregator ensures that the whole procedure will be polynomial. This is not the case for the OR language, however: OR-EVAL is essentially just WEIGHTED SET PACKING, which is a well-known NP-complete problem [Garey and Johnson, 1979].

Now, suppose that a goalbase language $\mathcal{L}$ is such that $\mathcal{U}(\text{OR}) \subseteq \mathcal{U}(\mathcal{L})$. That is, every utility function expressible in the OR language is expressible in $\mathcal{L}$. Since $\mathcal{U}(\text{OR}) \subseteq \mathcal{U}(\mathcal{L})$, we know that for each bid in the OR language, there is an equivalent goalbase in $\mathcal{L}$. Furthermore, so long as we have chosen a polynomially-computable aggregator, the bundle evaluation problem for $\mathcal{L}$, $\mathcal{L}$-EVAL, is in P. Therefore, if there were an algorithm capable of finding the $\mathcal{L}$-translations of arbitrary OR bids in polynomial time, we could reduce OR-EVAL to $\mathcal{L}$-EVAL. Since OR-EVAL is NP-complete and $\mathcal{L}$-EVAL is in P, it would follow immediately that $\mathsf{P} = \mathsf{NP}$. Put another way: It is *very unlikely* that there is any fast way to translate arbitrary OR bids into a goalbase language.[3] Nonetheless, we do not view this, or any of the results in this section, as negative results for goalbase languages. These results merely show that the XOR/OR/OR* family and goalbase languages have different representational "sweet spots".

6.4 Winner Determination

Intuitively, the Winner Determination Problem (WDP) is the problem of finding an optimal allocation of goods to bidders, given a set of bids. In this section, we discuss algorithms for solving the WDP. In order to give a precise definition of the WDP, we require some notation first.

6.4.1 Notation

Auctions are fundamentally about allocating items to bidders. If we consider a process in which items are allocated one at a time, then at any step an item could be allocated to a bidder or to no one.

Definition 6.4.1 (Allocations)**.**

- $\mathcal{A}$ is the set of agents bidding in any given auction. Each agent $i \in \mathcal{A}$ has a goalbase G_i defining his valuation over the goods in $\mathcal{PS}$.
- An *allocation* $A\colon \mathcal{PS} \to \mathcal{A} \cup \{*\}$ is a function which maps goods to the agents to which they are given. The symbol $*$ indicates no agent. We write $A(p) = *$ when A leaves good p unallocated, and in that case A is a (strictly) *partial* allocation. If A allocates all goods in $\mathcal{PS}$, then A is a *complete* allocation.
- The set $\text{und}(A) = \{p \in \mathcal{PS} \mid A(p) = *\}$ is the set of unallocated goods in allocation A; the set $\text{und}(A, \varphi)$ is the set of unallocated goods appearing as propositional variables in the formula φ.

[3]Note however that this *does not* imply that a polysize translation is impossible. For further discussion of this issue, see Chapter 4, p. 66.

An allocation induces a model (see Definition 2.2.2) for each agent, where the true proposition letters in the model are those corresponding to the goods allocated to the agent. We now give a precise definition and some notation for this:

Definition 6.4.2 (Allocation-Induced Models)**.**

- The model $M_i^A = \{p \in \mathcal{PS} \mid A(p) = i\}$ is the set of goods assigned to bidder i by allocation A.
- $M_i^A \models \varphi$ iff the goal φ is satisfied by M_i^A.
- $M_i^A \,?\, \varphi$ iff there is an allocation $A' \supseteq A$ such that $M_i^A \not\models \varphi$ and $M_i^{A'} \models \varphi$, or $M_i^A \models \varphi$ and $M_i^{A'} \not\models \varphi$.

In previous chapters when we dealt with models, we considered them in a static context, so had no need to represent as-of-yet unallocated items. Because some of the algorithms we discuss later construct partial allocations, we need to to distinguish between formulas which an allocation makes false for an agent and formulas which are merely left undetermined at that stage. While false formulas cannot be made true by allocating more items, undetermined formulas can. For example, suppose that agent 1 bids $\{(a \wedge b, 3), (c, 2), (\neg c \vee d, 2)\}$ and we have as a partial allocation $A = \{(a, 1), (b, 2), (c, 1), (d, *)\}$. Then $M_1^A = \{a, c\}$, and we have that $M_1^A \models c$ because agent 1 was allocated c, $M_1^A \models \neg(a \wedge b)$ because b was allocated to some other agent, and $M_1^A \,?\, \neg c \vee d$ because $\neg c \vee d$ could become true if d, which is unallocated in A, were later allocated to agent 1.

Note that the second clause of the definition of $M_i^A \,?\, \varphi$, in which φ goes from true in M_i^A to false in some extended allocation, never obtains for positive formulas. This clause is necessary so that undecided formulas containing negation are handled properly.

6.4.2 The Winner Determination Problem

As the WDP is a maximization problem, we must give an objective to maximize, which in this case is utilitarian social welfare. The *utilitarian social welfare* of an allocation A of a set of goods $\mathcal{PS}$ to agents with goalbases $\{G_i\}_{i \in \mathcal{A}}$ in $\mathcal{L}(\mathit{forms}, \mathbb{R}, \Sigma)$ is as follows:

$$\mathrm{sw}(A) = \sum_{i \in \mathcal{A}} \sum_{\substack{(\varphi, w) \in G_i \\ M_i^A \models \varphi}} w.$$

That is, the utilitarian social welfare of an allocation is the sum of the weights associated with all satisfied goals across all agents. Hence, the WDP (as a function problem) is to find a complete allocation A maximizing $\mathrm{sw}(A)$. By restricting

attention to complete allocations we are defining a WDP *without free disposal*. If desired, we can easily model auctions with free disposal by adding to any given auction instance a single bidder with an empty goalbase.

The decision-problem version of the WDP for almost all goalbase languages is NP-complete: MAX-CUF(Φ, W, F, Σ) from Section 5.6 *is* the WDP for bidders placing bids with goalbases in $\mathcal{L}(\Phi, W, F)$, and in all cases we examined, when the collective aggregator is sum and the language is sufficiently rich to express synergies between pairs of items, we have NP-completeness. Therefore, we cannot hope to have a single algorithm which will solve all WDP instances efficiently. Because NP-hardness is a worst-case notion, however, many WDP instances are still efficiently solvable, and there are several approaches we can take which perform acceptably on a variety of inputs.

The *brute force* algorithm for solving the WDP enumerates all complete allocations, computes the social welfare for each, and picks the one with the highest value. Naturally, such an approach will not scale. Good (but possibly suboptimal) results can be obtained using local search methods [Hoos and Boutilier, 2000] and simulated auctions [Fujishima et al., 1999]. For provably optimal results, the most common approach is integer programming, taken by Boutilier [2002] for one logic-based language [Boutilier and Hoos, 2001]. The integer programming approach to solving WDP instances (Section 6.4.3) relies on black-box general-purpose IP solvers like CPLEX [ILOG, 2009]. Such IP solvers are powerful tools, but not tuned specifically for any particular problem [Andersson, Tenhunen, and Ygge, 2000]. Another approach is to craft algorithms to solve WDP instances directly. In Section 6.4.4, we use a branch-and-bound algorithm, similar to the work of Sandholm [2002] and Fujishima et al. [1999]. Branch-and-bound produces provably optimal results, and uses heuristics to guide the search and prune the search tree. We restrict ourselves to the version of the WDP where sum is the collective aggregator. For some exploratory results about Nash product as the collective aggregator, see [Ramezani, 2008, Chapter 6].

Finally, we note that our IP formulation of the WDP could be easily modified for use with max as the collective aggregators instead of sum. Adjusting it to use max would permit it to solve the egalitarian version of the WDP studied by Bouveret [2007]. While we could do the same for branch-and-bound—the upper bound heuristics we develop later in this chapter are also admissible for egalitarian social welfare—the bounds would be quite loose, and so this is likely to be less fruitful than developing new heuristics.

6.4.3 An IP Formulation of the WDP

People wishing to solve instances of the WDP frequently resort to integer programming [Schrijver, 1986; Wolsey, 1998]. Boutilier [2002] compares the performance of an IP formulation of the WDP for generalized logical bids (GLBs, see also Section 2.3.10), which he calls *structured* bids, with an IP formulation of the

WDP for equivalent XOR bids, which he calls *flat* bids, and presents experimental results which indicate that the WDP can be solved faster for GLBs than for equivalent XOR bids. Similarly, we can take advantage of IP for solving the WDP for goalbases in $\mathcal{L}(\mathit{forms}, \mathbb{R}, \Sigma)$. Here we present a fully-general integer programming formulation of the WDP for sum languages, following that given by Boutilier [2002].

A linear program consists of a linear function known as the *objective function* and a set of linear inequalities called *constraints*. Linear programming is the technique of finding values for the variables which optimize the objective function subject to the constraints. In the standard (primal) form, we want to maximize the objective function $c_1x_1 + \ldots + c_nx_n$ while respecting the constraints

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & \leq & b_1 \\ a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & \leq & b_2 \\ \vdots & & \vdots & & \vdots & & \vdots & \vdots & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \ldots & + & a_{mn}x_n & \leq & b_m \end{array}$$

where the $c_i, b_i, a_{ij} \in \mathbb{R}$ are constants and the x_i are variables. Additionally, we require that all $x_i \geq 0$. This formulation suggests a matrix form, which is to maximize $\mathsf{c}^\mathsf{T}\mathsf{x}$ subject to $\mathsf{Ax} \leq \mathsf{b}$ and $\mathsf{x} \geq 0$, where A is the matrix formed by the a_{ij} and c, b, and x are the column vectors formed by the c_i, b_i, and x_i, respectively.

Every linear program has a dual form: minimize $\mathsf{c}^\mathsf{T}\mathsf{y}$ subject to $\mathsf{A}^\mathsf{T}\mathsf{y} \geq \mathsf{c}$ and $\mathsf{y} \geq 0$. If there are feasible solutions x^* and y^* for a primal and its dual such that $\mathsf{c}^\mathsf{T}\mathsf{x}^* = \mathsf{b}^\mathsf{T}\mathsf{y}^*$, then x^* and y^* are optima for the primal and dual, respectively. This fact is useful as it is sometimes easier to solve the dual of an LP than it is to solve the LP directly.

An integer program is a linear program for which the x_i are required to be integers. While LPs are solvable in polynomial time, the additional requirement that solutions be integer makes solving IPs NP-complete. Because IPs are useful for modeling many kinds of business problems, a great deal of research has been applied to finding optimal solutions for them, and as a result IPs are readily solvable despite their high computational complexity. Various methods exist for solving IPs, such as branch-and-bound, cutting planes, branch-and-cut, and column generation. For a survey of these methods, see [Wolsey, 1998]. A 0-1 integer program is one where the domains of all variables are binary. (Binary-valued variables are known as decision variables.) Solving 0-1 integer programs remains NP-complete, despite the domain restriction.

Now we define our 0-1 integer program for solving the WDP for $\mathcal{L}(\mathit{forms}, \mathbb{R}, \Sigma)$ goalbases: Let i index bidders, j index formulas, and r index goods, throughout. The jth formula in bidder i's goalbase G_i we refer to as φ_{ij}. Each w_{ij} is the weight in the goalbase G_i of the corresponding formula φ_{ij}. That is, for bidder i, his goalbase is $G_i = \{(\varphi_{ij}, w_{ij})\}_j$. Next, we define constraints on subformulas of the weighted formulas in a goalbase. For each good r and agent i, define a

binary decision variable y_{ir} which is intended to equal 1 iff agent i receives good r. For each subformula φ, define a binary decision variable s_φ, which is intended to equal 1 iff the allocation satisfies φ. Then, define a constraint for each φ, depending on its logical form:

$$\varphi = p: \qquad s_\varphi \leq y_{ir} \tag{6.1}$$

$$\varphi = \neg\psi: \qquad s_\varphi = 1 - s_{\neg\psi} \tag{6.2}$$

$$\varphi = \psi_1 \wedge \ldots \wedge \psi_n: \quad n \cdot s_\varphi \leq \sum_{k=1}^{n} s_{\psi_k} \tag{6.3}$$

$$\varphi = \psi_1 \vee \ldots \vee \psi_n: \qquad s_\varphi \leq \sum_{k=1}^{n} s_{\psi_k} \tag{6.4}$$

and finally,

$$\text{Maximize} \sum_{\varphi_{ij} \text{ an atomic bid}} w_{ij} s_{\varphi_{ij}} \text{ subject to:} \tag{6.5}$$

$$\sum_i y_{ir} = 1 \text{ for all } r, \text{ and} \tag{6.6}$$

$$\text{A constraint for each subformula of each atomic bid } \varphi_{ij}. \tag{6.7}$$

Note that we maximize only over the atomic bids and not their subformulas, as subformulas have no value independent of the atomic bids which contain them.[4] Additionally, we provide constraints for handling negation.[5] It it worth noting that the negation constraints are the only subformula constraints which contain equalities. This is necessary to ensure bivalence. If, e.g., only $s_\varphi \leq 1 - s_{\neg\varphi}$ were taken as a constraint, then we would not have ruled out $s_\varphi = s_{\neg\varphi} = 0$. The subformula constraints generated from (6.1)–(6.4) are *bid satisfaction* constraints, as they ensure that bids cannot be satisfied without the appropriate items being allocated. The constraints in (6.6) are a combination of *preemption* and *complete allocation*. That is, the $\leq$ direction allocates each item to at most one agent (preemption) and the $\geq$ direction allocates each item to at least one agent (complete allocation—i.e., without free disposal). The complete allocation direction of this constraint can be dropped for languages which contain only positive formulas and positive weights without affecting which allocations are optimal, but is necessary in cases (like task allocation) where bads are being allocated to agents. This is due to the fact that the objective will be maximized by the auctioneer keeping items which all bidders wish to avoid. Only in languages which permit mixed

[4]This differs from Boutilier's approach, where subformulas are permitted to have their own weights. E.g., $((p, 5) \wedge (q, 1), 2)$ is a formula with subweights in Boutilier's language, which we would write as the three weighted formulas $(p, 5), (q, 1), (p \wedge q, 2)$.

[5]Negation does not occur in the languages considered by Hoos and Boutilier [2000], Boutilier and Hoos [2001], and Boutilier [2002], so Boutilier provides no negation rule.

weights or negation as a connective can the desire to avoid receiving an item be expressed, so for these case we need the complete allocation constraint in order to find nontrivial optima.

Given a set of goalbases $\{G_i\}_i$ in $\mathcal{L}(forms, \mathbb{R}, \Sigma)$, the number of binary decision variables s_φ in the IP corresponding to it is bounded by the sum of the number of subformulas in each G_i (which is at most the sum of twice the number of formulas in each G_i) and the number of binary decision variables y_{ir} is equal to $|\mathcal{A}| \cdot |\mathcal{PS}|$. Similarly, there will be one constraint for each subformula and $|\mathcal{PS}|$ constraints to ensure preemption. Hence, both the number of decision variables and the number of constraints is linear in the number of agents, items, and size of the bid representations.

6.4.4 Branch-and-Bound WDP Algorithms

Branch-and-bound is a tree-based search algorithm which uses heuristics to prune branches from its search tree. We first describe branch-and-bound abstractly, and follow with an explanation of how we apply branch-and-bound to the WDP.

Consider any search problem with a finite search space. Branch-and-bound searches by building a tree from that search space. Each node in this tree is a subset of the search space, a set of solutions to our problem. The root of our search tree is S_0, the entire search space. We expand the tree by selecting a leaf S and adding as its children some proper nonempty subsets $S_1, \ldots, S_k$ $(k \geq 2)$ which cover S. (If S is a leaf *and* a singleton, it cannot be further expanded.)

Each element of our search space can be measured for its "quality", i.e., how good a solution it is. Similarly, we can measure the quality of sets of solutions. (How quality is measured is one of the design parameters, which will be discussed later. Suppose for now that we have some way of measuring quality.) For each node we build, we estimate the quality of the solutions it contains. To do this, we have two functions, g and $g + h$: $g(S)$ is a lower bound on the quality of solutions in S, while $g(S) + h(S)$ is an upper bound. These bounds are used to direct our search. For each new node S we create, we calculate $g(S)$ and $h(S)$. If there is some other node S' such that $g(S') \geq g(S) + h(S)$, then S' contains a solution which weakly dominates all solutions in S. Since S' contains solutions which are at least as good as the ones in S, there is no reason to continue searching in S; we may safely prune S from our search tree.

We repeat this procedure, choosing and expanding weakly undominated leaves, until we are left with a leaf which is a singleton $\{s\}$ that weakly dominates all leaves in the tree. If g is exact for singletons, then this solution, s, is optimal, and our search is complete. For a schematic representation of the branch-and-bound algorithm, see Figure 6.1.

In this description, four parts were left underspecified: First, we need a method for choosing which leaf to expand next; second, we need a method for subdividing the search space once we have chosen a leaf to expand; third and fourth, we need

```
S* = S_0
while ∃S a weakly undominated nonsingleton leaf do
  divide S into a covering ∅ ⊂ S_1, ..., S_k ⊂ S
  for all S_i do
    calculate g(S_i), h(S_i)
    if g(S_i) + h(S_i) > g(S*) then
      add S_i as a child of S
      if g(S_i) > g(S*) then
        S* := S_i
      end if
    end if
  end for
end while
return S*
```

Figure 6.1: The branch-and-bound algorithm.

lower and upper bounds for the quality of subsets of the search space so that we can prune away unpromising parts of the search tree. Any branch-and-bound implementation must fill in these four parts.

Branch-and-bound is well suited for optimization problems where the notion of a *partial* solution makes sense. A partial solution defines the set of complete solutions which extend it, which means that partial solutions can be nodes in a branch-and-bound tree. If we also have a way of bounding the quality of partial solutions, then we can use branch-and-bound to incrementally build an optimal solution.

The WDP for combinatorial auctions is just such a problem. Solutions to the WDP are complete allocations of the set of goods. Partial allocations simply leave some goods unallocated, and lower bounds on the values of partial allocations are easy to calculate from the bids received. We take this approach. When a leaf/allocation A is to be expanded/extended, we select a good p unallocated in A and produce as children of A all allocations A' which extend A by allocating p. Thus each expanded node will have one child for each bidder in the auction, since A could be extended by awarding p to any of the bidders in $\mathcal{A}$.

We start with an initial tree consisting of a single node where no goods have been allocated yet. We maintain a *frontier* of leaf nodes and a pointer to the current *top allocation* A^* delivering the highest social welfare so far. The algorithm then repeatedly applies the following steps:

1. Select a node (partial allocation) A from the frontier that still has a chance of beating the current top allocation A^*: $g(A^*) < g(A) + h(A)$. Any A not meeting this condition can be removed from the frontier.

2. Select a good not yet allocated in A: $p \in \text{und}(A)$.

3. Produce as children of A all allocations A' which extend A by allocating p. Thus each expanded node will have one child for each bidder in $\mathcal{A}$. Add all children to the frontier (and remove A from it).

We stop when there are no more viable partial allocations in the frontier to choose from (during step 1). As a solution we return (one of) the best (by now complete) allocations in the final frontier.

A function is called *admissible* for the purposes of branch-and-bound if the function is in fact a bound on the quality of a set of solutions. Whenever the branch-and-bound algorithm is provided with admissible upper- and lower-bound functions, it is guaranteed to eventually find an optimal solution.

For our purposes g is admissible if it never overestimates the value of the worst completion of a partial allocation, and $g+h$ is admissible if it never underestimates the value of the best completion of a partial allocation. Whether a given bound function is admissible depends on what utility functions the bidders are permitted to have, which in turn depends on the choice of bidding language. The trivial lower bound $g(A) = -\infty$ is admissible for any bidding language, but will prevent any pruning of the search space. It is easy to see that $g(A) = \text{sw}(A)$ is admissible so long as all agents' utility functions are monotone: There cannot be an agent who will become worse off by allocating him some items left unallocated in A. Therefore, this g which calculates attained value is admissible for any bidding language which permits the representation of monotone utility functions only. The trivial upper bound $g(A) + h(A) = \infty$ is admissible for any language, but, as with the trivial lower bound, makes branch-and-bound build the entire search tree.[6]

Other ways of applying branch-and-bound to the WDP are possible. Our algorithm branches on goods. Fujishima et al. [1999] and Sandholm and Suri [2003] have designed search algorithms (CASS and BOB, respectively) where branching decisions are taken by accepting or rejecting *atomic bids* (in the OR- or the XOR-language). According to Sandholm [2006], branch-on-bids formulations of the search problem tend to yield faster algorithms than branch-on-goods approaches.

The branch-on-bids formulation works as follows: At each node in the search tree, we select a bid and branch such that along one branch, we accept the selected bid and along the other we reject it. We reach a leaf in the search tree when a branch contains a set of bids which is maximal, in the sense that were we to accept any additional bids, we would no longer have a feasible allocation. The bids which are available for selection at any node in the search tree are the bids which are *undecided* at that node. The undecided bids at a node are the ones which have neither been accepted nor rejected along the branch leading to that

[6]The trivial bounds guarantee that every one of the $|\mathcal{PS}|^{|\mathcal{A}|+1}$ nodes in the search tree will be visited. A brute-force search will visit only the leaves of this search tree; thus a poor choice of bounding can make branch-and-bound *worse* than a brute-force search.

node, nor do they conflict with any bids which have been accepted along that branch. In order to keep track of which nodes are available for selection, a *conflict graph* is associated with each search tree node. The nodes in the conflict graph are the undecided bids, and the (undirected) edges mark conflicts between bids. A selected bid and its neighborhood (the bids it conflicts with) are removed from the conflict graph which the accepting branch inherits, while only the selected bid is removed from the conflict graph inherited by the rejecting branch. In this way, we can easily track which bids are still undecided, and once the conflict graph has no more edges, we have arrived at a leaf in the search tree and may finish by accepting all remaining bids in the conflict graph.

Branch-on-bids results in narrower, deeper search trees than branch-on-items does: A branch-on-items tree will be a $|\mathcal{A}|$-ary tree of depth $|\mathcal{PS}|$, while a branch-on-bids tree will be a binary tree with depth not exceeding the number of atomic bids.

For our bidding languages, branch-on-bids corresponds (roughly) to branching on *goals*: At each branching point we would have to decide whether to satisfy (or satisfy the negation of) a given goal of a given agent. Unfortunately, we cannot immediately adapt the methods developed for standard bidding languages to our situation. In the standard approach using the OR language, nodes in a branch-on-bids search tree correspond to partial allocations, but this is not always the case when branching on goals. For example, if agent i had $p \wedge q$ as a goal and we wished to branch on that goal, the accepting branch is straightforward—we would allocate items p and q to i—but the rejecting branch is not, since there could be many partial allocations where at least one of p and q is not given to agent i. Either we must associate sets of partial allocations with branches, or we must forgo binary branching.

Keeping a conflict graph is no longer straightforward, either. The negative entailment relation for the OR language is simple and symmetric: Two atomic bids conflict iff they overlap. Moreover, if any three bids conflict, then they do so because some pair of them conflicts. There is no positive entailment relation among atomic OR bids at all: There are no $\langle X, a\rangle$ and $\langle Y, b\rangle$ for which it is always the case that the second cannot be accepted without also accepting the first. Conflict among formulas is more complex, in that it is directed. For two formulas φ and ψ, they could be contraries ($\models \varphi \rightarrow \neg\psi$), subcontraries ($\models \neg\varphi \rightarrow \psi$), contradictories ($\models \varphi \leftrightarrow \neg\psi$), or fail to conflict altogether. There are sets of formulas where any pair is consistent but any three are inconsistent, which indicates that we will not have a conflict graph, but rather a *hyper*graph. Finally, it can be the case that $\varphi \models \psi$, and so we must also track positive entailment in our graph if we wish to use it for clearing dead formulas. OR bids may be treated as anonymous for the purposes of branch-and-bound. Which bidder placed which atomic bid is irrelevant for determining which atomic bids are accepted in an optimal allocation. When bidding with formulas it matters which formula belongs to which bidder. Consider that $a \wedge b$ entails b if both bids belong to the same bidder, but $a \wedge b$

entails $\neg b$ if they belong to different bidders. We leave the investigation of this alternative approach to another occasion.

Finally, it is worth noting that the A* algorithm for finding shortest paths [Hart, Nilsson, and Raphael, 1968] is an instance of branch-and-bound, where g is the actual distance from the start node to the current node and h is an underestimate of the distance from the current node to the goal node. (h is an underestimate because better paths are shorter. By *under*estimating the distance, h *over*estimates the path quality.) A* selects the node with the least (best) $g + h$ as the node to expand next. The algorithm calculates g and h for each neighbor of the current node, and they too are marked as visited. This continues until the goal node is reached.

6.5 Heuristics for Winner Determination

In this section, we present the required heuristics for using branch-and-bound to solve the WDP for three goalbase languages: $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$, $\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$, and $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$. As stated in the previous section, a branch-and-bound implementation requires:

- A lower-bound function,
- An upper-bound function,
- An *expansion policy*, a function which chooses which undominated node to expand next, and
- A *branching policy*, a function which creates the children of nodes chosen by the expansion policy.

NP-hardness of the WDP precludes the existence of a globally optimal ensemble of lower- and upper-bound functions and expansion and branching policies (unless, of course, P = NP). While we cannot find bounds and policies which *always* perform well, there might still be choices for these which perform well across a broad range of cases. For all three languages, we choose to define a lower bound function g and a marginal upper bound function h, thus making the upper bound function $g + h$.

6.5.1 Expansion and Branching Policies

Our expansion and branching policies differ across the three languages only insofar as they rely on language-specific g and h functions for input. Therefore, we present the expansion and branching policies first, before defining upper and lower bounds for each language.

An Expansion Policy

The most obvious expansion policy for any branch-and-bound implementation is to choose the unexpanded node with the greatest upper bound $g + h$ as the next node to expand. This is the node with the most potential according to our upper-bound heuristic, and ordering nodes by their upper bounds is an inexpensive operation. Ties may be resolved arbitrarily, though best lower bound could be used as a tie-breaker. Unlike the upper and lower bound functions, there is no way for an expansion policy to fail, so long as it chooses only unexpanded nodes.

Two Branching Policies

Let $\mathfrak{A}$ be the set of strictly partial allocations. A function $b\colon \mathfrak{A} \to \mathcal{PS}$ is a *branching policy* if for all strictly partial allocations A, $b(A) = p$ for some p which A does not allocate. For each language we consider here, its marginal contribution functions h is composed from one h^p function for each $p \in \mathcal{PS}$. In all cases, $h^p(A)$ is an upper bound on the marginal value of allocating item p.

Definition 6.5.1. We define two branching policies:

- The *lexical* branching policy is the branching policy b such that $b(A) = p$, where p is the lexically least good not allocated by partial allocation A.
- The *best-estimate first* branching policy is the branching policy b such that $b(A) = p$, where p is the lexically least good such that $h^p(A) = \max_{a \in \mathcal{PS}} h^a(A)$.

The lexical branching policy is equivalent to branching randomly. As with expansion policies, no branching policy can fail, though one might direct the search better than another.

6.5.2 Heuristics for Positive Cubes

The language $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ corresponds to the so-called *k-additive form* for representing utility functions [Grabisch, 1997; Chevaleyre et al., 2006].[7] Intuitively, the weight given to a cube of the form $p_1 \wedge \cdots \wedge p_m$ may be regarded as the marginal utility associated with obtaining all of items $p_1, \ldots, p_m$, beyond the utility associated with any subset of these. If none of the cubes has a length exceeding k, then the agent in question is said to have k-additive preferences. These kinds of languages have been widely used for preference modeling, and recently their relevance to the theory of combinatorial auctions has also been recognized [Conitzer et al., 2005].

[7] To be precise, $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ corresponds to the k-additive form with positive coefficients. See Theorems 3.4.6 and 3.4.15, and Corollary 3.4.9 for details.

A Lower Bound Heuristic for Positive Cubes

Because only monotone utility functions are representable in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$, the attained value of a partial allocation, $g(A) = \mathrm{sw}(A)$, is an admissible lower bound.

An Upper Bound Heuristic for Positive Cubes

We now define our upper-bound heuristic for $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$. For simplicity, we assume that prior to the use of this heuristic, all goalbases have had duplicate formulas collapsed, i.e., if (φ, x) and (φ, y) occur in G_i, they are replaced by $(\varphi, x + y)$.

Definition 6.5.2. Define the heuristic function $h^+_\wedge$ as

$$h^+_\wedge(A) = \sum_{p \in \mathcal{PS}} h^p(A),$$

where

$$\begin{aligned} h^p(A) &= \max_{i \in \mathcal{A}} h^p_i(A) \\ h^p_i(A) &= \sum_{(\varphi, w) \in G_i} h^p_i(A, \varphi) \\ h^p_i(A, \varphi) &= \begin{cases} \frac{w}{|\mathrm{und}(A,\varphi)|} & \text{if } (\varphi, w) \in G_i,\ p \in \mathrm{und}(A, \varphi), \text{ and } M^A_i \,?\, \varphi \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

The intuition embodied here is that we can estimate the marginal value of an item for an agent by assigning to each item a share of the weight of each positive cube in which it appears. For example, suppose agent 1 bids $\{(a \wedge b, 6), (a \wedge c, 8)\}$, and agent 2 bids $\{(a, 6), (b \wedge c, 10)\}$. Under the empty partial allocation $\emptyset$, for agent 1 we have that $h^a_1(\emptyset) = \frac{6}{2} + \frac{8}{2} = 7$, $h^b_1(\emptyset) = \frac{6}{2} = 3$, and $h^c_1(\emptyset) = \frac{8}{2} = 4$. For agent 2, we have $h^a_2(\emptyset) = 6$, $h^b_2(\emptyset) = \frac{10}{2} = 5$, and $h^c_2(\emptyset) = \frac{10}{2} = 5$. Since each item may be allocated to only one agent, the atom-wise components of the heuristic "award" each item to the agent for whom that item contributes most: $h^a(\emptyset) = 7$ since $h^a_1(\emptyset) > h^a_2(\emptyset)$, $h^b(\emptyset) = 5$ since $h^b_2(\emptyset) > h^b_1(\emptyset)$, and $h^c(\emptyset) = 5$ since $h^c_2(\emptyset) > h^c_1(\emptyset)$. The marginal upper bound $h^+_\wedge(\emptyset)$ is the sum of the maximum contributions of the atoms: $h^+_\wedge(\emptyset) = 7 + 5 + 5 = 17$. Notice that in this case the optimal value is 16, which is attained when agent 2 receives all three items; the heuristic overestimates the optimal value to be 17 instead.

Theorem 6.5.3. *The heuristic $g + h^+_\wedge$ is an admissible upper bound for the language $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$.*

Proof. We proceed by showing that $g(A') + h^+_\wedge(A') \leq g(A) + h^+_\wedge(A)$ for all allocations A' extending A, or equivalently, that $g(A') - g(A) \leq h^+_\wedge(A) - h^+_\wedge(A')$.

Fix an allocation A and an item p, where $(p, *) \in A$. Let A' extend A by assigning p to some agent k. That is, $A' = (A \setminus \{(p, *)\}) \cup \{(p, k)\}$, for some agent k.

If agent k is indifferent to item p (i.e., $p \notin \bigcup_{(\varphi,w)\in G_k} \text{atoms}(\varphi)$), then there is no change in utility from A to A': $g(A') - g(A) = 0$. h is a decreasing function (as allocations are extended), so $h^+_\wedge(A) - h^+_\wedge(A') \geq 0$.

Otherwise, suppose that agent k is not indifferent to item p. We have that

$$g(A') - g(A) = \sum_{\substack{(\varphi,w)\in G_k \\ M_k^A ? \varphi \\ M_k^{A'} \models \varphi}} w$$

since the attained utility of no agent other than k can change from A to A'. Only p is allocated between A and A', and for any φ which becomes true in A' it must be the case that p is the last undetermined atom in φ in A. In A, the atom p carries all of the potential weight of φ. That is, $h^p_k(A, \varphi) = w_\varphi$, so

$$\sum_{\substack{(\varphi,w)\in G_k \\ M_k^A ? \varphi \\ M_k^{A'} \models \varphi}} w = \sum_{\substack{(\varphi,w)\in G_k \\ M_k^A ? \varphi \\ M_k^{A'} \models \varphi}} h^p_k(A, \varphi).$$

We have that

$$\sum_{\substack{(\varphi,w)\in G_k \\ M_k^A ? \varphi \\ M_k^{A'} \models \varphi}} h^p_k(A, \varphi) \leq \sum_{(\varphi,w)\in G_k} h^p_k(A, \varphi)$$

because every $h^p_k(A, \varphi) \geq 0$. Further, it is the case that $h^p_k(A, \varphi) \leq \max_{i\in\mathcal{A}} h^p_i(A)$ and $\max_{i\in\mathcal{A}} h^p_i(A) = h^p(A)$ by definition. Thus far, we have that $g(A') - g(A) \leq h^p(A)$.

Now we work from the opposite end. Observe first that

$$\sum_{\substack{a\in\mathcal{PS} \\ a\neq p}} (h^a(A) - h^a(A')) \geq 0,$$

from which it follows that

$$h^p(A) \leq \Big(\sum_{\substack{a\in\mathcal{PS} \\ a\neq p}} (h^a(A) - h^a(A')) \Big) + h^p(A).$$

Now, $h^p(A') = 0$ because p is already allocated in A', and so

$$\begin{aligned} h^p(A) &\leq \Big(\sum_{\substack{a \in \mathcal{PS} \\ a \neq p}} (h^a(A) - h^a(A')) \Big) + h^p(A) - h^p(A') \\ &= \sum_{a \in \mathcal{PS}} (h^a(A) - h^a(A')) \\ &= h^+_\wedge(A) - h^+_\wedge(A'), \end{aligned}$$

which gives us that $h^p(A) \leq h^+_\wedge(A) - h^+_\wedge(A')$.

Together, the first and second parts of the proof show that

$$g(A') - g(A) \leq h^p(A) \leq h^+_\wedge(A) - h^+_\wedge(A'),$$

which was to be proved. □

We note that this heuristic is similar in concept, though not in execution, to the upper-bound heuristic for the OR language used by Fujishima et al. [1999] in CASS and Sandholm [2002] in BOB,

$$\sum_{i \in A} c(i) \quad \text{where } c(i) = \max_{j | i \in S_j} \frac{p_j}{|S_j|},$$

where the $\langle S_j, p_j \rangle$ are atomic bids in the OR language and A is the set of unallocated items along some path of the search tree. This upper bound heuristic assigns to each unallocated item an estimated value equal to its "fair" contribution to the best bundle which contains it. (E.g., each of items a, b, and c are considered to contribute 14 to the bundle $\langle \{a, b, c\}, 42 \rangle$.) Because the OR language does not permit the acceptance of overlapping atomic bids, there is no need for the heuristic to track which items are already allocated (all items in live bids are still unallocated) nor to which agent the items are being allocated, which allows it to be simpler than the one we present.

6.5.3 Heuristics for Positive Clauses

$\mathcal{L}(pclauses, \mathbb{Q}^+, \Sigma)$ is an interesting bidding language due to its ability to concisely express substitutes of unequal quality. For example, if I have a Phillips screw I want to turn, I would prefer to do the job with a Phillips screwdriver, though I could use an appropriately-sized flathead screwdriver in a pinch (at the risk of stripping the screw's head). Therefore, in an auction for a Phillips (p) and a flathead (f) screwdriver I might bid $\{(p, 5), (f \vee p, 1)\}$.

A Lower Bound Heuristic for Positive Clauses

As with $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$, the attained value of a partial allocation is an admissible lower bound heuristic for $\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$, due again to the monotonicity of all representable utility functions here.

An Upper Bound Heuristic for Positive Clauses

As before, we assume that duplicate formulas have been collapsed.

Definition 6.5.4. Define the heuristic function $h^+_\vee$ as

$$h^+_\vee(A) = \sum_{p \in \mathcal{PS}} h^p(A),$$

where

$$\begin{aligned} h^p(A) &= \max_{i \in \mathcal{A}} h^p_i(A) \\ h^p_i(A) &= \sum_{(\varphi,w) \in G_i} h^p_i(A, \varphi) \\ h^p_i(A, \varphi) &= \begin{cases} w & \text{if } (\varphi, w) \in G_i, p \in \text{und}(A, \varphi), M^A_i \,?\, \varphi \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

This heuristic is similar to $h^+_\wedge$; the difference is that we do not divide by the number of unallocated atoms in φ when defining $h^p_i(A, \varphi)$.

Theorem 6.5.5. *The upper-bound heuristic $g + h^+_\vee$ is admissible for the language $\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$.*

Proof. The argument exactly parallels that given for $h^+_\wedge$ in the proof of Theorem 6.5.3. □

6.5.4 Heuristics for Cubes

$\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$ is a potentially attractive bidding language due to its ability to express marginal utilities easily (which it inherits from $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$) as well as to indicate a desire to avoid certain bundles (useful, for example, if some items are not goods, but instead undesirable tasks which must be completed by their winner).

A Lower Bound Heuristic for Cubes

Unlike with $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ and $\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$, attained value of partial allocations is not an admissible lower bound heuristic for $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$. When all bids are expressed with goalbases containing only positive formulas with positive weights, the attained value of partial allocations over those bids is nondecreasing as more items are allocated; for cubes, allocating an additional item might cause a decline in overall value if some cubes contain negative literals. However, attained value *is* admissible for cubes so long as the bids meet the following (rather complex) condition, which is a sort of localized, conditional, item-specific free disposal:

> For every partial allocation A and each item p left unallocated by A, there exists a bidder i for whom every extension $A' \supseteq A$ which also does not allocate p is such that $u_{G_i,\Sigma}\big(M_i^{A' \cup \{(p,i)\}}\big) \geq u_{G_i,\Sigma}(M_i^{A'})$.

That is, attained value is admissible as a lower bound for cubes so long as we can always fob off an unwanted item onto some bidder who is not hurt by it. Fortunately, in an auction with a large number of goods and bidders this condition will almost always be satisfied. If for each item being auctioned, there is some bidder whose utility function is locally monotone with respect to that item, then the condition is met. This could happen if each item has at least one bidder who is indifferent towards it, or if there is even a single bidder with a monotone utility function. All test data used in the $\mathcal{L}(\mathit{cubes}, \mathbb{R}^+, \Sigma)$ experiments on which we report in Section 6.7.1 meet this condition, and so we were able to use attained value as a lower bound there.

If this condition cannot be met, then a different function must be used as a lower bound for $\mathcal{L}(\mathit{cubes}, \mathbb{R}^+, \Sigma)$. Subtracting the weights of satisfied nonpositive formulas from the attained value gives us one admissible lower bound, though it could be rather loose if the disregarded nonpositive formulas have significant weight. We can do better than this, however: Atoms corresponding to unallocated items default to false in bidders' individual models, and M_i^A does not preserve information about whether an atom a is false for agent i because a was allocated to some agent $j \neq i$, or because a is still unallocated. Due to this, false atoms may sometimes turn true for a bidder—but only those atoms which correspond to unallocated items. Once all of the items named in a formula are allocated, the truth value of that formula cannot change with the allocation of further items; hence, it is safe to include in the lower bound bids whose formulas have fixed truth values. So

$$\sum_{i \in \mathcal{A}} \sum_{\substack{(\varphi, w) \in G_i,\, M_i^A \models \varphi \\ \mathrm{Var}(\varphi) \cap \mathrm{und}(A) = \emptyset}} w$$

is an admissible lower bound for $\mathcal{L}(\mathit{cubes}, \mathbb{R}^+, \Sigma)$ without any further conditions.

An Upper Bound Heuristic for Cubes

The marginal upper bound heuristic for cubes is similar to $h^+_\wedge$ for positive cubes, though now we must take negative literals into account as well.

Definition 6.5.6. Define the heuristic function $h_\wedge$ as:

$$h_\wedge(A) = \sum_{p \in \mathcal{PS}} h^p(A),$$

where

$$h^p(A) = \max_{i \in \mathcal{A}} \Big(h_i^p(A) + \sum_{j \neq i} h_j^{\neg p}(A) \Big)$$

$$h_i^\ell(A) = \sum_{(\varphi, w) \in G_i} h_i^\ell(A, \varphi)$$

$$h_i^\ell(A, \varphi) = \begin{cases} \frac{w}{|L|} & \text{if } M_i^A \,?\, \varphi \text{ and } \ell \in L = \{\hat{\ell} \mid \hat{\ell} \text{ a literal in } \varphi, M_i^A \,?\, \hat{\ell}\} \\ 0 & \text{otherwise.} \end{cases}$$

The crucial difference from Definition 6.5.2 is for $h^p(A)$, which in this case maximizes over the estimated contribution of the agent receiving p as well the agents wishing to avoid p.

Theorem 6.5.7. *The upper-bound heuristic $g + h_\wedge$ is admissible for the language $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$.*

Proof. Fix an allocation A and extend it to A' by allocating p. If no agent gained utility from A to A', then $g(A') - g(A) \leq h(A) - h(A')$ trivially. Otherwise, suppose that agents $j_1, \ldots, j_n$ experienced a gain from A to A'. $g(A') - g(A) \leq h_{j_1}^{\ell_1} + \ldots + h_{j_n}^{\ell_n}$, where $\ell_i = p$ if $i = k$, and $\ell_i = \neg p$ otherwise. Then

$$h_{j_1}^{\ell_1} + \ldots + h_{j_n}^{\ell_n} \leq \max_{i \in \mathcal{A}} \Big(h_i^p(A) + \sum_{j \neq i} h_j^{\neg p}(A) \Big) = h^p(A).$$

From here, the proof is as for $g + h_\wedge^+$ in Theorem 6.5.3. □

6.6 Experimental Setup

Here we describe some design decisions we faced in our attempts to test the heuristics and IP described above. In particular, this section revolves around the generation of test data.

6.6.1 Principles for Generating Realistic Data

In order to test the feasibility of the WDP algorithms detailed above, we need data against which to test them. Data from real combinatorial auctions would be best. While we would not expect to find data sets already expressed as goalbases, these could be constructed from bidders' preference information if only it were available. Little has changed since 2000, when Andersson et al. [2000] lamented the lack of freely-available real-world data. Hence, we are forced to test against artificial, generated data—so we aim to make that data as realistic as possible. It is hard to know what counts as *realism* when we have never seen any *real* data before; absent that, we attempt to generate data in a principled way. We state here several principles which we believe that real-world valuations would follow:

- **Narrow range of values:** The valuations of all bidders for a given bundle of goods will fall within a narrow band. There should be no two agents who have dramatically different values for the same bundle. This is the case with everyday goods and typical bidders—the value of a bicycle to Alice will not be ten or a hundred times what it is to Bob.[8]

- **Few atomic bids:** The number of atomic bids for a given set of goods—that is, the number of equivalence classes of formulas over the atoms representing those goods—can be quite large. For example, the languages $\mathcal{L}(pcubes, \mathbb{R}, \Sigma)$, $\mathcal{L}(cubes, \mathbb{R}, \Sigma)$, and $\mathcal{L}(form, \mathbb{R}, \Sigma)$ have, respectively, $2^{|\mathcal{PS}|}$, $3^{|\mathcal{PS}|}$ and $2^{2^{|\mathcal{PS}|}}$ distinct nonequivalent formulas which could be weighted. For large sets of goods, bidders could not be expected to weight even a small fraction of the available formulas.

- **Short atomic bids:** Bidders are likely to have values for individual goods, and synergies among goods are more likely to be binary or ternary than they are to be 10-ary or 100-ary. This seems to reflect common experience. E.g., many items have strong synergies in pairs, triples, and quadruples (shoes, a necklace and a matching set of earrings, automobile tires), but hardly any have noticeable synergies at much larger sizes. (Are there any 100 goods in the world, collectibles excepted, where the set of 100 together are worth significantly more than the value of any 99-good subset plus the value of the hundredth good alone?)

- **No good unvalued:** Every good should contribute positive value to some bundle. Supposing that we are auctioning *goods* (and not *bads*) then it would be strange if there were some item for which no bidder was willing to offer a nonzero bid.

We do not argue that every real-world instance of the WDP would have these characteristics, only that it would not be incongruous if one did, and hence an instance of the WDP having these characteristics might be considered realistic.

We ultimately settled on testing against two data sets. The design of one of our test data sets is guided by these principles. The other data set, designed to be simple to generate rather than realistic, was used in our initial work on winner determination algorithms [Uckelman and Endriss, 2008a], and serves to highlight how the performance of our branch-and-bound heuristic varies with the type of problem instances it faces.

[8] We say *everyday* goods and *typical* bidders for good reason—for large differences in valuation might be possible if, e.g., the bicycle has sentimental value for Alice, or there is some information asymmetry among the bidders.

6.6.2 Data Generation

We now describe in detail three data generators—the two we used in our experiments, and one which we did not.

Why We Did Not Use CATS

Combinatorial auctions researchers have devised numerous data distributions for testing their algorithms [Sandholm, 2002; Fujishima et al., 1999; Boutilier, Goldszmidt, and Sabata, 1999b; de Vries and Vohra, 2003; Parkes, 1999; Sandholm, Suri, Gilpin, and Levine, 2005; Andersson et al., 2000]. Leyton-Brown, Pearson, and Shoham [2000] argued against using these data distributions for evaluating combinatorial auction algorithms on the grounds that they are ad hoc and have no clear connection to real-world auctions, and that the lack of standardization makes comparison of results difficult and so stymies progress in the field. As an alternative, they devised the Combinatorial Auctions Test Suite (CATS) which is able to generate test data based on five semi-realistic problems—paths in space, proximity in space, arbitrary relationships, temporal matching, and temporal scheduling—and can also generate data using fifteen ad hoc distributions found in the literature [Leyton-Brown and Shoham, 2006]. After the introduction of CATS, it was rapidly adopted by combinatorial auctions researchers for evaluating their algorithms [Sandholm et al., 2005; Gonen and Lehmann, 2000, 2001; Holte, 2001; Schuurmans, Southey, and Holte, 2001; Kastner, Hsieh, Potkonjak, and Sarrafzadeh, 2002; Zurel and Nisan, 2001]. The uptake of CATS has been so extensive that today, if an experimental paper on combinatorial auctions algorithms is submitted to a conference *without* testing against CATS, the authors will surely be asked by the referees to justify that decision. As we did not use CATS for our experiments, we now justify why not.

We did not use CATS because the output it produces is fundamentally unsuitable for use with algorithms which operate on goalbase languages. CATS generates XOR bids. Our succinctness results in Section 6.3.3 indicate that translation from XOR bids to a goalbase language sufficiently expressive to capture them could result in an exponential blow-up in bid size. In particular, Theorem 6.3.7 shows that the XOR language is strictly less succinct than $\mathcal{L}(\mathit{pforms}, \mathbb{R}^+, \Sigma)$; Theorem 6.3.3 shows that XOR is strictly less succinct than OR, and Theorem 6.3.5 shows that OR is not more succinct than $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$, from which follows that XOR is also not more succinct than $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$.

The possibility for exponential blow-up of bids when translating from XOR to goalbases poses two problems, one computational and the other methodological. The computational problem we face when doing such translations is that the only algorithms we know for carrying them out require exponential time. For example, we can use the Möbius inversion for finding weights if we are translating from XOR to $\mathcal{L}(\mathit{pcubes}, \mathbb{R}, \Sigma)$, but doing so means calculating a weight for every one of

the $2^{|\mathcal{PS}|}$ positive cubes (see Theorem 3.4.2 and p. 36). Even if we already know that there are polysize translations from XOR to our target goalbase language, this only guarantees us that there is some PSPACE algorithm for finding such a translation. Not only do we not have such an algorithm, but since NP $\subseteq$ PSPACE it could turn out to be harder to translate from XOR to our target goalbase language than to solve the WDP in the first place! That is, even if we had optimal algorithms for translating XOR bids into goalbases, it might be computationally prohibitive to run them.

The methodological problem with translating from XOR to goalbases is that by using translated input, we are measuring hammers for how well they drive screws. There is no sense in which the goalbases which would result from translation out of the XOR language would be natural input for goalbase WDP algorithms. Bidders creating bids in a goalbase language are unlikely to write the huge bids which could result from translation. What we are interested in is the performance of our algorithms on the input for which they were designed, not on input which is the result of translation; hence, we did not use CATS.

Random Data Generation

Our random data generators are quite simple:

- For $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ and a fixed number of goods m, we generate goalbases as follows: For each agent, we randomly choose an integer in $[1, 2m]$ to be the number of formulas in that agent's goalbase, with each potential atom appearing in a given positive cube with probability 0.5. (This makes positive cubes of middling length more probable than very short or very long ones.) Each cube thus constructed is given a random integer weight uniformly chosen from $[1, 10]$.

- For $\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$ and a fixed number of goods m, we generate goalbases as follows: For each agent, we randomly choose an integer in $[1, 2m]$ to be the number of formulas in that agent's goalbase, with each potential atom appearing in a given positive clause with probability 0.5. Each clause is given a random integer weight uniformly chosen from $[1, 10]$.

- For $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$ and a fixed number of goods m, we generate goalbases as follows: For each agent, we randomly choose an integer in $[1, 2m]$ to be the number of formulas in that agent's goalbase, with each potential atom appearing in a given cube with probability 0.5, and, if appearing, positive with probability 0.5 and negative otherwise. (That is, an atom will appear positively with probability 0.25, negatively with probability 0.25, and not at all with probability 0.5.) Each formula is given a random integer weight uniformly chosen from $[1, 10]$.

for all agents **do**
  for $k \in 1, \ldots, |\mathcal{PS}|$ **do**
    randomly choose $\left\lceil p_k \cdot \binom{|\mathcal{PS}|}{k} \right\rceil$ distinct k-pcubes
    for all chosen k-pcubes φ **do**
      choose a random $r \in [0, 1]$
      if $k = 1$ **then**
        $w_\varphi := \text{bin}(\varphi) \cdot \texttt{basevalue} \cdot (1 + (2r \cdot \texttt{variance}))$
      else
        $w_\varphi := u_G(\text{atoms}(\varphi)) \cdot \texttt{variance} \cdot r$
      end if
      $G := G \cup \{(\varphi, w_\varphi)\}$
    end for
  end for
end for

Figure 6.2: Principled distribution for $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$.

Principled Data Generation

Here we aim for test data which follows the principles outlined in the previous section. Because it is clearest how to follow these principles for $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$, we present a data generator for that language only.

We divide the atoms into a number of bins (`bins`), and specify a base value (`basevalue`) and variance (`variance`), such that the atoms in bin i have value $i \cdot \texttt{basevalue} \pm \texttt{variance}\%$. The function bin: $\mathcal{PS} \to \{1, \ldots, \texttt{bins}\}$ maps each atom to the bin it occupies. For each $k \in \{0, \ldots, |\mathcal{PS}|\}$, we specify a proportion $p_k \in [0, 1]$ of the number of k-pcubes which should receive weights. Each agent receives $\left\lceil p_1 \cdot |\mathcal{PS}| \right\rceil$ weighted atoms. Each atom is weighted by the base value of its bin, adjusted up or down by a variance factor chosen from a uniform distribution. For any k-pcube φ which is longer than a single atom, it will receive additional weight in proportion to the weight of the state where only its constituent atoms are true. For a formal statement of the procedure, see Figure 6.2.

The test data generator, `dist2`, takes as arguments the number of auction instances to produce, the number of agents and goods for each auction, and for each $k \in 0, \ldots, |\mathcal{PS}|$, the proportion of k-pcubes to be weighted. An example of the output produced by `dist2` is seen here, a single auction instance with two agents and two goods, no 0-pcubes and all 1- and 2-pcubes weighted:

```
[juckelma@plataan wdp-c]$ ./dist2 1 2 2 0 1.0 1.0
[ { 01,9.97086 10,20.3416 11,1.3874 }
  { 01,9.72613 10,19.9248 11,0.0111367 } ]
```

Auction instances are delimited by square brackets (`[]`), goalbases by curly

braces (`{}`). The pcubes are represented as binary vectors, where a 1 in position i indicates that the atom p_i occurs in the pcube. Weights are separated from formulas by commas. The proportion of weighted k-pcubes is assumed to be zero when not specified for some k.

6.7 Experimental Results

We conducted experiments with two different branch-and-bound solvers, the first written in Java, and the second a reimplementation of the first in C++, as well as with a C++ solver which feeds our IP formulation of the WDP to CPLEX. Here we recount the results of those experiments.

6.7.1 First Solver

The first solver uses test data from the random data generator. The test program, `TestHarness.java`, takes as arguments the number of goods and agents for each auction, the number of auctions to be run, the name of a class to generate goalbases for each auction, and a list of (class names of) WDP solvers to use for each auction. Every solver listed is tried for every auction instance, the CPU time expended by the solver is recorded (some solvers also record additional data), and the values of the allocations generated by each solver are compared to ensure equality.[9]

In order to verify the correctness of this solver, we implemented a brute-force search solver against which to check our branch-and-bound results. Due to the large search space, it was possible to run the brute-force solver only for small problem instances.

All experiments reported in this section were conducted on a Fedora 7 Linux system, using kernel 2.6.23 with a 2 GHz Intel Core 2 Duo T7300 CPU and 2 GB of RAM, and Sun's Java SE JVM, version 1.6.0_02.

Results for $\mathcal{L}(\boldsymbol{pcubes}, \mathbb{R}^+, \Sigma)$

For the language $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$, we tested these solvers:

`BruteForce` A brute-force search. Iterates over all complete allocations, returning the lexically first optimal one.

`PCubeLex` A branch-and-bound solver, using the upper-bound heuristic $g + h^+_\wedge$ (given in Definition 6.5.2) and the lexical branching policy.

`PCubeBF` A branch-and-bound solver, using the upper-bound heuristic $g + h^+_\wedge$ and the best-estimate-first branching policy.

[9] Because we are testing solvers which find provably optimal allocations, every allocation returned should have the same value—though it is possible that two properly-functioning solvers will find different, but equivalently-valued, optimal allocations.

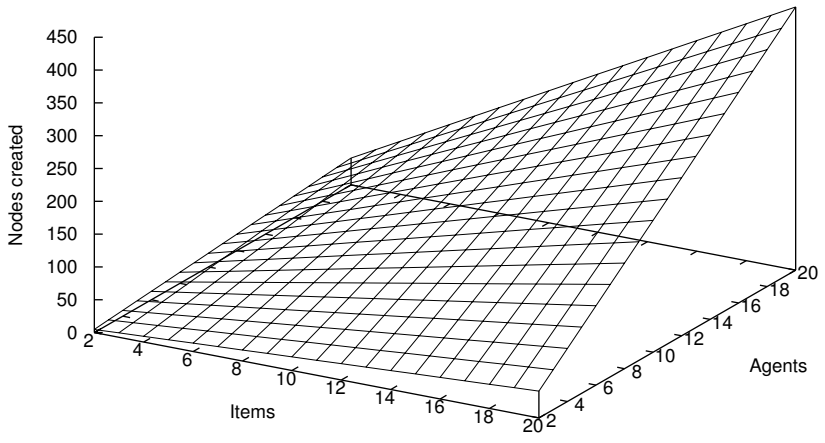


Figure 6.3: Nodes created by `PCubeLex`, averaged over 20 WDP instances of each size, random data.

Our first experiment tested the `BruteForce` solver against `PCubeLex`, both as proof-of-concept and to establish a baseline for comparison. As expected, the `BruteForce` solver is rapidly overwhelmed by the size of the solution space even for small problem instances. Instances of size $(8, 8)$ (8 agents, 8 goods) which take nearly 54 seconds to solve by `BruteForce` can be solved by `PCubeLex` in less than 0.01 seconds.

Our second experiment tested the branch-and-bound solvers to see how many partial allocations (nodes) were created for each WDP instance. The number of nodes created is a useful measure of efficiency for a branch-and-bound solver. The worst case for any WDP instance with n agents and m goods is that every partial allocation is built, which amounts to creating a complete n-ary tree of depth m. (This happens for all instances if the heuristic $h(A) = \infty$ is used, since no pruning can ever occur.) Figure 6.3 shows the average number of nodes built during twenty runs of `PCubeLex` at every size of WDP in $[2, 20] \times [2, 20]$. What can be seen here is that `PCubeLex` is quite parsimonious in building nodes. At $(20, 20)$, `PCubeLex` builds, on average, only 401 of the 5.5×10^{24} possible nodes. In general, the increase in number of nodes built is rather gentle: At $(20, 70)$, for example, most instances generate around 1400 nodes.

Our third experiment tested the effect on solver runtime of varying the branching policy. Intuitively, one would expect that a best-estimate-first branching policy would produce better performance than a lexical branching policy when used with

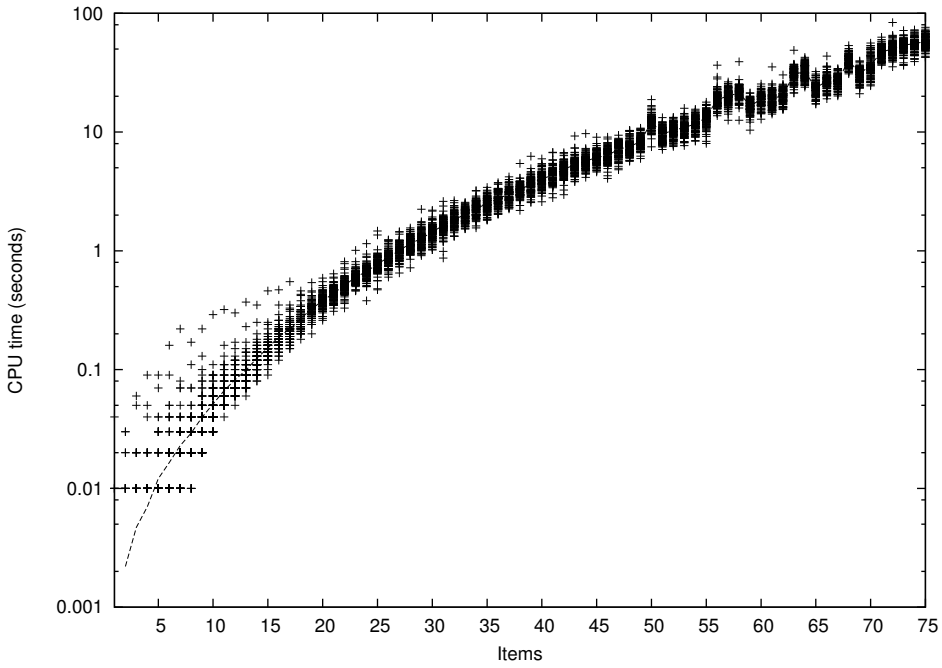


Figure 6.4: CPU time for WDP instances with 20 agents in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ using the `PCubeBF` solver, random data.

an upper-bound heuristic which is inexact. (When generating goalbases randomly, lexical branching is essentially the same as random branching.) However, for `PCubeLex` and `PCubeBF` this turned out not to be the case—the runtime saved in other parts of the solver by calculating which good to branch on next was usually consumed by the calculation itself, and so seldom was any gain realized this way. Our implementation of `PCubeBF` does not cache the values it calculates for h^p when determining the upper bound for a partial allocation. Essentially all of the time spent by `PCubeBF` in implementing its branching policy is spent recalculating these h^ps, so a caching implementation could achieve a small edge over `PCubeLex`, in the neighborhood of a few percent of total runtime.

In our fourth experiment, we fixed the number of agents at twenty and varied the number of goods from 1 to 75, and solved 100 randomly-generated (as above) WDP instances of each size using `PCubeBF`, the results of which can be seen in Figure 6.4. Due to our method of randomly generating goalbases, the number of atomic bids (i.e., weighted formulas) present in any WDP instance will be around $|\mathcal{A}| \cdot |\mathcal{PS}|$. For example, the average instance with 20 agents and 75 items will contain around 1500 atomic bids. Our best solver for $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ on this data set, `PCubeBF`, is capable of solving problems with nearly one hundred items and thousands of bids in under one minute.

Results for $\mathcal{L}(\boldsymbol{pclauses}, \mathbb{R}^+, \Sigma)$

The solvers used for $\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$ were:

`BruteForce` A brute-force search, as before.

`PClauseLex` A branch-and-bound solver using $g + h^+_\vee$ with the lexical branching policy.

`PClauseBF` A branch-and-bound solver using $g + h^+_\vee$ with the best-estimate-first branching policy.

Our first experiment tested the `BruteForce` solver against the two branch-and-bound solvers for $\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$, `PClauseLex` and `PClauseBF`. Instances of size $(8, 8)$ which take nearly 98 seconds to solve by `BruteForce` are solvable by `PClauseLex` in 0.65 seconds and by `PClauseBF` in 0.18 seconds.

Our second experiment compared the performance of the two branching policies. At each size in $[2, 10] \times [2, 10]$, we solved 20 WDP instances with both `PClauseLex` and `PClauseBF`, to compare the number of nodes generated in each case. The results are shown in Figure 6.5. Here, unlike for our $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ heuristics, the best-estimate-first branching policy contributes dramatically to reducing the number of nodes built. For instances of size $(9, 9)$, we found that, on average, `PClauseBF` built one-seventh the nodes that `PClauseLex` did.

The third experiment fixed the number of agents at twenty, and solved using the `PClauseBF` solver twenty randomly-generated WDP instances with each number of goods from 1 to 11. The results of this are shown in Figure 6.6. Of note here is the large variance in runtime between instances of the same size. For example, at $(20, 10)$ the easiest instance was solved in 0.06 seconds, while the hardest took 505 seconds, i.e., the hardest instance took 8400 times longer than the easiest. (For comparison, the hardest instances solved by `PCubeBF` at any given size usually took only two to three times as long as the easiest instances.)

Results for $\mathcal{L}(\boldsymbol{cubes}, \mathbb{R}^+, \Sigma)$

Finally, we present our results for the bidding language $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$, the most expressive language considered in this chapter. Specifically, $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$ permits the use of negation, a feature not commonly found in the literature on combinatorial auctions.

The solvers used for $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$ were:

`BruteForce` A brute-force search, as before.

`CubeLex` A branch-and-bound solver using $g + h_\wedge$ with the lexical branching policy.

`CubeBF` A branch-and-bound solver using $g + h_\wedge$ with the best-estimate-first branching policy.

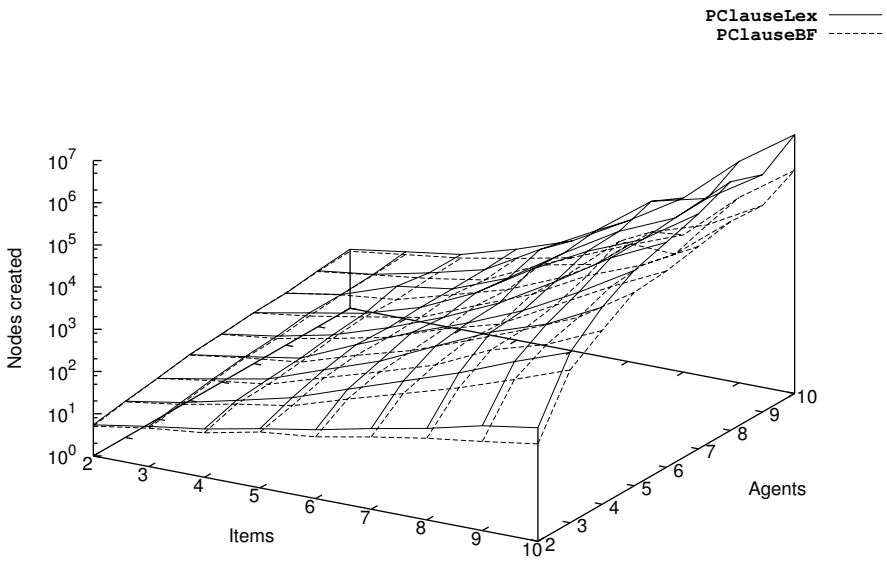


Figure 6.5: Nodes created by `PClauseLex` and `PClauseBF`, averaged over 20 instances of each size, random data.

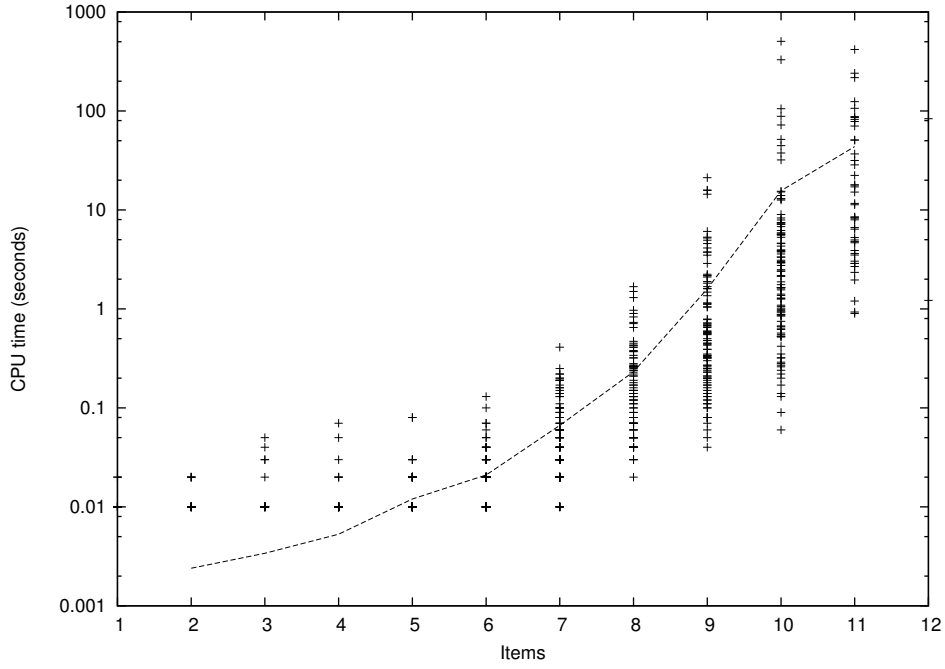


Figure 6.6: CPU time for WDP instances with 20 agents in $\mathcal{L}(pclauses, \mathbb{R}^+, \Sigma)$ using the `PClauseBF` solver, random data.

Our first experiment tested `BruteForce` against `CubeBF` on small instances in order to establish baseline performance. Instances of size $(8, 8)$ which `BruteForce` solves in 70 seconds can be solved by `CubeBF` in only 6 seconds. The impact which the addition of negation to the bidding language has can be seen here: $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ instances of size $(8, 8)$ were solved in less than 0.01 seconds by `PCubeBF`. Positive cubes are evaluated more rapidly in our implementation than general cubes of the same length.[10]

Our second experiment tested the effect of the branching policy on runtime. At each size in $[2, 9] \times [2, 9]$ we solved twenty WDP instances with both `CubeLex` and `CubeBF`, with the results appearing in Figure 6.7. (Here, as elsewhere, we display number of nodes constructed as a proxy for runtime.) Recall that in the case of $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ we found that best-estimate-first branching provides no advantage over lexical branching when using $h^+_\wedge$ as the upper-bound heuristic. Our experimental results show that this is emphatically not the case for $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$ and $h_\wedge$. `CubeBF` builds only 46% as many nodes on average as `CubeLex` does for random instances of size $(9, 9)$—with a corresponding reduction in runtime—and the advantage of `CubeBF` increases with the size of the WDP instance.

Our third experiment fixed the number of agents at twenty and varied the number of goods from 1 to 8. We used `CubeBF` to solve 20 randomly-generated WDP instances of each size, with the results displayed in Figure 6.8. Here it can be seen again how much the addition of negation to the bidding language magnifies the difficulty of WDP: The runtime for `CubeBF` increases tenfold with each additional item, while the marginal cost in runtime of additional items has a much gentler ascent for our `PCubeBF` solver (*cf.* Figure 6.4).

6.7.2 Second Solver and CPLEX

We tested the C++ implementation of our branch-and-bound solver, described above, against CPLEX 10.2 and the IP formulation of the WDP, on data from the principled data generator. The test harness program, `test`, reads input as formatted by `dist2` from standard input and copies the auction instance back to standard output, followed by the results for each solver. For example:

```
[ { 01,9.65404 10,19.5356 11,0.0675854 }
  { 01,10.212 10,20.862 11,1.1696 } ]
0:1 1:1 32.2436 32.2436 5 0
0:1 1:1 32.2436 32.2436 0 0
```

Each result line indicates an allocation of goods to agents, as `item:agent`, followed by the lower and upper bounds achieved by the allocation, the number of nodes built in the search tree, and the number of system clock ticks it took for the

[10] Our `PCube` class must check only two possibilities—present or not—for each atom when evaluating a positive cube, while our `Cube` class must check three—positive, negative, absent.

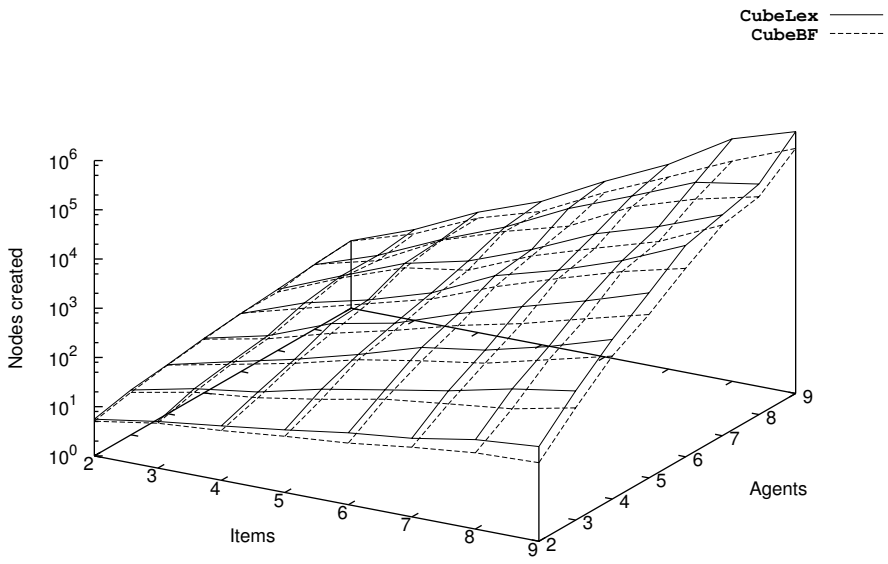


Figure 6.7: Nodes created by `CubeLex` and `CubeBF`, averaged over 20 WDP instances of each size, random data.

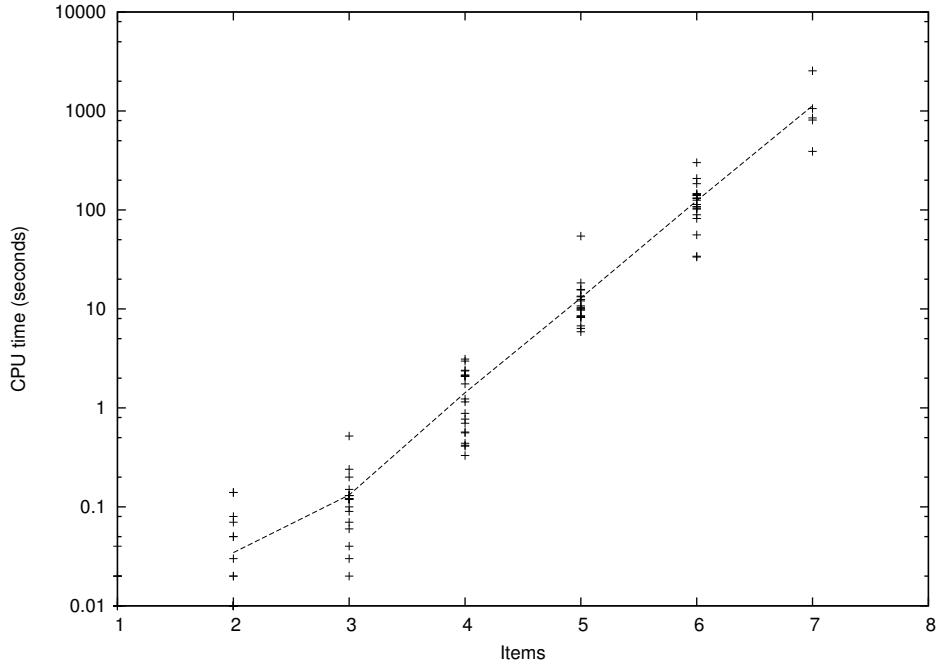


Figure 6.8: CPU time for WDP instances with 20 agents in $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$ using the `CubeBF` solver, random data.

solver to find the given allocation. The first line is the branch-and-bound solver, the second is CPLEX. In all cases, the lower and upper bounds on the value of the allocation should be equal to the actual value of the allocation. For the branch-and-bound solver, this serves as a check on the correctness of the algorithm; for CPLEX, the result it returns is simply printed twice. The number of nodes built during the search is relevant only for the branch-and-bound solver. For CPLEX, this is always given as zero. Finally, the number of clock ticks per second may vary from machine to machine. In the case of the machine we used, each clock tick is one-hundredth of a second. The example given above is so small that it was solved by both solvers in a time below the resolution of the system clock, hence the number of clock ticks for each is zero.

All experiments reported in this section were conducted on a Fedora 10 Linux system, using kernel 2.6.27.12 with a 2.13 GHz Intel Core 2 Duo E6420 CPU and 2 GB of RAM. For the generation of principled data, we used `bins` = 5, `basevalue` = 10.0, and `variance` = 0.05. (See Section 6.6.2 for the meaning of these parameters.) The effect of this is to give us five groups of similarly-valued items, with small but significant synergies among items.

Results for $\mathcal{L}(\boldsymbol{pcubes}, \mathbb{R}^+, \Sigma)$

In the first experiment, we tested the C++ implementation of our `PCubeBF` solver against CPLEX and the IP formulation of the WDP from Section 6.4.3, for $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$. Each solver was run on the same collection of auction instances to permit a direct performance comparison.

As noted above, both the branch-and-bound solver and CPLEX should always find allocations of the same value when confronted with the same problem instance. However, we found that this was not the case here: Sometimes our branch-and-bound solver found an allocation which had a few tenths or hundredths of a point of utility more than the allocation found by CPLEX. After checking several of these by hand, we determined that both solvers were accurately reporting the value of the allocations they had returned, and thus CPLEX was returning a (barely) suboptimal allocation. We suspect that this is caused by imprecision in floating-point arithmetic internal to CPLEX, but at present, we cannot conclusively account for why CPLEX fails in this way. Though our results suggest that the WDP can be solved in its IP formulation by CPLEX much faster than it can be solved by our branch-and-bound solver, we cannot conclude that CPLEX is always faster, specifically because in some cases it fails (barely) to find an optimal allocation.

In our second experiment, we ran the CPLEX solver alone on instances much larger than in the first experiment. As seen in Figures 6.9 and 6.10, the solution time (both average and worst-case) for our branch-and-bound solver was already growing dramatically for small problem sizes, while over the same range the growth of solution time for CPLEX remained low. Using the same data distribution as

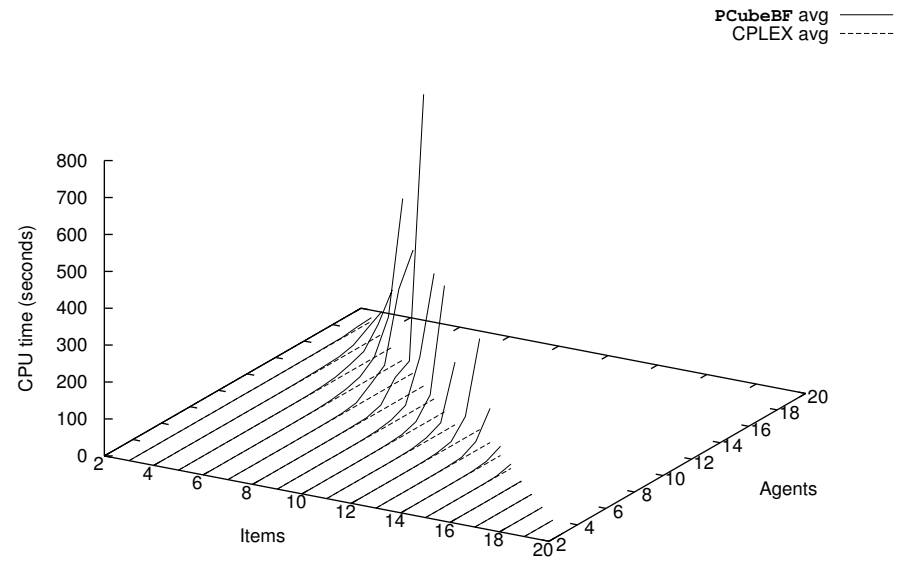


Figure 6.9: Average CPU time for WDP instances in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ using the C++ `PCubeBF` solver and CPLEX, principled data.

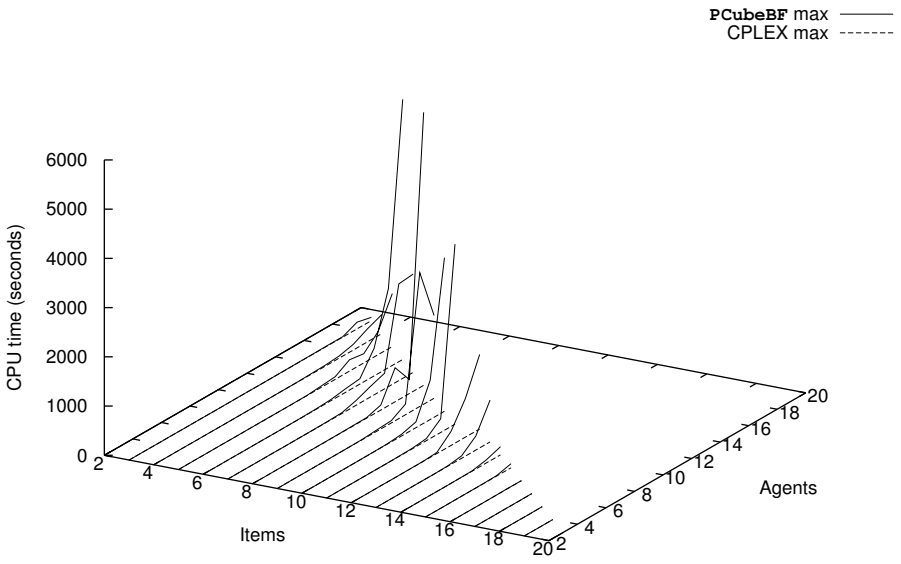


Figure 6.10: Maximum CPU time for WDP instances in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ using the C++ `PCubeBF` solver and CPLEX, principled data.

the first experiment, we solved 100 instances of each size in $[2, 50] \times [2, 50]$. As we approach $(50, 50)$, the maximum solution times have yet to exceed 90 seconds, while average solution times are near 25 seconds. Clearly solution times are beginning to ramp up for CPLEX as well, but at $(50, 50)$ our IP is nowhere near the limit of what CPLEX can handle. There is the possibility that, as in the first experiment with CPLEX, some of the allocations returned were fractionally suboptimal; however, in this case almost all problem instances were too large to solve independently with our branch-and-bound solver, so we have not identified *which* instances were not solved optimally.

Here (as in all experiments we ran) the separation between the best and worst cases is quite dramatic. This indicates that our data distributions generate at least a few trivial instances of all sizes.

6.7.3 Comparison of Solvers

It is immediately clear from comparing results for the Java `PCubeBF` solver on the random distribution in the first experiment with the results for the C++ `PCubeBF` solver on the realistic distribution in the second experiment that something strange is happening: In Figure 6.4, we can see that the Java `PCubeBF` solver was able to solve every $(20, 20)$ instance in less than one second, while from Figure 6.10 we can see that instances of size $(10, 10)$ were already taking so long for the C++ `PCubeBF` solver that it would never have succeeded in solving an $(20, 20)$ instance. What are we seeing here? Three possibilities present themselves:

- The Java `PCubeBF` solver is much better than the C++ version.
- The machine that the Java `PCubeBF` solver ran on is much faster than the C++ version.
- The random data set is much easier than the realistic one.

The first possibility is easily ruled out, as we found that the C++ solver is approximately twice as fast as the Java solver when run on the same machine with equivalent input.[11] The second possibility is also easily ruled out, since the apparently faster solver was run on a (slightly) slower CPU than the apparently slower solver; there is no reason to think that running the C++ solver on a slower CPU would speed it up.

This leaves us with the third possibility, namely that the random instances fed to the Java `PCubeBF` solver are dramatically easier than the realistic instances tackled by the C++ `PCubeBF` solver. In fact, the difference in difficulty is clearly observable in Figures 6.13 and 6.14. These two figures plot, for each instance size, the proportion of instances solved in which a single bidder was allocated all of the

[11] We say *equivalent* here because the two use a different input format, so we cannot give them identical input and expect both to operate properly.

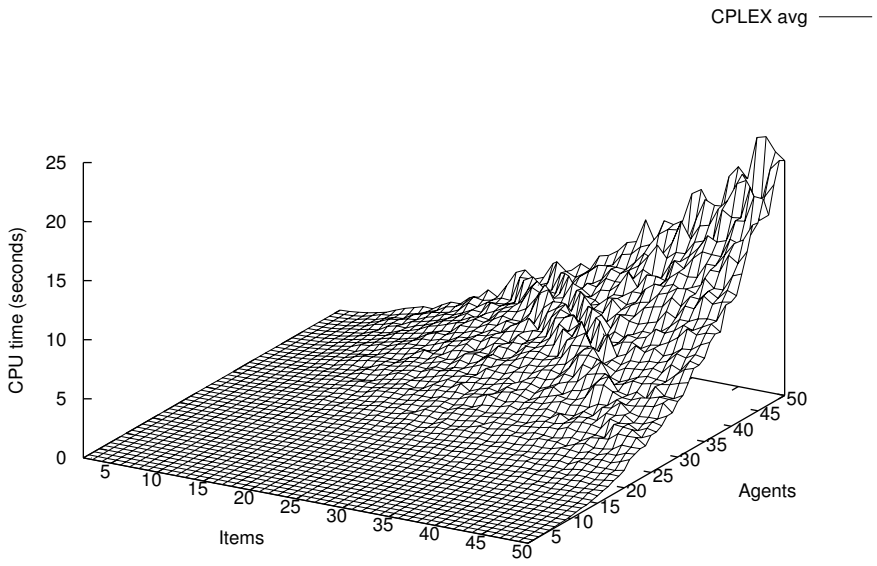


Figure 6.11: Average CPU time for WDP instances in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ using CPLEX, principled data.

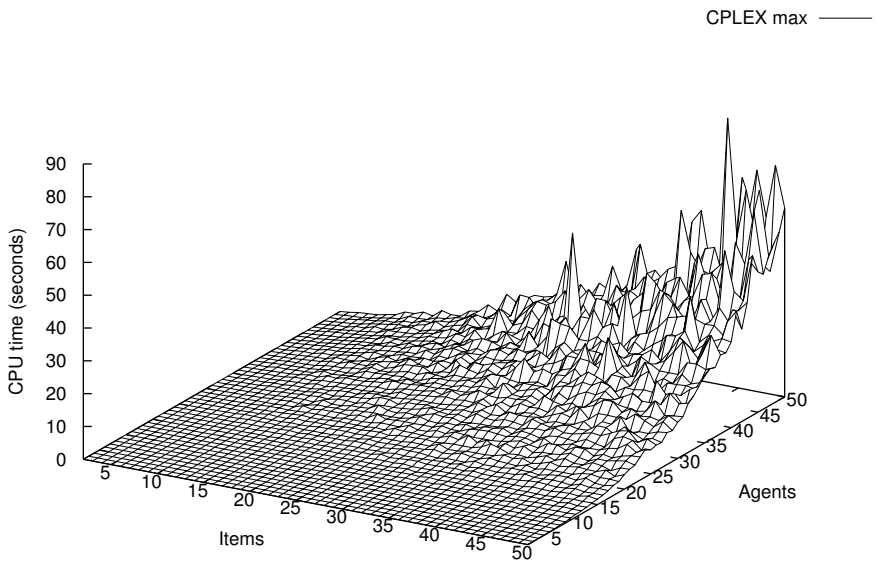


Figure 6.12: Maximum CPU time for WDP instances in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ using CPLEX, principled data.

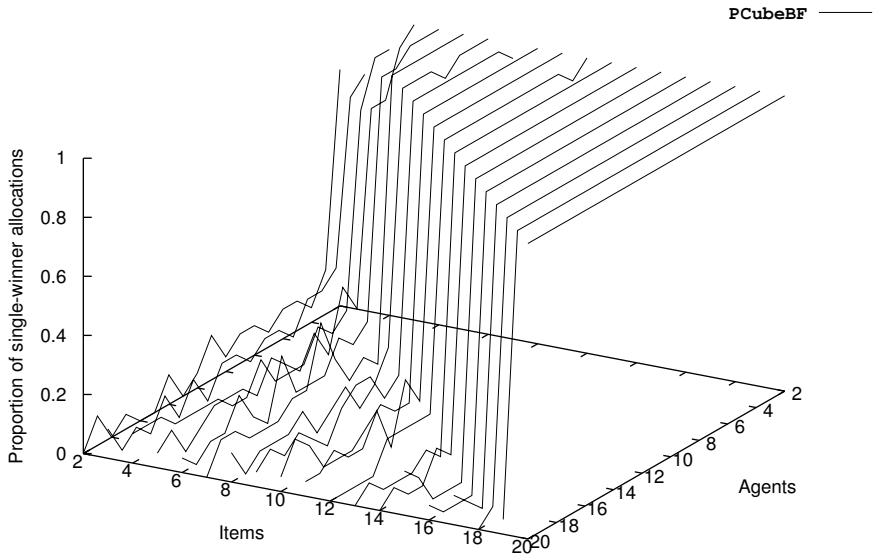


Figure 6.13: Proportion of WDP instances in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ where the optimal allocation found by the Java `PCubeBF` solver awarded all goods to one agent, random data.

goods. For the realistic data, this proportion drops very quickly as the number of items and agents increase, and is almost zero for instances of size $(10, 10)$ or larger.[12] For the random data, until the items began to outnumber the agents it was virtually always the case that all items were awarded to a single agent, and even thereafter a significant portion of the instances had a single winner. In cases where there is a single-winner optimum, this could indicate that only one bidder bid competitively, and in such cases most branches are pruned immediately; this will not be the case when several bidders place similar bids, as can happen with the realistic data.

In Sandholm's survey of WDP algorithms, we find the claim that "[o]n easier distributions ... optimal winner determination scales to hundreds or thousands of items and tens of thousands of bids in seconds" [Sandholm, 2006]. Our best solver for $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$, `PCubeBF`, is capable of solving problems with nearly one hundred items and thousands of bids in under one minute on our random distribution, which, as argued above, appears to be rather easy. Improvements in the performance of `PCubeBF` could likely be obtained by implementing some simple optimizations, such as removing satisfied and falsified formulas from goalbases

[12]The proportions of single-winner allocations were the same for both solvers, which is why only one series of data is visible in Figure 6.14. This is not surprising, as the proportion of single-winner allocations is a property of the input, not of the solver.

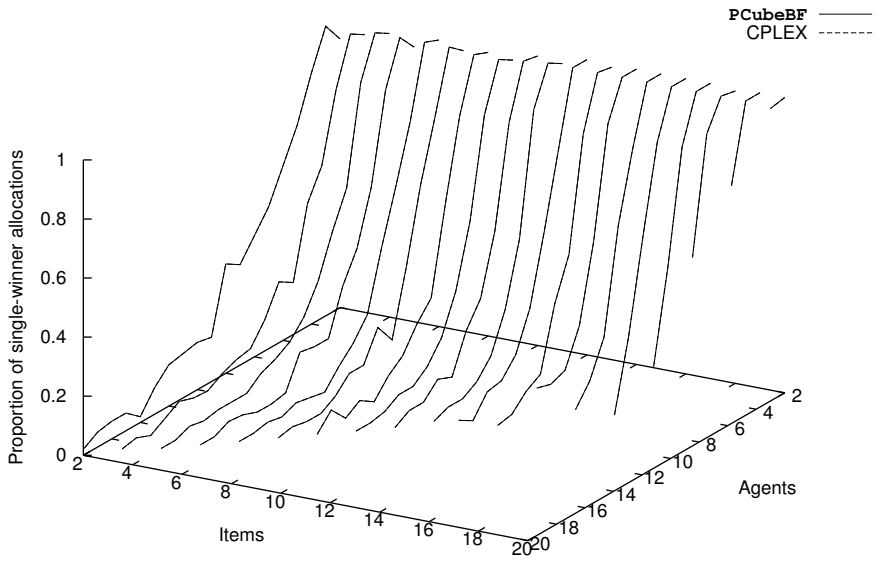


Figure 6.14: Proportion of WDP instances in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ where the optimal allocations found by the C++ `PCubeBF` solver and CPLEX awarded all goods to one agent, principled data.

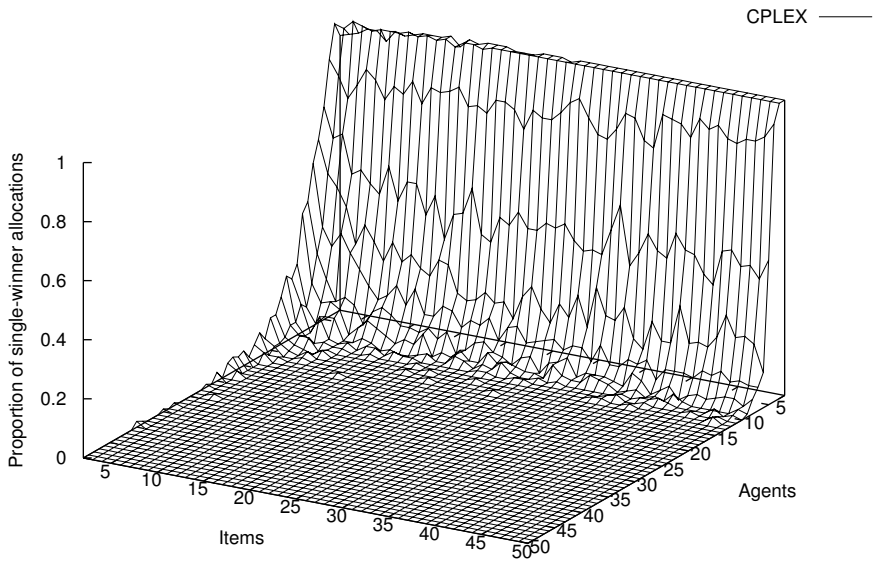


Figure 6.15: Proportion of WDP instances in $\mathcal{L}(pcubes, \mathbb{R}^+, \Sigma)$ where the allocation found by CPLEX awarded all goods to one agent, principled data.

as we build the tree of partial allocations, though optimizations of this sort are unlikely to garner us the several-orders-of-magnitude speedup it would take for `PCubeBF` to be competitive with state-of-the-art solvers. An upper-bound heuristic tighter than $h^+_\wedge$ might deliver better performance, as might the application of other combinatorial optimization techniques.

6.8 Conclusion

In this chapter, we set out to compare our goalbase languages with the OR/XOR family of languages and to demonstrate the feasibility of using goalbase languages for bidding in combinatorial auctions. As expected, neither the OR/XOR/OR* languages nor goalbase languages dominate in terms of succinctness. This tells us that the combinatorial auction designer will want languages from both families in his toolbox. Whether an OR/XOR or goalbase language should be preferred for any particular domain may additionally depend on issues such as cognitive relevance and ease of elicitation, and is a question we leave for future work.

In Section 6.4.4 we presented the branch-and-bound method for solving the WDP, and in Section 6.5 we gave examples of admissible heuristics for several goalbase languages. In Section 6.4.3 we presented a integer programming formulation of the WDP for sum languages. With the WDP written as an IP, we may take advantage of powerful off-the-shelf IP solvers such as CPLEX for finding solutions to the WDP. All of these we proceeded to test for performance on two kinds of data, one random and one generated according to certain principles of realism, found in Section 6.6.1.

The results of our experiments indicate, first, that using goalbase languages for bidding is feasible. The results for CPLEX in Section 6.7.2 show that the natural IP formulation of the WDP for goalbase languages could be used for auctions of moderate size without special tuning or resorting to expert knowledge about CPLEX. An expert in integer programming, CPLEX, or both might well be able to reduce the solution time we have achieved here. The speed of our branch-and-bound solver depends jointly on the tightness of the upper bound heuristic used and the speed with which the upper bound heuristic can be calculated. Tighter heuristics will prune more of the search tree, reducing the number of nodes which must be built before finding a provably optimal allocation. None of our upper-bound heuristics are especially tight, and with some effort could be tightened. What we do not know is where the balance point between gains from better pruning and losses from increased heuristic computation time is; further tightening would surely reduce the number of nodes built, but we do not know if further tightening would reduce computation time. We do know that tightening the upper bound heuristics would enable our solver to tackle larger problem instances, even if doing so would make the solver slower: The limiting factor for our solver on larger instances was not time—we could have allowed the

solver to run longer—but rather memory in which to store the enormous number of nodes it generated. Furthermore, other combinatorial optimization techniques could presumably be applied to obtain even better performance. As our goal was to demonstrate feasibility, not to produce a state-of-the-art solver ourselves, we are happy to leave this work to experts on combinatorial optimization.

Chapter 7

Voting

7.1 Introduction

The problem of electing a committee which satisfies as many voters as possible is one for which good solutions are scarce. There have been numerous attempts to devise methods for committee election, some which have their origins in single-winner voting methods [Brams, Kilgour, and Sanver, 2004, 2006, 2007], others of which are intended to produce outcomes which are proportional in some way [Chamberlin and Courant, 1983; Monroe, 1995]. Single-winner voting systems frequently fail to respect voter preferences when extended to a multi-winner setting, due mainly to the fact that they deny voters the ability to express interdependence among candidates. Moreover, the way in which such systems measure the "representativeness" of committees may not be at all similar to the way in which voters measure it.

In order to tackle the interdependence problem, we propose a voting method which uses goalbases as ballots, in the spirit of combinatorial vote as proposed by Lang [2004].

We begin in Section 7.2 by recalling a bit of voting theory. We present some known methods for committee selection in Section 7.3.1, and then in Section 7.3.2 find fault with these due to their lack of expressive power. In Section 7.4 we will show how the election methods presented earlier can be simulated and extended using goalbases, and in Section 7.5 we consider the computational complexity of finding winning committees using this method. In Section 7.6, we give an example of extending approval voting using goalbases as ballots, and in Section 7.7 we touch on several avenues for further investigation.

7.2 Background

Voting theory, as its name implies, deals with the formal properties of systems of voting. Prior to the 18th century, there were few, isolated attempts to study

voting systems, two of which resulted in the discovery of the Borda count: In the 13th century, Ramon Llull devised an iterative version of it to be used for selecting the abbess of a monastery [Llull, 1926], and in 1433 Nicholas of Cusa proposed it for electing the Holy Roman Emperor [Hägele and Pukelsheim, 2008]. It was not until the two decades prior to the French Revolution that members of the French Academy—Jean-Charles de Borda and Nicolas de Condorcet, in particular—began a sustained and systematic study of voting methods. Since then, there has been extensive investigation of voting rules and their properties. We recount here only as much as we need for the present chapter. For much, much more on the subject, see [Taylor, 2005].

In voting theory, the dramatis personae are a set of *candidates* and a set of *voters*. The voters cast *ballots* indicating their preferences over the candidates; the ballots are fed as input to a *voting rule*, the output of which indicates which candidate or candidates are *winners*. Numerous voting rules have been devised. Here we describe some which may be familiar, and others which figure in our discussion later in this chapter.

Some voting rules solicit relatively little preference information from voters:

unanimity Each voter may cast at most one vote for one candidate. A candidate is a winner iff every voter selects that candidate.

plurality Each voter may cast at most one vote for one candidate. A candidate is a winner iff no other candidate receives more votes.

approval Each voter may cast at most one vote for each candidate. A candidate is a winner iff no other candidate receives more votes.

The unanimity and plurality rules ask voters only to register their top choice, while approval voting permits voters to make no finer distinction than preferred versus nonpreferred. The plurality rule is the familiar first-past-the-post rule used in virtually all elections in the United States. Historically, a complex iterated version of approval voting was used during 1268–1789 by the Venetians to elect their doge [Lines, 1986], and another iterated version was used to elect new popes by the papal conclaves held between 1294 and 1621 [Colomer and McLean, 1998]. Modernly, numerous professional societies[1] have adopted approval voting for their elections, and the Secretary-General of the United Nations is elected by approval voting [Brams, 2007, Sections 1.2–1.4]. Unanimity is not practical for large groups—Poland's Sejm (diet) demonstrated this during the latter half of the

[1] Among them: the Mathematical Association of America (MAA), the American Mathematical Society (AMS), the Institute for Operations Research and Management Sciences (INFORMS), the American Statistical Association (ASA), the Institute of Electrical and Electronics Engineers (IEEE), the Public Choice Society, the Society for Judgment and Decision Making, the Social Choice and Welfare Society, the European Association for Logic, Language and Information, the Game Theory Society, the Econometric Society, and the International Joint Conference on Artificial Intelligence (IJCAI).

17th century [Roháč, 2008]—though a similar rule is frequently used in criminal trials, where the jury or panel of judges are the voters and the "candidates" are *guilty* and *innocent*.

Other voting rules require voters to supply full preference orders over the candidates:

Condorcet Each voter gives a strict linear order over the candidates. A candidate c is a winner iff for each other candidate c', a majority of voters rank c above c' ($c > c'$).

Borda Each voter gives a strict linear order over the candidates. From each ballot, a candidate c receives one point for each other candidate c' above whom he is ranked. A candidate is a winner iff no other candidate scores more points.

The Condorcet and Borda rules incorporate much more of the voters' preference information into their results than do unanimity, plurality, and approval.

The plurality and Borda rules are instances of a class of voting rule known as *positional scoring rules*. A positional scoring rule is one where voters submit strict linear orders and ballots are scored using a scoring vector $\langle s_1, s_2, \ldots, s_{n-1}, s_n \rangle$. A candidate ranked ith by a voter receives s_i points; the winner(s) are the highest-scoring candidate(s). The scoring vector for the plurality rule is $\langle 1, 0, \ldots, 0 \rangle$, while for Borda it is $\langle n-1, n-2, \ldots, 1, 0 \rangle$. We need not require that all voters use the same scoring vector. *General scoring rules* are ones where each ballot induces a score for each candidate, and winners may be determined by summing candidate scores across all ballots. (That is, positional scoring rules use one scoring vector globally, while general scoring rules permit each ballot to have its own scoring vector.) Approval voting is a general scoring rule, but not a positional one; the Condorcet rule is not a scoring rule at all.

The Borda rule does not preserve any intensity information which a voter might provide. Borda treats a voter who strongly prefers candidate a to candidate b the same as one who has a slight preference for a over b. A rule which we will return to later in this chapter, known as cumulative voting, preserves intensity of preference by asking voters to express cardinal rather than ordinal preferences:

cumulative Each voter is given k points which may be distributed among the candidates. A candidate is a winner iff no other candidate scores more points.

Cumulative voting is a general scoring rule, but it is not a positional scoring rule, since each voter is free to choose his own scoring vector. (If we take the ordering of the candidates as fixed by their order of appearance on the ballot, then a cumulative ballot essentially *is* a scoring vector.)

We now move on to some properties of voting rules. The Condorcet rule is rather weak, in the sense that it is easy to devise situations in which it will produce

no winners; however, this means that when a candidate *is* the winner according to the Condorcet rule—known as the *Condorcet winner*—then that candidate is quite strong. By definition, a Condorcet winner would defeat every other candidate in a head-to-head vote. Always electing a Condorcet winner, if one exists, is an intuitively desirable property for a voting rule to have, and so this is one property for which new voting rules are always examined. (Similarly, a Condorcet loser is a candidate who would lose every head-to-head vote, and we would also like for our voting rules never to elect a Condorcet loser.) Unfortunately, many voting rules fail to elect the Condorcet winner in some circumstances: Young [1975] proved that every positional scoring rule will sometimes fail to elect the Condorcet winner.

Other properties of interest are resoluteness, anonymity, neutrality, monotonicity, unanimity, non-imposition, Pareto, and strategyproofness. A rule is resolute if it always chooses a single winner. A rule is anonymous if all voters are treated the same; a rule is neutral if all candidates are treated the same. A rule is monotone if winners continue to be winners if their ranking on some ballot improves. A rule is unanimous if, when all voters have the same candidate as their first choice, that candidate wins; a rule is non-imposing when every candidate has some configuration of ballots which would cause him to win. A rule is Pareto if there is never a candidate which all voters prefer to the winner. A rule is strategyproof if voters have no incentive to misrepresent their preferences; rules which are not strategyproof are said to be manipulable.

Finally, we mention one more voting rule, one which plays a much larger role in voting theory than most voting theorists would like:

dictatorship Each voter may cast at most one vote for one candidate. A candidate is a winner iff the voter predesignated as the dictator votes for that candidate.

It is an unfortunate fact that the preponderance of results in voting theory are negative, sometimes of the form:

> *Any voting rule which satisfies desirable properties $X_1, \ldots, X_k$ when there are at least three candidates is a dictatorship.*

Famous instances of this include Arrow's Theorem [Arrow, 1970] and the Gibbard-Satterthwaite Theorem [Gibbard, 1973; Satterthwaite, 1975]: For Arrow, the properties are Pareto and independence of irrelevant alternatives;[2] for Gibbard-Satterthwaite the properties are resoluteness, non-imposition, and strategyproofness.

There are many other voting rules not mentioned here, as well as many other properties of interest. For a thorough overview of voting rules, see [Brams and Fishburn, 2002].

[2]While Arrow's Theorem is usually stated for social welfare functions, the version for voting rules is equivalent. See [Taylor, 2005, Section 3.4].

7.3 Multi-Winner Elections

The voting rules mentioned in the previous section are generally intended to be used for electing single winners, despite that some of them will occasionally produce ties. Less studied are voting rules intended for the election of multiple winners. In this section, we discuss some of the challenges associated with multi-winner elections.

7.3.1 Some Methods for Committee Election

Various methods for committee elections have been proposed. (For a general discussion of the difficulties of committee elections, see [Chevaleyre, Endriss, Lang, and Maudet, 2008b] and [Lang and Xia, 2009].) The naïve (and perhaps for that reason, most widely used) approach is an extension of the single-vote plurality method. With k seats to fill from a slate of n candidates, each voter may cast up to k votes, no more than one vote per candidate, and the top k candidates win. A similar naïve extension of approval voting to a multi-winner setting is possible: Again with k seats to fill from a slate of n candidates, each voter may cast up to n votes, no more than one per candidate, and again the top k candidates win. These two methods lie along a spectrum of voting methods where the maximum number of votes cast per voter is varied—the approval version anchoring one end, and a single-vote top-k method anchoring the other. We now give a formal definition:

Definition 7.3.1 (m-vote, top-k)**.** Call a voting method *m-vote* if each voter may cast single votes for up to m candidates, and *top-k* if the k candidates receiving the most votes are the winners.

Standard plurality voting as used in many elections for public office is a 1-vote top-1 method.

Top-k methods all share a feature which makes them rather unsuitable for committee elections, namely that they tend to quash minority representation.

Theorem 7.3.2. *If there are v voters in an m-vote top-k election, then a coordinated block of $\left\lfloor \frac{v}{2} + 1 \right\rfloor$ voters is sufficient to dictate the top m candidates.*

Proof. Strategy: $\left\lfloor \frac{v}{2} + 1 \right\rfloor$ voters cast votes for the same m candidates, $c_1, \ldots, c_m$. Result: Each c_i must receive at least $\left\lfloor \frac{v}{2} + 1 \right\rfloor$ votes, and no other candidate can receive more than $\left\lceil \frac{v}{2} - 1 \right\rceil$ votes, thus ensuring that $c_1, \ldots, c_m$ are the top m vote-getters. □

(Note that while the coordinated block of voters *can* dictate the top m candidates, it *cannot* dictate order among them: If the majority block skimps on votes for one of its candidates and the minority block is also coordinated, one of the majority's m candidates could tie with one or more candidates receiving votes

only form the minority block. Hence the votes which determine the order among the top m all come from voters outside of the majority block.)

If representation of minority views on a committee is important, this fact displays a flaw in m-vote top-k committee voting: When $m \geq k$, a majority block essentially has veto power over candidates. Only candidates supported by the majority block will receive seats. When $m < k$, the majority block are guaranteed to have all of their m candidates on the committee, which may allow voters outside that block to succeed in electing candidates as well, but setting the number of votes each voter may cast to be less than the number of seats available has its own drawbacks in terms of permitting voters to express their preferences. The fewer votes a voter is permitted to cast, the less information is gathered from his preference order by the voting method, and moreover, we run the risk of collecting misleading preference information. For example, if we want to fill five seats but permit only three votes per voter (i.e., we are using a 3-vote top-5 method), then it is hard to predict how voters will behave. Will a voter cast a ballot for the three candidates he thinks best, or for the three-candidate subcommittee he thinks best, or for something else? It's easy to envision a situation in which a voter's preferred three-candidate subcommittee is not a part of the same voter's preferred five-candidate full committee.

We must not lose sight of the fact that committees are not just elected, but elected *for some purpose*. Often, a committee—rather than an individual—is chosen to carry out some task so that the diversity of views held by the committee members may be brought to bear on the given task, or in order to have a decision-making body which is representative of the voters as a whole, and not just some subset of them. An m-vote top-k voting method will fail to produce a diverse or representative committee in the face of a coordinated majority block unless that majority block is itself committed to producing a diverse or representative committee. As we cannot hope for this in all but the most collegial circumstances, if we want a diverse or representative committee, then we should not elect our committee using an m-vote top-k method.

We have seen that outcomes of m-vote top-k elections are dictated by majority blocks regardless of what m and k are. The alternatives which we will now consider vary the winning criterion instead of the number of votes each voter may cast. Brams et al. [2004] describe what is known as the "minisum" method. Voters cast ballots as in approval voting, but we do not declare the top k vote-getters to be the winners—candidates do not win individually. Instead, winning committees are ones which *mini*mize the *sum* of the Hamming distances to the votes cast, hence the name "minisum". More precisely:

Consider each ballot as a binary vector $b_1 \ldots b_n$, where $b_i = 1$ if the voter casting the ballot approves of candidate i and $b_i = 0$ otherwise. The Hamming distance H between two ballots is the least number of bits which must be flipped to transform one ballot into the other. (For example, $\mathrm{H}(01\underline{010}, 01\underline{101}) = 3$.) Thus

we can state the minisum rule as

$$c \text{ is a winner} \iff \forall c' \in C\colon \sum_{b\in B} \mathrm{H}(c,b) \leq \sum_{b\in B} \mathrm{H}(c',b),$$

where C is the set of k-seat committees and B is the multiset of ballots cast, both with their members expressed as binary vectors. (Here, we say $c \in C \ldots$ because the minimum need not be unique.)

Intuitively, any winning committee is one which is as similar in membership as it can be to as many ballots as it can be. However, this intuition is wrong. Brams et al. [2007, Appendix, Proposition 4] show that any k-candidate minisum solution will consist of the k candidates receiving the most approval votes. Hence, we have the following:

Theorem 7.3.3. *The voting methods m-vote k-minisum and m-vote top-k are equivalent.*

Thus the minisum method, despite that it *sounds* more accommodating, is no more promising for electing representative committees than any of the m-vote top-k plurality methods are.

Brams et al. [2007] suggest minimax as an alternative to minisum. Rather than selecting committees which minimize the sum of Hamming distances to the ballots, minimax minimizes the maximum Hamming distance to any ballot. Formally, the minimax rule is

$$c \text{ is a winner} \iff \forall c' \in C\colon \max_{b\in B} \mathrm{H}(c,b) \leq \max_{b\in B} \mathrm{H}(c',b),$$

where, as before, C is the set of k-seat committees and B is the multiset of ballots cast, both with their members expressed as binary vectors.

The intuitive effect of minimax as a winning criterion is that it antagonizes outliers the least; or, rather, it antagonizes the farthest outlier the least. This gives rise to the following feature of the pure minimax criterion: The exact number of voters casting any particular ballot is irrelevant to the outcome; only *whether* particular ballots are cast matters. If one incorrigible voter casts the ballot 00101 while all others choose 11101, then it makes no difference if there are two, ten, or a million ballots cast in total—the winning three-member committees are 10101 and 01101. Usually this is not a desirable feature, though it arguably is appropriate in some circumstances (e.g., multilateral treaty negotiation, as described by Brams et al. [2004]). Brams et al. [2007] suggest a useful refinement which avoids this problem without reintroducing the tyranny of the majority, namely the addition of *proximity weights*.

The proximity weight w_b of a ballot b is defined as

$$w_b = \frac{m_b}{\sum_{b'\in B} m_{b'}\, \mathrm{H}(b,b')},$$

where m_b is the number of voters casting ballot b. The minimax criterion with proximity weights becomes

$$c \text{ is a winner} \iff \forall c' \in C\colon \max_{b \in B} w_b \, \mathrm{H}(c, b) \leq \max_{b \in B} w_b \, \mathrm{H}(c', b).$$

Weighting the Hamming distance to a ballot by its proximity to other ballots diminishes the influence of outliers while at the same time reintroducing the proportionality which pure minimax lacks.

Many other weightings are possible, but these are representative, and sufficient to illustrate our point in the next section.

7.3.2 Similar Committees Need Not Be Similarly Preferable

As Brams et al. [2006, pp. 83–84] say, '[w]e view the problem of identifying the most representative committee as that of identifying the subset that is "closest" to the collection of subsets specified by the voters.' In m-vote top-k methods, proximity is tied to support of individual candidates; the minisum and minimax criteria equate proximity with average and maximum Hamming distance, respectively. Taking the pure minimax criterion as our example, it is not hard to see that the *intent* is to minimize the dissatisfaction of the farthest outlier, while the *rule* is to minimize the dissimilarity between the farthest outlier's ballot and the winning committee. But why should we suppose that the farthest outlier (or any voter, for that matter) actually cares about his ballot's similarity to the winning committee? As we shall see now, it is quite reasonable to think that for many voters their committee preferences will not track the Hamming distance at all.

Taking the Hamming distance as a measure of similarity, we have the following two properties: If c is a voter v's preferred committee, then

- any substitution of n members in c is strictly better according to v than every substitution of m members, for $n < m$, and
- all committees c' which are Hamming-equidistant from c are equally preferred by v,

both of which are dubious when applied to voters electing real committees.

For example, suppose that we are electing a three-seat committee from the five candidates Alice, Bob, Charlie, Dave, and Elaine. Suppose further that one of the voters believes that

- Alice and Bob are the best candidates, so any committee with one of them is better than any committee with neither,
- Alice and Bob will fight if they are on the committee together, so any committee with both is worse than any committee with neither,

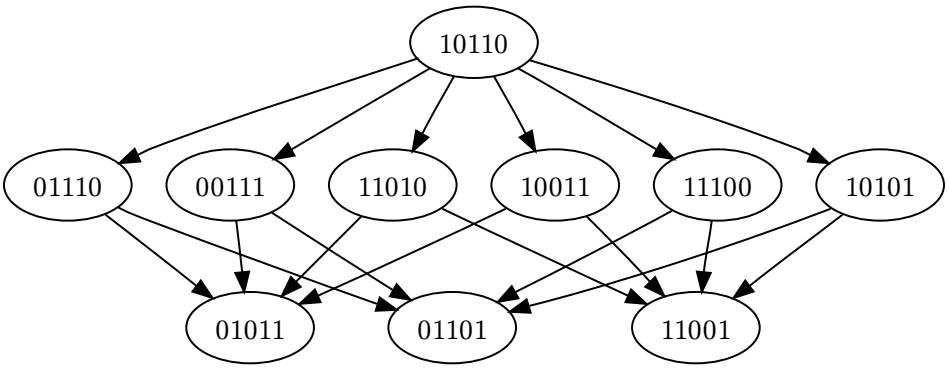


Figure 7.1: Order on ballots induced by Hamming distance from 10110.

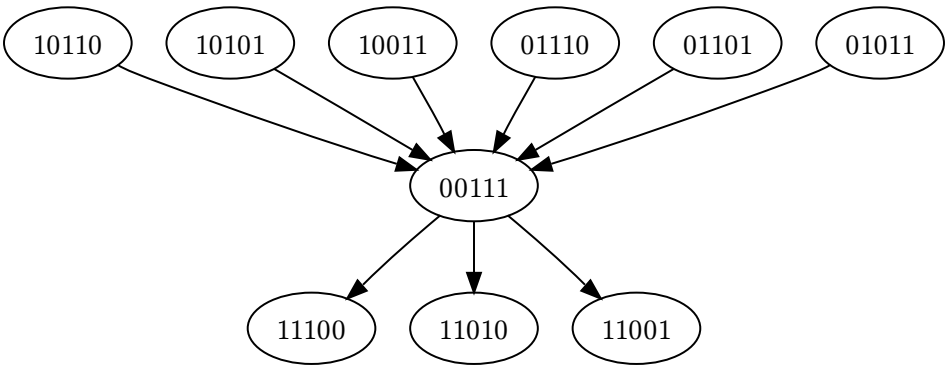


Figure 7.2: A more realistic order on ballots.

and that this voter is otherwise indifferent among potential committees.

Thus our voter ranks the committees in preference order as

$$ACD, ACE, ADE, BCD, BCE, BDE > CDE > ABC, ABD, ABE$$

as seen in Figure 7.2. This preference ordering is sensitive to small changes in committee composition. Each of the best committees is only one substitution away from some worst committee. (Notice that this is neither due to the size of the committee nor to the number of candidates, but to the way that two candidates interact in our example voter's preferences. We use a three-seat committee with five candidates only to keep the example manageable.) Put another way, the Hamming distance between some pairs of best and worst committees is 2, which is always the minimum Hamming distance between two committees of the same size.[3] From our voter's point of view, $ACD \sim BDE > ABC$; but $\mathrm{H}(10110, 01011) = 4$ while $\mathrm{H}(10110, 11100) = 2$, and so the ordering induced by the Hamming distance from ACD is $ACD > ABC > BDE$, thus putting an optimal committee last and one of our voter's least favored committees in second place. (*Cf.* Figures 7.1 and 7.2.) If we use a minisum or minimax procedure and have many voters with preferences like this one, we risk an outcome that is similar in composition to voters' first choices, yet is widely disliked.

The problem we have identified is that the question of committee membership for one candidate is not necessarily independent of the question of committee membership for some other candidate. In the language of utility functions, some voters have nonmodular preferences. (Benoît and Kornhauser [1991, 1994, 2006] identify a similar problem with the election of representative assemblies—not only might a voter have complex preferences over the composition of the assembly, but preferences over candidates for his district might depend on or involve candidates for other districts where he isn't even able to cast a vote.) It could be argued that the problem is caused not by the voting methods we examined, but rather because of the way they are applied: The candidates are individuals rather than committees. If committees were raised to the status of first-class citizens—that is, if voters were to vote for whole committees rather than for individuals—perhaps we would not have this problem. However, this approach is unhelpful implemented one way, and scales poorly implemented another. Brams et al. [2007] report that the 2003 Game Theory Society election filled 12 seats from a slate of 24 candidates, giving 2704156 possible committees for voters to consider. This is still manageable if the voter is queried for his most preferred committee only—but then we are left with no information about candidate interdependence, and so we are no better off than before. It is also likely that no two voters will share the same committee

[3] If c, c' are distinct k-member committees, then the least difference there could be is that c' is c but with one member replaced. Since committees are represented as bit vectors, a single-member substitution turns one 'on' bit off and one 'off' bit on. Hence the Hamming distance between two distinct committees is always even and at least 2.

as their first choice, so the result will be a many-way tie. If we ask voters for rankings, we begin to sink into the combinatorial morass: Voters would balk at ranking their top 0.001% of the possible committees let alone all 2.7 million of them, and even if the voters were able to rank all of the possible committees the vote tabulator would be overwhelmed by the data.

A more general argument for the unsuitability of single-candidate voting systems for electing committees can be given, by way of comparing how many voter profiles single-candidate voting methods can accommodate. The most expressive voting method considered above is approval voting, where every subset of candidates is a valid ballot. (All other methods mentioned restrict the set of valid ballots to a proper subset of the powerset of candidates.) In comparison, there are

$$\sum_{i=1}^{\binom{n}{k}} \sum_{j=1}^{i} (-1)^{i-j} \binom{i}{j} j^{\binom{n}{k}}$$

distinct voter profiles over k-seat committees chosen from n candidates, which, for any useful value of n, dwarfs the 2^n distinct approval ballots.[4] Taking the Game Theory Society election as our example, the largest term in the sum is $\binom{24}{12}^{\binom{24}{12}} = 2704156^{2704156}$, which is rather large.[5] Clearly, we need a different approach.

7.4 Simulating Voting Methods Using Goalbases

Many common election methods may be simulated by using goalbases as ballots, summing them to get a single goalbase, and then using some method for finding optimal models over the resulting goalbase.

Recall the goalbase summation operator $\oplus$ (see Definition 2.2.10). Suppose that there are voters $1, \ldots, n$. Then we can straightforwardly simulate the following voting methods:

- **plurality:** Each voter casts a single vote, with the candidate(s) receiving the most votes as the winner(s). Define a goalbase $G_i = \{(c_j, 1)\}$ for each voter i, where c_j is the candidate for which i casts his vote. Then find an optimal model for $G_1 \oplus \ldots \oplus G_n$ considering single-atom models only. Alternatively, let $G_i = \{(c_j \wedge \bigwedge_{j \neq k} \neg c_k, 1)\}$ and place no constraint on models.

[4] A set of size n may be partitioned into k nonempty subsets $\left\{ {n \atop k} \right\}$ ways (where $\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=1}^{k} (-1)^{k-j} \binom{k}{j} j^n$ denotes a Stirling number of the second kind [Graham, Knuth, and Patashnik, 1994, Section 6.1]), and in each case these k subsets may themselves be strictly ordered in $k!$ ways. Thus the number of (not necessarily strict) linear orders on n items is $\sum_{k=1}^{n} k! \left\{ {n \atop k} \right\} = \sum_{k=1}^{n} \sum_{j=1}^{k} (-1)^{k-j} \binom{k}{j} j^n$.

[5] For comparison, Haub [2002] estimates that 106 billion people had ever lived as of the year 2002. The profile space is more than adequate to permit every person who has ever lived a unique opinion on the 2003 Game Theory Society committee election.

- **unanimity:** Each voter casts a single vote. Any candidate receiving all n votes is the unique winner; otherwise, all candidates tie. Define G_i as for plurality, and find an optimal model having n utility, considering single-atom models only.

More generally, we can simulate the following parametrized voting rule:

- ***m*-vote top-*k*:** Each voter casts up to m votes. The k candidates receiving the most votes win. Define the goalbase $G_i = \{(c, 1)\}_{c \in V_i}$ where V_i is the set of m or fewer candidates favored by voter i. Find an optimal model for $\bigoplus_i G_i$, considering k-sized models only.

Many other election procedures mentioned by Taylor [2005], such as near-unanimity, omninomination, dictatorship, and oligarchy, may also be simulated in this fashion. In fact, we will now show that any *positional scoring rule* may be simulated using goalbases as ballots.

First, we need some notation for referring to (sets of) optimal models.

Definition 7.4.1 (Optimal Models)**.** Given a goalbase G, define $\mathrm{opt}(G)$ and $\mathrm{opt}_k(G)$ to be the sets such that

$$\mathrm{opt}(G) = \operatorname*{argmax}_{M \subseteq \mathcal{PS}} u_G(M), \qquad \mathrm{opt}_k(G) = \operatorname*{argmax}_{\substack{M \subseteq \mathcal{PS} \\ |M|=k}} u_G(M).$$

Clearly,

$$\mathrm{opt}(G) = \mathrm{argmax}\Big\{ u_G(M) \;\Big|\; M \in \bigcup_{k=0}^{|\mathcal{PS}|} \mathrm{opt}_k(G) \Big\},$$

because each model in the set of optimal models must also be an optimal model among the models of its own size. Note also that the problem of generating some member of the set $\mathrm{opt}(G)$ is an instance of the function problem corresponding to the decision problem MAX-UTIL (see Definition 5.3.1).

The next theorem shows that every single-winner positional scoring rule may be simulated by casting goalbases as ballots and finding the set of utility-maximizing models:

Theorem 7.4.2. *Let V be a single-winner (possibly nonresolute) positional scoring rule with scoring vector $\langle s_1, \ldots, s_m \rangle$. Let $b_1, \ldots, b_n$ be a sequence of ballots where $\mathrm{rank}_i(j)$ is the rank given to candidate j (denoted by c_j) by ballot b_i. Let $G_i = \{(c_j, s_{\mathrm{rank}_i(j)}) \mid 1 \leq j \leq m\}$. Then $V(b_1, \ldots, b_n) = \mathrm{opt}_1(\bigoplus_{i=1}^n G_i)$.*

Proof. Let $S(x)$ be the score of candidate x under rule V with ballots $b_1, \ldots, b_n$. If $x \in V(b_1, \ldots, b_n)$, then since V is a positional scoring rule, this implies that $S(x) \geq S(y)$ for all $y \in A$. Similarly, if $\{x\} \in \mathrm{opt}_1(\bigoplus_{i=1}^n G_i)$, then $u_{\bigoplus_{i=1}^n G_i}(\{x\}) \geq u_{\bigoplus_{i=1}^n G_i}(\{y\})$ for all singleton models $\{y\}$. Finally, observe that for all $x \in A$, $S(x) = u_{\bigoplus_{i=1}^n}(\{x\})$. □

Finding the set of utility-maximizing models is related to functional version of MAX-UTIL seen in Chapter 5. Though in general this is NP-complete, determining winners for positional scoring rules is always in P, so there is clearly no complexity-theoretic point to be made here. (The class of goalbases corresponding to positional scoring rules represents only modular utility functions.) Rather, what is noteworthy is that if a voting rule can be simulated using goalbases as ballots, then that voting rule can be *extended* by loosening the restrictions we imposed in order to simulate it.

Though we have defined ballots for positional scoring rules to be total preorders, we could also have defined them cardinally. Suppose that each voter were given a supply of *points* which he may assign to the candidates as he wishes, and as with positional scoring rules, the winners are the set of candidates receiving the maximal number of points. This voting rule is known as *cumulative voting*. Any positional scoring rule may be seen as a special case of cumulative voting, wherein the voters are not given free reign as to the assignment of points, but rather required to award points only in predefined, indivisible blocks. (For example, the plurality rule gives each voter a single, indivisible one-point block of votes, while the Borda rule gives each voter blocks of size $m, m-1, \ldots, 1$ when there are m candidates.) Positional scoring rules are able to use total preorders as ballots because each voter has the same scoring vector; hence, any positional scoring rule can use cardinal ballots simply by moving the rule's scoring vector into the ballot in this way.

Fact 7.4.3. *If no restrictions are placed on the ballot goalbases G_i, then* $\mathrm{opt}_1(\bigoplus_{i=1}^{n} G_i)$ *corresponds to cumulative voting without point limits.*

Cumulative voting without point limits is not a practical voting method, since for the voters it is equivalent to playing the game of which voter can write down the largest number. However, if we restrict the weights in the ballots G_i to some closed interval of the nonnegative reals, then we have the following correspondence:

Fact 7.4.4. *If each $G_i \in \mathcal{L}(\mathit{forms}, [0, m], \Sigma)$ and $\Sigma_{(\varphi,w)\in G_i} w \leq m \in \mathbb{R}^+$, then* $\mathrm{opt}_1(\bigoplus_{i=1}^{n} G_i)$ *corresponds to m-vote cumulative voting.*

Note that it is essential that the interval from which weights are chosen is closed rather than open on the right—otherwise, the voters are again playing the write-the-largest-number game, this time approaching m rather than infinity.

Many undesirable properties of voting rules—the failure to always elect a Condorcet winner, the possibility of electing a Condorcet loser, nonmonotonicity—are existence properties: A voting rule has them by virtue of some set of permissible ballots for which the rule yields a pathological result. Cumulative vote contains every positional scoring rule, in the sense that any collection of ballots which is permissible input for some positional scoring rule is also permissible input for cumulative voting, and on those ballots both rules will generate the same outcome.

As a result, a collection of ballots which is pathological for some positional scoring rule will yield the same pathological result under cumulative voting. For example: Borda ballots are legal cumulative ballots and Borda can fail to elect the Condorcet winner, so if all voters happen to submit ballots which would cause this defect under the Borda rule, then the same result will occur with those ballots under the cumulative voting rule. There is a minor subtlety here, in that what we are holding constant when comparing positional scoring rules with cumulative voting is the ballots cast, rather than the voters' preferences. It might well be the case that the same group of voters would not submit identical ballots under, say, Borda, and cumulative voting, due to the less constrained ballot space which the latter affords.

Though cumulative voting lacks some desirable properties, it also has a much larger ballot space than its subrules. How this affects the likelihood of encountering pathological ballot profiles in practice is unknown. Finally, the fact that cumulative voting can fail to elect the Condorcet winner does not obviously preclude there being some anonymous subrule which does always elect the Condorcet winner. That subrule cannot be a positional scoring rule, as proved by Young [1975], but there are many subrules of cumulative voting which are not positional scoring rules; we have not yet eliminated the possibility that some subrule of cumulative voting is a Condorcet rule. Whether any such rule exists we leave for future investigation.

7.5 The Complexity of Deciding Winning Slates

Determining the winner of an election where goalbases are ballots is related to solving MAX-UTIL for the sum of those goalbases. MAX-UTIL for languages with straightforward definitions tends to be either trivial or NP-complete. (For a thorough treatment of the complexity of MAX-UTIL, see Section 5.5.)

Because we are concerned here with electing committees of a size fixed prior to the election (as opposed to the open-ended committees discussed by Brams et al. [2007]), we cannot apply MAX-UTIL directly to the sum of voters' goalbases in order to determine the winners of the election. Doing that might yield a model with the wrong number of winners. We must do something to ensure that only models which fill k seats are potentially optimal. One approach to adapting our winner determination problem to MAX-UTIL is to augment the sum of the voters' goalbases with formulas which increase the utility of k-sized models (or decrease the utility of non-k-sized models).

First, some notation is required. Define the following formulas:

$$\varphi_{\geq k} = \bigvee \left\{ \bigwedge X \;\middle|\; X \subseteq \mathcal{PS} \text{ and } |X| = k \right\}$$

$$\varphi_{\leq k} = \neg \varphi_{\geq k+1}$$

$$\varphi_{=k} = \varphi_{\leq k} \wedge \varphi_{\geq k}$$

and the quantity

$$\delta = \sum \Big\{ |w| \;\Big|\; (\varphi, w) \in \bigoplus_i G_i \Big\}.$$

The formula $\varphi_{=k}$ is such that a model $M \models \varphi_{=k}$ iff $|M| = k$. The quantity δ is a (not necessarily tight) upper bound on the utility change between arbitrary models for $u_{\bigoplus_i G_i}$. Note that $\varphi_{\geq k}$ has $\binom{n}{k}$ disjuncts, and so is potentially a very long formula.[6] However, in the context of committee elections n and k—the numbers of candidates and seats—will tend to be small, and, as will be seen below, $\varphi_{=k}$ appears exactly once in the goalbase which represents the voters' preferences.

Suppose that $\bigoplus_i G_i$ is the sum of voter goalbases in a k-seat committee election. Let

$$G = \Big(\bigoplus_i G_i \Big) \oplus \{(\varphi_{=k}, \delta + 1)\}.$$

Since δ is an upper bound on utility change between models for $u_{\bigoplus_i G_i}$, we can say the following: If M, N are models such that $M \models \varphi_{=k}$ and $N \not\models \varphi_{=k}$, then $u_G(M) > u_G(N)$, as the greatest possible utility loss of moving from N to M in $u_{\bigoplus_i G_i}$ is δ, and making $\varphi_{=k}$ true results in a gain of $\delta + 1$. Thus, since any model of size k is strictly better than every model of any other size, we are guaranteed that all models which yield maximal utility are of size k. Moreover, since $\varphi_{=k}$ is true on *every* model of size k, it does not affect their utility relative to one another, so augmenting $\bigoplus_i G_i$ with $(\varphi_{=k}, \delta + 1)$ preserves the ordering of (relevant) models.

Therefore, we may easily adapt the input to force size-k models to the top of the ordering and use an off-the-shelf algorithm for deciding MAX-UTIL($\mathit{forms}, \mathbb{R}, \Sigma$) to determine winners—though this may be impractical due to the complexity of MAX-UTIL($\mathit{forms}, \mathbb{R}, \Sigma$). If $\bigoplus_i G_i$ is confined to something less than the full language, however, we may be able to use that to our advantage. The following theorem shows that when solving MAX-UTIL we may always reduce a goalbase outside a given language $\mathcal{L}$ to a goalbase inside the language by solving a simpler version of MAX-UTIL at most an exponential number of times:

Theorem 7.5.1. *If $G \in \mathcal{L}$, $G' \notin \mathcal{L}$, and $\mathcal{L}$ is closed under substitution of logical constants for atoms, then* MAX-UTIL *for $G \oplus G'$ may be solved with no more than $2^{|\mathrm{Var}(G')|}$ calls to a* MAX-UTIL *oracle for $\mathcal{L}$.*

[6] While it is not possible to shorten $\varphi_{\geq k}$ using standard Boolean connectives, we can write it more concisely if we are willing to augment our language with a cardinality operator. For example, Benhamou, Sais, and Siegel [1994] consider a variant of propositional logic in which there are *pair formulas* $(\rho, \mathcal{L})$, where $\mathcal{L}$ is a multiset of literals and ρ specifies how many elements of the multiset must be true in order for the $(\rho, \mathcal{L})$ to be true. Clearly $\left(\frac{|\mathcal{PS}|}{2}, \mathcal{PS}\right)$ is equivalent to $\varphi_{\geq |\mathcal{PS}|/2}$, but exponentially shorter. Hoos and Boutilier [2000] propose a similar, though less powerful, k-of operator—less powerful due to the fact that their (bidding) language lacks negation, and so any k-of operates on atoms only.

Proof. There are $2^{|\mathrm{Var}(G')|}$ models on just the variables occurring in formulas in G'. For each model over the variables in G', we substitute $\top$ and $\bot$ into $G \oplus G'$ as per the model and carry out MAX-UTIL on the modified $G \oplus G'$. □

If $\mathrm{Var}(G')$ is small and does not depend on $\mathcal{PS}$, decomposing a goalbase containing alien formulas in this way is potentially feasible. However, the formula $\varphi_{=k}$ contains every atom in $\mathcal{PS}$ at least once and hence the upper bound we get is exponential in $|\mathcal{PS}|$, which is unhelpful. (For a discussion of the substitution closure condition, see Section 5.7.1.)

An alternative approach is to modify the decision problem instead of the goalbase. Perhaps MAX-UTIL is not the right decision problem unless we have the same number of candidates as seats—in which case, why vote? Instead, we define a variant of MAX-UTIL where exactly k atoms must be true in any solution:

Definition 7.5.2 (k-MAX-UTIL)**.** The decision problem k-MAX-UTIL(Φ, W, F) is defined as: Given a goalbase $G \in \mathcal{L}(\Phi, W, F)$ and an integer K, is there is a model $M \in 2^{\mathcal{PS}}$ such that $u_G(M) \geq K$ and $|M| = k$?

k-MAX-UTIL is the decision-problem version of finding members of opt_k, just as MAX-UTIL is the decision-problem version of finding members of opt.

Fortunately, having a fixed number of seats we are trying to fill dramatically reduces the complexity of finding a voter's preferred ballot:

Theorem 7.5.3. k-MAX-UTIL$(\mathit{forms}, \mathbb{R}, \Sigma) \in \mathsf{P}$*, for fixed* $k \in \mathbb{N}$.

Proof. For any given k and $\mathcal{PS}$, there are $\binom{|\mathcal{PS}|}{k}$ models of size k to check. It is always the case that $\binom{n}{k} \leq \frac{n^k}{k!}$, which grows polynomially in $n = |\mathcal{PS}|$ for any fixed k. □

This makes *whatever* language we want for representing our voters' ballots computationally tractable (though not necessarily trivial) so long as the number of seats and candidates is not too large. In particular, it is well within the capabilities of contemporary desktop computers to determine the winners in committee elections of a size similar to that conducted by Game Theory Society in 2003, where there would be only 2.7 million models to check.

7.6 Extending Single-Winner Voting Methods

In this section, we consider ways in which single-winner voting methods may be extended using goalbase ballots, and provide a concrete example where we extend approval voting from the approval of single candidates to the approval of properties of outcomes.

The fact that we can easily simulate many single-step voting procedures by using goalbases and solving MAX-UTIL on them suggests a way of extending

these methods to better register the preferences of voters. Let us extend the expressiveness of plurality (in our terminology, 1-vote top-1) as an example. In a standard plurality election, each voter casts a single vote for a single candidate. This permits voters to express only single-peaked, modular, monochromatic utility functions—that is, the only voters who can accurately express their preferences using such a method are those who prefer all candidates equally, except for one candidate who is preferred over the others. This is an unusual preference ordering for a voter to have. (Think of the 2000 U.S. Presidential election: What sort of voter would most prefer Gore, but at the same time be indifferent between Nader and Bush?)

The goalbase simulation of plurality allows voters to weight a single atom each. What if, instead, voters were subject to fewer restrictions on the goalbases they submit? Suppose that we ease the restriction on our voting language so that instead of just one, voters may specify up to n $\{0, 1\}$-weighted atoms. Now we can additionally express preference orderings where more than one candidate is maximally preferred, and indeed, solving MAX-UTIL over singleton models will give us approval voting instead of plurality voting.

If we move to multi-winner voting as we have when electing committees, the fit between voter preferences and the expressivity of the voting language grows worse, as argued above. Using goalbases, it is not difficult to simulate top-k voting methods—in order to find the top k candidates in the aggregate preference order, we need only solve MAX-UTIL on the sum of voter goalbases, ignoring models electing more or fewer than k candidates. In order to gain more expressivity, we can further relax the restrictions on the formulas which may be weighted. Examples:

- Suppose that we restrict voters to positive clauses with binary weights. This language is sufficient for expressing any weak linear ordering of candidates, as we shall see later this section.

- Suppose that we restrict voters to positive cubes with binary weights. This language permits voters to assign a bonus to committees which contain favored combinations of candidates. If a voter believes that, *ceteris paribus*, committees with both A and B are preferable, then he may have a goalbase such that $(a \wedge b, 1) \in G$.

We mention here a several classes of formulas which voters electing committees might find useful:

- **Literals:** a and $\neg b$ are useful for expressing simple preferences, e.g., "I want Alice on the committee", or "I don't want Bob on the committee".

- **Positive cubes:** $a \wedge b$ is useful when the combination of some candidates is better than those candidates individually.

- **Negative Horn clauses:** $\neg a \vee \neg b$ is useful when the combination of some candidates is worse than those candidates individually.

This last class, negative Horn clauses, is exactly what is needed to overcome the difficulty described in Section 7.3.2, where two candidates may be individually desirable but collectively undesirable. Or, equivalently, we could use positive cubes with negative weights. The voter in our example who preferred Alice-committees and Bob-committees over neither-committees over both-committees could represent his preferences as $G = \{(a, 1), (b, 1), (a \wedge b, -3), (\top, 1)\}$. It is easily checked that u_G respects the voter's preference ordering:

$$u_G(X) = \begin{cases} 2 & \text{if } a \in X, b \notin X \text{ or } \textit{vice versa} \\ 1 & \text{if } a, b \notin X \\ 0 & \text{if } a, b \in X \end{cases}$$

In the general case where voters cast arbitrary goalbases G_i as ballots, we can determine a winning committee by solving MAX-UTIL for $\bigoplus_i G_i$ on k-seat models only.

Now we offer one example of how a single-winner voting method may have its expressivity extended through goalbase voting.

Call Property Approval Voting (PAV) the voting method in which properties of the outcome (rather than individual candidates) are the objects of approval or disapproval. Any goalbase $G \in \mathcal{L}(\mathit{forms}, \{1\}, \Sigma)$ constitutes an admissible PAV ballot. However, some formulas will be useless: Any formula which implies a positive cube longer than the intended number of winners, and any formula which implies a negative cube longer than the intended number of losers, will effectively be equivalent to $\bot$. A significant difference between AV and PAV is the range of preorders of which they permit representation. Every AV ballot induces a dichotomous order, while PAV supports much more. In the case where there are three candidates a, b, c, the PAV ballot $\{(a, 1), (a \vee b, 1)\}$ induces the (non-dichotomous) order $a > b > c$, since the state $\{a\}$ receives two points, $\{b\}$ one point, and $\{c\}$ zero points. (Only singleton states are relevant here, since we are considering the single-winner case.)

In fact, there is a general way of representing any strict linear order $a_1 > a_2 > \ldots > a_n$ with a PAV ballot:

$$\begin{gathered} (a_1 \vee \ldots \vee a_{n-2} \vee a_{n-1}, 1) \\ (a_1 \vee \ldots \vee a_{n-2}, 1) \\ \vdots \\ (a_1 \vee a_2, 1) \\ (a_1, 1) \end{gathered}$$

The clause which ends with a_i is the one which causes a_i to be ordered strictly above a_{i+1}, so by omitting that clause we can get a ballot where $a_i \sim a_{i+1}$. This is sufficient to induce any weak linear order over the candidates. Thus, in the single-winner case PAV is something like a nonresolute version of the Borda rule.

7.7 Future Work

There are a number of paths yet to be explored regarding voting with goalbases. In this section we give an overview of those of which we are aware.

In order to use goalbase ballots for multi-winner cumulative voting, we must place some restriction on the weights which are available to voters. As noted after Fact 7.4.3, cumulative voting without point limits is not a sensible voting method. Having established that restrictions are needed, we are now faced with the problem of selecting some—it is not presently obvious which restrictions are most suitable. The restriction which cumulative voting itself suggests is to limit the sum of weights in any goalbase: $\sum_{(\varphi,w)\in G} w \leq K$.[7] This is a limit on the input space. Another approach is to restrict the output space: For example, we might limit the utility of any admissible state: $u_G(M) \leq K$ for all $M \subseteq \mathcal{PS}$ where $|M| = k$.

There are advantages and disadvantages to both methods. If our voter is a person, then he will find it easier to cast a valid sum-limited ballot than a valid state-limited one. Input limits are not uncommon. For example, in the U.S., the State of Illinois used cumulative voting (over atoms) with a 3-point limit for electing members of its House of Representatives from 1870 to 1980 [Moore, 1909; Yale Law Journal, 1982]. Corporate boards of directors are usually elected using cumulative voting, where the point limit for each voter is the number of shares he owns. We know of no uses of output limits: Presumably this is because it is hard to see when working in the input space whether output limits are being respected; output limits expect too much of the average voter. However, output limits on elections of the size human voters are likely to face will not be difficult for machines to enforce, so might be useful if the voters are using a computer-aided voting system. This is a user-interface issue.

Point limits also raise a fairness issue. For simplicity, we use a single-winner example, though the problem it illustrates is general. The sum-limit $\sum_{(\varphi,w)\in G} w \leq K$ will not always produce utility functions which have equal sums for singleton models. E.g., consider the goalbase ballots $G_1 = \{(a, 10)\}$ and $G_2 = \{(a \vee b, 10)\}$. The latter has a singleton state sum of 20 ($u_{G_2}(\{a\}) = 10, u_{G_2}(\{b\}) = 10$), while singleton states for the former sum only to 10 ($u_{G_1}(\{a\}) = 10$). The effect of

[7] If we permit negative weights, then we would need to place an upper bound on the sum of the absolute values of the weights instead of on the sum of the weights. In this way we avoid ballots like $\{(a, -2^{1000}), (b, 2^{1000} + 10)\}$ which the voter could claim is a 10-point ballot according to the latter method.

the sum-limit is to give voters with top-heavy preferences more influence on the outcome than voters with balanced or bottom-heavy preferences. Note that this is not a failure of anonymity, as it has nothing to do with voters' names or order. We could try to account for this by "normalizing" formulas based on the number of states they affect, e.g., $(a \vee b, 10)$ could be translated to $(a, 5), (b, 5)$, but this would seem to disadvantage voters who have top-heavy preferences. If $(\bigvee \mathcal{PS} \setminus \{a\}, 10)$, after normalization, gives one point to everyone but candidate a, that is not likely to be very effective for voters who dislike a but otherwise do not distinguish among the other candidates. Or we could try other ways of normalizing—Lafage and Lang [2000, Section 3.2.3] suggest postprocessing (dis)utilities to equalize entropy across agents—which will potentially have some other differential effect on voters.

The basic question here seems to be how to set the value of preferences which are not over single states against those which are. What is an appropriate measure of voting power here? Input limits seem to favor top-heavy voters, output limits seem to favor bottom-heavy voters. One way of quantifying the effect that a proposed weight limit could have is by considering the *efficacy* of voters with different preferences under that weight limit. (The efficacy of a ballot for a voter is a measure of how often that voter will be pivotal if he casts that ballot.) Ideally, all voters would have equally efficacious ballots to cast. Brams and Fishburn [2007, Chapter 5] calculate the efficacy of ballots for approval voting and find that not all ballots are equally effective. If we assume that our voters are truthful, what this means is that approval voting is advantageous for voters with some kinds of preference orders and disadvantageous for others. A similar analysis could be done for cumulative voting with goalbase ballots, with an eye to which weight restrictions treat voters most equitably.

With any voting system, there are questions about whether it encourages or discourages strategic voting. The manipulability of a voting system must always be considered in the context of a notion of sincerity, for we cannot say whether a voter is *mis*representing his preferences if we cannot first say what it would be for a voter to represent his preferences accurately.

Consider, first, voting systems with ordinal ballots. Many standard systems—e.g., plurality, approval, Borda—use ballots which contain purely ordinal information. In the case where there is an allowable ballot which induces the same preorder over outcomes as the voter's true preorder, then any reasonable notion of sincerity should deem that ballot sincere (and any ballot which does not induce that same preorder, insincere). This means, for example, that for voters with strict linear orders, there will always be unique sincere plurality and Borda ballots; and similarly, for voters whose preferences are dichotomous (and not wholly indifferent), there will be a unique sincere approval ballot. However, this will not be the case for voters with other kinds of preorders. There are no approval ballots which express nondichotomous preferences (e.g., $x > y > z$); standardly, Borda does not permit ties, so voters with weak (instead of strict) orders will have no ballots which express their preferences exactly.

What should count as sincere in the space of cardinal ballots is not immediately obvious. A voter's preferences may be inexpressible as a result of restrictions on the ballot language, and this can result in the existence of multiple sincere ballots which the voter could cast. Endriss [2007] explores the existence of multiple sincere ballots for approval voting and shows that the Gibbard-Satterthwaite Theorem is avoidable in that context; Endriss, Pini, Rossi, and Venable [2009] present several measures of sincerity for languages where ballots are preorders, and examine the consequences for strategyproofness under these. This line of research could be continued for goalbase ballots, first by developing reasonable notions of sincerity, and secondly by determining which language restrictions induce sincerity in rational voters. Meir, Procaccia, Rosenschein, and Zohar [2008] avoid the problem of sincerity in multi-winner voting altogether by defining manipulation as an optimization problem asking whether, given the ballots of some other voters, there is a ballot which the manipulating voter may cast which yields him at least t utility. The question of whether a better ballot exists is more general than, and serves as a proxy for, the question of whether a better insincere ballot exists—though this still leaves open the possibility that some ballot which is optimal is nonetheless also sincere, and so does not exactly capture classical manipulability.

Finally, we might consider questions about the difficulty of finding a sincere ballot given a voter's preferences. It would not be surprising to learn that for some languages, it is always in the voter's best interests to cast a sincere ballot, but nonetheless quite difficult for him to determine which ballots are sincere for him. Strategyproofness is not worth much in this case. A method for constructing sincere ballots will be essential for any language intended for human voters.

7.8 Conclusion

In this chapter we introduced some methods for electing committees and demonstrated that they lack certain properties which are desirable when conducting multi-winner elections. In particular, single-winner voting methods lack the expressivity to extend well to the multi-winner case. The observation that it is possible to simulate many single-winner voting methods using goalbases and MAX-UTIL suggests one way of extending the expressivity of existing voting methods for use in a multi-winner setting. Because multi-winner elections tend to have the number of winners fixed beforehand, the complexity of MAX-UTIL is limited, even when the goalbase language is not. Along these lines, we suggest a multi-winner extension of approval voting, which we call Property Approval Voting. Finally, we discuss the possibilities for future work: the need to find useful limits on weights in goalbase ballots; the fairness of these limits, since they may differentially affect voters with dissimilar preferences; and issues related to sincerity and strategic voting.

Chapter 8

Conclusion

In this dissertation we have presented a framework for compactly representing cardinal preferences over combinatorial domains and shown the feasibility of using this framework for auctions and voting.

Goalbase languages are formed by the parameters restricting the available formulas and weights. Due to their parametric nature, these languages are scattered all across the representational landscape. In order to make practical use of goalbases, we must first know the lay of the land. In Part I, we have explored the landscape of goalbase languages in three directions:

Expressivity. In Chapter 3, we considered this question: Given a goalbase language, what utility functions are representable in it? Many goalbase languages with natural definitions were revealed to correspond exactly to classes of utility functions having well-known properties. Furthermore, we showed that some goalbase languages have precisely one representation for any representable utility function, and provided methods for finding these representations. For a summary of these results, see Figures 3.1 and 3.2, and the accompanying explanatory text in Sections 3.4.4 and 3.5.4.

Succinctness. In Chapter 4, we pursued the problem of concision. Given two goalbase languages, are the smallest representations in one significantly smaller than equivalent smallest representations in the other? We systematically compared more than two hundred pairs of languages to determine which languages were more succinct. For a summary of these results, see Tables 4.1 and 4.2.

Complexity. In Chapter 5, we examined how the structure of a goalbase language affects the computational complexity of three decision problems: MAX-UTIL, MIN-UTIL, and MAX-CUF. More expressive languages tended to have NP-complete MAX-UTIL and MAX-CUF problems, and coNP-complete MIN-UTIL problems; for those which are solvable in polynomial time, we provided algorithms demonstrating

that. For a summary of the results for MAX-UTIL and MIN-UTIL, see Table 5.1, and for MAX-CUF see Table 5.2. Finally, we considered an alternative version of MAX-UTIL, which focuses on true atoms in optimal states instead of the existence of models which reach a given utility level.

In Part II of this dissertation, we considered two possible applications of goalbase languages:

Auctions. In Chapter 6, we introduced combinatorial auctions and the widely-studied XOR/OR family of bidding languages. Goalbase languages are sometimes more and sometimes less succinct than languages in the XOR/OR family, depending on the utility functions to be represented. This is as expected, and supplies some of our motivation for the investigations in Part I: An auction designer cannot choose the most appropriate bidding language without knowing the expressivity and succinctness characteristics of the languages on offer. We went on to describe the Winner Determination Problem, both formulating it as an integer program and defining branch-and-bound heuristics for solving it directly. Finally, we presented some experimental results showing the performance of our IP formulation and branch-and-bound solver, which demonstrated the feasibility of using goalbase languages for auctions of moderate size.

Voting. In Chapter 7, we considered the problem of insufficiently expressive voting methods, and suggested voting with goalbases as ballots as a possible remedy. Common single-winner voting methods do not extend well to multi-winner settings like committee elections due to interactions between candidates. We noted that finding winners when goalbases are used as ballots is similar to MAX-UTIL, and that in practice the complexity will tend to be manageable due to small numbers of candidates and seats. We suggested an extension to approval voting, where properties of the outcome are approved (or not) rather than particular outcomes as in standard approval voting. Finally, we discussed a number of directions for further investigation.

Together, these chapters provide a clear view of the power of goalbase languages for preference representation, and point to potential areas of application.

Open Questions and Future Work. Here we mention some open questions and directions for future work:

In Chapter 2, we have presented only a limited range of restrictions on formulas. It would be interesting to investigate other, less straightforward properties of formulas, such as being read-once, or representable by a Boolean circuit with certain features, to see what effect these might have on expressivity, succinctness, and complexity. Additionally, sum and max are only the most obvious aggregators; in principle, any function $F\colon \mathbb{N}^{\mathbb{R}} \to \mathbb{R}$ could be used, so there are others (e.g., min) which might be of interest.

In Chapter 3, there were a few languages for which we were unable to exactly characterize their expressivity. (See Figures 3.1 and 3.2.) In particular, $\mathcal{L}(k\text{-}forms, \mathbb{R}^+, \Sigma)$ represents some subset of the nonnegative k-additive utility functions, but which subset or even whether the inclusion is proper is unknown. For $\mathcal{L}(clauses, \mathbb{R}^+, \Sigma)$, it is known to represent a proper subset of the nonnegative utility functions, but here, again, exactly what other properties that subset has is unknown.

In Chapter 4, some open succinctness questions can be seen in Table 4.1. We do not expect any of the proofs which would resolve the remaining open questions to be easy, as all involve pairs of languages where neither language has unique representations. We would particularly like to know the relation between $\mathcal{L}(cubes, \mathbb{R}^+, \Sigma)$ and $\mathcal{L}(clauses, \mathbb{R}^+, \Sigma)$, as well as between $\mathcal{L}(pforms, \mathbb{R}^+, \Sigma)$ and these two; but resolving any of the remaining open questions would be useful, since the combination of Fact 4.2.3 and the numerous results we already have mean that any new result can be leveraged to answer several open questions at once.

In Chapter 5, we resolved for most languages whether MAX-UTIL and MAX-CUF are polynomial or NP-complete. What this leaves open is where the boundary is: For the languages which have polynomial decision problems, how much can we loosen the restrictions on their structure and still remain polynomial? Or, from the other direction: How little can we take away from the NP-complete languages before they become polynomial? We have also not investigated how amenable these decision problems are to approximation, nor whether for MAX-CUF any of the languages are strategyproof and if not, how hard manipulation is.

We believe that our WDP algorithms in Chapter 6 could be improved, either through tighter heuristics or other combinatorial optimization techniques. In a practical vein, it would be gratifying to see one of our languages used for real combinatorial auctions.

The avenues for further work on voting in Chapter 7 are too numerous to repeat here; for a full accounting of them, see Section 7.7. Of technical interest are methods for finding a maximally sincere ballot when no ballot matches a voter's preferences and determining whether a voting rule advantages or disadvantages voters with certain preferences. Of practical interest is how to design a goalbase voting system which human voters would find usable.

Bibliography

A. Andersson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. In *4th International Conference on Multi-Agent Systems (ICMAS 2000)*, pp. 39–46. IEEE Computer Society, 2000. Cited on pp. 147, 161, 163.

H. Anton. *Elementary Linear Algebra*. Wiley & Sons, seventh ed., 1994. Cited on p. 39.

K.R. Apt, F. Rossi, and K.B. Venable. Comparing the notions of optimality in CP-nets, strategic games and soft constraints. *Annals of Mathematics and Artificial Intelligence*, 52(1):25–54, 2008. Cited on p. 15.

K.J. Arrow. *Social Choice and Individual Values*. Yale University Press, second ed., 1970. Cited on p. 184.

G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, 1999. Cited on p. 105.

F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second ed., 2007. Cited on p. 23.

F. Bacchus and A.J. Grove. Graphical models for preference and utility. In P. Besnard and S. Hanks, eds., *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, August 18–20, 1995, Montreal, Quebec, Canada*, pp. 3–10. Morgan Kaufmann, 1995. Cited on p. 26.

M.O. Ball, G.L. Donohue, and K. Hoffman. Auctions for the safe, efficient, and equitable allocation of airspace system resources. In Cramton et al. [2006], pp. 507–538. Cited on p. 3.

B. Benhamou, L. Sais, and P. Siegel. Two proof procedures for a cardinality based language in propositional calculus. In P. Enjalbert, E.W. Mayr, and K.W. Wagner, eds., *STACS 94, 11th Annual Symposium on Theoretical Aspects of Computer Science, Caen, France, February 24–26, 1994, Proceedings*, vol. 775 of Lecture Notes in Computer Science, pp. 71–82. Springer, 1994. Cited on p. 195.

J.-P. Benoît and L.A. Kornhauser. Voting simply in the election of assemblies. Technical Report RR 91-32, New York University Starr Center for Applied Economics, June 1991. Cited on p. 190.

J.-P. Benoît and L.A. Kornhauser. Only a dictatorship is efficient or neutral. Technical report, New York University School of Law, 2006. Cited on p. 190.

J.-P. Benoît and L.A. Kornhauser. Social choice in a representative democracy. *American Political Science Review*, 88(1):185–192, 1994. Cited on p. 190.

S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3):199–240, 1999. Cited on p. 24.

L. Blumrosen and N. Nisan. Combinatorial auctions. In N. Nisan, T. Roughgarden, É. Tardos, and V.V. Vazirani, eds., *Algorithmic Game Theory*, pp. 267–300. Cambridge University Press, 2007. Cited on p. 119.

E. Bonzon, M.-C. Lagasquie-Schiex, J. Lang, and B. Zanuttini. Compact preference representation and Boolean games. *Autonomous Agents and Multi-Agent Systems*, 18(1):1–35, 2009. Cited on p. 24.

C. Boutilier. Solving concisely expressed combinatorial auction problems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI 2002)*. AAAI Press, 2002. Cited on pp. 144, 147, 148, 149.

C. Boutilier, ed. *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-2009)*, 2009. Cited on pp. 209, 212, 220.

C. Boutilier and H.H. Hoos. Bidding languages for combinatorial auctions. In Nebel [2001], pp. 1211–1217. Cited on pp. 27, 28, 144, 147, 149.

C. Boutilier, R. Brafman, C. Geib, and D. Poole. A constraint-based approach to preference elicitation and decision making. In *AAAI Spring Symposium on Qualitative Decision Theory*, 1997. Cited on p. 14.

C. Boutilier, R.I. Brafman, H.H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In K.B. Laskey and H. Prade, eds.,

UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30–August 1, 1999, pp. 71–80. Morgan Kaufmann, 1999a. Cited on p. 14.

C. Boutilier, M. Goldszmidt, and B. Sabata. Sequential auctions for the allocation of resources with complementarities. In T. Dean, ed., *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 99)*, pp. 527–523. Morgan Kaufmann, 1999b. Cited on p. 163.

C. Boutilier, F. Bacchus, and R.I. Brafman. UCP-networks: A directed graphical representation of conditional utilities. In J.S. Breese and D. Koller, eds., *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2–5, 2001*, pp. 56–64. Morgan Kaufmann, 2001. Cited on p. 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 Artifcial Intelligence Research (JAIR)*, 21:135–191, 2004. Cited on pp. 15, 16, 35.

S. Bouveret. *Allocation et partage équitables de ressources indivisibles: modélisation, complexité et algorithmique*. PhD thesis, Supaéro/University of Toulouse, 2007. Cited on pp. 101, 102, 120, 147.

S. Bouveret and J. Lang. Efficiency and envy-freeness in fair division of indivisible goods: Logical representation and complexity. *Journal of Artifcial Intelligence Research (JAIR)*, 32:525–564, 2008. Cited on p. 102.

S. Bouveret, H. Fargier, J. Lang, and M. Lemaître. Allocation of indivisible goods: A general model and some complexity results. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M.P. Singh, and M. Wooldridge, eds., *4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25–29, 2005, Utrecht, The Netherlands*, pp. 1309–1310. ACM, 2005. Cited on p. 120.

S. Bouveret, U. Endriss, and J. Lang. Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods. In Boutilier [2009], pp. 67–72. Cited on p. 16.

R.I. Brafman and C. Domshlak. Introducing variable importance tradeoffs into CP-nets. In A. Darwiche and N. Friedman, eds., *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1–4, 2002*, pp. 69–76. Morgan Kaufmann, 2002. Cited on p. 16.

R.I. Brafman, C. Domshlak, and T. Kogan. Compact value-function representations for qualitative preferences. In D.M. Chickering and J.Y. Halpern, eds., *UAI '04, Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, July 7–11 2004, Banff, Canada*, pp. 51–59. AUAI Press, 2004. Cited on p. 26.

S. Brams and P. Fishburn. Voting procedures. In K. Arrow, A.K. Sen, and K. Suzumura, eds., *Handbook of Social Choice and Welfare, Volume 1*, no. 19 in Handbooks in Economics. North-Holland, 2002. Cited on p. 184.

S.J. Brams. *Mathematics and Democracy: Designing Better Voting and Fair-Division Procedures*. Princeton University Press, 2007. Cited on p. 182.

S.J. Brams and P.C. Fishburn. *Approval Voting*. Springer, second ed., 2007. Cited on p. 200.

S.J. Brams, D.M. Kilgour, and M.R. Sanver. A minimax procedure for negotiating multilateral treaties. In M. Wiberg, ed., *Reasoned Choices: Essays in Honor of Academy Professor Hannu Nurmi*, pp. 108–139. Finnish Political Science Association, Turku, Finland, 2004. Cited on pp. 181, 186, 187.

S.J. Brams, D.M. Kilgour, and M.R. Sanver. How to elect a representative committee using approval balloting. In *Mathematics and Democracy*, Studies in Choice and Welfare, pp. 83–95. Springer, 2006. Cited on pp. 181, 188.

S.J. Brams, D.M. Kilgour, and M.R. Sanver. A minimax procedure for electing committees. *Public Choice*, 132(3–4):401–420, 2007. Cited on pp. 181, 187, 190, 194.

M. Cadoli, F.M. Donini, P. Liberatore, and M. Schaerf. Comparing space efficiency of propositional knowledge representation formalisms. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5–8, 1996*, pp. 364–373. Morgan Kaufmann, 1996. Cited on p. 66.

M. Cadoli, F.M. Donini, P. Liberatore, and M. Schaerf. The size of a revised knowledge base. *Artifcial Intelligence*, 115(1):25–64, 1999a. Cited on p. 66.

M. Cadoli, L. Palopoli, and F. Scarcello. Propositional lower bounds: Algorithms and complexity. *Annals of Mathematics and Artificial Intelligence*, 27(1–4): 129–148, 1999b. Cited on p. 66.

M. Cadoli, F.M. Donini, P. Liberatore, and M. Schaerf. Space efficiency of propositional knowledge representation formalisms. *Journal of Artificial Intelligence Research (JAIR)*, 13:1–31, 2000. Cited on pp. 62, 66.

M. Cadoli, F.M. Donini, P. Liberatore, and M. Schaerf. Preprocessing of intractable problems. *Information and Computation*, 176(2):89–120, 2002. Cited on p. 66.

E. Cantillon and M. Pesendorfer. Auctioning bus routes: The London experience. In Cramton et al. [2006], pp. 573–591. Cited on p. 3.

C. Caplice and Y. Sheffi. Combinatorial auctions for truckload transportation. In Cramton et al. [2006], pp. 539–571. Cited on p. 3.

C. Cayrol, M.-C. Lagasquie-Schiex, and T. Schiex. Nonmonotonic reasoning: From complexity to algorithms. *Annals of Mathematics and Artificial Intelligence*, 22 (3-4):207–236, 1998. Cited on p. 18.

A. Chagrov and M. Zakharyaschev. *Modal Logic*, vol. 35 of Oxford Logic Guides. Clarendon Press, 1997. Cited on p. 34.

J.R. Chamberlin and P.N. Courant. Representative deliberations and representative decisions: Proportional representation and the Borda rule. *The American Political Science Review*, 77(3):718–733, 1983. Cited on p. 181.

Y. Chevaleyre, U. Endriss, and J. Lang. Expressive power of weighted propositional formulas for cardinal preference modelling. In Doherty, Mylopoulos, and Welty [2006], pp. 145–152. Cited on pp. 6, 12, 62, 67, 73, 155.

Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet. Multiagent resource allocation in k-additive domains: Preference representation and complexity. *Annals of Operations Research*, 163(1):49–62, 2008a. Cited on pp. 33, 73, 105.

Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. Preference handling in combinatorial domains: From AI to social choice. *AI Magazine, Special Issue on Preferences*, 29(4):37–46, 2008b. Cited on p. 185.

J.M. Colomer and I. McLean. Electing popes: Approval balloting and qualified-majority rule. *Journal of Interdisciplinary History*, 29(1):1–22, 1998. Cited on p. 182.

Complexity Zoo. `http://qwiki.stanford.edu/wiki/Complexity_Zoo`, 2009. Cited on p. 97.

V. Conitzer and T. Sandholm. Computing Shapley values, manipulating value division schemes, and checking core membership in multi-issue domains. In McGuinness and Ferguson [2004], pp. 219–225. Cited on p. 68.

V. Conitzer, T.W. Sandholm, and P. Santi. Combinatorial auctions with k-wise dependent valuations. In M.M. Veloso and S. Kambhampati, eds., *Proceeings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pp. 248–254. AAAI Press, 2005. Cited on pp. 33, 155.

S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pp. 151–158. ACM, 1971. Cited on p. 99.

S. Coste-Marquis, J. Lang, P. Liberatore, and P. Marquis. Expressive power and succinctness of propositional languages for preference representation. In Dubois, Welty, and Williams [2004], pp. 203–212. Cited on pp. 19, 20, 34, 62, 66.

P. Cramton, Y. Shoham, and R. Steinberg, eds. *Combinatorial Auctions*. MIT Press, 2006. Cited on pp. 101, 138, 207, 210, 215, 217, 219.

E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33:374–425, 2001. Cited on p. 125.

A. Darwiche and P. Marquis. Compiling propositional weighted bases. *Artificial Intelligence*, 157:81–113, 2004. Cited on p. 66.

A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002. Cited on p. 67.

A.P. Dempster. Upper and lower probabilities induced by a multivaluated mapping. *Annals of Mathematical Statistics*, 38(2):325–339, 1967. Cited on p. 49.

X. Deng and C.H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994. Cited on p. 68.

P. Doherty, J. Mylopoulos, and C.A. Welty, eds. *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2–5, 2006*, 2006. AAAI Press. Cited on pp. 211, 220.

D. Dubois, C.A. Welty, and M.-A. Williams, eds. *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR-2004), Whistler, Canada, June 2–5, 2004*, 2004. AAAI Press. Cited on pp. 211, 213.

P.E. Dunne, W. van der Hoek, S. Kraus, and M. Wooldridge. Cooperative Boolean games. In Padgham, Parkes, Müller, and Parsons [2008], pp. 1015–1022. Cited on p. 24.

F. Dupin de Saint-Cyr, J. Lang, and T. Schiex. Penalty logic and its link with Dempster-Shafer theory. In R. López de Mántaras and D. Poole, eds., *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI-1994)*, pp. 204–211. Morgan Kaufmann, 1994. Cited on p. 16.

E. Elkind, L.A. Goldberg, P.W. Goldberg, and M. Wooldridge. A tractable and expressive class of marginal contribution nets and its applications. *Mathematical Logic Quarterly*, 55(4):362–376, 2009. Cited on pp. 26, 68, 77, 93, 131.

U. Endriss. Vote manipulation in the presence of multiple sincere ballots. In Samet [2007], pp. 125–134. Cited on p. 201.

U. Endriss, M.S. Pini, F. Rossi, and K.B. Venable. Preference aggregation over restricted ballot languages: Sincerity and strategy-proofness. In Boutilier [2009], pp. 122–127. Cited on p. 201.

Federal Election Commission. Federal elections 2008: Election results for the U.S. President, the U.S. Senate and the U.S. House of Representatives. `http://www.fec.gov/pubrec/fe2008/federalelections2008.pdf`, July 2009. Cited on p. 1.

P.C. Fishburn. *Utility Theory for Decision Making*. John Wiley & Sons, 1970. Cited on p. 26.

Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In T. Dean, ed., *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 99)*, pp. 548–553. Morgan Kaufmann, 1999. Cited on pp. 101, 139, 147, 152, 158, 163.

M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979. Cited on pp. 104, 105, 107, 113, 119, 144.

A. Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41 (4):587–601, 1973. Cited on p. 184.

R. Gonen and D. Lehmann. Linear Programming helps solving large multi-unit combinatorial auctions. In *Electronic Market Design Workshop, The Proceedings of INFORMS 2001*. Institute for Operations Research and the Management Sciences, 2001. Cited on p. 163.

R. Gonen and D.J. Lehmann. Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-2000)*, pp. 13–20, 2000. Cited on p. 163.

C. Gonzales and P. Perny. GAI networks for utility elicitation. In Dubois et al. [2004], pp. 224–234. Cited on p. 26.

C. Gonzales, P. Perny, and S. Queiroz. Preference aggregation in combinatorial domains using GAI-nets. In D. Bouyssou, F. Roberts, and A. Tsoukiàs, eds.,

Proceedings of the DIMACS-LAMSADE Workshop on Voting Theory And Preference Modelling, vol. 6 of Annales du LAMSADE, pp. 165–179, Paris, October 2006. Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision. Cited on pp. 25, 26.

M. Grabisch. k-order additive discrete fuzzy measures and their representation. *Fuzzy Sets and Systems*, 92(2):167–189, 1997. Cited on pp. 33, 36, 155.

R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, second ed., 1994. Cited on p. 191.

R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. A compendium of problems complete for P. Technical report, University of Alberta, Computer Science Department, 1992. URL `http://citeseer.ist.psu.edu/greenlaw91compendium.html`. Cited on p. 125.

G. Hägele and F. Pukelsheim. The electoral systems of Nicholas of Cusa in the *Catholic Concordance* and beyond. In G. Christianson, T.M. Izbicki, and C.M. Bellitto, eds., *The Church, the Councils, & Reform: The Legacy of the Fifteenth Century*, pp. 229–249. The Catholic University of America Press, 2008. Cited on p. 182.

P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2):100–107, 1968. Cited on p. 154.

C. Haub. How many people have ever lived on earth? *Population Today*, 30(8): 3–4, November/December 2002. Cited on p. 191.

S. van Hoesel and R. Müller. Optimization in electronic markets: examples in combinatorial auctions. *Netnomics*, 3(1):23–33, 2001. Cited on p. 119.

R.C. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In E. Stroulia and S. Matwin, eds., *Advances in Artificial Intelligence, 14th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI 2001, Ottawa, Canada, June 7–9, 2001, Proceedings*, vol. 2056 of Lecture Notes in Computer Science, pp. 57–66. Springer, 2001. Cited on p. 163.

H.H. Hoos and C. Boutilier. Solving combinatorial auctions using stochastic local search. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence (AAAI/IAAI 2000)*, pp. 22–29. AAAI Press, 2000. Cited on pp. 27, 144, 147, 149, 195.

S. Ieong and Y. Shoham. Marginal contribution nets: A compact representation scheme for coalitional games. In J. Riedl, M.J. Kearns, and M.K. Reiter, eds., *Proceedings, 6th ACM Conference on Electronic Commerce (EC-2005)*, pp. 193–202. ACM Press, 2005. Cited on pp. 26, 35, 67, 68, 77, 131.

ILOG, 2009. ILOG CPLEX. `http://www.ilog.com/products/cplex/`, 2009. Cited on p. 147.

B. Jaumard and B. Simeone. On the complexity of the maximum satisfiability problem for Horn formulas. *Information Processing Letters*, 26(1):1–4, 1987. Cited on p. 106.

R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations*. Plenum Press, 1972. Cited on p. 119.

R. Kastner, C. Hsieh, M. Potkonjak, and M. Sarrafzadeh. On the sensitivity of incremental algorithms for combinatorial auctions. In *Proceedings of the Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS)*, pp. 81–88. MIT Press, 2002. Cited on p. 163.

R. Kaye. Minesweeper is NP-complete. *Mathematical Intelligencer*, 22(2):9–15, 2000. Cited on p. 107.

P. La Mura and Y. Shoham. Expected utility networks. In K.B. Laskey and H. Prade, eds., *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30–August 1, 1999*, pp. 366–373. Morgan Kaufmann, 1999. Cited on p. 26.

C. Lafage and J. Lang. Logical representation of preferences for group decision making. In A.G. Cohn, F. Giunchiglia, and B. Selman, eds., *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR-2000)*, pp. 457–468. Morgan Kaufmann, 2000. Cited on pp. 12, 14, 18, 22, 200.

J. Lang. Logical preference representation and combinatorial vote. *Annals of Mathematics and Artificial Intelligence*, 42(1–3):37–71, 2004. Cited on pp. 12, 21, 22, 102, 116, 181.

J. Lang and L. Xia. Sequential composition of voting rules in multi-issue domains. *Mathematical Social Sciences*, 57(3):304–324, 2009. Cited on pp. 15, 185.

T. Lee. *Kolmogorov Complexity and Formula Size Lower Bounds*. PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, 2006. ILLC Publication DS-2006-01. Cited on p. 76.

B. Lehmann, D. Lehmann, and N. Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55:270–296, 2006a. Cited on pp. 32, 35.

D. Lehmann, R. Müller, and T. Sandholm. The winner determination problem. In Cramton et al. [2006], pp. 288–317. Cited on pp. 117, 138.

K. Leyton-Brown and Y. Shoham. A test suite for combinatorial auctions. In Cramton et al. [2006], pp. 451–478. Cited on p. 163.

K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-2000)*, pp. 66–76, 2000. Cited on p. 163.

P. Liberatore. Monotonic reductions, representative equivalence, and compilation of intractable problems. *Journal of the ACM*, 48(6):1091–1125, 2001. Cited on p. 65.

M. Lines. Approval voting and strategy analysis: A Venetian example. *Theory and Decision*, 20(2):155–172, 1986. Cited on p. 182.

R.J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In J.S. Breese, J. Feigenbaum, and M.I. Seltzer, eds., *Proceedings 5th ACM Conference on Electronic Commerce (EC-2004)*, pp. 125–131. ACM, 2004. Cited on p. 131.

R. Llull. *Blanquerna: a thirteenth century romance*. Jarrolds, London, 1926. Translated by E.A. Peers. Cited on p. 182.

Y. Mansour. Learning Boolean functions via the Fourier transform. In V. Roychowdhury, K.-Y. Siu, and A. Orlitsky, eds., *Theoretical Advances in Neural Computation and Learning*. Kluwer, 1994. Cited on p. 74.

M. Mavronicolas, B. Monien, and K.W. Wagner. Weighted Boolean formula games. In X. Deng and F.C. Graham, eds., *Internet and Network Economics, Third International Workshop, WINE 2007, San Diego, CA, USA, December 12–14, 2007, Proceedings*, vol. 4858 of Lecture Notes in Computer Science, pp. 469–481. Springer, 2007. Cited on p. 24.

R.P. McAfee and J. McMillan. Auctions and bidding. *Journal of Economic Literature*, 25(2):699–738, 1987. Cited on p. 136.

D.L. McGuinness and G. Ferguson, eds. *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25–29, 2004, San Jose, California, USA*, 2004. AAAI Press / The MIT Press. Cited on pp. 211, 218.

N. Megiddo and C.H. Papadimitriou. On total functions, existence theorems, and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. Cited on p. 122.

R. Meir, A.D. Procaccia, J.S. Rosenschein, and A. Zohar. Complexity of strategic behavior in multi-winner elections. *Journal of Artificial Intelligence Research (JAIR)*, 33:149–178, 2008. Cited on p. 201.

P. Milgrom. *Putting Auction Theory to Work*. Churchill Lectures in Economics. Cambridge University Press, 2004. Cited on p. 136.

B.L. Monroe. Fully proportional representation. *The American Political Science Review*, 89(4):925–940, 1995. Cited on p. 181.

B.F. Moore. *The History of Cumulative Voting and Minority Representation in Illinois, 1870–1908*, vol. III.3 of The University Studies. University Press, Urbana-Champaign, Illinois, 1909. Cited on p. 199.

H. Moulin. *Axioms of Cooperative Decision Making*, vol. 15 of Econometric Society Monographs. Cambridge University Press, 1988. Cited on pp. 33, 116.

R. Müller. Tractable cases of the winner determination problem. In Cramton et al. [2006], pp. 319–336. Cited on pp. 101, 138.

National Odd Shoe Exchange. The history of the National Odd Shoe Exchange. `http://www.oddshoe.org/history.php`, 2009. Cited on p. 137.

B. Nebel, ed. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4–10, 2001*, 2001. Morgan Kaufmann. Cited on pp. 208, 219.

N. Nisan. Bidding languages for combinatorial auctions. In Cramton et al. [2006], pp. 215–232. Cited on pp. 27, 32, 35, 51, 68, 138, 139, 141.

N. Nisan. Bidding and allocation in combinatorial auctions. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pp. 1–12, 2000. Cited on pp. 68, 101.

L. Padgham, D.C. Parkes, J. Müller, and S. Parsons, eds. *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12–16, 2008*, 2008. IFAAMAS. Cited on pp. 212, 219.

C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994a. Cited on pp. 95, 106, 113, 122.

C.H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994b. Cited on p. 111.

D.C. Parkes. *i*Bundle: An efficient ascending price bundle auction. In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99)*, pp. 148–157. ACM, 1999. Cited on p. 163.

E. Pilotto, F. Rossi, K.B. Venable, and T. Walsh. Compact preference representation in stable marriage problems. In F. Rossi and A. Tsoukias, eds., *Algorithmic Decision Theory: First International Conference, ADT 2009. Venice, Italy, October 2009. Proceedings*, vol. 5783 of Lecture Notes in Artificial Intelligence. Springer, 2009. Cited on p. 15.

G. Pinkas. Propositional nonmonotonic reasoning and inconsistency in symmetric neural networks. In J. Mylopoulos and R. Reiter, eds., *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-1991)*, pp. 525–531. Morgan Kaufmann, 1991. Cited on p. 16.

A. Ragone, T.D. Noia, F.M. Donini, E.D. Sciascio, and M.P. Wellman. Computing utility from weighted description logic preference formulas. In *Declarative Agent Languages and Technologies VII (DALT-2009)*, Lecture Notes in Computer Science. Springer, 2009a. To appear. Cited on p. 23.

A. Ragone, T.D. Noia, F.M. Donini, E.D. Sciascio, and M.P. Wellman. Weighted description logics preference formulas for multiattribute negotiation. In *Scalable Uncertainty Management. Third International Conference, SUM 2009, Washington, DC, USA, September 28–30, 2009. Proceedings*, vol. 5785 of Lecture Notes in Computer Science, pp. 193–205. Springer, 2009b. Cited on p. 23.

S. Ramezani. *Nash Social Welfare in Multiagent Resource Allocation*. Master's thesis, Institute for Logic, Language and Computation, University of Amsterdam, 2008. ILLC Publication MoL-2008-09. Cited on p. 147.

J. Rawls. *A Theory of Justice*. Harvard University Press, 1971. Cited on p. 117.

D. Roháč. The unanimity rule and religious fractionalisation in the Polish-Lithuanian Republic. *Constitutional Political Economy*, 19(2), 2008. Cited on p. 183.

J.S. Rosenschein and G. Zlotkin. *Rules of Encounter*. MIT Press, 1994. Cited on p. 33.

F. Rossi, K.B. Venable, and T. Walsh. mCP Nets: Representing and reasoning with preferences of multiple agents. In McGuinness and Ferguson [2004], pp. 729–734. Cited on p. 15.

G.-C. Rota. On the foundations of combinatorial theory I: Theory of Möbius functions. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 2 (4):340–368, 1964. Cited on pp. 36, 143.

M.H. Rothkopf, A. Pekeč, and R.M. Harstad. Computationally manageable combinational auctions. *Management Science*, 44(8):1131–1147, 1998. Cited on p. 101.

D. Samet, ed. *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-2007), Brussels, Belgium, June 25–27, 2007*, 2007. Presses Universitaires de Louvain. Cited on pp. 212, 220.

T. Sandholm. Expressive commerce and its application to sourcing: How we conducted $35 billion of generalized combinatorial auctions. *AI Magazine*, 28 (3):45–58, 2007. Cited on p. 3.

T. Sandholm, S. Suri, A. Gilpin, and D. Levine. CABOB: A fast optimal algorithm for winner determination in combinatorial auctions. *Management Science*, 51 (3):374–390, 2005. Cited on p. 163.

T.W. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1–2):1–54, 2002. Cited on pp. 139, 147, 158, 163.

T.W. Sandholm. Optimal winner determination algorithms. In Cramton et al. [2006], pp. 337–368. Cited on pp. 138, 152, 177.

T.W. Sandholm and S. Suri. BOB: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence*, 145(1–2):33–58, 2003. Cited on p. 152.

M.A. Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975. Cited on p. 184.

A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986. Cited on p. 147.

D. Schuurmans, F. Southey, and R.C. Holte. The exponentiated subgradient algorithm for heuristic Boolean programming. In Nebel [2001], pp. 334–341. Cited on p. 163.

G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, NJ, 1976. Cited on p. 49.

M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997. Cited on p. 95.

A.D. Taylor. *Social Choice and the Mathematics of Manipulation*. Cambridge University Press, 2005. Cited on pp. 182, 184, 192.

L. Trevisan. Lecture notes 25, CS278: Computational complexity, UC-Berkeley, 1 December 2004. `http://www.cs.berkeley.edu/~luca/cs278/notes/lecture25.pdf`. Cited on p. 75.

J. Uckelman and U. Endriss. Winner determination in combinatorial auctions with logic-based bidding languages. In Padgham et al. [2008], pp. 1617–1620. Short Paper. Cited on pp. 7, 12, 162.

J. Uckelman and U. Endriss. Preference representation with weighted goals: Expressivity, succinctness, complexity. In J. Doyle, J. Goldsmith, U. Junker, and J. Lang, eds., *Proceedings of the AAAI Workshop on Preference Handling for Artificial Intelligence (AiPref-2007)*, pp. 85–92, Vancouver, British Columbia, July 2007. AAAI Press. Technical Report WS-07-10. Cited on pp. 6, 12, 62.

J. Uckelman and U. Endriss. Preference modeling by weighted goals with max aggregation. In G. Brewka and J. Lang, eds., *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR-2008)*, pp. 579–587, September 2008b. Cited on pp. 6, 12.

J. Uckelman and A. Witzel. Logic-based preference languages with intermediate complexity. In J. Chomicki, V. Conitzer, U. Junker, and P. Perny, eds., *Proceedings of the 4th Multidisciplinary Workshop on Advances in Preference Handling (MPREF-2008)*, pp. 123–127, Chicago, 2008. AAAI Press. Cited on pp. 6, 12.

J. Uckelman, Y. Chevaleyre, U. Endriss, and J. Lang. Representing utility functions via weighted goals. *Mathematical Logic Quarterly*, 55(4):341–361, 2009. Cited on pp. 6, 12.

W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961. Cited on p. 136.

S. de Vries and R.V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, 15(3):284–309, 2003. Cited on p. 163.

M. Wachter and R. Haenni. Propositional DAGs: A new graph-based language for representing Boolean functions. In Doherty et al. [2006], pp. 277–285. Cited on pp. 66, 67.

I. Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, 1987. Cited on p. 65.

L.A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998. Cited on pp. 147, 148.

L. Xia and J. Lang. A dichotomy theorem on the existence of efficient or neutral sequential voting correspondences. In Boutilier [2009], pp. 342–347. Cited on p. 15.

L. Xia, J. Lang, and M. Ying. Strongly decomposable voting rules on multiattribute domains. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22–26, 2007, Vancouver, British Columbia, Canada*, pp. 776–781. AAAI Press, 2007a. Cited on p. 15.

L. Xia, J. Lang, and M. Ying. Sequential voting rules and multiple elections paradoxes. In Samet [2007], pp. 279–288. Cited on p. 15.

L. Xia, V. Conitzer, and J. Lang. Voting on multiattribute domains with cyclic preferential dependencies. In D. Fox and C.P. Gomes, eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13–17, 2008*, pp. 202–207. AAAI Press, 2008. Cited on p. 15.

Yale Law Journal, 1982. Alternative voting systems as remedies for unlawful at-large systems. *The Yale Law Journal*, 92(1):144–160, 1982. Cited on p. 199.

H.P. Young. Social choice scoring functions. *SIAM Journal on Applied Mathematics*, 28(4):824–838, 1975. Cited on pp. 184, 194.

H. Zhang and M.E. Stickel. An efficient algorithm for unit propagation. In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI-MATH '96), Fort Lauderdale, Florida, USA*, pp. 166–169, 1996. Cited on p. 128.

E. Zurel and N. Nisan. An efficient approximate allocation algorithm for combinatorial auctions. In *Proceedings, 3rd ACM Conference on Electronic Commerce (EC-2001)*, pp. 125–136, 2001. Cited on p. 163.

List of Symbols

Index

Samenvatting

In deze dissertatie presenteren we een theorie waarmee we compact cardinale voorkeuren over combinatorische domeinen kunnen weergeven en tonen aan dat het uitvoerbaar is deze theorie te gebruiken voor veilingen en stemprocedures.

Deze theorie gebruikt zogenaamde 'goalbases'. Dit zijn verzamelingen gewogen propositionele formules waarmee we utiliteitsfuncties kunnen representeren. We berekenen de utiliteitswaarde van een alternatief als de geagregeerde waarde van de gewichten van de doelen die het alternatief vervullen. Goalbase talen worden beschreven door de formules en gewichten die in de goalbases voor kunnen komen te beperken. Deze beperkingen laten zich zien als parameters waarmee de familie van goalbase talen zich laat beschrijven. Voor het praktische gebruik is het nodig eerst vast te stellen hoe de talen die bij toepassingen worden gebruikt feitelijk met deze parameters omgaan. We zullen het landschap van goalbase talen in relatie tot een drietal verschillende eigenschappen verkennen:

Uitdrukbaarheid: Gegeven een goalbase taal welke utiliteitsfuncties zijn hierin te representeren? Veel goalbase talen met natuurlijke definities komen precies overeen met klassen van utiliteitsfuncties die bekende eigenschappen hebben. Voorts laten we zien dat somige goalbase talen een unieke representatie hebben voor de representeerbare utiliteitsfuncties. Tevens verschaffen we methoden om deze representaties te vinden.

Beknoptheid: Gegeven twee goalbase talen, zijn de kortste representaties in de ene taal significant kleiner dan equivalente kortste representaties in de andere taal? Op systematische wijze bepalen we voor meer dan tweehonderd paren van talen wat de meer beknopte taal is.

Complexiteit: Gegeven een goalbase taal, hoe moeilijk is het om vragen te beantwoorden over goalbases die in deze taal zijn beschreven? Dit wordt toegespitst op een specifiek probleem. Gegeven een uitdrukking in een goalbase taal zoeken wij naar de optimale, resp. de slechtste toestand voor

individuen dan wel groepen. Dit probleem blijkt onhanteerbaar te zijn voor de meer expressieve goalbase talen; voor het geval dat het probleem wel effectief oplosbaar is geven wij algoritmen met een polynomiaal begrensde rekentijd die deze effectieve oplosbaarheid aantonen.

Nadat we de eigenschappen van veel goalbase talen hebben vastgesteld beschouwen we twee mogelijke toepassingen:

Combinatorische veilingen: Combinatorische veilingen kunnen over het algemeen niet uitgevoerd worden zonder een exacte bied-taal. Goalbase talen kunnen gebruikt worden als bied-taal en zijn, soms meer en soms minder, beknopt dan de bied-talen die reeds in gebruik zijn. Wij presenteren een formulering als geheeltallig lineair programmerings probleem van het *Winner Determination Problem* (WDP) waarbij de goalbases als biedingen wor den gebruikt, en een 'branch-and-bound' heuristiek voor het rechtstreeks oplossen van het WDP. Daarnaast presenteren we experimentele resultaten die de haalbaarheid van het gebruik van goalbase talen voor veilingen van gemiddelde grootte aantonen.

Stemming: We beschouwen het probleem van onvoldoend expressieve stemprocedures en komen met een voorstel voor het stemmen met goalbases als stembiljet als mogelijke oplossing. Gewone stemmethodes die een enkele winnaar aanwijzen zijn niet gemakkelijk uit te breiden naar situaties waarbij een groep winnaars moet worden aangewezen, zoals bijvoorbeeld de verkiezing van een comité; dit wordt veroorzaakt door de interactie tussen de kandidaten. We stellen een uitbreiding van 'Approval Voting' voor, waarbij in plaats van oordelen over losse groepen (zoals bij 'Approval Voting' gebruikelijk) voorkeuren over groepen worden weergegeven via preferenties voor eigenschappen die deze groepen al dan niet bezitten.

Samenvattend, dit proefschrift geeft een goed beeld van het nut van goalbase talen voor het representeren van voorkeuren en een indicatie van de potentiële toepassingsgebieden.

Abstract

In this dissertation we present a framework for compactly representing cardinal preferences over combinatorial domains and show the feasibility of using this framework for auctions and voting.

Our framework uses goalbases—sets of weighted propositional formulas—to represent utility functions. We compute the utility of an alternative as the aggregated value of the weights of the goals the alternative satisfies. Goalbase languages are formed by restricting the formulas and weights which may appear in goalbases. Due to their parametric nature, these languages are scattered all across the representational landscape. In order to make practical use of goalbases, we must first know the lay of the land. In particular, we explore the landscape of goalbase languages in three directions:

Expressivity: Given a goalbase language, what utility functions are representable in it? Many goalbase languages with natural definitions correspond exactly to classes of utility functions having well-known properties. Furthermore, we show that some goalbase languages have precisely one representation for any representable utility function, and provide methods for finding these representations.

Succinctness: Given two goalbase languages, are the smallest representations in one significantly smaller than equivalent smallest representations in the other? We systematically compare more than two hundred pairs of languages to determine which languages are more succinct.

Complexity: Given a goalbase language, how difficult is it to answer queries about goalbases which are its members? We consider the computational complexity of finding optimal states for individuals and groups, and finding pessimal states for individuals. These problems tend to be intractable for more expressive languages; for those which are solvable in polynomial time, we provide algorithms demonstrating that.

After determining the properties of many goalbase languages, we consider two possible applications of them:

Combinatorial Auctions: Combinatorial auctions cannot generally be conducted without concise bidding languages. Goalbase languages may be used as bidding languages, and are sometimes more and sometimes less succinct than bidding languages already in use. We give an integer programming formulation of the Winner Determination Problem using goalbases as bids, as well as branch-and-bound heuristics for solving the WDP directly, and present experimental results which demonstrate the feasibility of using goalbase languages for auctions of moderate size.

Voting: We consider the problem of insufficiently expressive voting methods, and suggest voting with goalbases as ballots as a possible remedy. Common single-winner voting methods do not extend well to multi-winner settings like committee elections due to interactions between candidates. We suggest an extension to Approval Voting, where properties of the outcome are approved (or not) rather than particular outcomes as in standard Approval Voting.

In summary, this dissertation provides a clear view of the power of goalbase languages for preference representation, and points to potential areas of application.

Titles in the ILLC Dissertation Series:

ILLC DS-2001-01: **Maria Aloni**
Quantification under Conceptual Covers

ILLC DS-2001-02: **Alexander van den Bosch**
Rationality in Discovery - a study of Logic, Cognition, Computation and Neuropharmacology

ILLC DS-2001-03: **Erik de Haas**
Logics For OO Information Systems: a Semantic Study of Object Orientation from a Categorial Substructural Perspective

ILLC DS-2001-04: **Rosalie Iemhoff**
Provability Logic and Admissible Rules

ILLC DS-2001-05: **Eva Hoogland**
Definability and Interpolation: Model-theoretic investigations

ILLC DS-2001-06: **Ronald de Wolf**
Quantum Computing and Communication Complexity

ILLC DS-2001-07: **Katsumi Sasaki**
Logics and Provability

ILLC DS-2001-08: **Allard Tamminga**
Belief Dynamics. (Epistemo)logical Investigations

ILLC DS-2001-09: **Gwen Kerdiles**
Saying It with Pictures: a Logical Landscape of Conceptual Graphs

ILLC DS-2001-10: **Marc Pauly**
Logic for Social Software

ILLC DS-2002-01: **Nikos Massios**
Decision-Theoretic Robotic Surveillance

ILLC DS-2002-02: **Marco Aiello**
Spatial Reasoning: Theory and Practice

ILLC DS-2002-03: **Yuri Engelhardt**
The Language of Graphics

ILLC DS-2002-04: **Willem Klaas van Dam**
On Quantum Computation Theory

ILLC DS-2002-05: **Rosella Gennari**
Mapping Inferences: Constraint Propagation and Diamond Satisfaction

ILLC DS-2002-06: **Ivar Vermeulen**
A Logical Approach to Competition in Industries

ILLC DS-2003-01: **Barteld Kooi**
Knowledge, chance, and change

ILLC DS-2003-02: **Elisabeth Catherine Brouwer**
Imagining Metaphors: Cognitive Representation in Interpretation and Understanding

ILLC DS-2003-03: **Juan Heguiabehere**
Building Logic Toolboxes

ILLC DS-2003-04: **Christof Monz**
From Document Retrieval to Question Answering

ILLC DS-2004-01: **Hein Philipp Röhrig**
Quantum Query Complexity and Distributed Computing

ILLC DS-2004-02: **Sebastian Brand**
Rule-based Constraint Propagation: Theory and Applications

ILLC DS-2004-03: **Boudewijn de Bruin**
Explaining Games. On the Logic of Game Theoretic Explanations

ILLC DS-2005-01: **Balder David ten Cate**
Model theory for extended modal languages

ILLC DS-2005-02: **Willem-Jan van Hoeve**
Operations Research Techniques in Constraint Programming

ILLC DS-2005-03: **Rosja Mastop**
What can you do? Imperative mood in Semantic Theory

ILLC DS-2005-04: **Anna Pilatova**
A User's Guide to Proper names: Their Pragmatics and Semanics

ILLC DS-2005-05: **Sieuwert van Otterloo**
A Strategic Analysis of Multi-agent Protocols

ILLC DS-2006-01: **Troy Lee**
Kolmogorov complexity and formula size lower bounds

ILLC DS-2006-02: **Nick Bezhanishvili**
Lattices of intermediate and cylindric modal logics

ILLC DS-2006-03: **Clemens Kupke**
Finitary coalgebraic logics

ILLC DS-2006-04: **Robert Špalek**
Quantum Algorithms, Lower Bounds, and Time-Space Tradeoffs

ILLC DS-2006-05: **Aline Honingh**
The Origin and Well-Formedness of Tonal Pitch Structures

ILLC DS-2006-06: **Merlijn Sevenster**
Branches of imperfect information: logic, games, and computation

ILLC DS-2006-07: **Marie Nilsenova**
Rises and Falls. Studies in the Semantics and Pragmatics of Intonation

ILLC DS-2006-08: **Darko Sarenac**
Products of Topological Modal Logics

ILLC DS-2007-01: **Rudi Cilibrasi**
Statistical Inference Through Data Compression

ILLC DS-2007-02: **Neta Spiro**
What contributes to the perception of musical phrases in western classical music?

ILLC DS-2007-03: **Darrin Hindsill**
It's a Process and an Event: Perspectives in Event Semantics

ILLC DS-2007-04: **Katrin Schulz**
Minimal Models in Semantics and Pragmatics: Free Choice, Exhaustivity, and Conditionals

ILLC DS-2007-05: **Yoav Seginer**
Learning Syntactic Structure

ILLC DS-2008-01: **Stephanie Wehner**
Cryptography in a Quantum World

ILLC DS-2008-02: **Fenrong Liu**
Changing for the Better: Preference Dynamics and Agent Diversity

ILLC DS-2008-03: **Olivier Roy**
Thinking before Acting: Intentions, Logic, Rational Choice

ILLC DS-2008-04: **Patrick Girard**
Modal Logic for Belief and Preference Change

ILLC DS-2008-05: **Erik Rietveld**
Unreflective Action: A Philosophical Contribution to Integrative Neuroscience

ILLC DS-2008-06: **Falk Unger**
Noise in Quantum and Classical Computation and Non-locality

ILLC DS-2008-07: **Steven de Rooij**
Minimum Description Length Model Selection: Problems and Extensions

ILLC DS-2008-08: **Fabrice Nauze**
Modality in Typological Perspective

ILLC DS-2008-09: **Floris Roelofsen**
Anaphora Resolved

ILLC DS-2008-10: **Marian Counihan**
Looking for logic in all the wrong places: an investigation of language, literacy and logic in reasoning

ILLC DS-2009-01: **Jakub Szymanik**
Quantifiers in TIME and SPACE. Computational Complexity of Generalized Quantifiers in Natural Language

ILLC DS-2009-02: **Hartmut Fitz**
Neural Syntax

ILLC DS-2009-03: **Brian Thomas Semmes**
A Game for the Borel Functions

ILLC DS-2009-04: **Sara L. Uckelman**
Modalities in Medieval Logic

ILLC DS-2009-05: **Andreas Witzel**
Knowledge and Games: Theory and Implementation

ILLC DS-2009-06: **Chantal Bax**
Subjectivity after Wittgenstein. Wittgenstein's embodied and embedded subject and the debate about the death of man.

ILLC DS-2009-07: **Kata Balogh**
Theme with Variations. A Context-based Analysis of Focus

ILLC DS-2009-08: **Tomohiro Hoshi**
Epistemic Dynamics and Protocol Information

ILLC DS-2009-09: **Olivia Ladinig**
Temporal expectations and their violations

ILLC DS-2009-10: **Tikitu de Jager**
"Now that you mention it, I wonder...": Awareness, Attention, Assumption

ILLC DS-2009-11: **Michael Franke**
Signal to Act: Game Theory in Pragmatics

ILLC DS-2009-12: **Joel Uckelman**
More Than the Sum of Its Parts: Compact Preference Representation Over Combinatorial Domains