More than the sum of its parts: compact preference representation over combinatorial domains
Uckelman, J.D.

Citation for published version (APA):
Uckelman, J. D. (2009). More than the sum of its parts: compact preference representation over combinatorial domains Amsterdam: Institute for Logic, Language and Computation

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
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 \textsc{max-util}, \textsc{min-util}, and \textsc{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 \textsc{max-util} and \textsc{min-util} (Section 5.5) and \textsc{max-cuf} (Section 5.6), followed by an exploration of an alternative version of \textsc{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 $\varphi$ 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., 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 $f, g : \mathbb{N} \to \mathbb{N}$ 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).

- $\text{TIME}(t(n))$ is the class of languages decidable by a deterministic $O(t(n))$-time Turing machine.

- $\text{NTIME}(t(n))$ is the class of languages decidable by a nondeterministic $O(t(n))$-time Turing machine.

- $\mathbb{P} = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$. 
5.2. Background

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

In other words, \( \text{P} \) is the class of languages for which there are deterministic polynomial-time algorithms to decide them, and \( \text{NP} \) is the class of languages for which there are non-deterministic polynomial-time algorithms to decide them. An alternative characterization of \( \text{NP} \), one of which we shall frequently make use, is that \( \text{NP} \) 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 \( \text{NP} \), 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 \( \text{P} \) 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 \( C \), \( \text{co} C = \{ \overline{L} \mid L \in C \} \) is its complementary complexity class.

In particular, we are interested \( \text{coNP} \), 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 \( \text{coP} \) because \( \text{coP} = \text{P} \).

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

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 \text{WIDGET} \text{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 \( f : \{0, 1\}^* \to \{0, 1\}^* \) is \textit{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 \( 1\)
Figure 5.1: Some complexity classes for decision problems.
5.2. Background

$O(g(n))$ 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 $C$ could be reduced, we might say that this problem was at least as hard as every problem in $C$.

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

A problem which is hard for $C$ we call $C$-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 $C$-hardness is not the direct method suggested by the definition—i.e., demonstrating directly that arbitrary languages in $C$ 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 $C$-hard problem, then by definition every problem in $C$ reduces to it; therefore, if we can reduce $A$ to $B$, then by transitivity every problem in $C$ reduced to $B$ also, and hence we have shown $B$ to be $C$-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 \textsc{sat} to be \textsf{NP}\text{-hard}. Since then, a bewildering number of other problems have been shown to be \textsf{NP}\text{-hard}, and we are free to reduce from whichever is most convenient when showing \textsf{NP}\text{-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 $C$ if both $L \in C$ and $L$ is $C$-hard.

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

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

The decision problem \textsc{min-util} is the pessimal version of \textsc{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 \textsc{min-util}).** The decision problem \textsc{min-util}($\Phi, W, F$) is defined as: Given a goalbase $G \in \mathcal{L}(\Phi, W, F)$ and an integer $K$, are all models $M \in 2^{PS}$ 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 $\sigma : \mathbb{R}^* \rightarrow \mathbb{R}$.

A collective utility function takes a tuple of individual utilities as its input, and aggregates them into a single group utility.\footnote{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.} The decision problem \textsc{max-cuf} is like \textsc{max-util}, but lifted from an individual agent to a group of agents.
Definition 5.3.4 (The Decision Problem MAX-CUF). The decision problem MAX-CUF(Φ, W, F, σ) for n agents is defined as: Given goalbases G_1, ..., G_n ∈ L(Φ, W, F), a collective utility function σ, and an integer K, is there a partition (M_1, ..., M_n) of PS such that σ(u_{G_1,F}(M_1), ..., u_{G_n,F}(M_n)) ≥ 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 ∧, ∨, 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 lexicin as the individual aggregator, and some using lexicin 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 \(\mathcal{P}\) or being merely NP-complete to being complete for classes in the Boolean and polynomial hierarchies, such as \(\text{coBH}_2\), \(\Theta^p_2\), \(\Delta^p_2\), and \(\Sigma^p_2\). 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 \((t \land p, -\infty)\) 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 \textsc{max-util} and \textsc{min-util}

Who (if anyone) needs to solve \textsc{max-util} depends on the context in which our preference representation languages are being applied. Take auctions, for example: Whether \textsc{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 \textsc{max-util} if the resources are shareable. Specifically, \textsc{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 \textsc{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 \textsc{max-util} in order to solve the Winner Determination Problem, the complexity of \textsc{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 \textsc{max-util}, as it contains \textsc{max-util} as a subproblem.

Finally, for an agent herself it is useful to solve \textsc{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 \textsc{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 \textsc{max-util} becomes even more important in that setting.

Our strategy in each of the following subsections is as follows: Where \textsc{max-util} is polynomial, we show that for the most expansive languages for which we know it holds. Where \textsc{max-util} is \textsc{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 \textsc{max-util} trivializes to the prototypical \textsc{NP}-complete problem \textsc{sat}, since deciding \textsc{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 \textsc{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 \( \Phi \) and polynomially-computable aggregation function \( F \), \( \text{MAX-UTIL}(\Phi, Q, F) \in \mathbb{NP} \).

**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 \( \mathbb{NP} \)-completeness results for MAX-UTIL implicitly rely on this lemma for the \( \mathbb{NP} \) 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 \( \text{MAX-UTIL}(\text{forms}, Q, \Sigma) \) in order to show that MAX-UTIL is \( \mathbb{NP} \)-complete for the unrestricted language.

**Theorem 5.5.2.** \( \text{MAX-UTIL}(\text{forms}, Q, \Sigma) \) is \( \mathbb{NP} \)-complete.

**Proof.** By reduction from the well-known \( \mathbb{NP} \)-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 \( \mathbb{NP} \)-completeness even after this dramatic reduction in our stock of formulas.

**Theorem 5.5.3.** \( \text{MAX-UTIL}(k\text{-cubes}, Q^+, \Sigma) \) is \( \mathbb{NP} \)-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 \( \mathcal{P}\mathcal{S} \) and an integer \( K \), check whether there is a model \( M \in 2^{\mathcal{P}\mathcal{S}} \) which satisfies at least \( K \) of the \( k \)-cubes in \( C \). \( \text{MAX-UTIL}(k\text{-cubes}, Q^+, \Sigma) \) is a weighted version of MAX \( k \)-CONSTRAINT SAT, which is \( \mathbb{NP} \)-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 \( (p \land q, \pi^e + \sqrt{2}) \) is required, users will find that that \( \mathbb{Q} \) affords them arbitrarily close rational approximations.
Note that we are able to prove the stronger result, namely that we have \( \mathsf{NP} \)-completeness even when we know that all formulas in our goalbases are satisfiable formulas, due to the fact that \( \mathsf{UNSAT} \) 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 \( \text{MAX-UTIL}(2\text{-cubes}, Q^+, \Sigma) \) is a subproblem of \( \text{MAX-UTIL}(\text{forms}, Q, \Sigma) \), we may state as a corollary the following even stronger result for the unrestricted language:

**Corollary 5.5.4.** \( \text{MAX-UTIL}(\text{forms}, Q, \Sigma) \) is \( \mathsf{NP} \)-complete, even if goalbases contain only satisfiable formulas.

In the remaining \( \mathsf{NP} \)-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.** \( \text{MAX-UTIL}(k\text{-clauses}, Q^+, \Sigma) \) is \( \mathsf{NP} \)-complete for \( k \geq 2 \).

**Proof.** \( \text{MAX-UTIL}(2\text{-clauses}, Q^+, \Sigma) \) is a weighted version of the well-known \( \mathsf{NP} \)-complete problem \( \text{MAX } 2\text{-SAT} \) [Garey and Johnson, 1979]. Furthermore, \( \text{MAX-UTIL}(k\text{-clauses}, Q^+, \Sigma) \) contains \( \text{MAX-UTIL}(k\text{-clauses}, Q^+, \Sigma) \) for \( k \geq 2 \).

We have now seen that neither short cubes nor short clauses will keep \( \text{MAX-UTIL} \) from being \( \mathsf{NP} \)-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.** \( \text{MAX-UTIL}(k\text{-spcubes}, Q, \Sigma) \) is \( \mathsf{NP} \)-complete for \( k \geq 2 \).

**Proof.** We show \( \mathsf{NP} \)-hardness for \( k = 2 \) by reduction from \( \text{MAX } 2\text{-SAT} \) [Garey and Johnson, 1979], using a construction previously employed by Chevaleyre et al. [2008a] to show \( \mathsf{NP} \)-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 \( K \leq |S| \). \( \text{MAX } 2\text{-SAT} \) 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 \( p \lor q \in S \), add \((p, 1), (q, 1)\), and \((p \land q, -1)\) to \( G \).
- For any clause \( p \lor \neg q \in S \), add \((\top, 1), (q, -1)\), and \((p \land q, 1)\) to \( G \).
- For any clause \( \neg p \lor \neg q \in S \), add \((\top, 1)\) and \((p \land q, -1)\) 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, $Q, \Sigma$) must be at least as hard as MAX 2-SAT.

Similarly, we cannot avoid \textbf{NP}-completeness by using short positive clauses if we want also to have negative weights.

**Theorem 5.5.7.** \textbf{MAX-UTIL}($k$-pclauses, $Q, \Sigma$) is \textbf{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 $p \lor q \in S$, add $(p \lor q, 1)$ to $G$.
- For any clause $p \lor \neg q \in S$, add $(\top, 1)$, $(p, 1)$, and $(p \lor q, -1)$ to $G$.
- For any clause $\neg p \lor \neg q \in S$, add $(\top, 1)$, $(p, -1)$, $(q, -1)$, and $(p \lor q, 1)$ 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 $P \neq \textbf{NP}$). We might try to avoid \textbf{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 \textbf{NP}-complete [Papadimitriou, 1994a, Corollary, p. 176]. Unfortunately, the restriction to Horn clauses is still not severe enough:

**Theorem 5.5.8.** \textbf{MAX-UTIL}($k$-Horn, $Q^+, \Sigma$) is \textbf{NP}-complete for $k \geq 2$.

**Proof.** The problem MAX HORN 2-SAT is \textbf{NP}-complete [Jaumard and Simeone, 1987, Proposition 3.1]. We exhibit a polynomial reduction from MAX HORN 2-SAT to MAX-UTIL($2$-Horn, $Q^+, \Sigma$): 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, $Q^+, \Sigma$) is contained in MAX-UTIL($k$-Horn, $Q^+, \Sigma$) for $k \geq 2$, so MAX-UTIL($k$-Horn, $Q^+, \Sigma$) is \textbf{NP}-complete for $k \geq 2$. 

\qed
5.5. The Complexity of \textsc{max-util} and \textsc{min-util}

1. For each \((\bigwedge X, w) \in G\):
   \begin{enumerate}
   \item Let \(X' = \{x \mid x \in X\} \cup \{ar{x} \mid \neg x \in X\}\).
   \item Put \((\bigwedge X', w) \in G'_0\).
   \end{enumerate}

2. Normalize \(G'_0\) to \([-1, 1]\).

3. Let \(\delta = \sum_{(\phi, w) \in G'_0} |w|\).

4. Let \(\alpha = \delta + 1\), and \(\beta = -3\delta - 3\).

5. For each \(x \in \mathcal{PS}\), put \((x, \alpha), (\bar{x}, \alpha), (x \land \bar{x}, \beta) \in G'_1\).

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 \textsc{NP}-completeness guarantees that there is an abundance of different proofs reducing one \textsc{NP}-complete problem to another. In principle, any two \textsc{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 \textsc{NP}-hardness reductions from \textsc{traveling salesman} [Garey and Johnson, 1979] or even \textsc{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 \textsc{max-util}(k\text{-cubes}, \mathbb{Q}, \Sigma) to \textsc{max-util}(k\text{-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:

\textbf{Proof.} \textsc{max-util}(k\text{-cubes}, \mathbb{Q}^+, \Sigma) is \textsc{NP}-complete for \(k \geq 2\) by Theorem 5.5.3; \textsc{max-util}(k\text{-cubes}, \mathbb{Q}, \Sigma) contains it for any fixed \(k\), so is \textsc{NP}-complete also. Now we exhibit a polynomial reduction from \textsc{max-util}(k\text{-cubes}, \mathbb{Q}, \Sigma) to \textsc{max-util}(k\text{-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 \(\overline{\mathcal{PS}} = \{\overline{p} \mid p \in \mathcal{PS}\}\). 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 $|u_{G'}(M) - u_{G'}(N)| \leq \delta$. $\delta$ 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 $u_{G'}(A \cup \{x\})$ and $u_{G'}(A \cup \{x, \bar{x}\})$:

$$u_{G'}(A \cup \{x\}) = u_{G'_0}(A \cup \{x\}) + u_{G'_1}(A \cup \{x\})$$
$$= u_{G'_0}(A \cup \{x\}) + u_{G'_1}(A) + w'_x$$
$$= u_{G'_0}(A \cup \{x\}) + u_{G'_1}(A) + \delta + 1$$
$$\geq u_{G'_0}(A) - \delta + u_{G'_1}(A) + \delta + 1$$
$$= u_{G'_0}(A) + u_{G'_1}(A) + 1$$
$$> u_{G'_0}(A) + u_{G'_1}(A) = u_{G'}(A)$$

Similarly, $u_{G'}(A \cup \{x, \bar{x}\}) > u_{G'}(A)$. Finally,

$$u_{G'}(A \cup \{x, \bar{x}\}) = u_{G'_0}(A \cup \{x, \bar{x}\}) + u_{G'_1}(A \cup \{x, \bar{x}\})$$
$$= u_{G'_0}(A \cup \{x, \bar{x}\}) + u_{G'_1}(A) + w'_x + w'_\bar{x}$$
$$= u_{G'_0}(A \cup \{x, \bar{x}\}) + u_{G'_1}(A) - \delta - 1$$
$$\leq u_{G'_0}(A) + \delta + u_{G'_1}(A) - \delta - 1$$
$$= u_{G'_0}(A) + u_{G'_1}(A) - 1$$
$$< u_{G'_0}(A) + u_{G'_1}(A) = u_{G'}(A)$$

(Lemma 5.5.9) $\square$

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 $u_{G'}$, since all full, bivalent models have the same value ($w_x = w_{\bar{x}}$ and $w_{x \land \bar{x}} = w_{y \land \bar{y}}$ 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 \((x, \alpha), (\bar{x}, \alpha), (x \lor \bar{x}, \beta) \in G'_1\).
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 \(\alpha\), which is order-preserving; and increases \(w_{x \land \bar{x}}\) by \(\beta\), which has no effect at all since \(x \land \bar{x}\) 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 \(\text{MAX-UTIL}(k\text{-cubes}, \mathbb{Q}, \Sigma)\) to \(\text{MAX-UTIL}(k\text{-pclauses}, \mathbb{Q}, \Sigma)\). 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 \(\text{MAX-UTIL}(k\text{-pclauses}, \mathbb{Q}, \Sigma)\) is \(\text{NP}\)-complete.

This method can easily be adapted to obtain the analogous \(\text{NP}\)-completeness result for positive clauses, giving an alternative proof of Theorem 5.5.7:

**Proof.** Similar to the proof for the \(\text{NP}\)-completeness of \(\text{MAX-UTIL}(k\text{-pclauses}, \mathbb{Q}, \Sigma)\). 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. \((x \lor \bar{x}\) is true in every full, bivalent model and hence the disjunctive weights do not disturb the ordering.) Hence by the same argument, \(\text{MAX-UTIL}(k\text{-clauses}, \mathbb{Q}, \Sigma)\) reduces polynomially to \(\text{MAX-UTIL}(k\text{-pclauses}, \mathbb{Q}, \Sigma)\), and hence \(\text{MAX-UTIL}(k\text{-pclauses}, \mathbb{Q}, \Sigma)\) is \(\text{NP}\)-complete.

### 5.5.2 Easiness Results for MAX-UTIL

The previous subsection might leave us wondering whether we can ever avoid \(\text{NP}\)-completeness for \(\text{MAX-UTIL}\) for sum languages, as we found \(\text{NP}\)-completeness...
wherever we looked. For two sum languages (and all their sublanguages), however, we can obtain easiness results. The first of these is $\text{MAX-UTIL}(\text{pforms}, Q^+, \Sigma)$, where we take advantage of the fact that the largest optimal state is easy to construct.

**Theorem 5.5.10.** $\text{MAX-UTIL}(\text{pforms}, Q^+, \Sigma) \in P$.

*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 $\text{MAX-UTIL}(\text{pforms}, Q^+, \Sigma)$; 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 $\mathcal{U}(\text{literals}, Q, \Sigma)$ are modular to decide for each item whether it should be in or out of an optimal model.

**Theorem 5.5.11.** $\text{MAX-UTIL}(\text{literals}, Q, \Sigma) \in P$.

*Proof.* A simple polynomial algorithm: Fix a goalbase $G \in \mathcal{L}(\text{literals}, Q, \Sigma)$. Keep for each atom $p$ a number $\delta_p$, the difference between the sum of $p$’s positive occurrences and the 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 $(-p, 5)$, we subtract 5 from $\delta_p$.) 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 $\delta_p$.

In contrast to the hardness results we have for most sum languages, solving $\text{MAX-UTIL}$ for any max language is trivial:

**Theorem 5.5.12.** $\text{MAX-UTIL}(\Phi, W, \text{max})$ is linear in the size of the goalbase, for any $\Phi \subseteq \mathcal{L}_{PS}$ and any $W \subseteq Q$, so long as goalbases contain only satisfiable formulas.

*Proof.* An algorithm solving $\text{MAX-UTIL}$ for any max language simply has to iterate over the formulas in the goalbase, answer affirmatively as soon as it encounters a $(\varphi, w)$ for which $w \geq K$, and answer negatively otherwise.

This complexity result requires some discussion. First, without the restriction to satisfiable formulas, $\text{MAX-UTIL}(\Phi, W, \text{max})$ is $\text{NP}$-complete, as lifting this restriction imposes the additional requirement of checking whether $\varphi$ 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 \((\phi, w)\) 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 \( \phi \), \( fsat \) will return either a satisfying model \( M \), or “no” if there is no satisfying model. If we somehow know already that \( \phi \) 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, \( FP \subseteq TFNP \subseteq FNP \), but no more beyond that is known. If \( FP = TFNP \), this would imply that \( P = NP \cap coNP \), 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 \( L(\text{forms}, W, \text{max}) \) 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,
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 \( \{ \{ \top, 1 \} \}, 1 \) 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 \( \Phi \), MIN-UTIL(\( \Phi \), Q, \( \Sigma \)) \( \in \) 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.** MIN-UTIL(form, Q, \( \Sigma \)) is coNP-complete.

**Proof.** coNP membership follows from Lemma 5.5.13. For coNP-hardness: Let \( \varphi \) be an instance of UNSAT, and \( \{ \{ \neg \varphi, 1 \} \}, 1 \) an instance of MIN-UTIL(form, Q, \( \Sigma \)). It is easy to see that if \( \varphi \) is not satisfiable, then \( u_{\{ \neg \varphi, 1 \}}(M) = 1 \) for all models \( M \), and vice versa. Hence UNSAT reduces to MIN-UTIL(form, 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 \( C \) 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. MAX-UTIL(\( \Phi \), W, \( \Sigma \)) \( \in \) \( C \) iff MIN-UTIL(\( \Phi \), \( -W \), \( \Sigma \)) \( \in \) co\( C \).

2. MAX-UTIL(\( \Phi \), W, \( \Sigma \)) is \( C \)-hard iff MIN-UTIL(\( \Phi \), \( -W \), \( \Sigma \)) is co\( C \)-hard.

**Proof.** Recall that

\[
\text{MAX-UTIL}(\Phi, W, \Sigma) = \{ \langle G, K \rangle \mid \langle G, K \rangle \notin \text{MAX-UTIL}(\Phi, W, \Sigma) \}.
\]

Since MAX-UTIL is the complementary problem to MAX-UTIL we have by definition that MAX-UTIL \( \in \) co\( C \) iff MAX-UTIL \( \in \) \( C \). If \( G \) is a goalbase, let \(-G = \{ (\varphi, -w) \mid (\varphi, w) \in G \} \). Clearly we have that

\[
\langle G, K \rangle \in \text{MAX-UTIL}(\Phi, W, \Sigma) \iff \langle -G, -K \rangle \in \text{MIN-UTIL}(\Phi, -W, \Sigma)
\]

which shows that MAX-UTIL(\( \Phi \), W, \( \Sigma \)) and MIN-UTIL(\( \Phi \), \( -W \), \( \Sigma \)) are interreducible, and hence that MIN-UTIL(\( \Phi \), \( -W \), \( \Sigma \)) \( \in \) co\( C \) also. Similarly, if MAX-UTIL is \( C \)-hard, then MAX-UTIL is co\( C \)-hard by definition, and due to the interreducibility of MAX-UTIL(\( \Phi \), W, \( \Sigma \)) and MIN-UTIL(\( \Phi \), \( -W \), \( \Sigma \)), we have that MIN-UTIL(\( \Phi \), \( -W \), \( \Sigma \)) is co\( C \)-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\((k\text{-}spcubes, Q, \Sigma)\) is \textit{coNP}-complete for \(k \geq 2\).

**Theorem 5.5.17.** MIN-UTIL\((k\text{-}pclauses, Q, \Sigma)\) is \textit{coNP}-complete for \(k \geq 2\).

**Theorem 5.5.18.** MIN-UTIL\((\text{literals}, Q, \Sigma)\) \(\in\) \(\text{P}\).

We may apply Lemma 5.5.15 in these cases because the rationals are closed under negation (that is, \(Q = -Q\)). Theorem 5.5.18 relies on the fact that deterministic complexity classes are closed under complementation, so we may say \(P\) there instead of \textit{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\((\text{literals}, Q, \Sigma)\) 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\((\text{literals}, Q, \Sigma)\) is \(O(n \log n)\).)

For MIN-UTIL\((pforms, Q^+, \Sigma)\), 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\((pforms, Q^+, \Sigma)\) \(\in\) \(\text{P}\).

**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\((pforms, Q^+, \Sigma)\) iff \(u_G(\emptyset) \geq K\), which can be verified polynomially. \(\square\)

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\((k\text{-}clauses, Q^+, \Sigma)\) is \textit{coNP}-complete for \(k \geq 2\).

**Proof.** \textit{coNP} membership follows from Lemma 5.5.13. For \textit{coNP}-hardness, we give a reduction from (the complement of) MIN 2-SAT, which is \textit{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\((2\text{-}clauses, Q^+, \Sigma)\) instance \(\{\{\varphi, 1\} \mid \varphi \in C\}, K + 1\)\). 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, \(\{\{\varphi, 1\} \mid \varphi \in C\}, K + 1\) \(\in\) MIN-UTIL\((2\text{-}clauses, Q^+, \Sigma)\) iff \(\langle C, K \rangle \notin\) MIN 2-SAT. Hence MIN-UTIL\((2\text{-}clauses, Q^+, \Sigma)\) is \textit{coNP}-complete. \(\square\)

**Theorem 5.5.21.** MIN-UTIL\((k\text{-}cubes, Q^+, \Sigma)\) is \textit{coNP}-complete for \(k \geq 2\), even if goalbases contain only satisfiable formulas.
Proof. \textit{coNP} membership follows from Lemma 5.5.13. For \textit{coNP}-hardness: We first note that \textsc{min 2-constraint sat}, which is the minimization analog of \textsc{max 2-constraint sat}, is \textsc{NP}-complete: Let \(\langle C, K \rangle\) be a \textsc{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 \(\langle C', |C| - K \rangle \in \textsc{min 2-constraint sat}\) iff \(\langle C, K \rangle \in \textsc{max 2-sat}\), since every false member of \(C\) is a true member of \(C'\). Having established that \textsc{min 2-constraint sat} is \textsc{NP}-hard, its complement may be reduced to \textsc{min-util}(\textit{k-cubes}, \(Q^+, \Sigma\)) using the same construction as in Theorem 5.5.20.

We have seen that for sum languages, \textsc{min-util} behaves similarly to \textsc{max-util}. However, this is not the case for max languages:

**Theorem 5.5.22.** \textsc{min-util}(\textit{forms}, \(Q\), \textsc{max}) is \textit{coNP}-complete.

Proof. For \textit{coNP} membership: Any purported counterexample state \(M\) is polynomially checkable, simply evaluating \(u_{G,\text{max}}(M)\) to see if it is less than \(K\).

For \textit{coNP}-hardness: The reduction from \textsc{unsat} to \textsc{min-util}(\textit{forms}, \(Q\), \(\Sigma\)) 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 \textsc{unsat} to \textsc{min-util}(\textit{forms}, \(Q\), \textsc{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 \textsc{min-util}(\textit{forms}, \(Q\), \textsc{max}):**

**Theorem 5.5.23.** \textsc{min-util}(\textit{forms}, \(Q\), \textsc{max}) \(\in \text{P}\), when restricted to goalbases containing no superfluous formulas.

Proof. Since no \((\varphi, w) \in G\) is superfluous, any such \(\varphi\) will determine the value of \(u_{G,\text{max}}(M)\) for some model \(M\). Hence, the value of the worst state may be found simply by finding the \((\varphi, w)\) with the least \(w\). If that \(w \geq K\), the \textsc{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 \textsc{min-util} remains polynomial in the absence of any further restrictions. This may be seen in the following three theorems.

**Theorem 5.5.24.** \textsc{min-util}(\textit{pforms}, \(Q\), \textsc{max}) \(\in \text{P}\).

Proof. Same proof as for \textsc{min-util}(\textit{pforms}, \(Q\), \(\Sigma\)) in Theorem 5.5.19. The least-valued state is always \(\emptyset\). Check whether \(u_{G,\text{max}}(\emptyset) \geq K\).

**Theorem 5.5.25.** \textsc{min-util}(\textit{liters}, \(Q\), \textsc{max}) \(\in \text{P}\).

Proof. We present a polynomial-time algorithm: Find \(\delta_p\) 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 \(u_{G,\text{max}}(\mathcal{PS} \setminus M) \geq K\).
5.5. The Complexity of max-util and min-util

**Theorem 5.5.26.** min-util\((cubes, Q, \max) \in \mathbb{P}\).

*Proof.* We argue that it is easy to identify and remove superfluous formulas from goalbases in \(L(cubes, Q, \max)\); once \(G\) contains no superfluous formulas, we may invoke Theorem 5.5.23 to show that min-util\((cubes, Q, \max) \in \mathbb{P}\).

First, observe that when \(X, Y, X', Y' \subseteq \mathcal{P}\mathcal{S}\),
\[
| = (\bigwedge X \land \bigwedge \neg Y) \rightarrow (\bigwedge X' \land \bigwedge \neg Y')
\]
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 \(G \in L(cubes, Q, \max)\) like so:

For each pair of cubes \((\bigwedge X \land \bigwedge \neg Y, w), (\bigwedge X' \land \bigwedge \neg Y', w') \in G\), if \(w' > w\) and either \(X' \subseteq X\) and \(Y' \subseteq Y\) or \(X \cap Y \neq \emptyset\), then \((\bigwedge X \land \bigwedge \neg Y, w)\) 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 \(w \geq K\). \(\square\)

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 \(L(2-p\text{clauses}, Q, \Sigma)\), \(L(2-\text{spcubes}, Q, \Sigma)\), \(L(2-cubes, Q^+, \Sigma)\), or \(L(2-\text{clauses}, Q^+, \Sigma)\). This covers every sum language mentioned in Chapter 3 except \(L(p\text{forms}, Q^+, \Sigma)\) and \(L(\text{literals}, Q, \Sigma)\) and their sublanguages, for which max-util is in \(\mathbb{P}\). Theorem 5.5.12 shows that max-util is in \(\mathbb{P}\) for all max languages.

For sum languages, min-util is like max-util but reflected into complementary complexity classes—NP into coNP, \(\mathbb{P} = \text{coP}\) 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 \(L(cubes, Q, \max)\) and \(L(\text{forms}, Q, \max)\) (see Corollary 3.5.5),...
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 lexicin 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(\( \Phi, W, F, \sigma \)) \( \in \) NP whenever \( F \) and \( \sigma \) are polynomially-computable functions.

This holds because whenever \( F \) and \( \sigma \) 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 \( f: \mathbb{R}^* \to \mathbb{R} \) 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 \( (\varphi, w) \) 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(\( \Phi, W, F, \sigma \)) is at least as hard as MAX-UTIL(\( \Phi, W, F \)) for any singleton-consistent \( \sigma \).
Here, singleton consistency ensures that $\sigma$ behaves as the identity function when only a single agent is involved, and hence for such $\sigma$ 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 $\sigma$:

1. MAX-CUF($2$-pclauses, $Q$, $\Sigma$, $\sigma$),
2. MAX-CUF($2$-spcubes, $Q$, $\Sigma$, $\sigma$),
3. MAX-CUF($2$-clauses, $Q^+$, $\Sigma$, $\sigma$),
4. MAX-CUF($2$-cubes, $Q^+$, $\Sigma$, $\sigma$).

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—$L(pforms, Q^+, \Sigma)$, $L(literals, Q, \Sigma)$, and $L(forms, Q, max)$—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 $L(pforms, Q^+, \Sigma)$ and $L(literals, Q, \Sigma)$:

**Theorem 5.6.6.**

1. MAX-CUF($pforms$, $Q^+$, $\Sigma$, max) $\in$ P.
2. MAX-CUF($literals$, $Q$, $\Sigma$, 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 $L(literals, Q, \Sigma)$:

**Theorem 5.6.7.** MAX-CUF($literals$, $Q$, $\Sigma$, $\Sigma$) $\in$ P.

**Proof.** For each $a \in PS$, allocate $a$ to the agent $i$ who maximizes

$$u_{G_i, \Sigma}(\{a\}) + \sum_{a \in PS \backslash \{a\}} u_{G_j, \Sigma}(\emptyset).$$

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. \( \text{max-cuf}(\Phi, W, \text{max}, \text{max}) \) is linear in the combined size of the goalbases, for any \( \Phi \subseteq L_{PS} \) and any \( W \subseteq Q \), 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 \( \text{NP} \)-complete. Recall that both \( \text{max-cuf}(2\text{-cubes}, Q^+, \Sigma, \Sigma) \) and \( \text{max-cuf}(2\text{-spcubes}, Q, \Sigma, \Sigma) \) are \( \text{NP} \)-complete. We might ask whether their “intersection”, \( \text{max-cuf}(2\text{-spcubes}, Q^+, \Sigma, \Sigma) \) is also \( \text{NP} \)-complete. Let us approach from a different direction. We can say that the \( \text{max-cuf}(\text{spcubes}, Q^+, \Sigma, \Sigma) \) problem is identical to the single-minded bidder allocation problem, proved to be \( \text{NP} \)-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. \( \text{max-cuf}(3\text{-spcubes}, Q^+, \Sigma, \Sigma) \) is \( \text{NP} \)-complete.

Proof. The problem \text{set packing} asks whether given a collection \( C \) of sets and an integer \( K \), there are at least \( K \) mutually disjoint sets in \( C \). \text{set packing} is \( \text{NP} \)-complete, even when all \( C \in C \) have \(|C| = 3 \) [Garey and Johnson, 1979].

We reduce \text{set packing} for sets of size 3 to \( \text{max-cuf}(3\text{-spcubes}, Q^+, \Sigma, \Sigma) \). For each \( C \in C \), 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 \( \langle \{G_C\}_{C \in C}, K \rangle \in \text{max-cuf}(3\text{-spcubes}, Q^+, \Sigma, \Sigma) \) iff \( \langle C, K \rangle \in \text{set packing} \) restricted to size-3 sets. \( \square \)

Note that this reduction does not go through for 2-spcubes, since \text{set packing} is in \( \text{P} \) when all \( C \in C \) have \(|C| \leq 2 \). This is tantalizingly close to the result we were seeking, but whether \( \text{max-cuf}(2\text{-spcubes}, Q^+, \Sigma, \Sigma) \) is \( \text{NP} \)-complete remains open.

Several of the languages mentioned in Theorem 5.6.5 may be restricted quite severely and yet \( \text{max-cuf} \) remains \( \text{NP} \)-complete for them over a variety of aggregators.

Theorem 5.6.10. All of the following problems are \( \text{NP} \)-complete:

1. \( \text{max-cuf}(2\text{-spcubes}, \{0, 1\}, \text{max}, \Sigma) \)
2. \( \text{max-cuf}(2\text{-spcubes}, \{0, 1\}, F, \sigma) \), where \( F \in \{\text{max}, \Sigma\} \) and \( \sigma \in \{\text{min}, \Pi\} \).

Proof. For the first case, we follow van Hoesel and Müller [2001, Theorem 2] by reducing the known \( \text{NP} \)-complete problem \text{triplpartite matching} [Karp, 1972] to \( \text{max-cuf}(2\text{-spcubes}, \{0, 1\}, \text{max}, \Sigma) \). Instances of \text{triplpartite matching}
Chapter 5. Complexity

are \( (X, Y, Z, T) \), where the sets \( X, Y, Z \) are such that \( |X| = |Y| = |Z| \) and \( T \subseteq X \times Y \times Z \). An instance \( (X, Y, Z, T) \) 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-spacubes. For each \( (x, y, z) \in T \), put \( (y \land z, 1) \in G_x \).

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 \( (X, Y, Z, T) \in \text{TRIPARTITE MATCHING} \) iff \( \langle \{G_x\}_{x \in X}, |X| \rangle \in \text{MAX-CUF}(2\text{-spacubes}, \{0, 1\}, \text{max}, \Sigma) \).

In the other cases, use the same reduction from \( \text{TRIPARTITE MATCHING} \) 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 \textsc{set packing}) apply to \( \text{MAX-CUF}(\text{pforms}, \mathbb{Q}, \text{max}, \text{min}) \), which contains the problem \( \text{MAX-CUF}(2\text{-spacubes}, \{0, 1\}, \text{max}, \text{min}) \) that we have just proved to be \( \text{NP}\)-complete.

Finally, we give a slightly different result for \( \text{MAX-CUF} \) 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 \( \text{MAX-CUF} \) 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.** \( \text{MAX-CUF}(2\text{-pucubes}, \{1, 2\}, \text{max}, \Pi) \) is \( \text{NP}\)-complete.

**Proof.** We first exhibit a reduction from \( \text{MAX-CUF}(2\text{-pucubes}, \mathbb{Q}^+, \text{max}, \Sigma) \) to \( \text{MAX-CUF}(2\text{-pucubes}, \mathbb{Q}^+, \text{max}, \Pi) \). 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 \( \text{MAX-CUF}(2\text{-pucubes}, \mathbb{Q}^+, \text{max}, \Pi) \) 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 \( \text{MAX-CUF} \) problem must be at least as hard as the utilitarian \( \text{MAX-CUF} \) problem.

Then, observe that the same reduction works for the restricted case of \( \text{MAX-CUF}(2\text{-pucubes}, \{0, 1\}, \text{max}, \Sigma) \) to \( \text{MAX-CUF}(2\text{-pucubes}, \{1, 2\}, \text{max}, \Pi) \), the former of which was proven to be \( \text{NP}\)-complete in Theorem 5.6.10. \( \square \)
5.6. The Complexity of Collective Utility Maximization

<table>
<thead>
<tr>
<th>Decision Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX-CUF pforms</td>
<td>$Q^+$</td>
</tr>
<tr>
<td>MAX-CUF literals</td>
<td>$Q$</td>
</tr>
<tr>
<td>MAX-CUF literals</td>
<td>$Q$</td>
</tr>
<tr>
<td>MAX-CUF satisfiable $\varphi$</td>
<td>$Q$</td>
</tr>
<tr>
<td>MAX-CUF 2-cubes</td>
<td>$Q^+$</td>
</tr>
<tr>
<td>MAX-CUF 2-clauses</td>
<td>$Q^+$</td>
</tr>
<tr>
<td>MAX-CUF 2-pclauses</td>
<td>$Q$</td>
</tr>
<tr>
<td>MAX-CUF 2-spcubes</td>
<td>$Q$</td>
</tr>
<tr>
<td>MAX-CUF 3-spcubes</td>
<td>$Q^+$</td>
</tr>
<tr>
<td>MAX-CUF 2-spcubes</td>
<td>${0, 1}$</td>
</tr>
<tr>
<td>MAX-CUF 2-spcubes</td>
<td>${0, 1}$</td>
</tr>
<tr>
<td>MAX-CUF 2-spcubes</td>
<td>${0, 1}$</td>
</tr>
<tr>
<td>MAX-CUF 2-spcubes</td>
<td>${0, 1}$</td>
</tr>
<tr>
<td>MAX-CUF 2-spcubes</td>
<td>${0, 1}$</td>
</tr>
<tr>
<td>MAX-CUF 2-pcubes</td>
<td>${1, 2}$</td>
</tr>
</tbody>
</table>

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($\text{literals, } Q, \Sigma, \text{min}$) and MAX-CUF($\text{literals, } Q, \Sigma, \Pi$). 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($\text{2-sp-cubes, } \{0, 1\}, \Sigma, \Sigma$) 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 \( G \in L(\Phi, W, F) \), find a model \( M \in 2^{PS} \) 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(forms, Q, Σ) 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 \( O(|PS|) \) 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]:
5.7. An Alternate Formulation of \textsc{max-util} 123

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

\[
\sum_{(\varphi,w) \in G \atop w < 0} w,
\]

nor more than

\[
\sum_{(\varphi,w) \in G \atop w > 0} w,
\]

so it follows that \( \Omega \), the value of an optimal state for \( G \), lies in this range. Without loss of generality, multiply out 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 \textsc{max-util} oracle to do a binary search for \( \Omega \). Ask the oracle whether \( (G, \ell + 2^{n-1}) \in \textsc{max-util} \). 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 \( \Omega \) in \( O(\log(h - \ell)) \) steps. Now recover the original (unmultiplied) \( \Omega \) by dividing by whatever factor we multiplied by to eliminate fractions.

2. **Find an optimal state:** Recall that \( \varphi[\nu/\omega] \) is the formula \( \varphi \) with all occurrences of \( \omega \) replaced by \( \nu \), and \( G[\nu/\omega] \) the goalbase with the same substitution applied to all of its formulas. For each item \( p \in \mathcal{P} \mathcal{S} \): Use the \textsc{max-util} oracle to decide whether \( G[\top/p] \) can yield \( \Omega \) 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 \textsc{max-util} by making a single call to a function problem \textsc{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 \( (\varphi, w) \in \mathcal{L}(\Phi, W, \Sigma), p \in \mathcal{P} \mathcal{S} \), and \( p \) occurs in \( \varphi \), but that \( \varphi[\top/p] \notin \Phi \). In this case, we cannot query a \textsc{max-util}(\( \Phi, W, \Sigma \)) oracle about \( (G[\top/p], K) \), 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 \( (G[\top/p], K) \notin \textsc{max-util}(\Phi, W, \Sigma) \), 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 $p \land \neg p$ 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 $p \lor \neg p$ 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 $(\varphi, w)$ from $G$ instead of substituting $\top$ into it, and reset $K$ to $K - w$ to account for having satisfied $(\varphi, w)$. 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 $C$ of goalbases:

$$C = \{ ((\varphi_i, w_i))_i \mid \bigwedge \varphi_i \text{ is satisfiable and all } w_i \geq 0 \}.$$  

Here, it is possible that we might leave the class $C$ on our first substitution. The goalbase $G = \{ (p \rightarrow q, 1), (\neg q, 1)\} \in C$, but $G[\top/p]$ clearly is not since $\{ T \rightarrow q, \neg q \}$ is not a satisfiable set. While the decision problem for $C$ 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\textsuperscript{*}).** Given a goalbase $G \in L$ and an atom $p \in PS$, 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 $\{ (p \land \neg q, 1), (\neg p \land q, 1)\}$ 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\textsuperscript{*} problem $|PS|$ 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* $\text{LM}(S)$ of $S$ is a smallest set $M \subseteq \mathcal{P}S$ 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 $p \in \mathcal{P}S$, is $p \in \text{LM}(S)$?

**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 \textit{negated conjunctions}; we can think of them as “negative synergies”, or exclusions of certain combinations of goods: “A committee with \textit{both} Alice \textit{and} Bob on it would be a disaster.”, “I would appreciate not having \textit{both} my defense \textit{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, \textsc{max-util} is \(\textbf{NP}\)-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 \textsc{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 \textsc{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 \textsc{max-util}\(^*\) for the resulting languages, we take a problem which has the desired complexity and find a class of goalbases whose \textsc{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, \((\lnot a \lor b, 1)\) translates into positive cubes as \(\{(\top, 1), (a, -1), (a \land b, 1)\}\), while \((\lnot a \lor \lnot b, 1)\) becomes \(\{(\top, 1), (a \land b, -1)\}\). 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 \(\text{P}\)-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 \(\text{P}\)-complete goalbase classes related to Horn clauses.
5.7. An Alternate Formulation of \textsc{max-util}

**PLP Goalbases**

**Definition 5.7.9.** The language $\mathcal{L}_{\text{PLP}}$ of PLP goalbases consists of all goalbases

$$G = \{(\varphi_i, w_i)\} \cup \left\{ \left( p, -\frac{m}{|\mathcal{P}S| + 1} \right) \right\}_{p \in \mathcal{P}S}$$

where

- the $\varphi_i$ are strict Horn clauses,
- the $w_i$ are positive, and
- $m = \min_i \{w_i\}$.

$L_P(G) = \{\varphi_i\}$ 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 $i$,

$$w_i > \sum_{p \in \mathcal{P}S} \frac{m}{|\mathcal{P}S| + 1}.$$  

**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., $\text{LM}(L_P(G))$.

**Proof.** $\text{LM}(L_P(G))$ 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. $\Box$

**Lemma 5.7.12.** The \textsc{max-util} decision problem for PLP goalbases is in $\text{P}$.  

**Proof.** Given $G \in \mathcal{L}_{\text{PLP}}$ and $p \in \mathcal{P}S$, $L_P(G)$ can be computed in linear time, and then $p \in \text{LM}(L_P(G))$ is decidable in polynomial time due to Fact 5.7.6. Due to Corollary 5.7.11, this yields the answer to the \textsc{max-util} decision problem. $\Box$

**Lemma 5.7.13.** PLP can be reduced in logarithmic space to the \textsc{max-util} decision problem for PLP goalbases.

**Proof.** Given a logic program $S = \{\varphi_i\}$ and $p \in \mathcal{P}S$, let

$$G = \bigcup_{i=1}^n \{(\varphi_i, 1)\} \cup \bigcup_{p \in \mathcal{P}S} \left\{ \left( p, -\frac{1}{|\mathcal{P}S| + 1} \right) \right\}.$$  

Obviously, $G \in \mathcal{L}_{\text{PLP}}$, and due to Corollary 5.7.11, solving the \textsc{max-util} decision problem instance $\langle G, p \rangle$ yields the answer to the PLP decision problem instance $(S, p)$. $\Box$
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 $L_{HS}$ 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 $G \in L_{HS}$, 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, 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.
5.7. An Alternate Formulation of max-util

Proof. Given a set $S = \{\varphi_1, \ldots, \varphi_n, \varphi'_1, \ldots, \varphi'_m\}$ of strict ($\varphi_i$) and non-strict ($\varphi'_i$) 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. \qed

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. \qed

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
Chapter 5. Complexity

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 \textsc{min-util} and \textsc{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 \textsc{core-membership} and \textsc{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 \textsc{max-util}, \textsc{min-util}, and \textsc{max-cuf}.

Finally, we point out two directions which we do not pursue here, namely approximation and truthfulness. For each \textsc{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 \textsc{max-util} and \textsc{max-cuf} can be approximated in the cases where they are \textsc{NP}-complete. When we consider \textsc{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.