Computational Social Choice: Spring 2009

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

Plan for Today

Collective decision making is driven by the interests of individuals, who must be able to *communicate preferences* (directly through explicit preference revelation, or indirectly via "moves" in a game).

- So far, we have treated this topic only very *abstractly*, by saying that agents "have" some preference structure.
  - could be a *linear order* (assumption so far) or similar
  - could be a *utility function*
- Today we want to introduce *languages* for representing such preference structures in a compact manner, which is of critical importance for social choice in *combinatorial domains*.
- We shall be interested in the properties of these languages, such as *expressive power* and *comparative succinctness*.

Cardinal and Ordinal Preferences

A *preference structure* represents an agent's preferences over a (finite) set of alternatives $\mathcal{X}$. Different types of preferences:

- A *cardinal* preference structure is a (*utility* or *valuation*) function $u : \mathcal{X} \to \mathit{Val}$, where *Val* is usually a set of numerical values such as $\mathbb{N}$ or $\mathbb{R}$.
- An *ordinal* preference structure is a *binary relation* $\preceq$ on $\mathcal{X}$:
  - *linear order:* reflexive, transitive, antisymmetric, complete
  - *weak order:* reflexive, transitive, complete
  - partial orders have also been considered (nonstandard)

Every complete ordinal preference relation $\preceq$ is *representable* by a utility function u: $x \preceq y$ iff $u(x) \leq u(y)$.

Appropriateness of Representation

- Real-world/cognitive considerations: how close should the representation be to how humans express their preferences?
  - Can we assume that an agent can assign a real number reflecting preference to each alternative?
  - How do we *elicit* preference information?
- "Axiomatic" considerations: never mind the real world—what aspects of preference matter for our theorem/algorithm?
  - Ordinal preferences don't permit *interpersonal comparison* ("Ann likes x more than Bob likes y").
  - Ordinal preferences cannot express *preference intensity*; cardinal preferences can (subject to *Val* being numerical).
- Computational considerations: main topic of this lecture ...

Combinatorial Domains

A *combinatorial domain* is a Cartesian product $\mathcal{D} = D_1 \times \cdots \times D_n$ of n finite domains. Many collective decision-making problems of practical interest have a combinatorial structure:

- During a *referendum* (in Switzerland, California, places like that), voters may be asked to vote on n different propositions.
- Elect a *committee* of k members from amongst n candidates.
- Find a good *allocation* of n indivisible goods to agents.

Seemingly small problems generate huge numbers of alternatives:

- $\binom{10}{3} = 120$ possible 3-member committees from 10 candidates; i.e., $120! \approx 6.7 \times 10^{198}$ possible linear (more weak) orders
- Allocating 10 goods to 5 agents: $5^{10} = 9765625$ allocations and $2^{10} = 1024$ bundles for each agent to think about

Therefore: we need good *languages* for representing preferences!

Preference Representation Languages

Concerning computational considerations, the following questions should be addressed when choosing a language:

- *Elicitation:* How difficult is it to elicit the preferences of an agent so as to represent them in the chosen language?
- *Expressive power:* Can the chosen language encode all the preference structures we are interested in?
- *Succinctness:* Is the representation of (typical) structures succinct? Is one language more succinct than the other?
- *Complexity:* What is the computational complexity of related decision problems, such as comparing two alternatives?

We are going to concentrate on the final three.

Representing Set Functions

Recall: a combinatorial domain is a product $\mathcal{D} = D_1 \times \cdots \times D_n$.

A typical scenario is the domain of *resource allocation* problems:

- For a given set $\mathcal{G}$ of goods, we want to represent an agent's *utility function* $u : 2^{\mathcal{G}} \to \mathbb{R}$, mapping bundles to the reals.
- That is, here each D_n is a *binary* domain, and $n = |\mathcal{G}|$.
- More abstractly: we want to represent a set function.

Explicit Representation

The *explicit form* of representing a utility function u consists of a table listing for every bundle $S \subseteq \mathcal{G}$ the utility $u(S)$.
By convention, table entries with $u(S) = 0$ may be omitted.

- the explicit form is *fully expressive:* any utility function $u : 2^{\mathcal{G}} \to \mathbb{R}$ may be so described
- the explicit form is *not concise:* it may require up to 2^n entries

Even very simple utility functions may require exponential space: e.g., the function $u : S \mapsto |S|$ mapping bundles to their cardinality.

▶ We now introduce one example for a more sophisticated type of language, give exemplary expressivity, succinctness and complexity results, and then briefly review other languages from the literature.

Weighted Propositional Formulas

A compact representation language for modelling utility functions over products of binary domains —

Notation: finite set of propositional letters PS; we can use the propositional language $\mathcal{L}_{PS}$ over PS to describe goals.

A *goalbase* is a set $G = \{(\varphi_i, w_i)\}_i$ of pairs, each consisting of a (consistent) propositional formula $\varphi_i \in \mathcal{L}_{PS}$ and a real number w_i. The utility function u_G generated by G is defined by

$$u_G(M) \quad = \quad \sum\{w_i \mid (\varphi_i, w_i) \in G \text{ and } M \models \varphi_i\}$$

for all models $M \in 2^{PS}$. G is called the *generator* of u_G.

Example: $\{(p \vee q \vee r, 7), (p \wedge q, -2), (\neg s, 1)\}$

A Family of Languages

By imposing different restrictions on formulas and/or weights we can design different representation languages.

Regarding *formulas*, we may consider restrictions such as:

- *positive* formulas (no occurrence of $\neg$)
- *clauses* and *cubes* (disjunctions and conjunctions of literals)
- *k-formulas* (formulas of length $\leq k$), e.g. 1-formulas = literals
- combinations of the above, e.g. k-pcubes

Regarding *weights*, interesting restrictions would be $\mathbb{R}^+$ or $\{0, 1\}$.

If $H \subseteq \mathcal{L}_{PS}$ is a restriction on formulas and $H' \subseteq \mathbb{R}$ a restriction on weights, then $\mathcal{L}(H, H')$ is the language conforming to H and H'.

Properties

We are interested in the following types of questions:

- Are there restrictions on goalbases such that the utility functions they generate enjoy natural structural properties?
- Are some goalbase languages more succinct than others?
- What is the computational complexity of reasoning about preferences expressed in a given language?

The results on the following slides are from Uckelman *et al.* (2009).

J. Uckelman, Y. Chevaleyre, U. Endriss, and J. Lang. Representing Utility Functions via Weighted Goals. *Mathematical Logic Quarterly*, 2009. In press.

Expressive Power

We first give an example for a language that is fully expressive:

Theorem 1 (Expressivity of pcubes) *$\mathcal{L}(pcubes, \mathbb{R})$, the language of positive cubes, can express all utility functions.*

Proof: Let $u : 2^{PS} \to \mathbb{R}$ be any utility function. Define goalbase G:

$(\top, w_\top)$ with $w_\top = u(\{\,\})$
(p, w_p) with $w_p = u(\{p\}) - w_\top$
$(p \wedge q, w_{p,q})$ with $w_{p,q} = u(\{p, q\}) - w_p - w_q - w_\top$ ...
$(\bigwedge P, w_P)$ with $w_P = u(P) - \sum_{Q \subset P} w_Q$

Clearly, G thus defined will generate the function u. ✓

Observe that the proof also demonstrates that $\mathcal{L}(pcubes, \mathbb{R})$ has a *unique* way of representing any given function.

$\mathcal{L}(cubes, \mathbb{R})$, for example, is also fully expressive but *not unique:*
$\{(p \wedge q, 5), (p \wedge \neg q, 5), (\neg p \wedge q, 3), (\neg p \wedge \neg q, 3)\} \equiv \{(\top, 3), (p, 2)\}$

Expressive Power: Modular Functions

A function $u : 2^{PS} \to \mathbb{R}$ is *modular* if for all $M_1, M_2 \subseteq 2^{PS}$ we have:

$$u(M_1 \cup M_2) = u(M_1) + u(M_2) - u(M_1 \cap M_2)$$

Here's a nice characterisation of the modular functions:

Theorem 2 (Expressivity of literals) *$\mathcal{L}(literals, \mathbb{R})$ can express all modular utility functions, and only those.*

Proof sketch: Modular functions are like *additive* functions, except that we allow non-zero values for $\{\}$. Easy to see that $\mathcal{L}(atoms, \mathbb{R})$ expresses exactly the additive functions.

So $\mathcal{L}(atoms \cup \{\top\}, \mathbb{R})$ expresses exactly the modular functions.

To see that adding negation does not increase expressive power, observe that $G \cup \{(\neg\varphi, w)\} \equiv G \cup \{(\top, w), (\varphi, -w)\}$. ✓

Expressive Power: Monotonic Functions

A function $u : 2^{PS} \to \mathbb{R}$ is *monotonic* if for all $M_1, M_2 \subseteq 2^{PS}$:

$$M_1 \subseteq M_2 \text{ implies } u(M_1) \leq u(M_2)$$

We state this next theorem without proof:

Theorem 3 (Expressivity of pforms/pos) *$\mathcal{L}(pforms, \mathbb{R}^+)$, the language of positive formulas with positive weights, can express all nonnegative monotonic utility functions, and only those.*

Relative Succinctness

If two languages can express the same class of utility functions, which should we use? An important criterion is *succinctness*.

Let $\mathcal{L}$ and $\mathcal{L}'$ be two languages that can define all utility functions belonging to some class $\mathcal{U}$.

We say that $\mathcal{L}'$ is at least as succinct as $\mathcal{L}$ ($\mathcal{L} \preceq \mathcal{L}'$) over $\mathcal{U}$ if there exist a mapping $f : \mathcal{L} \to \mathcal{L}'$ and a *polynomial* function p such that for all expressions $G \in \mathcal{L}$ with the corresponding function $u_G \in \mathcal{U}$:

- $G \equiv f(G)$ (i.e., they represent the same function: $u_G = u_{f(G)}$);
- and $size(f(G)) \leq p(size(G))$ (polysize reduction).

$\mathcal{L}$ is *less succinct* than $\mathcal{L}'$ ($\mathcal{L} \prec \mathcal{L}'$) iff $\mathcal{L} \preceq \mathcal{L}'$ and not $\mathcal{L}' \preceq \mathcal{L}$. *Equivalence* ($\sim$) and *incomparability* ($\perp$) are defined accordingly.

The Effect of Negation

We have seen that positive cubes are fully expressive. Hence, $\mathcal{L}(pcubes, \mathbb{R})$ and $\mathcal{L}(cubes, \mathbb{R})$ have the same expressivity.

Theorem 4 (Succinctness) $\mathcal{L}(pcubes, \mathbb{R}) \prec \mathcal{L}(cubes, \mathbb{R})$.

Proof: Clearly, $\mathcal{L}(pcubes, \mathbb{R}) \preceq \mathcal{L}(cubes, \mathbb{R})$, because any positive cube is also a cube.

Now consider u with $u(\{\}) = 1$ and $u(M) = 0$ for all $M \neq \{\}$:

- $G = \{(\neg p_1 \wedge \cdots \wedge \neg p_n, 1)\} \in \mathcal{L}(cubes, \mathbb{R})$ has *linear* size and generates u.
- $G' = \{(\bigwedge X, (-1)^{|X|}) \mid X \subseteq PS\} \in \mathcal{L}(pcubes, \mathbb{R})$ has *exponential* size and also generates u.

But there can be not better way of expressing u in pcubes, because we have seen that pcube representations are *unique*. ✓

More Succinctness Results

Some more examples for succinctness theorems (without proof):

- Cubes and clauses are equally succinct:

  Theorem 5 $\mathcal{L}(cubes, \mathbb{R}) \sim \mathcal{L}(clauses, \mathbb{R})$

- But *positive* cubes and clauses are incomparable:

  Theorem 6 $\mathcal{L}(pcubes, \mathbb{R}) \perp \mathcal{L}(pclauses, \mathbb{R})$

- If we restrict the weights, we get equal succinctness again:

  Theorem 7 $\mathcal{L}(pcubes, \mathbb{R}^+) \sim \mathcal{L}(pclauses, \mathbb{R}^+)$

  In fact, this result is more about expressivity: it is a corollary of other results showing that only modular functions are expressible in *both* $\mathcal{L}(pcubes, \mathbb{R}^+)$ and $\mathcal{L}(pclauses, \mathbb{R}^+)$.

- The most difficult result, showing that we really gain further succinctness as we enrich the formula language:

  Theorem 8 $\mathcal{L}(cubes, \mathbb{R}) \prec \mathcal{L}(forms, \mathbb{R})$

Computational Complexity

Other interesting questions concern the complexity of reasoning about preferences. Consider the following decision problem:

MAX-UTILITY(H, H')
Instance: Goalbase $G \in \mathcal{L}(H, H')$ and $K \in \mathbb{Z}$
Question: Is there an $M \in 2^{PS}$ such that $u_G(M) \geq K$?

Some basic results are straightforward:

- MAX-UTILITY(H, H') is *in NP* for any $H \subseteq \mathcal{L}_{PS}$ and $H' \subseteq \mathbb{Q}$, because we can always check $u_G(M) \geq K$ in polynomial time.
- MAX-UTILITY$(forms, \mathbb{Q})$ is *NP-complete*, certainly if we do not assume that formulas are satisfiable (reduction from SAT).

More interesting questions would be: are there either (1) "large" sublanguages for which MAX-UTILITY is still polynomial, or (2) "small" sublanguages for which it is already NP-hard?

Three Complexity Results

Theorem 9 MAX-UTILITY$(k\text{-}cubes, \mathbb{Q})$ *is NP-complete, even for* $k = 2$ *and assuming all formulas are satisfiable.*

Proof: By reduction from MAX2SAT (NP-hard): "Given a set of 2-clauses, is there a satisfiable subset with cardinality $\geq K$?". Given a MAX2SAT instance, for each clause $p \vee q$ create a goal $(\neg p \wedge \neg q, -1)$. Add $(\top, N)$, where N is the number of clauses. Answer YES for the MAX2SAT instance iff max. utility is $\geq K$. ✓

Remark: NP-hardness persists if $\mathbb{Q}$ is replaced by $\{1\}$.

Theorem 10 MAX-UTILITY$(literals, \mathbb{Q})$ *is in P.*

Proof: Assuming that G contains every literal exactly once (possibly with weight 0), making p true iff the weight of p is greater than the weight of $\neg p$ results in a model with maximal utility. ✓

Theorem 11 MAX-UTILITY$(pforms, \mathbb{Q}^+)$ *is in P.*

Proof: Making *all* variables true yields maximal utility. ✓

Preference Representation Languages

We have looked into goalbase languages in detail, to see examples for the kinds of technical results that people have been looking into (others have done similar things for other languages).

Next, some examples for other languages, for representing both utility functions and ordinal preferences.

The k-additive Form

- A utility function is *k-additive* iff the utility assigned to a bundle X can be represented as the sum of marginal utilities for subsets of X with cardinality $\leq k$ (*limited synergies*).
- The *k-additive form* of representing utility functions:

$$u(X) = \sum_{T \subseteq X} \alpha^T \qquad \text{with } \alpha^T = 0 \text{ whenever } |T| > k$$

  Example: $u = 3.x_1 + 7.x_2 - 2.x_2.x_3$ is a 2-additive function
- That is, specifying a utility function in this language means specifying the *coefficients* α^T for bundles $T \subseteq \mathcal{R}$.
- In the context of resource allocation, the value α^T can be seen as the additional benefit incurred from owning the items in T *together*, i.e., beyond the benefit of owning all proper subsets.
- Just a notational variant of $\mathcal{L}(k\text{-}pcubes, \mathbb{R})$!

Bidding Languages

In *combinatorial auctions* the process of bidding amounts to transmitting a cardinal preference structure (valuation function).

People have developed special *bidding languages* for this purpose. Example for a bid using the so-called OR-language:

$$\langle \{a\}, 2 \rangle \text{ OR } \langle \{b\}, 2 \rangle \text{ OR } \langle \{c\}, 1 \rangle \text{ OR } \langle \{a, b\}, 5 \rangle$$

This expresses that the bidder is happy to buy any of the given sets at the prices specified, provided the sets selected do not overlap.

We will discuss bidding languages later on in the course.

Program-based Representations

Yet another approach to representing preferences would be to define utilities in terms of a *program:* input bundle, output utility value. But not just any program will do. Requirements:

- it must be possible to efficiently validate that a given string constitutes a *syntactically correct program*; and
- we have to have an effective method of *computing the output* of the program for any given input.

Dunne *et al.* (2005) propose such a program-based approach based on so-called *straight-line programs* (warning: rather technical).

One result says that any function computable by a deterministic TM in time T is representable by an SLP with $O(T \log T)$ lines.

P.E. Dunne, M. Wooldridge, and M. Laurence. The Complexity of Contract Negotiation. *Artificial Intelligence*, 164(1–2):23–46, 2005.

Ordinal Preferences

Next we are going to look into different languages for representing *ordinal* preference structures.

Note that an *explicit representation* of an ordinal preference relation $\preceq$ over 2^n alternatives requires space up to $O(2^n \cdot 2^n)$: for each pair of alternatives, say which one is preferred.

Prioritised Goals

Again, associate goods with propositional letters in PS and bundles with models $M \in 2^{PS}$. *Goals* can be expressed as formulas in the propositional language $\mathcal{L}_{PS}$.

Instead of weights, we now have a *priority relation* over goals. Assuming this priority relation is a linear order, it can be represented by a function $rank : \mathbb{N} \to \mathbb{N}$ mapping each (index of a) goal to its rank. By convention, a *lower rank* means *higher priority*.

A *goalbase* is now a finite set of goals with an associated rank function: $G = \langle \{\varphi_1, \ldots, \varphi_m\}, rank \rangle$.

▶ Ideally, all goals will get satisfied. But if not, how can we extend a priority relation over goals to a preference relation over models?

Combining Priorities

There are several options (convention: $\min(\{\}) = +\infty$):

- *Best-out ordering:*

  $M \preceq M'$ iff $\min\{rank(i) \mid M \not\models \varphi_i\} \leq \min\{rank(i) \mid M' \not\models \varphi_i\}$

  That is, preference depends (only) on the rank of the most important goal that is being violated.

- *Discrimin ordering:*
  Let $d(M, M') = \min\{rank(i) \mid M \not\models \varphi_i \text{ and } M' \models \varphi_i\}$ be the rank of the most important "discriminating" goal.

  $$M \preceq M' \text{ iff } d(M, M') \leq d(M', M) \text{ or } \{\varphi_i \mid M \models \varphi_i\} = \{\varphi_i \mid M' \models \varphi_i\}$$

Combining Priorities (cont.)

- *Leximin ordering:*
  Let $d_k(M) = |\{\varphi_i \mid M \models \varphi_i \text{ and } rank(\varphi_i) = k\}|$ be the number of goals of rank k that are satisfied by alternative M.

  $M \preceq M'$ iff (1) for all k: $d_k(M) = d_k(M')$ or
  (2) there exists a k such that $d_k(M) < d_k(M')$ and for all $j < k$: $d_j(M) = d_j(M')$

Properties

- None of the three variants of combining prioritised goals leads to a *fully expressive* preference representation language.
- For the strict parts of the preference relations we have:
  - *best-out preference* entails *discrimin preference*; and
  - *discrimin preference* entails *leximin preference*

Ceteris Paribus Preferences

In the language of *ceteris paribus* preferences, preferences are expressed as statements of the form $C : \varphi > \varphi'$, meaning:

> *"If C is true, all other things being equal, I prefer alternatives satisfying $\varphi \wedge \neg\varphi'$ over those satisf. $\neg\varphi \wedge \varphi'$."*

The "other things" are the truth values of the propositional variables not occurring in φ and φ'. A preference relation can be constructed as the transitive closure of the union of individual preference statements.

Discussion: interesting from a *cognitive* point of view (arguably close to human intuition), but of rather *high complexity*.

An important sublanguage of *ceteris paribus* preferences, imposing various restrictions on goals, are *CP-nets* ($\rightsquigarrow$ next week).

Summary

We have reviewed several preference representation languages for both cardinal and ordinal preference structures.

- The computational aspects of preference representation are crucial in *combinatorial domains* (such as resource allocation).
- We have emphasised *expressivity*, *succinctness* and *complexity*.
- Languages considered (there are more):
  - *cardinal:* explicit form, weighted goals, k-additive form, bidding languages, and program-based representations
  - *ordinal:* prioritised goals and *ceteris paribus* statements

References

For an in-depth survey of logic-based languages for representing preferences, refer to:

- J. Lang. Logical Preference Representation and Combinatorial Vote. *Annals of Mathematics and Artificial Intelligence*, 42(1):37–71, 2004.

For a concise overview of the role of preference representation in the context of multiagent resource allocation, consult:

- Y. Chevaleyre *et al.* Issues in Multiagent Resource Allocation. *Informatica*, 30:3–31, 2006. (Sect. Preference Representation)

What next?

The aim of this lecture has been to present some preference representation languages and to give examples for the kinds of properties that we might want to prove about them.

Preferences will play a central role throughout the course. Specifically, they will come up again on two occasions:

- Next week, we will introduce *CP-nets* in the context of discussing voting in combinatorial domains.
- We will see a number of expressivity and succinctness results for *bidding languages* later on in the course, when we will cover combinatorial auctions.