Introduction to Computational Social Choice

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

Lecture 3

Many social choice problems have a combinatorial structure:

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

Today will be an introduction to *compact preference representation languages* and to *voting* in *combinatorial domains*. Specifically:

- Properties of preference representation languages: expressivity, succinctness, and complexity
- Approaches to voting in combinatorial domains: to what extent can we vote issue-by-issue?

Cardinal and Ordinal Preferences

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

- A *cardinal* preference structure is a (*utility* or *valuation*) function $u : \mathcal{X} \to Val$, where Val is usually a set of numerical values such as $\mathbb{N}$ or $\mathbb{R}$.
- An *ordinal* preference structure is a *binary relation* $\preceq$ over the set of alternatives (reflexive, transitive and complete).

People have developed a number of different *languages* for representing such preference structures.

Preference Representation Languages

The following are relevant questions when we have to choose a preference representation language:

- *Cognitive relevance:* How close is a given language to the way in which humans would express their preferences?
- *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?

Combinatorial Domains

A *combinatorial domain* is a Cartesian product $\mathcal{X} = D_1 \times \cdots \times D_p$ of p finite domains. We want to represent utility functions over $\mathcal{X}$.

A good example are *resource allocation* problems. If we have a set $\mathcal{G}$ of indivisible goods, then each agent should have a utility function $u : 2^{\mathcal{G}} \to \mathbb{R}$ mapping bundles of goods to the reals.

That is, here each D_i is a binary domain, and $p = |\mathcal{G}|$.

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^p 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 list 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*; propositional language $\mathcal{L}_{PS}$ over PS to describe requirements.

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

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

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 $\{0, 1\}$.

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

Properties

We are interested in the following types of question:

- 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 complexity of reasoning about preferences expressed in a given language?

The results on the following slides are from Chevaleyre et al. (2006).

Y. Chevaleyre, U. Endriss, and J. Lang. *Expressive Power of Weighted Propositional Formulas for Cardinal Preference Modelling*. Proc. KR-2006.

Expressive Power

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

Theorem 1 (Expressivity of pcubes) *$\mathcal{L}(pcubes, all)$, 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:

$(\top, \alpha_\top)$ with $\alpha_\top = u(\emptyset)$
(p, α_p) with $\alpha_p = u(\{p\}) - \alpha_\top$
$(p \wedge q, \alpha_{p,q})$ with $\alpha_{p,q} = u(\{p,q\}) - \alpha_p - \alpha_q - \alpha_\top$ $\quad \ldots$
$(\bigwedge P, \alpha_P)$ with $\alpha_P = u(P) - \sum_{Q \subset P} \alpha_Q$

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

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

$\mathcal{L}(cubes, all)$, 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) *$\mathcal{L}(literals, all)$, the language of literals, 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 $\emptyset$. Easy to see that $\mathcal{L}(atoms, all)$ expresses exactly the additive functions.

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

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

Definition of 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 $u_G \in \mathcal{U}$:

- $G \equiv f(G)$ (they both represent the same function); and
- $size(f(G)) \leq p(size(G))$ (polysize reduction).

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

Relative Succinctness

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

Theorem 3 (Succinctness) $\mathcal{L}(pcubes, all) \prec \mathcal{L}(cubes, all)$.

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

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

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

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 choice of H and H', because we can always check $u_G(M) \geq K$ in polynomial time.
- MAX-UTILITY(all, all) is *NP-complete* (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 4 *MAX-UTILITY(k-clauses, positive) is NP-complete, even for $k = 2$.*

Proof: Reduction from MAX2SAT (NP-complete): "Given a set of 2-clauses, is there a satisfiable subset with cardinality $\geq K$?". ✓

Theorem 5 *MAX-UTILITY(literals, all) 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 6 *MAX-UTILITY(positive, positive) is in P.*

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

For more advanced complexity results see the paper cited below.

J. Uckelman and U. Endriss. *Preference Representation with Weighted Goals: Expressivity, Succinctness, Complexity*. Proc. AiPref-2007.

More Preference Representation Languages

Other languages for modelling utility functions include:

- weighted goals with other aggregators, e.g. *max*.
- k-additive form (actually isomorphic to $\mathcal{L}(k\text{-}pcubes, all)$)
- bidding languages for combinatorial auctions
- program-based representations (so-called SLP form)

Languages for modelling ordinal preference relations include:

- prioritised goals (e.g. go by most important goal violated)
- CP-nets: conditional *ceteris paribus* preferences

See Chevaleyre et al. (2006) for references.

Y. Chevaleyre, P.E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J.A. Rodríguez-Aguilar, and P. Sousa. Issues in Multiagent Resource Allocation. *Informatica*, 30:3–31, 2006.

The Paradox of Multiple Elections

Suppose 13 voters are asked to each vote *yes* or *no* on three issues; and we use the plurality rule for each issue independently to select a winning combination:

- 3 voters each vote for YNN, NYN, NNY.
- 1 voter each votes for YYY, YYN, YNY, NYY.
- No voter votes for NNN.

But then NNN wins: 7 out of 13 vote *no* on each issue.

This is an instance of the *paradox of multiple elections:* the winning combination receives the fewest number of votes.

S.J. Brams, D.M. Kilgour, and W.S. Zwicker. The Paradox of Multiple Elections. *Social Choice and Welfare*, 15(2):211–236, 1998.

Voting in Combinatorial Domains

The problem of voting in combinatorial domains:

- Domain: variables $X_1, \ldots, X_p$ with finite domains $D_1, \ldots, D_p$
- Voters have preferences over set of combinations $D_1 \times \cdots \times D_p$.
- What should be the winning combination in $D_1 \times \cdots \times D_p$?

(1) Here we only consider *binary* variables. (2) Sometimes there are additional *constraints:* e.g., if we need to elect a committee of size k and each variable represents one candidate, then the number of yes-values in the winning combination has to be exactly k.

Possible Solutions

(1) Use your favourite voting rule with the full set of combinations as the set of candidates.

(2) Do as for (1) but only require voters to rank their k most preferred combinations (e.g. for plurality we'd have $k = 1$).

(3) Select a small number of combinations and then use your favourite voting rule to elect a candidate from amongst those.

(4) Ask voters to report their preferences using a compact representation language and apply your favourite voting rule to the succinctly encoded ballots received ("combinatorial vote").

(5) Vote separately on each issue (as in the paradox), but —
  (a) identify conditions under which the paradox can be avoided;
  (b) find a novel way of aggregating the votes to select a winner;
  (c) do so sequentially rather than simultaneously.

Solution (1): Explicit Vote for Combinations

Idea: Vote for combinations directly: use your favourite voting rule with the full set of combinations as the set of candidates.

Problem: This will only be possible in very small domains, in particular when the voting rule requires a complete ranking of all candidates (such as the Borda rule).

Example: Suppose there are six binary issues. This makes $2^6 = 64$ possible combinations. Hence, under the Borda rule, each voter has to choose between $64! \approx 1.27 \cdot 10^{89}$ possible ballots.

Solution (2): Vote for Top Combinations only

Idea: Vote for combinations directly, but only require voters to rank their k most preferred combinations, for some small number k.

Specifically, for the plurality rule $k = 1$ will suffice.

This clearly addresses the communication problems of Solution (1).

Problem: This may lead to almost random decisions, unless domains are very small and there are many voters.

Example: Suppose there are 10 binary issues to be decided upon and 100 voters. Then there are $2^{10} = 1024$ combinations to vote for. Under the plurality rule, chances are that no combination receives more than one vote (so the tie-breaking rule decides).

Solution (3): Vote for Selected Combinations only

Idea: Select a small number of combinations and then use your favourite voting rule to elect a candidate from amongst those.

Problem: Who selects the candidate combinations? It is not at all clear what criteria should be used here. This gives the chooser (probably the election chair) undue powers and opens up new opportunities for controlling elections.

Solution (4): Combinatorial Vote

Idea: Ask voters to report their preferences using a compact preference representation language and apply your favourite voting rule to the succinctly encoded ballots received.

Lang (2004) calls this approach *combinatorial vote*.

Discussion: This seems the most promising approach so far, although not too much is known to date what would be good choices for preference representation languages or voting rules, or what algorithms to use to compute the winners. Also, complexity can be expected to be very high.

J. Lang. Logical Preference Representation and Combinatorial Vote. *Annals of Mathematics and Artificial Intelligence*, 42(1–3):37–71, 2004.

Single Goals and Generalised Plurality

Next an example for a complexity result ... We will work with the following preference representation language and voting rule:

- The *language of single goals* is an instance of the framework of weighted propositional formulas. Each agent specifies just one goal (arbitrary propositional formula) with weight 1. We are only interested in the ordinal preference structure induced.
- Under the *generalised plurality rule*, a voter gives 1 point to each undominated candidate.

Here are two examples, for the set of variables $\{A, B\}$:

- The goal $\neg A \wedge B$ induces the order $\bar{A}B \succ AB \sim A\bar{B} \sim \bar{A}\bar{B}$, so only combination $\bar{A}B$ receives 1 point.
- The goal $A \vee B$ induces the order $AB \sim \bar{A}B \sim A\bar{B} \succ \bar{A}\bar{B}$, so combinations AB, $\bar{A}B$, $A\bar{B}$ receive 1 point each.

Winner Verification under Plurality

Define the following decision problem, for some preference representation language $\mathcal{L}$ and some voting rule f:

> AMONG-WINNERS($\mathcal{L}, f$)
> **Instance:** Voter preferences in $\mathcal{L}$; candidate combination c.
> **Question:** Is c amongst the winners if voting rule f is used?

The following result (proof omitted) is due to Lang (2004):

Theorem 7 AMONG-WINNERS *is coNP-complete for the language of single goals and the generalised plurality rule.*

Recall that coNP is the complement of the complexity class NP and the complexity class of validity checking in propositional logic.

J. Lang. Logical Preference Representation and Combinatorial Vote. *Annals of Mathematics and Artificial Intelligence*, 42(1–3):37–71, 2004.

Solution (5a): Vote Separately Anyway

Idea: Try to identify conditions under which voting separately on each issue does not lead to paradoxes.

Discussion: To be precise, on the face of it the "paradox" can never be avoided, but for some types of voter preferences this is not a worry. Recall that NNN won, even though it was nobody's favourite combination. If NNN were everyone's *least* favourite combination (which is possible in general), then this is really dramatic. But if voters only care about the individual decisions, then NNN is a reasonable outcome, as 9 out of 13 voters voted 2×N, for instance.

Specifically, if all voter preferences are *separable* (preferences over one variable do not depend on instantiations of other variables), then voting separately is fine.

Solution (5b): Vote Separately and Aggregate

Idea: Vote separately on each issue. However, don't just promote the issue-wise winners to the winning combination, but rather investigate other ways of aggregating the ballot information.

Discussion: Potentially interesting approach, *if* we can come up with a reasonable aggregation rule. We should choose a combination that somehow *minimises the distance* of the winning combination to the combinations the voters voted for (ballots).

For example, Brams et al. (2007) suggest choosing the combination that minimises the maximal *Hamming distance* to any ballot.

S.J. Brams, D.M. Kilgour, and M.R. Sanver. A Minimax Procedure for Electing Committees. *Public Choice*, 132:401–420, 2007.

Solution (5c): Sequential Vote

Idea: Vote separately on each issue, but do so *sequentially* to give voters the opportunity to make their vote for one issue dependent on other issues already decided.

Literature: Lang (2007) and Xia et al. (2007) have followed this approach, in particular for the case where voter preferences can be modelled using *CP-nets* (Boutilier et al., 2004), which relaxes the separability assumption mentioned earlier. They have analysed e.g. under what circumstances Condorcet-consistency of the local voting rules will transfer to the global rule.

J. Lang. *Vote and Aggregation in Combinatorial Domains with Structured Preferences.* Proc. IJCAI-2007.

L. Xia, J. Lang, and M. Ying. *Sequential Voting Rules and Multiple Election Paradoxes.* Proc. TARK-2007.

C. Boutilier, R.I. Brafman, C. Domshlak, H.H. Hoos, and D. Poole. CP-nets: A Tool for Representing and Reasoning with Conditional *Ceteris Paribus* Preference Statements. *Journal of AI Research*, 21:135–191, 2004.

Sequential Voting and Condorcet Losers

A *Condorcet loser* is a candidate that loses against any other candidate in a pairwise competition. Electing a CL is very bad.

Example: NNN (the winning combination) in the original multiple election paradox may very well have been such a Condorcet loser.

Lacy and Niou (2000) show that sequential voting can avoid this:

Theorem 8 *Sequential voting (with plurality) over binary issues never results in a winning combination that is a Condorcet loser.*

Proof sketch: Just think what happens during the election for the final issue. The winning combination cannot be a Condorcet loser, because it does, at least, win against the other combination that was still possible after the penultimate election. ✓

D. Lacy and E.M.S. Niou. A Problem with Referendums. *Journal of Theoretical Politics*, 12(1):5–31, 2000.

Summary

We have explored several aspects of collective decision making in combinatorial domains:

- We need a language for compactly representing individual preferences. Weighted goals are one such language. There are many more, and people will continue to devise new ones.
- We need to understand the properties of these languages. We have seen expressivity, succinctness, and complexity results.
- We have reviewed different approaches for conducting elections in combinatorial domains. A lot more work is needed here.

Literature

For an exposition of the problem of collective decision making in combinatorial domains and references to the literature, see:

- Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. Preference handling in Combinatorial Domains: From AI to Social Choice. *AI Magazine*, 2008. In press.

For a discussion of compact preference representation languages (focusing on ordinal preferences) and more on combinatorial vote, see:

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

For a succinct overview of preference representation languages, specifically those attractive for resource allocation problems, see:

- Y. Chevaleyre, P.E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J.A. Rodríguez-Aguilar, and P. Sousa. Issues in Multiagent Resource Allocation. *Informatica*, 30:3–31, 2006.

Wednesday Morning Quiz

In the context of representing utility functions by means of weighted formulas, determine the *relative succinctness* of

- $\mathcal{L}(pcubes, all)$, the language of *positive cubes*, and
- $\mathcal{L}(positive, all)$, the language of *positive formulas*.