Computational Social Choice 2024

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

[http://www.illc.uva.nl/~ulle/teaching/comsoc/2024/]

Plan for Today (and the Near Future)

Exciting trend in *computational social choice:* use of *SAT solvers* to *automate* some of our tasks as researchers. *Very cool*. *But difficult*.

The next few lectures will be dedicated to covering this approach:

- Today: Putting basic machinery in place
- Next: Automating the proof of a classical impossibility theorem
- Later: Critique and refinement of the basic approach
- Later: Expanding the approach, with focus on explainability
- Later: Broader considerations of modelling SCT using logic

Hands-on: You can reproduce everything you see here directly on your own machine, using the *Jupyter Notebook* provided. *Try it!*

Need for New Techniques

The original proof of *Arrow's Theorem* was not quite correct (though the theorem itself was always fine). It took some years to fix this.

And the *G-S Theorem* is a deep result that long seemed elusive:

- People tried and failed to design strategyproof rules for centuries.
- After Arrow's Theorem a result *à la* G-S seemed to be "in the air".
- It still took two decades to find the right formulation and prove it.
- The original proofs are hard to digest.

Today the proofs of Arrow's and the G-S Theorem are well understood. But new results of this kind are still hard to discover and then prove.

Thus: *need much better methodology to reason about social choice!*

Proving the Gibbard-Satterthwaite Theorem

Recall that the G-S Theorem says that every resolute voting rule that is surjective and strategyproof must be a dictatorship.

This slight reformulation (which is equivalent) will be more convenient:

Gibbard-Satterthwaite Theorem: *For $m \geqslant 3$ alternatives, no resolute voting rule is strategyproof, surjective, and nondictatorial.*

Let's try to get a computer to prove it for us! But proving it for *all* $n \geqslant 1$ (voters) and $m \geqslant 3$ (alternatives) is too ambitious for now ...

Exercise: *For which values of n and m is the theorem most surprising?*

A. Gibbard. Manipulation of Voting Schemes. *Econometrica*, 1973.

M.A. Satterthwaite. Strategy-proofness and Arrow's Conditions. *JET*, 1975.

Base Case

So let's prove G-S for $n = 2$ voters and $m = 3$ alternatives!

Credo: Even if (*formally*) the full theorem might not follow easily from this 'base case', (*intuitively*) it will then be entirely unsurprising.

Proof Idea

Go through *all voting rules* for $n = 2$ and $m = 3$ and *check* one by one whether they satisfy our requirements. Confirm theorem if none do.

Exercise: *How many (resolute) voting rules do we need to check?*

Better Idea: Logic Encoding

Bad news: there are a total of $m^{(m!^n)} = 3^{36} = 150094635296999121$ resolute voting rules for us to check. *So this won't work.*

Instead, let's try to describe what we need in a compact way . . .

Idea: Define a logical language with propositional *variables* $p_{r,x}$ to say that in *profile* r the outcome should include *alternative* x.

This will allow us to describe the behaviour of any irresolute voting rule in a simple formal language using a fairly small number of variables.

Exercise: *Count the variables for $n = 2$ voters and $m = 3$ alternatives!*

Remark: During the lectures on working with SAT solvers, we will use r rather than $\boldsymbol{R}$ for profiles, to hint at the fact that we will think of r as a number *referring to* a profile $\boldsymbol{R}$ rather than *being* a profile itself.

Example

Let us refer to the voters as 0 and 1, and the alternatives as 0, 1, and 2.

There are $3! \times 3! = 36$ profiles, so let us enumerate them from 0 to 35.

The exact enumeration does not matter (as long as we keep it fixed), but suppose we have chosen an enumeration with these features:

Profile 2	Profile 5
$1 \succ 0 \succ 2$	$2 \succ 1 \succ 0$
$0 \succ 1 \succ 2$	$0 \succ 1 \succ 2$

Then *strategyproofness* requires that, if we want to elect 0 in profile 2, then we must *not* elect 1 in profile 5. Exercise: *Explain why!*

Using our propositional language, we can express this as an implication:

$$p_{2,0} \to \neg p_{5,1}$$

Correspondence

Let's focus on *irresolute* voting rules F for now:

$$F : \mathcal{L}(A)^n \to 2^A \setminus \{\emptyset\}$$

Every *assignments of truth values* to variables $p_{r,x}$ corresponds to a *function from profiles to sets of alternatives*, i.e., a *voting rule*.

This is so because fixing the truth values for all variables $p_{r,x}$ amounts to saying which alternatives x are (or are not) elected in a profile r.

Exercise: *This is almost true, but not quite. Do you see the problem?*

Modelling Voting Rules and Axioms

A voting rule must return *at least one* alternative x for every profile r:

$$\varphi_{\text{at-least-one}} \quad = \quad \bigwedge_{r} \left(\bigvee_{x} p_{r,x} \right)$$

We obtain a perfect correspondence between *voting rules* and *models* ($=$ satisfying truth assignments) of this formula. *Nice!*

Can use similar formulas to encode *axioms* of interest. Then:

models satisfying formulas	$\widehat{=}$	voting rules satisfying axioms
unsatisfiability	$\widehat{=}$	impossibility theorem

SAT Solving

Can use a *SAT solver* to check formulas (in *CNF*) for unsatisfiability.

DIMACS format: use *list of lists of positive and negative integers* to represent *set of clauses of positive and negative literals*. Example:

`[[1,-2,3],[4,-1]]` represents $(p_1 \vee \neg p_2 \vee p_3) \wedge (p_4 \vee \neg p_1)$

Need: script to generate such formulas!

A. Biere, M. Heule, H. van Maaren, and T. Walsh (eds), *Handbook of Satisfiability*. IOS Press, 2009.

A. Ignatiev, A. Morgado, and J. Marques-Silva. PySAT: A Python Toolkit for Prototyping with SAT Oracles. SAT-2018.

Preferences and Profiles

Fix an enumeration of voters, alternatives, preferences, profiles. Then represent everything as integers: *voters* from 0 to `n-1`, *alternatives* from 0 to `m-1`, *preferences* from 0 to `m!-1`, *profiles* from 0 to $\mathtt{m!}^{\mathtt{n}}-1$.

Next we implement some basic methods to explore this model:

- `allVoters()`, `allAlternatives()`, `allProfiles()`
- `voters(c)`, `alternatives(c)`, `profiles(c)` for condition `c`
- `prefers(i,x,y,r)` — does voter `i` prefer `x` to `y` in profile `r`?
- `top(i,x,r)` — does voter `i` top-rank `x` in profile `r`?
- `iVariants(i,r1,r2)` — are profiles `r1` and `r2` i-variants?
- `strProf(r)` — return a string representation for profile `r`

Implementation

Let's inspect the Jupyter Notebook to understand the implementation of these methods for preferences and profiles and run some examples . . .

Detail: Extracting Preferences from Profiles

Maybe the most complicated bit in this part of the implementation . . .

Think of profiles as numbers with n *digits* in the number system with *base* $m!$. So voter i's preference in r is the ith digit (from the back):

```
def preference(i, r):
    base = factorial(m)
    return ( r % (base ** (i+1)) ) // (base ** i)
```

For comparison, this is how, given a number in the decimal system, you would extract the 3rd digit (counting backwards from the "0th digit"):

$$(\,975474 \bmod 10^{3+1}\,) \;/\; 10^3 = 5.474$$

Exercises

Exercise: *Write code to print the representations of all 36 profiles!*

```
(012,012)
(021,012)
(102,012)
(120,012)
(201,012)
⋮
```

Exercise: *Now just print those in which both voters prefer* `0` *to* `2`*!*

```
(012,012)
(021,012)
(102,012)
(012,021)
(021,021)
⋮
```

Summary

We understood that the Gibbard-Satterthwaite Theorem is at its most baffling for the *base case* of $n = 2$ voters and $m = 3$ alternatives.

We understood that the question of whether there *exists* an irresolute voting rule for some fixed number of voters (such as $n = 2$) and some fixed number of alternatives (such as $m = 3$) can be *reduced* to the question of whether a given propositional formula is *satisfiable*.

To prepare for exploiting this correspondence later on, we saw how to implement *simple methods* in Python for reasoning about profiles and preferences (main idea: *everything is a number!*).

What next? Proving the base case of the G-S Thm with a SAT solver.