Advanced Topics in Computational Social Choice

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

Course Outline

Obtaining axiomatic results in SCT is hard: eliminating various minor errors from the original proof of Arrow's Theorem took several years; the Gibbard-Satterthwaite Theorem was conjectured at least a decade before it was proved correct; getting new results is really challenging.

Maybe *automated reasoning*, as studied in AI, can help? *Yes!*

In particular, *SAT solvers* (for checking whether a, possibly very large, propositional formula in CNF is satisfiable) have been used successfully to prove a range of *impossibility theorems* in SCT (and related areas):

- automated *verification* of classical results
- automated *proofs* of new theorems
- automated *discovery* of new theorems

We are going to learn how to use this approach, study successful examples from the literature, and try to obtain new results ourselves.

Plan for Today

Today we will introduce the SAT-based approach to SCT by means of a case study: the Gibbard-Satterthwaite Theorem. This involves:

- encoding a social choice scenario into *propositional logic*
- writing a *Python program* to generate that (large) formula
- proving the formula to be unsatisfiable using a *SAT solver*

Consult Geist and Peters (2017) for an introduction to this approach.

C. Geist and D. Peters. Computer-Aided Methods for Social Choice Theory. In U. Endriss (ed.), *Trends in Computational Social Choice*. AI Access, 2017.

Reminder: The Model

Fix a finite set $A = \{a, b, c, \ldots\}$ of *alternatives*, with $|A| = m \geqslant 2$.

Let $\mathcal{L}(A)$ denote the set of all strict linear orders R on A. We use elements of $\mathcal{L}(A)$ to model (true) *preferences* and (declared) *ballots*.

Each member i of a finite set $N = \{1, \ldots, n\}$ of *voters* supplies us with a ballot R_i, giving rise to a *profile* $\boldsymbol{R} = (R_1, \ldots, R_n) \in \mathcal{L}(A)^n$.

We write $N^{\boldsymbol{R}}_{x \succ y}$ for the set of voters ranking x above y in profile $\boldsymbol{R}$.

A (resolute) *voting rule* (or *social choice function*) for N and A selects one winner for every profile of preferences:

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

Remark: Most natural voting rules in fact are *irresolute* and have to be paired with a tie-breaking rule to always get a unique election winner.

Reminder: Two Axioms

- F is *strategyproof* if for no voter $i \in N$ there exist a profile $\boldsymbol{R}$ (including the "truthful preference" R_i of i) and a linear order R_i' (representing the "untruthful" ballot of i) such that:

$$F(R_i', \boldsymbol{R}_{-i}) \text{ is ranked above } F(\boldsymbol{R}) \text{ according to } R_i$$

- F is *surjective* if for every alternative $x \in A$ there is a profile $\boldsymbol{R}$ such that $F(\boldsymbol{R}) = x$. So no x is excluded from winning *a priori*.

The Gibbard-Satterthwaite Theorem

F is a *dictatorship* if there exists an $i \in N$ such that $F(\boldsymbol{R}) = \mathrm{top}(R_i)$ for every profile $\boldsymbol{R}$. Recall this central result of SCT:

Theorem 1 (Gibbard-Satterthwaite) *There exists no resolute SCF for $\geqslant 3$ alternatives that is surjective, strategyproof, and nondictatorial.*

Remarks:

- The theorem does not hold for $m = 2$ alternatives. (*Why?*)
- The theorem is trivially true for $n = 1$ voter. (*Why?*)

We will use a *SAT solver* to automatically prove that the theorem holds for the *smallest nontrivial case* (with $n = 2$ and $m = 3$).

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

M.A. Satterthwaite. Strategy-proofness and Arrow's Conditions. *Journal of Economic Theory*, 1975.

Approach

Technology: We use the solver *Lingeling* (`fmv.jku.at/lingeling/`). Lingeling can check whether a given formula in CNF is satisfiable. The formula must be represented as a *list of lists of integers*, corresponding to a *conjunction of disjunctions of literals*. Positive (negative) numbers represent positive (negative) literals.

Example: `[[1,-2,3], [-1,4]]` represents $(p \vee \neg q \vee r) \wedge (\neg p \vee s)$.

Idea: We use a *Python* script (Python3) to generate a propositional formula φ_{GS} that is satisfiable iff there exists a resolute SCF for $n = 2$ voters and $m = 3$ alternatives that is surjective, SP, and nondictatorial. Using Lingeling, we will show that φ_{GS} *is not satisfiable*.

Practicalities: To access Lingeling from Python we use the library `pylgl`, providing a function `solve` (`pypi.org/project/pylgl/`).

Example: `solve([[1], [-1,2], [-2]])` will result in `'UNSAT'`. ✓

Representing Basic Features of the Model

We choose to represent all basic features of the model as numbers:

- *voters* are represented as integers from 0 to $n-1$
- *alternatives* are represented as integers from 0 to $m-1$
- *preferences* are represented as integers from 0 to $m!-1$
- *profiles* are represented as integers from 0 to $(m!)^n-1$

In our Python program, we first fix n and m:

```
n = 2
m = 3
```

Basic functions to retrieve lists of all voters and so forth:

```
def allVoters():
    return range(n)

def allAlternatives():
    return range(m)
```

```
from math import factorial

def allProfiles():
    return range(factorial(m) ** n)
```

Let's Try!

All the code from this slide set is available as the Python script gs.py.

Compile it and try out the basic functions just defined:

```
$ python3 -i gs.py

>>> allAlternatives()
range(0, 3)
>>> list(allAlternatives())
[0, 1, 2]

>>> allProfiles()
range(0, 36)
>>> list(allProfiles())
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
 30, 31, 32, 33, 34, 35]

>>> exit()
```

Extracting Preferences from Profiles

Think of profiles as numbers with n *digits* in the number system with *base* $m!$. So voter i's preference in $\boldsymbol{R}$ 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"):

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

Interpreting Preferences

It can be useful to have an alternative representation of voter i's preference in a given profile $\boldsymbol{R}$ in the form of a list of alternatives:

```
from itertools import permutations

def preflist(i, r):
    preflists = list(permutations(allAlternatives()))
    return preflists[preference(i,r)]
```

We now can provide functions to check whether voter i prefers x to y in a given profile $\boldsymbol{R}$ and whether x is her top alternative:

```
def prefers(i, x, y, r):
    mylist =  preflist(i, r)
    return mylist.index(x) < mylist.index(y)

def top(i, x, r):
    mylist =  preflist(i, r)
    return mylist.index(x) == 0
```

Let's Try!

Look up the preference list of the first voter in the first profile:

```
>>> preflist(0,0)
(0, 1, 2)
```

And the one of the last voter in the last profile:

```
>>> preflist(1,35)
(2, 1, 0)
```

Let's inspect profile 17:

```
>>> preflist(0,17), preflist(1,17)
((2, 1, 0), (1, 0, 2))
```

Does voter 1 prefer alternative 1 to alternative 2 in that profile?

```
>>> prefers(1,1,2,17)
True
>>> prefers(1,2,1,17)
False
```

Restricting the Range of Quantification

When formulating axioms, we sometimes need to quantify over all alternatives that satisfy a certain (boolean) condition:

```
def alternatives(condition):
    return [x for x in allAlternatives() if condition(x)]
```

Let's try! You can now generate the list of all alternatives that meet the condition of being different from 1 (`condition` $= \lambda x.(x \neq 1)$).

```
>>> alternatives(lambda x : x!=1)
[0, 2]
```

And the corresponding functions for voters and profiles:

```
def voters(condition):
    return [i for i in allVoters() if condition(i)]

def profiles(condition):
    return [r for r in allProfiles() if condition(r)]
```

Literals

We can specify any (possibly irresolute) SCF $F : \mathcal{L}(A)^n \to 2^A \setminus \{\emptyset\}$ by saying whether or not $x \in F(\boldsymbol{R})$ for every profile $\boldsymbol{R}$ and alternative x.

So create a propositional variable $p_{\boldsymbol{R},x}$ for every profile $\boldsymbol{R} \in \mathcal{L}(A)^n$ and every alternative $x \in A$, with the intended meaning that:

$$p_{\boldsymbol{R},x} \text{ is } \textit{true} \ \ \underline{\text{iff}} \ \ x \in F(\boldsymbol{R})$$

Exercise: *How many variables for $n = 2$ voters and $m = 3$ alternatives?*

Need to decide *which number* to use to represent $p_{\boldsymbol{R},x}$. Good option:

```
def posLiteral(r, x):
    return r * m + x + 1
```

Recall: `r` $\in \{0, \ldots, (m!)^n - 1\}$
and `x` $\in \{0, \ldots, m-1\}$

And negative literals are represented by negative numbers:

```
def negLiteral(r, x):
    return (-1) * posLiteral(r, x)
```

Let's Try!

The very first literal (1) tells us whether the first alternative (0) is winning in the first profile (0), while the last literal (108) tells us whether the last alternative (2) is winning in the last profile (35).

```
>>> posLiteral(0,0)
1
>>> posLiteral(35,2)
108
```

Modelling Social Choice Functions

Every assignment of truth values to our *108 variables* $p_{\boldsymbol{R},x}$ corresponds to a function $F : \mathcal{L}(A)^n \to 2^A$ (in case $n = 2$ and $m = 3$).

But: a (possibly irresolute) SCF is a function $F : \mathcal{L}(A)^n \to 2^A \setminus \{\emptyset\}$.

Fix this by restricting attention to models of this formula:

$$\varphi_{\text{at-least-one}} \quad = \quad \bigwedge_{\boldsymbol{R} \in \mathcal{L}(A)^n} \left(\bigvee_{x \in A} p_{\boldsymbol{R},x} \right)$$

The following function will generate this formula:

```
def cnfAtLeastOne():
    cnf = []
    for r in allProfiles():
        cnf.append([posLiteral(r,x) for x in allAlternatives()])
    return cnf
```

Let's Try!

Let's have a look at this formula (in the so-called *DIMACS format*):

```
>>> cnfAtLeastOne()
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15],
 [16, 17, 18], [19, 20, 21], [22, 23, 24], [25, 26, 27], [28,
 29, 30], [31, 32, 33], [34, 35, 36], [37, 38, 39], [40, 41,
 42], [43, 44, 45], [46, 47, 48], [49, 50, 51], [52, 53, 54],
 [55, 56, 57], [58, 59, 60], [61, 62, 63], [64, 65, 66], [67,
 68, 69], [70, 71, 72], [73, 74, 75], [76, 77, 78], [79, 80,
 81], [82, 83, 84], [85, 86, 87], [88, 89, 90], [91, 92, 93],
 [94, 95, 96], [97, 98, 99], [100, 101, 102], [103, 104, 105],
 [106, 107, 108]]
```

Nice: We really get $(3!)^2 = 36$ clauses of 3 positive literals each.

Resoluteness

We now write a similar function for each one of our axioms.

F is *resolute* if for *all* profiles $\boldsymbol{R}$ and *all* alternatives $x \neq y$ it is the case that $x \notin F(\boldsymbol{R})$ *or* $y \notin F(\boldsymbol{R})$. So: *at most one winner* per profile.

Note: Can restrict quantification to $x < y$ (when taken as numbers).

$$\varphi_{\text{resolute}} \quad = \quad \bigwedge_{\boldsymbol{R} \in \mathcal{L}(A)^n} \left(\bigwedge_{x \in A} \left(\bigwedge_{\substack{y \in A \\ \text{s.t. } x < y}} \neg p_{\boldsymbol{R},x} \vee \neg p_{\boldsymbol{R},y} \right) \right)$$

```
def cnfResolute():
    cnf = []
    for r in allProfiles():
        for x in allAlternatives():
            for y in alternatives(lambda y : x < y):
                cnf.append([negLiteral(r,x), negLiteral(r,y)])
    return cnf
```

Surjectivity

Surjectivity is most naturally expressed as a conjunction of disjunctions of conjunctions. (*How?*) Could translate to CNF, but this is easier:

If F is already known to be *resolute*, then F is *surjective* if: for *all* alternatives x there *exists* a profile $\boldsymbol{R}$ such that $x \in F(\boldsymbol{R})$.

$$\varphi_{\text{surjective}} \quad = \quad \bigwedge_{x \in A} \left(\bigvee_{\boldsymbol{R} \in \mathcal{L}(A)^n} p_{\boldsymbol{R},x} \right)$$

```
def cnfSurjective():
    cnf = []
    for x in allAlternatives():
        cnf.append([posLiteral(r,x) for r in allProfiles()])
    return cnf
```

Preparation for Modelling Strategyproofness

To model strategyproofness we need to be able to model two profiles being so-called *i-variants* (for some voter $i \in N$):

$$\boldsymbol{R} =_{-i} \boldsymbol{R}' \quad \underline{\text{iff}} \quad R_j = R'_j \text{ for all voters } j \in N \setminus \{i\}$$

Recall: `preference(j,r)` returns the preference of voter `j` in profile `r`

Now our implementation is straightforward:

```
def iVariants(i, r1, r2):
    return all(preference(j,r1) == preference(j,r2)
                   for j in voters(lambda j : j!=i))
```

Note: Here the Python function `all()` tests whether all the boolean expressions in the list it is applied to evaluate to *true*.

Strategyproofness

Resolute F is *strategyproof* if for *all* voters i, *all* (truthful) profiles $\boldsymbol{R}_1$, *all* of its *i-variants* $\boldsymbol{R}_2$, *all* alternatives x, and *all* alternatives y *dispreferred* to x by i in $\boldsymbol{R}_1$ we have: $F(\boldsymbol{R}_1) = y$ *implies* $F(\boldsymbol{R}_2) \neq x$.

$$\varphi_{\mathsf{SP}} = \bigwedge_{i \in N} \left(\bigwedge_{\boldsymbol{R}_1 \in \mathcal{L}(A)^n} \left(\bigwedge_{\substack{\boldsymbol{R}_2 \in \mathcal{L}(A)^n \\ \text{s.t. } \boldsymbol{R}_1 =_{-i} \boldsymbol{R}_2}} \left(\bigwedge_{x \in A} \left(\bigwedge_{\substack{y \in A \\ \text{s.t. } i \in N^{\boldsymbol{R}_1}_{x \succ y}}} \neg p_{\boldsymbol{R}_1, y} \vee \neg p_{\boldsymbol{R}_2, x} \right)\right)\right)\right)$$

```
def cnfStrategyProof():
    cnf = []
    for i in allVoters():
        for r1 in allProfiles():
            for r2 in profiles(lambda r2 : iVariants(i,r1,r2)):
                for x in allAlternatives():
                    for y in alternatives(lambda y : prefers(i,x,y,r1)):
                        cnf.append([negLiteral(r1,y), negLiteral(r2,x)])
    return cnf
```

Nondictatorship

Resolute F is *nondictatorial* if for *all* voters i there *exists* a profile $\boldsymbol{R}$ such that $F(\boldsymbol{R}) \neq x$ for alternative $x = \text{top}_i(\boldsymbol{R})$.

$$\varphi_{\text{nondictatorial}} \quad = \quad \bigwedge_{i \in N} \left(\bigvee_{\boldsymbol{R} \in \mathcal{L}(A)^n} \left(\bigvee_{\substack{x \in A \\ \text{s.t. } x = \text{top}_i(\boldsymbol{R})}} \neg p_{\boldsymbol{R},x} \right) \right)$$

this works as $x = \text{top}_i(\boldsymbol{R})$ for just <u>one</u> x

```
def cnfNondictatorial():
    cnf = []
    for i in allVoters():
        clause = []
        for r in allProfiles():
            for x in alternatives(lambda x : top(i,x,r)):
                clause.append(negLiteral(r,x))
        cnf.append(clause)
    return cnf
```

Proving the (Special Case of the) Theorem

Putting it all together:

```
>>> cnf = ( cnfAtLeastOne() + cnfResolute() + cnfSurjective()
...        + cnfStrategyProof() + cnfNondictatorial() )
```

This is a conjunction of 1445 clauses (using 108 variables, as we saw):

```
>>> len(cnf)
1445
```

We make Lingeling available like this:

```
from pylgl import solve
```

And now the moment of truth has come:

```
>>> solve(cnf)
'UNSAT'
```

Done! So the G-S Theorem really holds for $n = 2$ and $m = 3$. Nice. ✓

Exercise: *Reproduce this result on your own machine!*

Discussion: Confidence in Computer Proofs?

Some will object to this approach. *Can we trust this kind of proof?*
Your computer-generated proof using a SAT solver is valid <u>only if:</u>

- your *encoding* of your question into propositional logic is correct
- the implementation of the *SAT solver* is correct
- the *environment* the solver is running in works to specification

Arguments in favour of the approach:

- If your encoding of the problem is short, clean, and systematic, then it can be *proof-read* in the same way as a regular proof.
- Due to standardised input/output format for SAT solvers, you can verify the correctness of your proof using *third-party tools*.

<u>Still:</u> This proof does not provide insight into *why* finding a suitable SCF is impossible. (We will return to this issue next time.)

Proving the Full Theorem

We now know that the Gibbard-Satterthwaite Theorem is true for the special case of $n = 2$ and $m = 3$. *What about larger values?*

- Intuitively, it seems that things will only get "more impossible" when we increase n or m. So we should be fine.
- But formally proving this intuition to be correct actually is not straightforward. (To be discussed next time.)

Other Uses of the Program

For $n = 2$ and $m = 3$, how many resolute rules are strategyproof?
This is a question we can answer with the help of our program.

First, construct the corresponding CNF:

```
>>> cnf = cnfAtLeastOne() + cnfResolute() + cnfStrategyProof()
```

We can use `solve()` to find *one* rule satisfying our requirements:

```
>>> solve(cnf)
[-1, 2, -3, -4, -5, 6, -7, 8, -9, -10, 11, -12, ..., 108]
```

Exercise: *Write a program to make rule specs such as this readable.*

Using `itersolve()`, we can get *all* such rules (and count them):

```
>>> from pylgl import itersolve
>>> rules = itersolve(cnf)
>>> len(list(rules))
17
```

Exercise: *What are those 17 rules? Provide a suitable classification.*

Exercise: Duggan-Schwartz Theorem

To help you get some practice, I want you to try and encode the simplest nointrivial instance of the *Duggan-Schwartz Theorem*.

So what's the D-S Theorem? Generalising G-S to irresolute rules . . .

Manipulability w.r.t. Psychological Assumptions

To analyse manipulability when we might get a set of winners, we need to make assumptions on how voters rank *sets of alternatives*, e.g.:

- A voter is an *optimist* if she prefers X over Y whenever she prefers her favourite $x \in X$ over her favourite $y \in Y$.
- A voter is a *pessimist* if she prefers X over Y whenever she prefers her least preferred $x \in X$ over her least preferred $y \in Y$.

Now we can speak about manipulability by certain types of voters:

- F is called *immune to manipulation by optimistic voters* if no optimistic voter can ever benefit from voting untruthfully.
- F is called *immune to manipulation by pessimistic voters* if no pessimistic voter can ever benefit from voting untruthfully.

Axiom: Nonimposition

Let F be an *irresolute* voting rule $F : \mathcal{L}(A)^n \to 2^A \setminus \{\emptyset\}$.

- F is *nonimposed* if for every alternative x there exists a profile $\boldsymbol{R}$ under which x is the unique winner: $F(\boldsymbol{R}) = \{x\}$.

For comparison, *surjectivity* means that for every element in the co-domain of F there is an input producing that element. Thus:

$$\text{resolute} \Rightarrow (\text{nonimposed} = \text{surjective})$$

Dictatorships for Irresolute Rules

Let F be an *irresolute* voting rule $F : \mathcal{L}(A)^n \to 2^A \setminus \{\emptyset\}$.

There are two natural notions of dictatorship for such rules:

- Voter $i \in N$ is called a (strong) *dictator* if $F(\boldsymbol{R}) = \{\text{top}(R_i)\}$ for every profile $\boldsymbol{R} \in \mathcal{L}(A)^n$.
- Voter $i \in N$ is called a *weak dictator* if $\text{top}(R_i) \in F(\boldsymbol{R})$ for every profile $\boldsymbol{R} \in \mathcal{L}(A)^n$. (Such a voter is also called a *nominator*.)

F is called *weakly dictatorial* if it has a weak dictator.

Otherwise F is called *strongly nondictatorial*.

The Duggan-Schwartz Theorem

There are several extensions of the G-S Theorem for irresolute rules. The D-S Theorem is regarded as the most important such result.

Our statement of the theorem follows Taylor (2002):

Theorem 2 (Duggan and Schwartz, 2000) *Any voting rule for $\geqslant 3$ alternatives that is nonimposed and immune to manipulation by both optimistic and pessimistic voters is weakly dictatorial.*

Observe that the G-S Theorem is a direct corollary. (*Why?*)

J. Duggan and T. Schwartz. Strategic Manipulation w/o Resoluteness or Shared Beliefs: Gibbard-Satterthwaite Generalized. *Social Choice and Welfare*, 2000.

A.D. Taylor. The Manipulability of Voting Systems. *The American Mathematical Monthly*, 2002.

Summary

This has been an introduction to the use of SAT solvers to obtain axiomatic results in social choice theory.

We focused on a hands-on example: using the approach to prove the "base case" of the Gibbard-Satterthwaite Theorem.

Tentative plan for the next (couple of) meeting(s):

- discussing your solutions to the Duggan-Schwartz exercise
- proving the full Gibbard-Satterthwaite Theorem by induction
- trying to obtain human-readable proofs using SAT technology
- applications beyond impossibility theorems
- broader discussion of using logic and automated reasoning for SCT

Duggan-Schwartz Exercise: Full Details (1)

Prove the Duggan-Schwartz Theorem for the special case of $n = 2$ voters and $m = 3$ alternatives using the SAT solving technique.

Reuse as much of our code as you like (but make it a habit to always indicate what you have copied and what you have changed exactly, if anything).

This is a difficult exercise, although modelling the requirement of the voting rule being strongly nondictatorial is relatively straightforward. So start with that. Modelling the two strategyproofness axioms requires some careful thinking, but you should end up with a fairly simple implementation as well. The main challenge is modelling nonimposition, which most immediately corresponds to a conjunction of disjunctions of conjunctions of literals. Translating this into CNF is impractical: the resulting formula would be huge (a conjunction of almost half a quintillion clauses of length 36).

But you can use this trick: Introduce *auxiliary variables* $q_{\boldsymbol{R},x}$ with the intended meaning that in profile $\boldsymbol{R}$ alternative x is the *only* winner.

Duggan-Schwartz Exercise: Full Details (2)

Then express the axiom of nonimposition with the help of these auxiliary variables, and fix their meaning by adding clauses that together enforce the following constraint for all profiles $\boldsymbol{R}$ and (distinct) alternatives x, y, and z:

$$q_{\boldsymbol{R},x} \leftrightarrow p_{\boldsymbol{R},x} \wedge \neg p_{\boldsymbol{R},y} \wedge \neg p_{\boldsymbol{R},z}$$

To make it easier for us to compare solutions, for each of your axioms, please use `len()` to count the *number of clauses* involved.

Besides proving the theorem, also demonstrate that for each of the four axioms featuring in the theorem it is *possible to design a voting rule* that satisfies the other three axioms (again, for the case of $n = 2$ and $m = 3$).

Try to find out *how many* such voting rules there are for each of those four cases. Keep in mind that this corresponds to very demanding queries for the SAT solver, so you may not be able to obtain an answer in a reasonable amount of time (just treat anything above 15 minutes as a *timeout*).