Computational Aspects of Voting Paradoxes and Axioms

Farhad Mohsin

April 8, 2024

Voting Rules

A voting rule is a mapping from a collection of individual preferences to a single decision.[1]

- For a set of alternatives, A, let $\mathcal{L}(A)$ be the set of all rankings (linear orders) over the items in A.
- $P \in \mathcal{L}(A)^n$ is called a preference profile.
- For n agents, a voting rule, $r : \mathcal{L}(A)^n \mapsto A$ outputs a winner.
- Since $\mid \mathcal{L}(A) \mid = m!$, we can abuse notation a bit and define $r : [0,1]^{m!} \mapsto A$ to be a function of a *normalized* preference profile.

[1] Felix Brandt et al. *Handbook of computational social choice*. Cambridge University Press, 2016.

Direction 1: Voting Paradoxes

We expect some axiomatic properties from voting rules. For example,

- Condorcet property: If a beats every other alternative in pairwise competition, a should win.
- Consistency: If $r(P) = a$ and $r(Q) = a$, then $r(P \cup Q) = 1$.
- Participation: For any voter (or group of voters), participating truthfully should always increase the likelihood of winning for your preferred alternative.

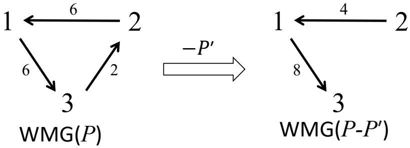
Voting paradoxes occur when these axiomatic properties are violated in a preference profile.

(Group) No-show Paradox (GNSP)

Definition (**Group no-show paradox**)

A *group no-show paradox* (GNSP) occurs in profile P if there exists a subset profile P', all of whom prefer $r(P - P')$ to $r(P)$, incentivizing them to abstain from voting.[a]

[a] Farhad Mohsin et al. "Computational Complexity of Verifying the Group No-show Paradox". In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems.* 2023, pp. 2877–2879.



For this case, Copeland winner shifts from 1 to 2.

Computational Verification of GNSP

GNSP doesn't occur for voting rules like plurality, Borda (positional scoring rules), but is a problem for pairwise comparison-based or runoff-based voting rules.

Theorem (Simplified)

The verification of whether GNSP occurs under a voting rule for a preference profile is an NP-complete to compute.[2]

So, *when* can we computationally verify the occurrence of GNSP?

- Small number of voters?
- Small number of alternatives?
- Never?

[2] NP-complete in terms of number of alternatives and size of preference profile.

Integer Linear Program (ILP) formulation for GNSP

Defining Variables

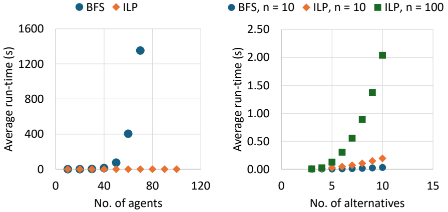
- Assume $r(P) = a$, for some voting rule, r
- We want to know if b can be $r(P)$ instead by GNSP.
- For all rankings, R, with $b \succ a$, $x_R \leq n_R$ is a variable

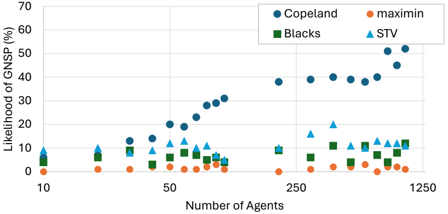
Defining ILP

- Different for each voting rule
- Convert voting rule definition to linear constraints
- Minimize number of voters needed to abstain ($\min \sum_R (n_R - x_R)$)

Experimental Results

ILP performs efficiently as long as number of voters or number of unique rankings in preference profile is small.





Conclusions about GNSP

- If only no-showing is enough for manipulating the results, then the voting rule lacks "robustness"
- Pre-conditions too strong? Need knowledge of all other voters.
- How about knowledge about voter distribution?

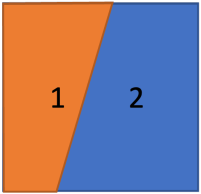
Machine Learning to Design Voting Rules

No voting rule is robust against *all voting paradoxes*.

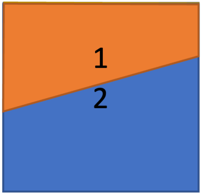
- So, we always want to design *better voting rules*.
- We may also get new requirements, e.g., voting rule fair to different demographics.
- Goal: Demographically Fair voting rules[3]
    - Create simulated election dataset
    - Mix profiles with fair winner and profiles with regular winner
    - Train classifier on mixed dataset
    - (Note: Gradient boosted models work best)
    - Use classifier as new voting rule
- Expectation
    - Fairer than traditional, efficient rule
    - More efficient than purely fair rule

[3] Farhad Mohsin et al. "Learning to design fair and private voting rules". In: *Journal of Artificial Intelligence Research* 75 (2022), pp. 1139–1176.

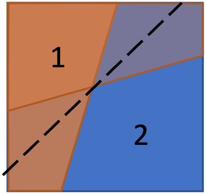
ML to Design Voting Rules (contd.)



Efficient Rule

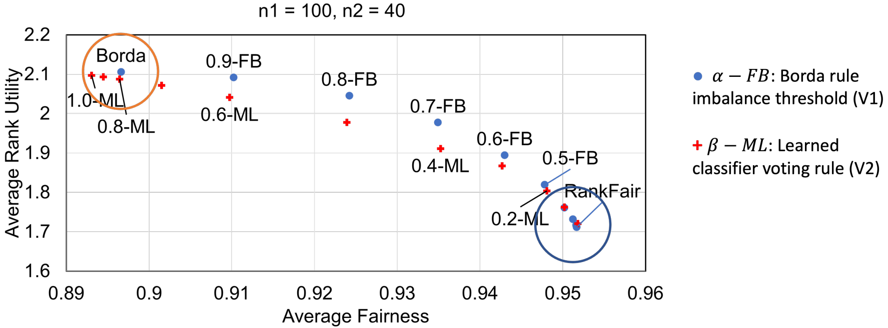


Fair Rule



Mixed labels e.g.
70% efficient + 30% fair

Empirical Results



Computational Way of Designing Voting Rules

Let's revisit some axioms with normalized profiles. First, **Consistency.**

- If $r(P) = r(Q) = a$, then $r(P + U) = a$
- Generalize this to: if $r(P) = r(Q) = a$, then $r(\lambda P + (1 - \lambda)Q) = a$.
- Convex Sets?

As it turns out, for all *always-consistent* voting rules, all preference profiles with the same winner belong in a convex set.[4]

[4] Lirong Xia. "Generalized scoring rules: a framework that reconciles Borda and Condorcet". In: *ACM SIGecom Exchanges* 12.1 (2013), pp. 42–48.

Embeddings and Better Design of Voting Rules

- Neural Networks essentially learn vector embeddings of inputs
- If we learn embeddings that maintain convexity, we get consistency for free.
- Possible using Input Convex Neural Networks[5]
- How about other properties?
- Still thinking/experimenting on it

[5] Brandon Amos, Lei Xu, and J Zico Kolter. "Input convex neural networks". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 146–155.