Algorithms and Systems for Computational Social Choice : Incorporating Context into Preference Aggregation

M1 internship, New York University

Théo Delemazure

Ecole normale supérieure

September 5, 2019

Internship

Supervisor : Pr. Julia Stoyanovich

Project : *Databases meet Computational Social Choice* (DBCOMSOC)

What is an election ?

A **voting profile** + a **voting rule**

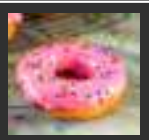
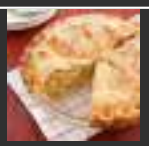
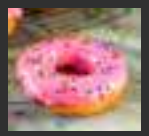
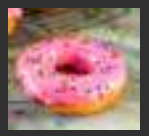
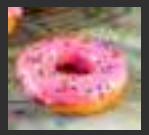
	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}
1						
2						
3						
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
n						

TABLE – Example of a Voting profile ($m = 6$)

What is an election ?

A **voting profile** + a **voting rule**

In this project, we are interested in positional scoring rules.

	1^{st}	2^{nd}	3^{rd}	$\dots$	$m-1^{th}$	m^{th}
Borda	$m-1$	$m-2$	$m-3$	$\dots$	1	0
2-approval	1	1	0	$\dots$	0	0
Plurality	1	0	0	$\dots$	0	0
Veto	1	1	1	$\dots$	1	0
Formula 1	25	18	15	$\dots$	0	0

TABLE – Some positional scoring rules

What if we have partial preferences ?

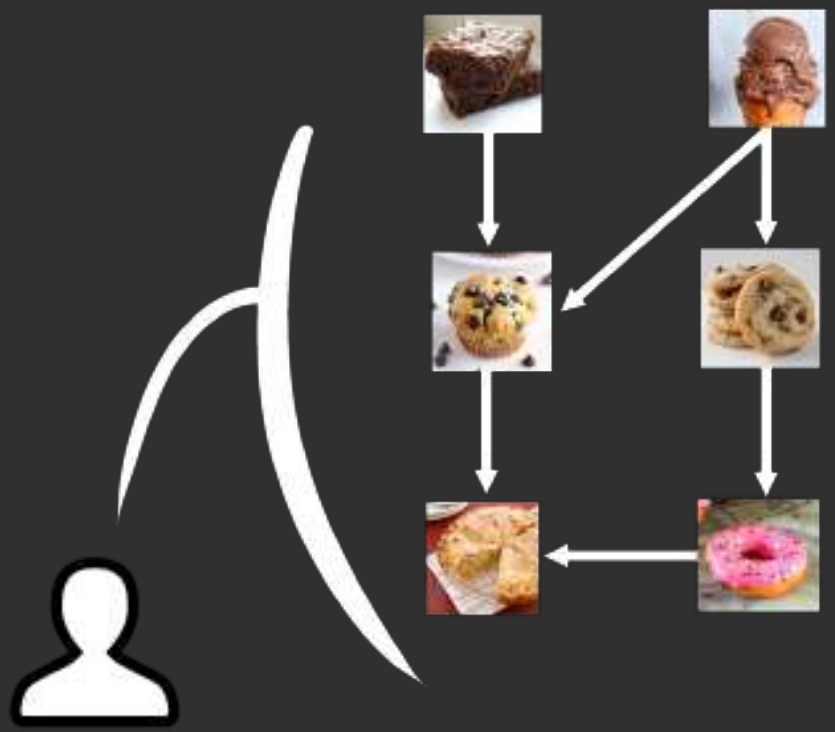


FIGURE – Which dessert is preferred between the Brownie and the Ice cream ?

Families of partial orders

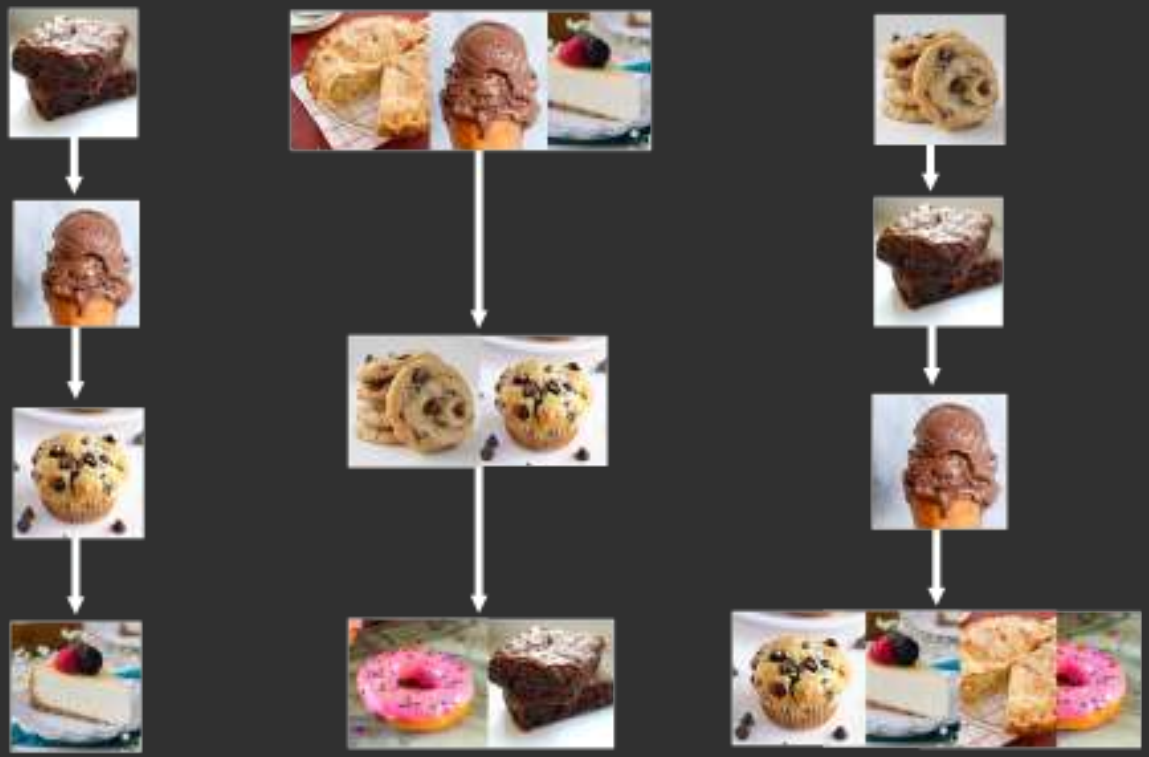


FIGURE – Some families of partial orders : linear posets, partitioned posets and top-k posets.

Definitions

A complete voting profile $\mathcal{T} = (T_1, \ldots, T_n)$ is a set of n total orders.

A partial voting profile $\mathcal{P} = (P_1, \ldots, P_n)$ is a set of n partial orders.

A completion of $\mathcal{P}$ is a complete voting profile $\mathcal{T}$ such that $\forall i, T_i$ is a total order that **extends** the partial order P_i.

NW and PW problems

Let r be a voting rule and $\mathcal{P}$ a partial voting profile.

- A candidate c is a necessary winner with respect to r and $\mathcal{P}$ if it is a winner in every completion of $\mathcal{P}$ for the voting rule r.
- A candidate c is a possible winner with respect to r and $\mathcal{P}$ if it is a winner in at least one completion of $\mathcal{P}$ for the voting rule r.

Summary

Summary

Why we needed data ?

We needed to test our algorithms for Necessary and Possible Winners. However, there is no partial order datasets available online, so we use three methods to obtain such datasets :

1. Generate synthetic data with partial order generation models.
2. Collect our own dataset of partial orders.
3. Transform preferences datasets available online into partial orders datasets.

Data generation methods

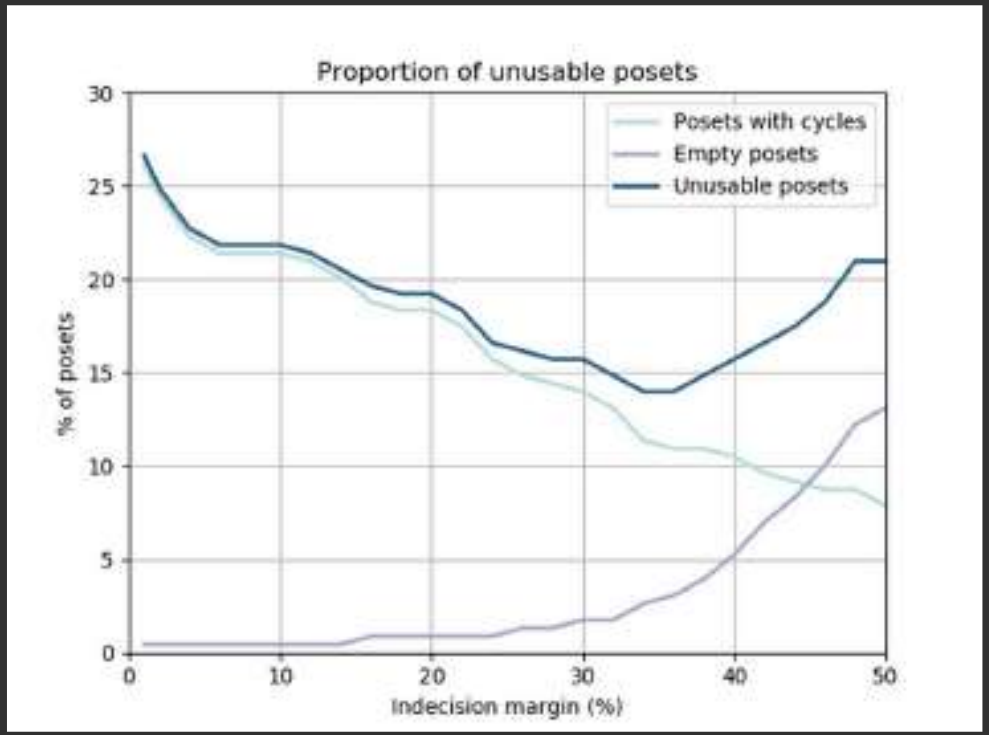
We used three different methods to **generate synthetic data** :

1. Drop candidate : After sampling a total order, we randomly drop k candidate from it. It generate only linear order.
2. Top-k : After sampling a total order, randomly choose k the number of candidates in top of the partial order.
3. The *Random Selection Model* (RSM) : A new method I proposed, largely inspired from the *Random Insertion Model*, but generalizable to partial orders.

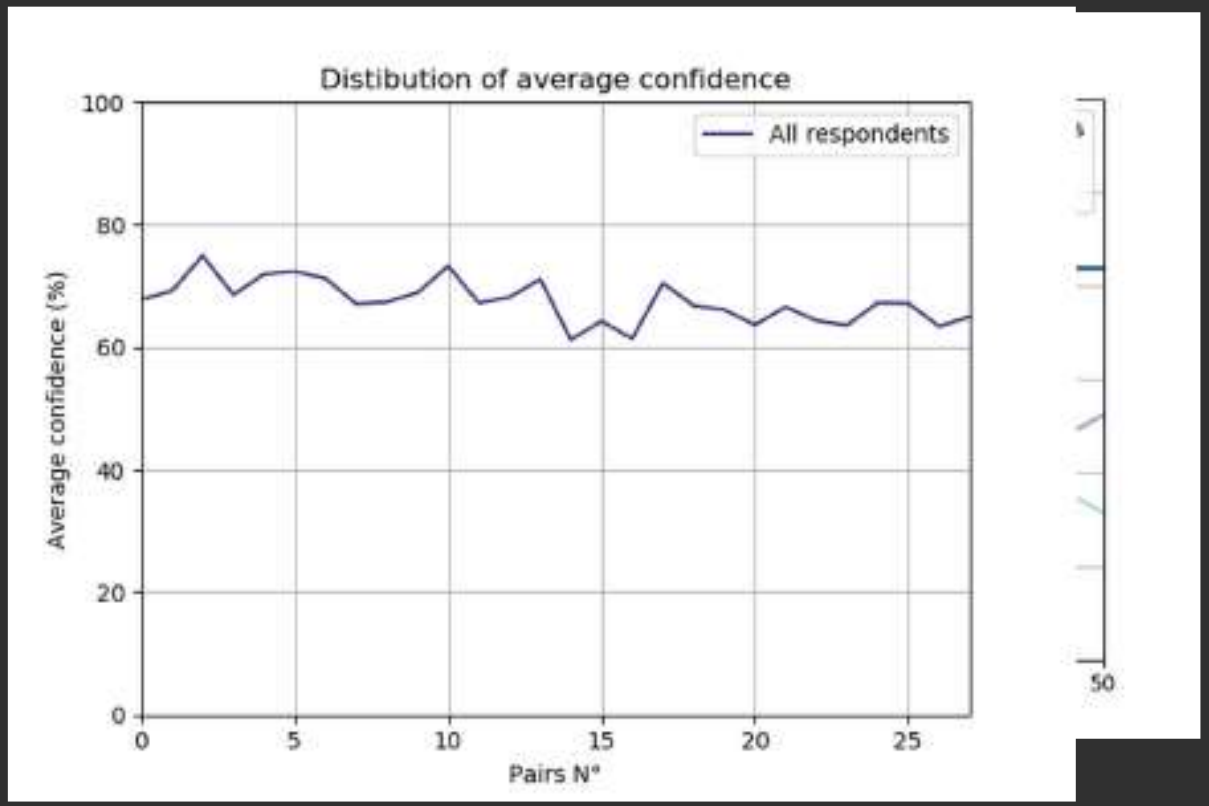
The dessert experiment : Demonstration

Demonstration

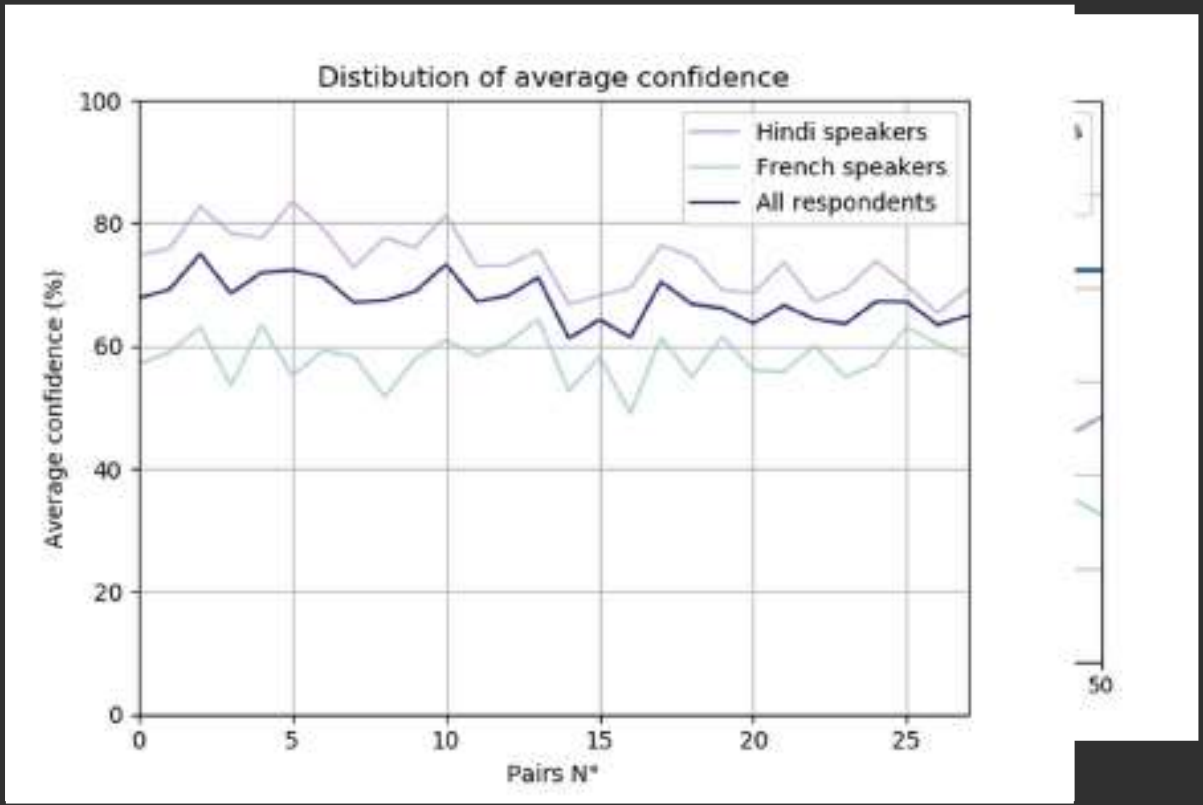
The dessert experiment : Results analysis



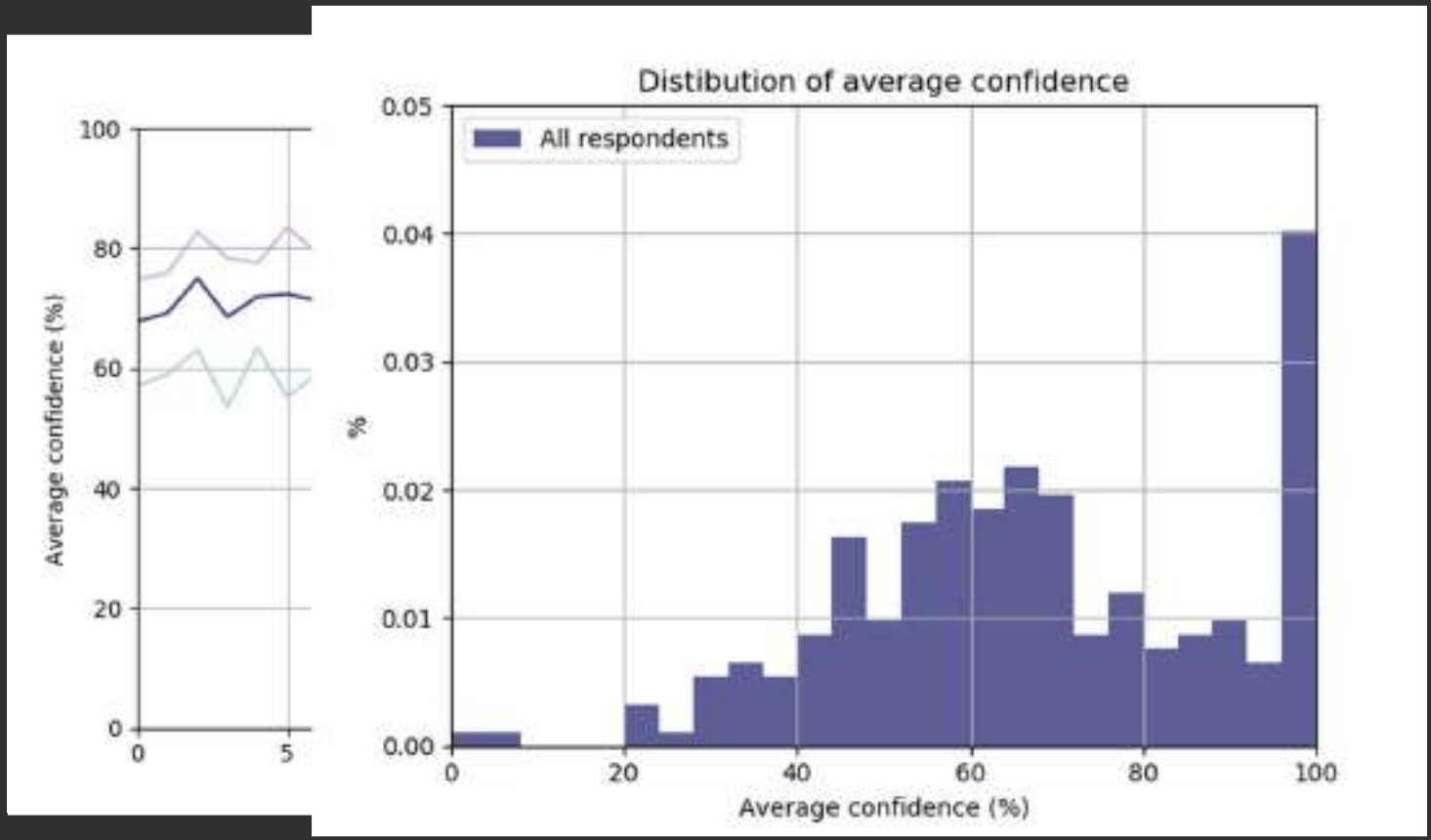
The dessert experiment : Results analysis



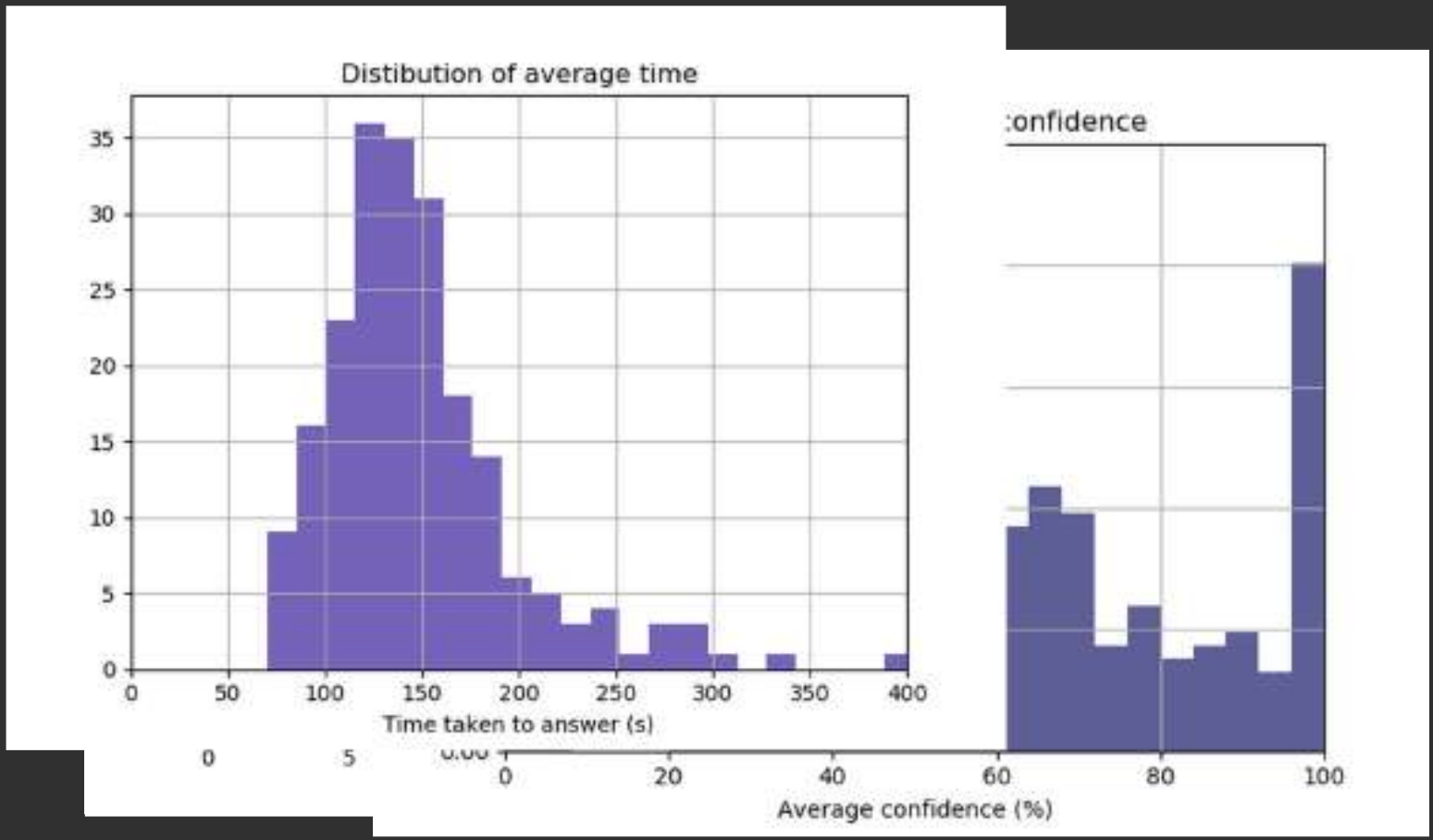
The dessert experiment : Results analysis



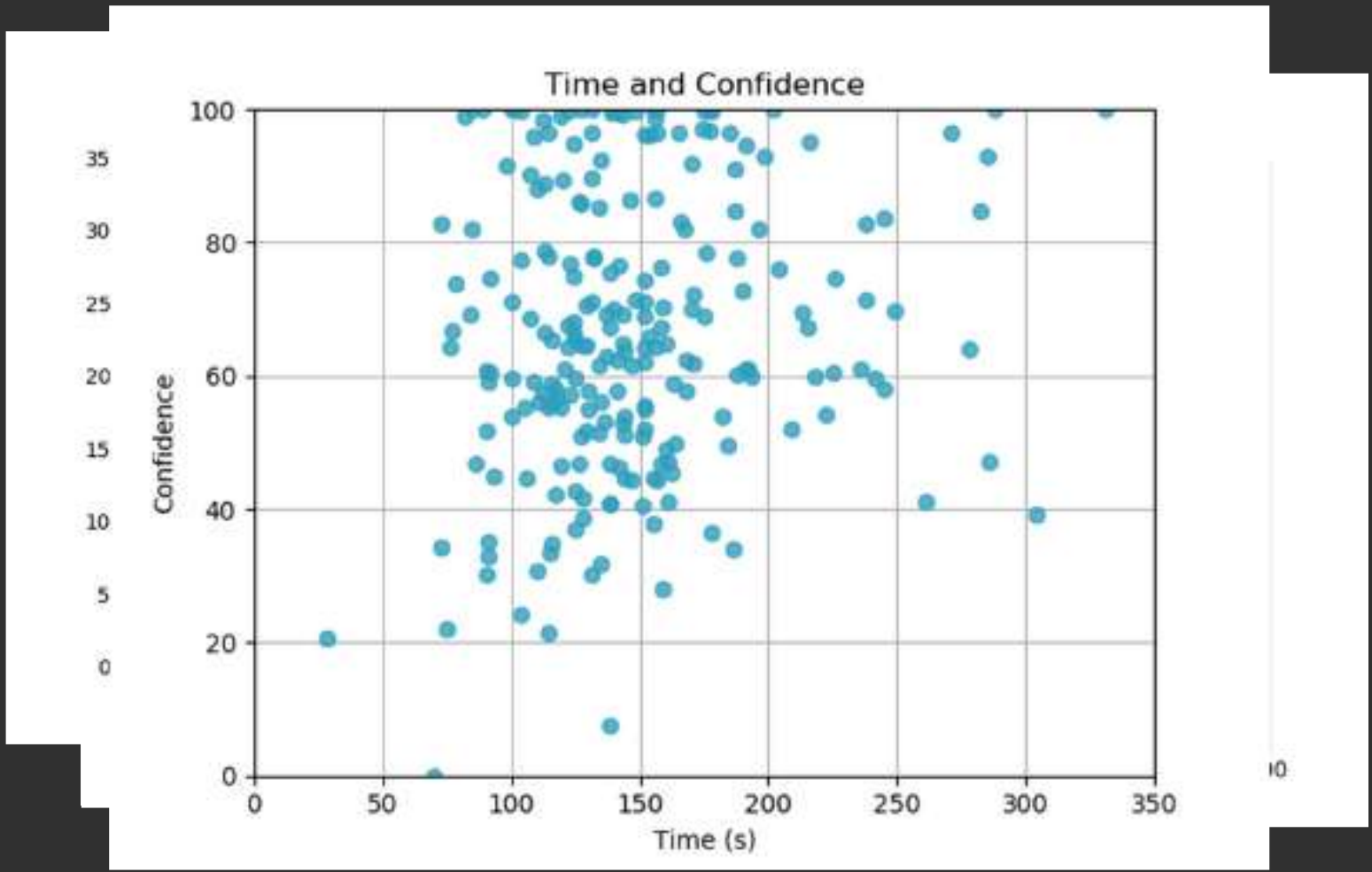
The dessert experiment : Results analysis



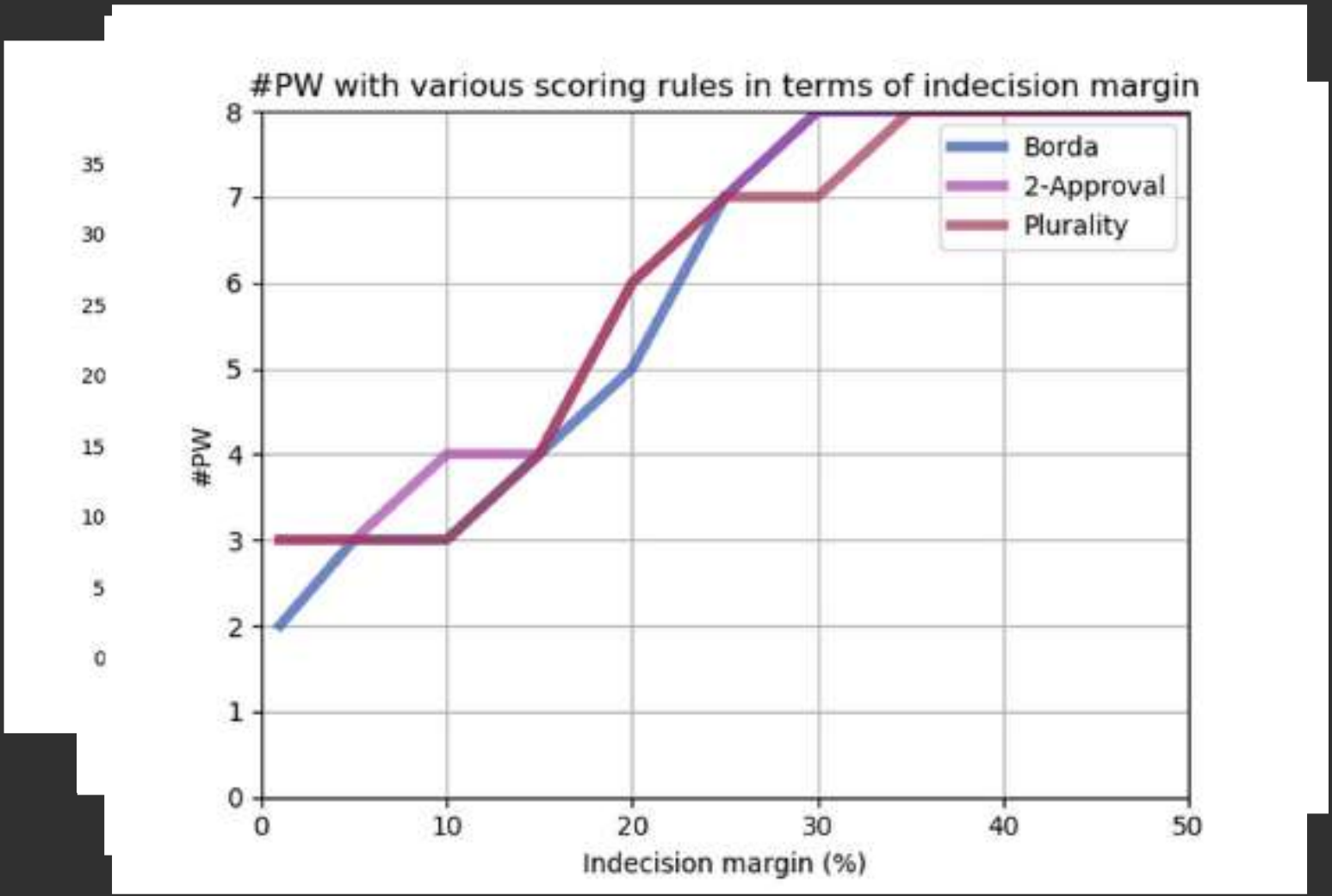
The dessert experiment : Results analysis



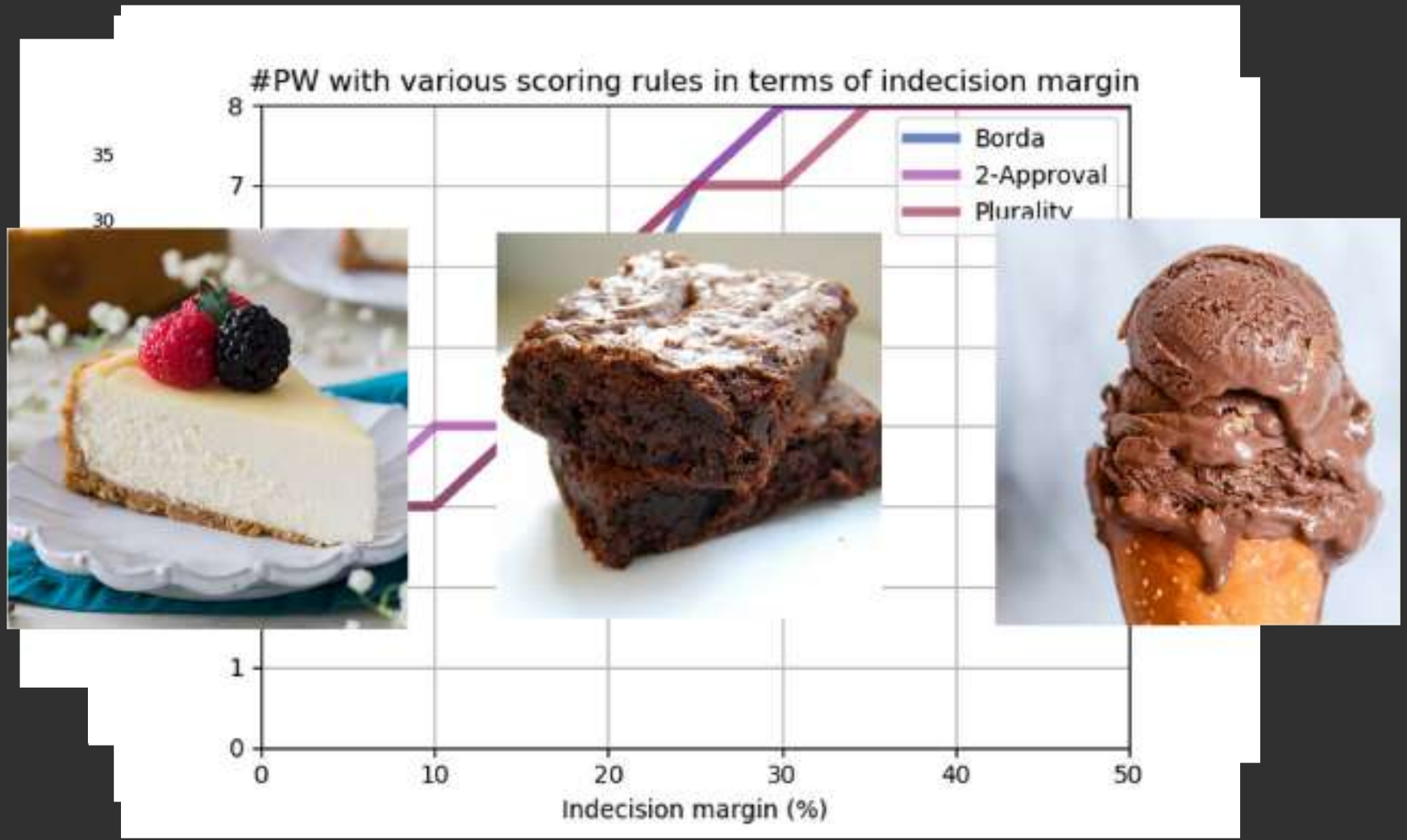
The dessert experiment : Results analysis



The dessert experiment : Results analysis



The dessert experiment : Results analysis



Data transformation : From ratings to rankings

Books dataset

candidates	241
voters	1288
density	0.02

Travel dataset

candidates	24
voters	5456
density	0.91

Summary

Introduction

Determining possible and necessary winners under common voting rules given partial orders (2011), *Xia and Conitzer*.

This algorithm can be summarized into two steps :

1. Compute `Up` and `Down`.
2. Simulate competitions between candidates to find a Necessary Winner.

Step 1 : Compute Up and Down

- General case : We use a **BFS-like** algorithm to compute these values. *Complexity :* $O(m^2)$*/voter*.
- Optimization for linear and partitioned poset : Easier computation for these kind of partial order. *Complexity :* $O(m)$*/voter*.

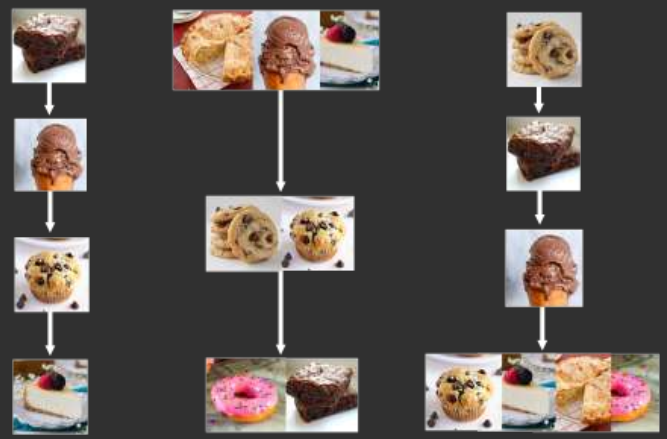


FIGURE – Families of partial orders

Step 2 : Simulate competitions

 is a Necessary Winner

$\Leftrightarrow$  always has a higher score than any other candidate

$\Leftrightarrow$ The value $\min$ *score*() - *score*(c) is ≥ 0 for every candidate $c \neq$  .

$\Rightarrow$ To minimize the difference of total score between two candidates  and  , we simply minimize the difference of score between them for every voter individually. There is two cases :

 $\succ_P$ 

$\Rightarrow$ Complexity $O(m)$

 $\not\succ_P$ 

$\Rightarrow$ Complexity $O(1)$

Reduce the number of competitions

- Original algorithm : $O(m^2)$ competitions in the worst case.
- Which candidates to test : The only candidates who can be NW are the candidates with the highest maximal score.
- Order the opponents : We order the opponents in the decreasing order of their maximal score.

	#**Competitions** ***(0 NW)***	#**Competitions** ***(1 NW)***
Non optimized	91	134
Optimized	1	49

TABLE – Example for $m = 50$ (average on 25 datasets, 4 with a NW)

Reduce the running time of a competition

- Original algorithm : $O(m)$ per voter in the worst case.
- Optimization for Borda and k-approval : $O(1)$ per voter.
- For other rules, we can use a preprocessing step to reduce the running time to $O(1)$ per voter.

	Borda	*k-approval*	*Formula 1*
Non optimized	$O(mn)$	$O(mn)$	$O(mn)$
Optimized	$O(n)$	$O(n)$	$O(\min(m^3+n, mn))$

TABLE – Complexities of optimized and non optimized version for different voting rules.

Time of each step

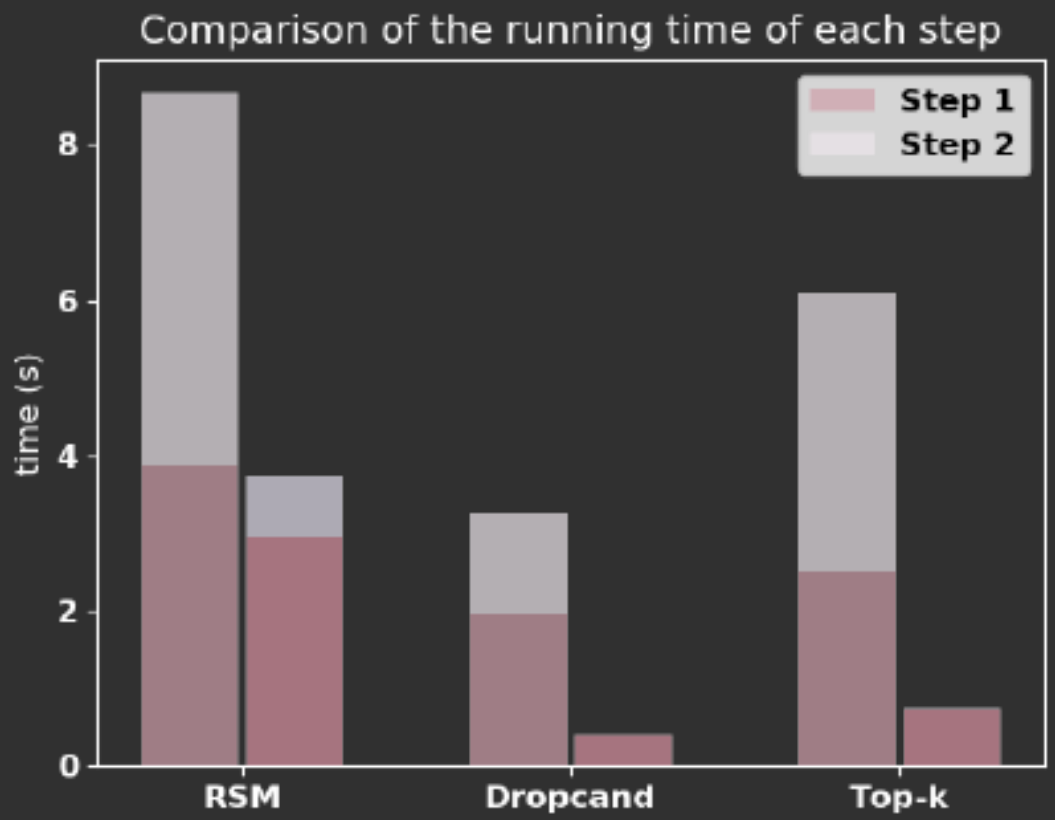
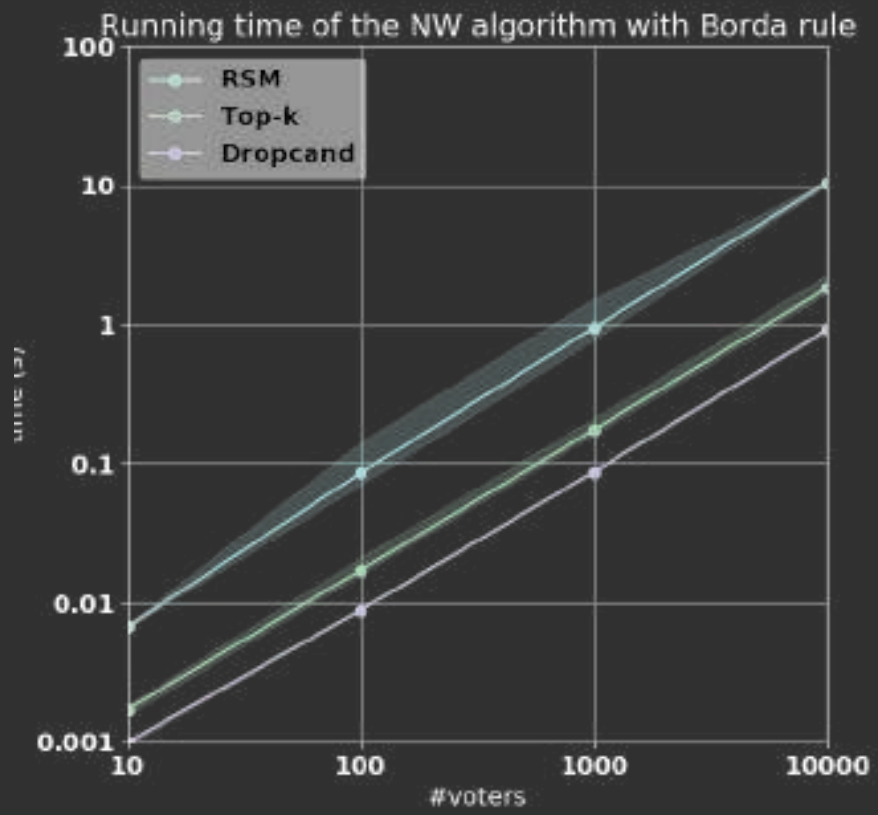
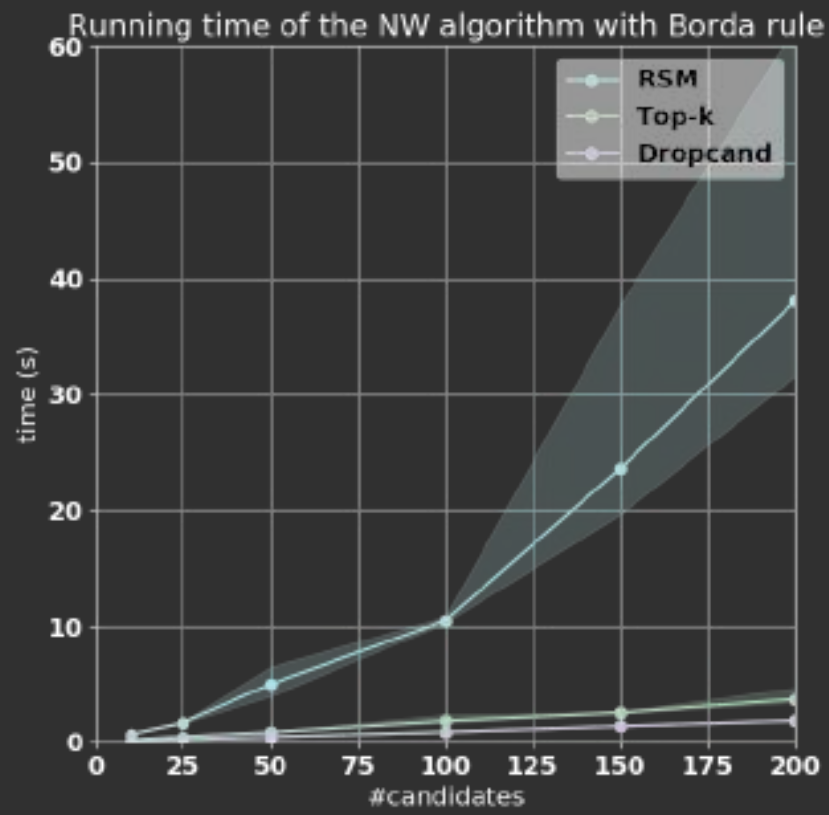


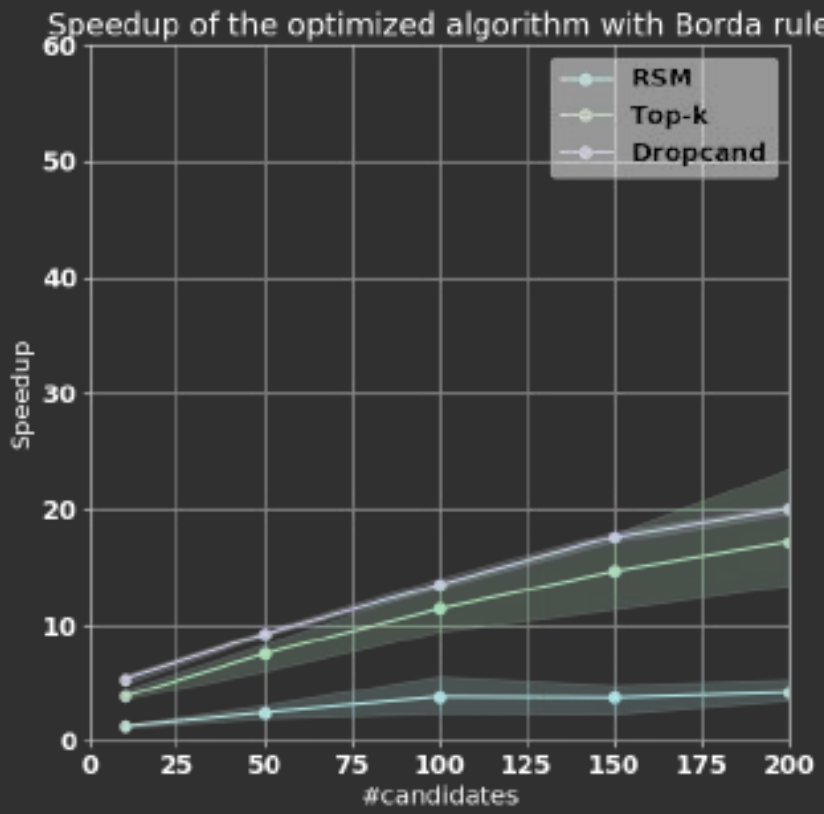
FIGURE – Comparisons of the running time of the two steps for optimized and non optimized version (Borda rule, $m = 50$, $n = 10,000$)

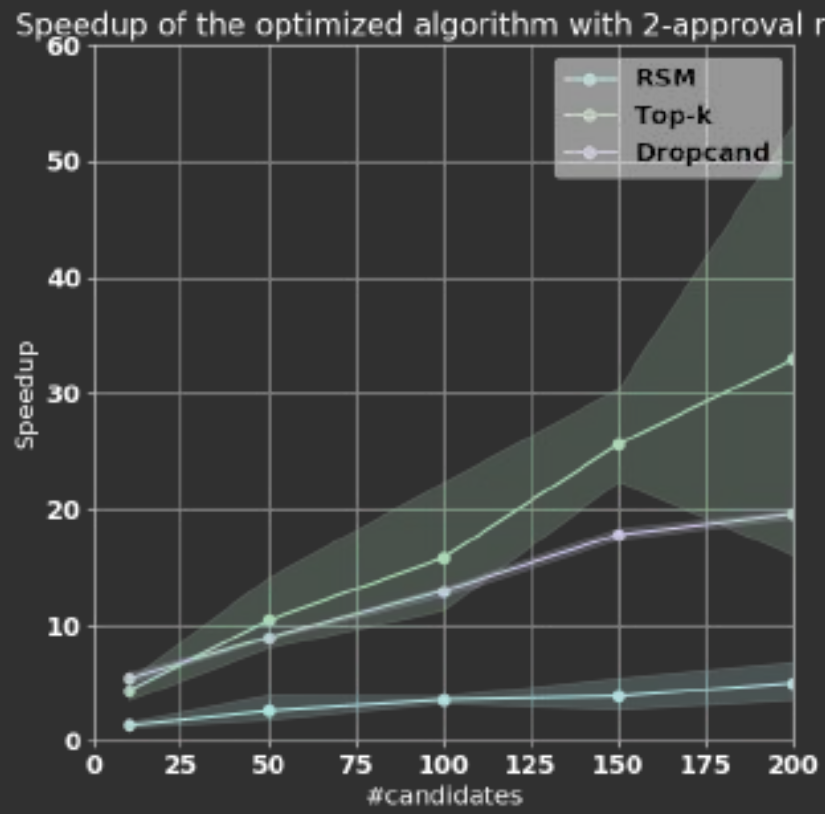
Running time (Borda)





Speedup


Speedup of the optimized algorithm with Borda rule
RSM
Top-k
Dropcand
Speedup
#candidates
0
10
20
30
40
50
60
0
25
50
75
100
125
150
175
200


Speedup of the optimized algorithm with 2-approval rule
RSM
Top-k
Dropcand
Speedup
#candidates
0
10
20
30
40
50
60
0
25
50
75
100
125
150
175
200

Summary

Dichotomy of the PW problem

Towards a dichotomy for the possible winner problem in elections based on scoring rules (2010) by *Betzler and Dorn*

Plurality and **Veto** :
PTIME

⇒ I implemented and optimized it.

Dataset	Time (s)
RSM	1.06
Top-k	0.44
drop-cand	31.27

TABLE – Running time of the PW algorithm for plurality with $m = 200$ and $n = 10,000$.

Borda and **k-approval** :
NP-complete

⇒ 3-steps algorithm :

1. Use the NW algorithm to eliminate candidates.
2. Try to build completions to find possible winners.
3. Use an ILP solver to find possible winners.

Step 1 : Pruning

- There is some obvious winners (for instance, the candidate with the highest maximal score) and obvious losers.
- We can also use the competitions from the Necessary Winner algorithm : *If*  *always has a better score than*  *, then*  *is not a possible winner.*

Dataset	*Drop cand.*		*Top-k*		*RSM*	
Rule	*Borda*	*2-app*	*Borda*	*2-app*	*Borda*	*2-app*
Step 1	18.3%	22.9%	71.6%	96.4%	59.4%	63.4%

TABLE – Proportion of candidates for which we can conclude after this step.

Step 2 : Build a completion

To prove that a candidate is a possible winner, the most intuitive way is to show one completion of the partial profile voting in which this candidate is a winner. I used a lot of technical heuristics **to increase the success rate** of this step.

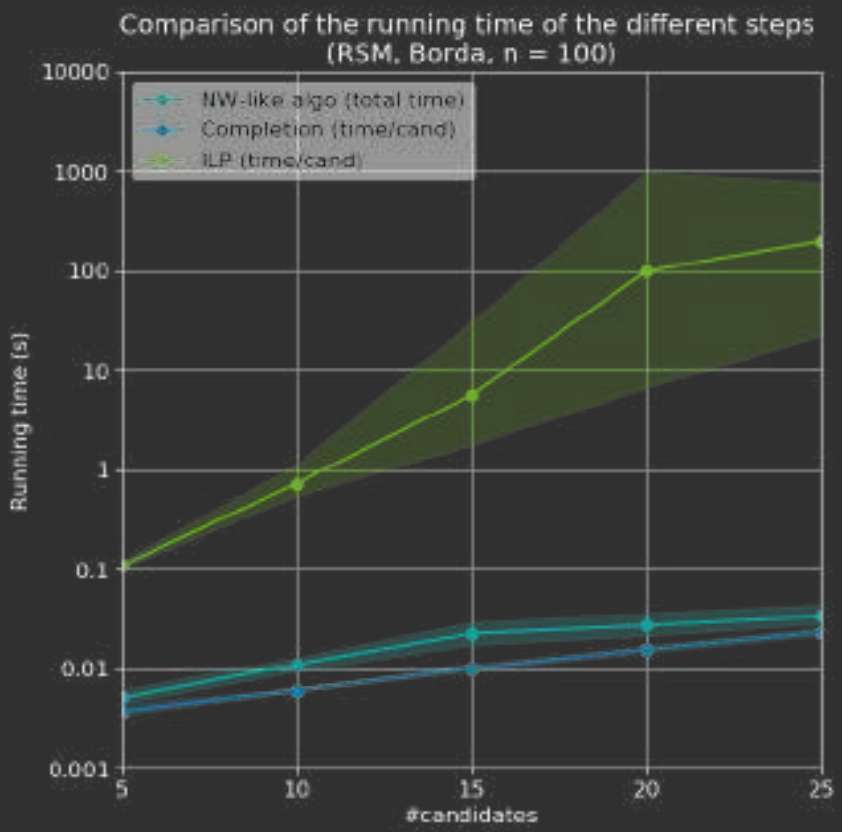
Dataset	*Drop cand.*		*Top-k*		*RSM*	
Rule	*Borda*	*2-app*	*Borda*	*2-app*	*Borda*	*2-app*
Step 2	100%	99.9%	99.9%	100%	97%	90.4%
Step 1+2	100%	99.9%	99.9%	100%	98.8%	96.4%

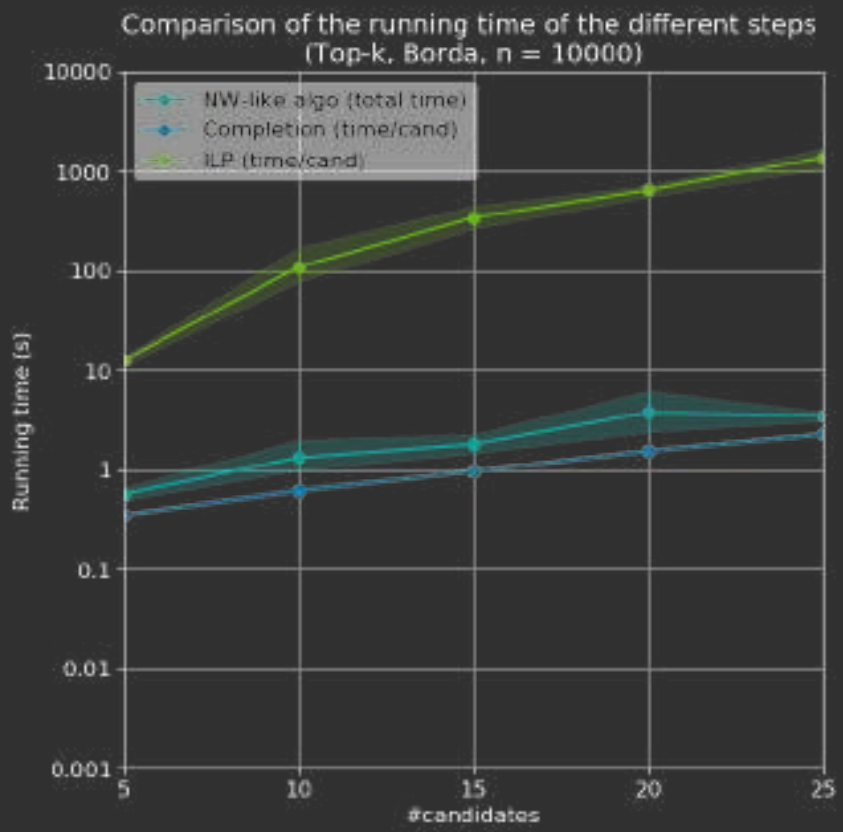
TABLE – Proportion of candidates for which we can conclude after this step.

Step 3 : ILP solver

ILP = *Integer linear programming*

Running time of each step





$\Rightarrow$ For instance, for *RSM* dataset and *Borda* rule, with $m = 25$ and $n = 100$, the version with step 2 is in average 1961 times faster

Summary

Introduction : Queries

The real goal of the *DBCOMSOC* project is to answer queries on election databases, such as :

- Query 1 : *Does a winner contains chocolate ?*
- Query 2 : *Is there a winner who contains chocolate and another winner who does not contain chocolate ?*
- Query 3 : *Is there a winner who contains chocolate and another winner who does not contain chocolate, and the two winners are sold in the same bakery of New York ?*

Introduction : Necessary and Possible answers

Query : *Does a winner contains chocolate ?*

The Possible Answer problem

Is it possible that a winner contains chocolate ?

	$q \in$	`possib`(q)
Plurality and Veto	$\mathcal{C}_D$	P
	$\mathcal{C}_C$	P
Other rules		NP-comp.

The Necessary Answer problem

Is it necessary that a winner contains chocolate ?

	$q \in$	`necess`(q)
Plurality and Veto	$\mathcal{C}_D$	P
	$\mathcal{C}_C$	coNP-comp.
Other rules		coNP-comp.

Possible answers

Is it possible that a winner contains chocolate ?

1. With a SQL query, get the list $\mathcal{L}$ of the desserts which contain chocolate.
2. Use the **Possible Winner algorithm** of the wanted voting rule on candidates on $\mathcal{L}$. If one is a possible winner, then return `True`. Otherwise, return `False`.

$\Rightarrow$ This can be generalized to more complex queries, by testing if sets of candidate are **possible winner sets**.

Necessary answers

Is it necessary that a winner contains chocolate ?

1. With a SQL query, get the list $\mathcal{L}$ of the desserts which contain chocolate.
2. Use the **Possible Winner algorithm** of the wanted voting rule on candidates not on $\mathcal{L}$. If one is a possible winner without a candidate from $\mathcal{L}$ as co-winner, then return `False`. Otherwise, return `True`.

$\Rightarrow$ This can be easily generalized to more complex queries that can be divided into simple subqueries : *Is there a winner who contains chocolate and another winner who does not contain chocolate ?*

For other queries, I implemented an a-priori algorithm to minimize the use of the ILP solver.

Results

	NECESSITY(q)			POSSIBILITY(q)		
	Plurality	*Borda*	*3-Appr.*	*Plurality*	*Borda*	*3-Appr.*
Query 1	$0.03s$	$1.4s$	$1.5s$	$0.03s$	$1.4s$	$1.5s$
Query 2	$0.03s$	$1.4s$	$1.5s$	$0.05s$	$3.5s$	$3.5s$
Query 3	$0.03s$	$4.3s$	$4.3s$	$0.07s$	$97.9s$	$7.4s$

TABLE – Running time of NECESSITY(q) and POSSIBILITY(q) for 3 different queries and different voting rules. $n = 10,000$ and $m = 8$

Conclusion

- I proposed an experiment to collect a real dataset of partial order.
- I proposed a general method to generate synthetic partial orders.
- We implemented and optimized the Necessary Winner algorithm.
- I implemented and optimized the Possible Winner algorithms for *Plurality* and *Veto*.
- I proposed a way to "approximate" the set of possible winners for other positional scoring rule in much less time than the ILP solver.
- I implemented a first framework to solve Possible and Necessary Answers problems.
- We are going to submit a part of this work to AAAI-2020.

Acknowledgments

Pr. Julia Stoyanovich

Supervisor

Kunal Relia

PhD student

References

Betzler, N. & Dorn, B. Towards a dichotomy for the Possible Winner problem in elections based on scoring rules. *Journal of Computer and System Sciences* **76,** 812–836. ISSN : 0022-0000 (2010).

Brandt, F., Conitzer, V., Endriss, U., Lang, J. & Procaccia, A. D. *Handbook of Computational Social Choice.* 1st. ISBN : 1107060435, 9781107060432 (Cambridge University Press, New York, NY, USA, 2016).

Kimelfeld, B., Kolaitis, P. G. & Stoyanovich, J. *Computational Social Choice Meets Databases.* in *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (AAAI Press, Stockholm, Sweden, 2018), 317–323. ISBN : 978-0-9992411-2-7.

Xia, L. & Conitzer, V. *Determining Possible and Necessary Winners under Common Voting Rules Given Partial Orders.* in. **41** (jan. 2008), 196–201.

Thanks for your attention !

TheoDlmz/DBCOMSOC