University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Kacper Harasimowicz
Student no. 429221

Cumulative Voting in Participatory Budgeting: Analysis and Evaluation of Selection Methods

Master's thesis
in COMPUTER SCIENCE

Supervisor:
dr hab. Piotr Skowron
MIM UW

Warsaw, August 2025

Abstract

Participatory budgeting is a process in which voters select a subset of proposed projects whose total cost fits within a fixed budget. Compared to the commonly used approval voting method, cumulative voting allows for a more nuanced expression of voter preferences. This thesis analyzes methods for selecting projects based on cumulative votes, including simple greedy algorithms and various variants of the Cumulative Single Transferable Vote (CSTV). The evaluation uses multiple metrics to assess the effectiveness and fairness of these methods on real-world data. As part of the project, I fix and properly test the implementation of those rules in the *pabutools* library and implement the methods used for analysis.

Keywords

Participatory budgeting, Cumulative voting, Greedy utilitarian method, Cumulative Single Transferable Vote (CSTV), Fairness metrics, Computational social choice, Voting system evaluation, pabutools library

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatyka

Subject classification

- Theory of computation
  - ~ Theory and algorithms for application domains
    - ~ Algorithmic game theory and mechanism design
      - ~ Algorithmic game theory
- Applied computing
  - ~ Computers in other domains
    - ~ Computing in government
      - ~ Voting / election technologies
  - ~ Operations research
    - ~ Decision analysis

Tytuł pracy w języku polskim

Głosowanie kumulatywne w budżetach partycypacyjnych: analiza i ocena metod wyboru projektów

Contents

Introduction . . . 5

1. Preliminaries . . . 7
1.1. Model . . . 7
1.2. Greedy rules . . . 7

2. Cumulative Single Transferable Vote (CSTV) rules . . . 9
2.0.1. Project Selection . . . 10
2.0.2. Excess Redistribution . . . 12
2.0.3. non-eligible projects resolution . . . 15
2.0.4. inclusive maximality post-procedure . . . 18

3. Analysis methods . . . 23
3.1. Utility score . . . 23
3.2. Exclusion ratio . . . 23
3.3. Power inequality . . . 24
3.4. Improvement margin . . . 24
3.5. Extended Justified Representation+ . . . 25

4. Analysis results . . . 27

5. Summary . . . 31

Bibliography . . . 33

Introduction

Participatory budgeting is a method for selecting a set of projects whose total cost does not exceed a given budget, based on voters' preferences. Among the various methods used in participatory budgeting, cumulative voting is relatively rare. This method allows voters to split their contribution between candidates however they want. However, to determine the effectiveness of cumulative voting, it is essential to analyze the performance of the election rules that process its results, especially using real-world election data.

One such rule, Cumulative Single Transferable Vote (CSTV), was proposed as an alternative to commonly used greedy selection rules [1]. This thesis evaluates the performance of CSTV rules using data from real participatory budgeting elections, comparing them to established Greedy rules. The analysis also examines CSTV variants based on the *greedy by support* rule, which has received little attention in prior research.

To carry out the study, the existing *pabutools* library was updated to include a fully functional CSTV implementation. Additionally, methods for calculating evaluation metrics were implemented, inspired by those used in evaluation of other methods [2]. Together, these tools enable systematic testing of CSTV performance on cumulative elections drawn from the *pabulib* dataset.

The remainder of this thesis is structured as follows: Chapter 1 introduces key definitions and concepts related to cumulative elections. Chapter 2 presents a detailed description of CSTV rules and their variants. Chapter 3 explains the methodology and the rationale behind the selected metrics. Finally, Chapter 4 presents and discusses the results of the analysis, followed by concluding remarks and suggestions for future research.

Chapter 1

Preliminaries

1.1. Model

Cumulative election (further referred to as *election*) is defined as a tuple $E = (P, V, \text{cost}, b)$ where

$P = \{p_0, p_1, ..., p_m\}$ is a set of *projects* (also referred to as *candidates*) with the associated function $\text{cost}(p_j)$ determining the cost of picking that candidate.

$V = \{v_1, v_2, ..., v_n\}$ is a set of *voters* (also referred to as *ballots*), with each voter being a contribution (also referred to as an allocation) function $v_i(p_j)$ determining how big a part of the budget each voter wishes to contribute to each project. In calculations, it is assumed that each voter splits 1 worth of contribution, that is $\sum_{p_j \in P} v_i(p_j) \leq 1$ with strict inequality if said voter wishes not to contribute his entire share of the budget. In real-world data, contributions are often scaled, so that $\sum_{p_j \in P} v_i(p_j) \leq c$ for some constant $c > 0$. Any algorithm described in this thesis can still be applied by first scaling all contributions by $1/c$.

b is a total budget that can be spent on projects.

This notation naturally extends to subsets $P' \subseteq P$, where $\text{cost}(P') = \sum_{p_j \in P'} \text{cost}(p_j)$ and $v_i(P') = \sum_{p_j \in P'} v_i(p_j)$.

An *aggregation method*, (also referred to as a *rule*) is a function that, based on election E, gives a subset of available projects $B \subseteq P$ that fit into the budget $\text{cost}(B) \leq b$.

1.2. Greedy rules

Greedy rules are simple aggregation methods that repeatedly select the most valuable remaining project that still fits within the budget.

Those rules start with an empty set of chosen projects $B = \emptyset$ and among the remaining projects that still fit into the budget $P' = \{p_j \in P \setminus B \mid \text{cost}(B \cup \{p_j\}) \leq b\}$ choose the one that maximizes value based on the type of the Greedy rule, possibly scaled by single voter's share of the budget $\frac{b}{n}$:

GS: Greedy by support: $f_{\text{GS}}(E, p_j) = \sum_{v_i \in V} v_i(p_j)$

GE: Greedy by excess: $f_{\text{GE}}(E, p_j) = \left(\sum_{v_i \in V} v_i(p_j)\frac{b}{n}\right) - \text{cost}(p_j)$

GSC: Greedy by support over cost: $f_{\text{GSC}}(E, p_j) = \left(\sum_{v_i \in V} v_i(p_j)\frac{b}{n}\right) / \text{cost}(p_j)$

Then adds the chosen project to the result set B if its cost fits into the remaining budget ($\text{cost}(p_j) \leq b$) or removes that project from P if it doesn't fit ($\text{cost}(p_j) > b$).

This process is then repeated until all projects are checked.

Algorithm 1: Greedy rule

Data: $E = (P, V, \text{cost}, b), f \in \{f_{\text{GS}}, f_{\text{GE}}, f_{\text{GSC}}\}$
Result: $B \subseteq P$
$B \leftarrow \varnothing$;
while $P \neq \varnothing$ **do**
  $p \leftarrow p_0$;
  $x \leftarrow -\infty$;
  for $p_j \in P$ **do**
    if $x < f(E, p_j)$ **then**
      $x \leftarrow f(E, p_j)$;
      $p \leftarrow p_j$;
    end
  end
  if $\text{cost}(p) \leq b$ **then**
    $b \leftarrow b - \text{cost}(p)$;
    $B \leftarrow B \cup \{p\}$;
  end
  $P \leftarrow P \setminus \{p\}$;
end
return B

Example 1 *Let's consider an election* $E = (\{A, B, C\}, \{v_1\}, cost, 100)$ *with functions as described in the table:*

project name	*cost*	v_1	f_{GS}	f_{GE}	f_{GSC}
A	*50*	*0.2*	*0.2*	*-30*	*0.4*
B	*55*	*0.3*	*0.3*	*-25*	$0.\overline{54}$
C	*100*	*0.5*	*0.5*	*-50*	*0.5*

In this situation, Greedy rules would choose the biggest number from their corresponding column, so greedy by support would choose project C, greedy by excess would choose project B, and greedy by support over cost would choose project A. After the highest-value project is chosen, the rule examines the next-highest option, but discards it since it no longer fits in the remaining budget. Similarly, no third project would be chosen.

Chapter 2

Cumulative Single Transferable Vote (CSTV) rules

CSTV rules are based on a simple idea: each voter has their own budget share and distributes it among available projects. This creates projects that are eligible for funding:

$$\text{eligible}(E) = \left\{ p_j \in P \mid \left(\sum_{v_i \in V} v_i(p_j) \frac{b}{n} \right) \geq \text{cost}(p_j) \right\}$$

Those projects can then be picked for the solution.

However, stopping at this point has drawbacks. A lot of support may be wasted either as excess from over-supporting popular projects or unused by allocating it to non-eligible projects. CSTV rules address this issue by transferring voters' allocations that cannot be used.

Thus, CSTV rule algorithm iteratively updates the election E' and the selected set of candidates B' from the previous state using 4 sub-procedures:

To better understand CSTV, explanations of the 4 sub-procedures include their effect on two examples with $v_i(p_j)\frac{b}{n}$ values already calculated:

No eligible projects example: $E_1 = (\{A, B, C, D\}, \{v_1, v_2\}, \text{cost}, 44)$

project name	cost	$\frac{b}{n}v_1$	$\frac{b}{n}v_2$
A	20	15	1
B	26	7	7
C	30	0	10
D	30	0	4

Greedy rule example: $E_2 = (\{A, B, C, D, E, F\}, \{w_1, w_2, w_3\}, \text{cost}, 7551)$

project name	cost	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
A	2200	2000	10	500
B	1720	10	2000	10
C	2800	500	500	2000
D	400	5	1	1
E	400	1	5	1
F	400	1	1	5

Algorithm 2: Cumulative Single Transferable Vote

Data: $E = (P, V, \text{cost}, b)$, project_selection, excess_redistribution, non_eligible_projects_resolution, inclusive_maximality_post_procedure
Result: $B \subseteq P$
$B \leftarrow \varnothing$;
$R \leftarrow \varnothing$ ▷ set of eliminated projects
while $P \neq \varnothing$ **do**
  $p \leftarrow$ project_selection(E);
  ▷ Check if any p was selected
  if $p \in P$ **then**
    $V \leftarrow$ excess_redistribution(E, p);
    $B \leftarrow B \cup \{p\}$;
    $P \leftarrow P \setminus \{p\}$;
    $b \leftarrow b - \text{cost}(p)$;
  else
    $(E, R, \text{do\_continue}) \leftarrow$ non_eligible_projects_resolution(E, R);
    if not do_continue **then**
      $B \leftarrow$ inclusive_maximality_post_procedure(E, R, B);
      return B
    end
  end
end

2.0.1. Project Selection

First, the single best remaining project $p_j \in \text{eligible}(E)$ is chosen, based on the current round's voters' allocation. The standard approach uses a single step of the Greedy rule $\{f_{GS}, f_{GE}, f_{GSC}\}$ as described in section **1.2** among all eligible projects. The next repeat is run with updated $P' = P \setminus \{p_j\}$ and $b' = b - \text{cost}(p_j)$.

The *eligible* sub-procedure can be implemented as follows:

Algorithm 3: eligible

Data: $E = (P, V, \text{cost}, b)$
Result: $P' \subseteq P$
$P' \leftarrow \varnothing$;
$n \leftarrow |V|$;
for $p_j \in P$ **do**
  if $\text{cost}(p_j) \leq \sum_{v_i \in V} v_i(p_j) \frac{b}{n}$ **then**
    $P' \leftarrow P' \cup \{p_j\}$;
  end
end
return P'

And project selection sub-procedure would look like this:

Example 2 *Let's consider how project selection will behave on the no eligible projects example:* $E_1 = (\{A, B, C, D\}, \{v_1, v_2\}, cost, 44)$

Algorithm 4: Project Selection

Data: $f \in \{f_{\text{GS}}, f_{\text{GE}}, f_{\text{GSC}}\}, E = (P, V, \text{cost}, b)$
Result: $p \in P$
$p \leftarrow$ `None`;
▷ None means no eligible project was found, we assume that None $\notin P$;
$x \leftarrow -\infty$;
for $p_j \in \text{eligible}(E)$ **do**
  if $x < f(E, p_j)$ **then**
    $x \leftarrow f(E, p_j)$;
    $p \leftarrow p_j$;
  end
end
return p

project name	*cost*	$\frac{b}{n}v_1$	$\frac{b}{n}v_2$
A	*20*	*15*	*1*
B	*26*	*7*	*7*
C	*30*	*0*	*10*
D	*30*	*0*	*4*

In this example, all sums of $\sum_{v_i} v_i(p_j)\frac{b}{n}$ *are smaller than project costs, so the eligible function would return an empty set, meaning that no project can be selected for funding, and the excess redistribution procedure will be triggered.*

Example 3 *Now let's also consider how project selection will behave on the Greedy rule example:* $E_2 = (\{A, B, C, D, E, F\}, \{w_1, w_2, w_3\}, \textit{cost}, 7551)$

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
A	*2200*	*2000*	*10*	*500*
B	*1720*	*10*	*2000*	*10*
C	*2800*	*500*	*500*	*2000*
D	*400*	*5*	*1*	*1*
E	*400*	*1*	*5*	*1*
F	*400*	*1*	*1*	*5*

The eligible function will return the set ***{A, B, C}****, then one of them will be selected based on the Greedy rule* $f \in \{f_{GE}, f_{GSC}, f_{GS}\}$.

project name	f_{GE}	f_{GSC}	$\frac{b}{n}f_{GS}$
A	*310*	≈ 1.14	*2510*
B	*300*	≈ 1.18	*2020*
C	*200*	≈ 1.07	*3000*

Since the project with the highest Greedy rule value is chosen, then f_{GE} *picks project* ***A****,* f_{GSC} *picks project* ***B*** *and* f_{GS} *picks project* ***C****.*

In future loops, after excess redistribution, project selection will pick 3 more projects before no eligible remain, depending on the rule: f_{GE} *will pick* ***C, B, E****,* f_{GSC} *will pick* ***C, A, D****, and* f_{GS} *will pick* ***A, B, E****. The allocation functions leading to these sequences will be detailed in the next subsection.*

2.0.2. Excess Redistribution

If project p_j was chosen to be added to B, it's necessary to redistribute excess support allocated to it in order to avoid punishing voters for supporting popular projects. Proportional redistribution is done only in subset $\text{trans}(p_j)\{v_i \in V | v_i(P) > v_i(p_j)\}$ where voters have other candidates they support. Formally, such $\gamma \leq 1$ needs to be chosen that

$$\sum_{v_i \in tran(p_j)} \gamma v_i(p_j)\frac{b}{n} + \sum_{v_i \notin tran(p_j)} v_i(p_j)\frac{b}{n} = \text{cost}(p_j)$$

and then redistribute $(1-\gamma)$ of each voter's contribution to p_j proportionally among other projects, creating a new allocation v_i':

$$\forall_{v_i \in V} \forall_{p_k \in P \setminus \{p_j\}} v_i'(p_k) = v_i(p_k) + \frac{v_i(p_k)}{v_i(P \setminus \{p_j\})}(1-\gamma)v_i(p_j)$$

In rare cases, it can happen that $\sum_{v_i \notin tran(p)} v_i(p_j)\frac{b}{n} > \text{cost}(p_j)$. If that happens, γ should be set to 0, and the excess from $v_i \notin tran(p_j)$ will be lost until the *inclusive maximality post-procedure*, described in 2.0.4, is considered.

Implementation of this sub-procedure would look like this:

Example 4 *Since in the 'no eligible projects' example the project selection sub-procedure couldn't pick a candidate to be funded, this sub-procedure is not called.*

Example 5 *Now let's follow the execution of excess redistribution on the Greedy rule example:* $E_2 = (\{A, B, C, D, E, F\}, \{w_1, w_2, w_3\}, cost, 7551)$

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
A	*2200*	*2000*	*10*	*500*
B	*1720*	*10*	*2000*	*10*
C	*2800*	*500*	*500*	*2000*
D	*400*	*5*	*1*	*1*
E	*400*	*1*	*5*	*1*
F	*400*	*1*	*1*	*5*

Since the selected project depends on the Greedy rule used, we get 3 different results of redistribution depending on the rule:

*$f = f_{GE}$ with **A** selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
B	*1720*	≈ 14.78	≈ 2001	≈ 10.3
C	*2800*	≈ 739	≈ 500.25	≈ 2061.23
D	*400*	≈ 7.39	≈ 1	≈ 1.03
E	*400*	≈ 1.48	≈ 5	≈ 1.03
F	*400*	≈ 1.48	≈ 1	≈ 5.15

*$f = f_{GSC}$ with **B** selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
A	*2200*	≈ 2001.18	≈ 15.75	≈ 500.3
C	*2800*	≈ 500.3	≈ 787.26	≈ 2001.18
D	*400*	≈ 5	≈ 1.57	≈ 1
E	*400*	≈ 1	≈ 7.87	≈ 1
F	*400*	≈ 1	≈ 1.57	≈ 5

Algorithm 5: Excess Redistribution

Data: $E = (P, V, \text{cost}, b), p \in P$
Result: V'
$V' \leftarrow V$;
$n \leftarrow |V|$;
$d \leftarrow 0$;
$\gamma \leftarrow \text{cost}(p) / \left(\sum_{v_i \in V} v_i(p) \frac{b}{n}\right)$;
repeat_loop $\leftarrow$ True;
while repeat_loop **do**
    repeat_loop $\leftarrow$ False;
    for $v'_i \in V'$ **do**
        $s \leftarrow \sum_{p_j \in P \setminus \{p\}} v'_i(p_j) \frac{b}{n}$;
        if $s = 0$ **then**
            repeat_loop $\leftarrow$ True;
            $V' \leftarrow V' \setminus \{v'_i\}$;
            $d \leftarrow d + v'_i(p)$;
            $\gamma \leftarrow \max\left(0, (\text{cost}(p) - d) / \left(\sum_{v'_i \in V'} v'_i(p) \frac{b}{n}\right)\right)$;
        end
    end
end
for $v'_i \in V'$ **do**
    $s \leftarrow \sum_{p_j \in P \setminus \{p\}} v'_i(p_j) \frac{b}{n}$;
    if $s > 0$ **then**
        $e \leftarrow v'_i(p) - \gamma v'_i(p)$;
        $v'_i(p) \leftarrow \gamma v'_i(p)$;
        for $p_j \in P \setminus \{p\}$ **do**
            if $v'_i(p_j) > 0$ **then**
                $v'_i(p_j) \leftarrow v'_i(p_j) + e v'_i(p_j)/s$;
            end
        end
    end
end
return V'

$f = f_{GS}$ *with* $\boldsymbol{C}$ *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
$\boldsymbol{A}$	*2200*	≈ 2033.05	≈ 10.17	≈ 628.95
$\boldsymbol{B}$	*1720*	≈ 10.17	≈ 2033.05	≈ 12.58
$\boldsymbol{D}$	*400*	≈ 5.08	≈ 1.02	≈ 1.26
$\boldsymbol{E}$	*400*	≈ 1.02	≈ 5.08	≈ 1.26
$\boldsymbol{F}$	*400*	≈ 1.02	≈ 1.02	≈ 6.29

After the second loop step, the second selected project and support result after redistribution will look like this:

$f = f_{GE}$ *after* $\boldsymbol{A}$, $\boldsymbol{C}$ *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
$\boldsymbol{B}$	*1720*	≈ 80.67	≈ 2076.58	≈ 194.12
$\boldsymbol{D}$	*400*	≈ 40.34	≈ 1.4	≈ 19.42
$\boldsymbol{E}$	*400*	≈ 8.08	≈ 5.19	≈ 19.42
$\boldsymbol{F}$	*400*	≈ 8.08	≈ 1.04	≈ 97.06

$f = f_{GSC}$ *with* $\boldsymbol{B}$, $\boldsymbol{C}$ *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
$\boldsymbol{A}$	*2200*	≈ 2075.27	≈ 84.61	≈ 793.59
$\boldsymbol{D}$	*400*	≈ 5.19	≈ 8.43	≈ 1.59
$\boldsymbol{E}$	*400*	≈ 1.04	≈ 42.28	≈ 1.59
$\boldsymbol{F}$	*400*	≈ 1.04	≈ 8.43	≈ 7.93

$f = f_{GS}$ *with* $\boldsymbol{C}$, $\boldsymbol{A}$ *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
$\boldsymbol{B}$	*1720*	≈ 218.16	≈ 2034.84	≈ 77.94
$\boldsymbol{D}$	*400*	≈ 108.97	≈ 1.02	≈ 7.8
$\boldsymbol{E}$	*400*	≈ 21.88	≈ 5.08	≈ 7.8
$\boldsymbol{F}$	*400*	≈ 21.88	≈ 1.02	≈ 38.97

Next, after 3 loop steps, projects $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ *will be selected regardless of the* f *chosen, but redistributed support will look completely different:*

$f = f_{GE}$ *after* $\boldsymbol{A}$, $\boldsymbol{C}$, $\boldsymbol{B}$ *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
$\boldsymbol{D}$	*400*	≈ 55.81	≈ 80.8	≈ 26.87
$\boldsymbol{E}$	*400*	≈ 11.18	≈ 403.25	≈ 26.87
$\boldsymbol{F}$	*400*	≈ 11.18	≈ 80.8	≈ 134.29

$f = f_{GSC}$ *with* $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{A}$ *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
$\boldsymbol{D}$	*400*	≈ 383.13	≈ 11.51	≈ 30.56
$\boldsymbol{E}$	*400*	≈ 76.77	≈ 51.71	≈ 30.56
$\boldsymbol{F}$	*400*	≈ 76.77	≈ 11.51	≈ 152.43

$f = f_{GS}$ *with* $\boldsymbol{C}$, $\boldsymbol{A}$, $\boldsymbol{B}$ *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
D	*400*	≈ 150	≈ 77.84	≈ 10.74
E	*400*	≈ 30.12	≈ 387.69	≈ 10.74
F	*400*	≈ 30.12	≈ 77.84	≈ 53.64

Lastly, after 4 loop steps, projects will select their last eligible project $f = f_{GE}$ *after* ***A, C, B, E*** *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
D	*400*	≈ 56.69	≈ 99.75	≈ 27.29
F	*400*	≈ 11.36	≈ 99.75	≈ 136.39

$f = f_{GSC}$ *with* ***B, C, A, D*** *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
E	*400*	≈ 88.21	≈ 52.27	≈ 30.86
F	*400*	≈ 88.21	≈ 11.64	≈ 153.95

$f = f_{GS}$ *with* ***C, A, B, E*** *selected:*

project name	*cost*	$\frac{b}{n}w_1$	$\frac{b}{n}w_2$	$\frac{b}{n}w_3$
D	*400*	≈ 150.68	≈ 90.85	≈ 10.86
F	*400*	≈ 30.46	≈ 90.85	≈ 54.24

2.0.3. non-eligible projects resolution

If there are no eligible projects, then vote transfer needs to happen in order to make one of the non-eligible projects eligible. There are two standard methods that can be performed to achieve this.

EWT: Elimination with Transfers

One method is to remove the worst candidate according to the same Greedy rule as used in project selection: f_{GS}, f_{GE} or f_{GSC}. Elimination then is performed the same way as excess redistribution with $\text{cost}(p_j) = 0$. This can be implemented this way:

Example 6 *The 'no eligible projects' example will start with this sub-procedure, let's follow its execution:* $E_1 = (\{A, B, C, D\}, \{v_1, v_2\}, cost, 44)$

project name	*cost*	$\frac{b}{n}v_1$	$\frac{b}{n}v_2$
A	*20*	*15*	*1*
B	*26*	*7*	*7*
C	*30*	*0*	*10*
D	*30*	*0*	*4*

The worst project according to f *will now get removed. Regardless of the Greedy rule used, project* ***D*** *is the worst by a big margin, so it is removed first and its support is redistributed to other projects, resulting in:*

project name	*cost*	$\frac{b}{n}v_1$	$\frac{b}{n}v_2$
A	*20*	*15*	≈ 1.22
B	*26*	*10*	≈ 8.56
C	*30*	*0*	≈ 12.2

Algorithm 6: Elimination with Transfers

Data: $f \in \{f_{\mathrm{GS}}, f_{\mathrm{GE}}, f_{\mathrm{GSC}}\}, E = (P, V, \mathrm{cost}, b), R$
Result: $p \in P$
$p \leftarrow$ `None`;
▷ `None` means no worst project was found, we assume that `None` $\notin P$;
$x \leftarrow \infty$;
for $p_j \in P$ **do**
  if $x > f(E, p_j)$ **then**
    $x \leftarrow f(E, p_j)$;
    $p \leftarrow p_j$;
  end
end
if p is None **then**
  return $(E' = (P', V', \mathrm{cost}, b), R,$ False)
end
$V' \leftarrow V$;
$n \leftarrow |V|$;
for $v_i \in V$ **do**
  $s \leftarrow \sum_{p_j \in P \setminus \{p\}} v_i'(p_j)\frac{b}{n}$;
  if $s > 0$ **then**
    $e \leftarrow v_i(p)$;
    $v_i'(p) \leftarrow 0$;
    for $p_j \in P \setminus \{p\}$ **do**
      if $v_i(p_j) > 0$ **then**
        $v_i'(p_j) \leftarrow v_i(p_j) + ev_i(p_j)/s$
      end
    end
  end
end
return $(E' = (P \setminus \{p\}, V', \mathrm{cost}, b), R \cup \{p\},$ True)

There are still no eligible projects, so the next worst, ***C****, is removed:*

project name	*cost*	$\frac{b}{n}v_1$	$\frac{b}{n}v_2$
A	*20*	*15*	≈ 2.74
B	*26*	*10*	≈ 19.24

Project ***B*** *is now eligible, so it is picked and goes through normal excess redistribution. There is not enough budget to pick* ***A****, so the program finishes with reverse elimination.*

Example 7 *In the 'Greedy rule' example, elimination with transfers receives E_2 in different states depending on the Greedy rule f used, but in all cases, the project that gives the worst result with f is eliminated, and the remaining project becomes eligible.*

This results in selections: for f_{GE}: ***A, C, B, E, F****, for f_{GSC}:* ***B, C, A, D, F****, and for f_{GS}:* ***C, A, B, E, D****.*

MT: Minimal Transfers

Second method is to pick an underfunded project and transfer just enough support to cover its cost $\text{cost}(p_j)$.

Project p_j is eligible for selection with transfers if it can be funded in case that all voters voting positively for it transferred all their votes to p_j, which can be formally described as:

$$\text{eligible}_t(E) = \left\{ p_j \in P \mid \sum_{v_i(p_j)>0} v_i(P)\frac{b}{n} \geq \text{cost}(p_j) \right\}$$

Among all projects eligible with transfers, we can then perform modified project selection and excess redistribution that proportionally transfers the deficit to the chosen project.

Modified selection uses Greedy rules only over a subset of supporters that has positive support towards chosen projects, described as maximizing $f\left((P, \{v_i \in V | v_i(p_j) > 0\}), p_j\right)$.

Modified redistribution, additionally, would use $\gamma > 1$ that satisfies

$$\sum_{v_i \notin \text{trans}(p_j)} v_i(p_j)\frac{b}{n} + \sum_{v_i \in \text{trans}(p_j)} min(\gamma v_i(p_j), v_i(P))\frac{b}{n} = \text{cost}(p_j)$$

And consume allocation proportionally:

$$\forall_{\gamma v_i(p_j) \geq v_i(P)} \forall_{p_k \in P - \{p_j\}} v_i'(p_k) = 0$$

$$\forall_{\gamma v_i(p_j) < v_i(P)} \forall_{p_k \in P - \{p_j\}} v_i'(p_k) = v_i(p_k) - \frac{v_i(p_k)}{v_i(P - \{p_j\})}(\gamma - 1)v_i(p_j)$$

This can be implemented as:

Example 8 *In this version 'no eligible projects' example also starts skips project selection and excess redistribution: $E_1 = (\{A, B, C, D\}, \{v_1, v_2\}, cost, 44)$*

project name	*cost*	$\frac{b}{n}v_1$	$\frac{b}{n}v_2$
A	*20*	*15*	*1*
B	*26*	*7*	*7*
C	*30*	*0*	*10*
D	*30*	*0*	*4*

Algorithm 7: eligible$_t$

```
Data: E = (P, V, cost, b)
Result: P' ⊆ P
P' ← ∅;
n ← |V|;
for p_j ∈ P do
    s ← 0 for v_i ∈ V do
        if v_i(p_j) > 0 then
            s ← s + Σ_{p_k ∈ P} v_i(p_k)
        end
    end
    if cost(p) ≤ s then
        P' ← P' ∪ {p_j};
    end
end
return P'
```

*Both projects **A** and **B** can get the full budget of* 50 *transferred to them, so both are eligible with transfers. **C** and **D** can only get transfers from* v_2*, so they can only get* 25 *with transfers, which is smaller than their cost, so neither is eligible with transfers.*

*Next project **A** is better than **B** according to all Greedy rules* f*, so it is added to the solution. Remaining support functions look like this after transfers:*

$E_1 = (\{A, B, C, D\}, \{v_1, v_2\}, cost, 24)$

project name	*cost*	$\frac{b}{n}v_1$	$\frac{b}{n}v_2$
B	*26*	*3.25*	≈ 6.92
C	*30*	*0*	≈ 9.88
D	*30*	*0*	≈ 3.95

Now the entire remaining budget: 24 *is not enough to cover the cost of any project, so no projects are eligible with transfers, so minimal transfers end with the result: {**A**}.*

Example 9 *Minimal transfers receive 'Greedy rule' example in different states depending on the Greedy rule* f *used, but in all cases both projects are eligible with transfers and the project that gives a better result with* f *is selected.*

This results in selections: for f_{GE}*: **A, C, B, E, F**, for* f_{GSC}*: **B, C, A, D, F**, and for* f_{GS}*: **C, A, B, E, D**.*

2.0.4. inclusive maximality post-procedure

Both procedures for excess redistribution may be unable to find new projects to add to result B despite there still being enough budget to fund more projects. Based on the procedure used in the previous step, we pick one of the methods.

Reverse Elimination

If the **Elimination with Transfers** was used, then remaining projects can be selected by greedily checking eliminated projects starting from those eliminated last. If the cost of the checked project fits inside the remaining budget, then it is selected. It can be implemented like this:

Algorithm 8: Minimal Transfers

Data: $f \in \{f_{\mathrm{GS}}, f_{\mathrm{GE}}, f_{\mathrm{GSC}}\}, E = (P, V, \mathrm{cost}, b), R$
Result: $p \in P$
$p \leftarrow$ None;
$x \leftarrow -\infty$;
for $p_j \in \mathrm{eligible}_t(E)$ **do**
  if $x < f(E, p_j)$ **then**
    $x \leftarrow f(E, p_j)$;
    $p \leftarrow p_j$;
  end
end
if p is None **then**
  return $(E, R,$ False$)$
end
$V' \leftarrow V$;
$n \leftarrow |V|$;
$d \leftarrow 0$;
$\gamma \leftarrow \mathrm{cost}(p) / \left(\sum_{v_i \in V} v_i(p) \frac{b}{n}\right)$;
repeat_loop $\leftarrow$ True;
while repeat_loop **do**
  repeat_loop $\leftarrow$ False;
  for $v'_i \in V'$ **do**
    $s \leftarrow \sum_{p_j \in P \setminus \{p\}} v'_i(p_j) \frac{b}{n}$;
    if $s > 0$ **then**
      $e \leftarrow v'_i(p) - \gamma v'_i(p)$ ▷ usually $e < 0$
      for $p_j \in P \setminus \{p\}$ **do**
        if $v'_i(p_j) + e v'_i(p_j)/s < 0$ **then**
          repeat_loop $\leftarrow$ True;
          $v'_i(p) \leftarrow v'_i(p) + v'_i(p_j)$;
          $v'_i(p_j) \leftarrow 0$;
          $\gamma \leftarrow (\mathrm{cost}(p) - d) / \left(\sum_{v_i \in V} v_i(p) \frac{b}{n}\right)$;
        end
      end
    else
      repeat_loop $\leftarrow$ True;
      $V' \leftarrow V' \setminus \{v'_i\}$;
      $d \leftarrow d + v'_i(p)$;
      $\gamma \leftarrow (\mathrm{cost}(p) - d) / \left(\sum_{v_i \in V} v_i(p) \frac{b}{n}\right)$;
    end
  end
end

```
for v'_i ∈ V' do
    s ← Σ_{p_j ∈ P\{p}} v'_i(p_j) b/n;
    if s > 0 then
        e ← v'_i(p) − γ v'_i(p)                          ▷ usually e < 0
        v'_i(p) ← γ v'_i(p);
        for p_j ∈ P \ {p} do
            if v'_i(p_j) > 0 then
                v'_i(p_j) ← v'_i(p_j) + e v'_i(p_j)/s;
            end
        end
    end
end
return (E' = (P, V', cost, b), R, True)
```

Algorithm 9: Reverse Elimination

```
Data: E = (P, V, cost, b), R, B, f ∈ {f_GS, f_GE, f_GSC}
Result: B' ⊆ P
B' ← B;
for p_j ∈ reverse(R) do
    if cost(p_j) ≤ b then
        b ← b − cost(p_j);
        B' ← B' ∪ {p_j};
    end
end
return B'
```

Example 10 *In 'no eligible projects' reverse elimination checks, in order, projects* ***A, C, D****, but all of them have costs higher than the remaining budget, so none are added to the solution.*

Example 11 *In the 'Greedy rule' example, reverse elimination checks the single remaining candidate, but in all cases, its cost exceeds the remaining budget, so the result remains unchanged from EWT.*

Acceptance of Under Supported Projects

If the **minimal transfers** was used, then remaining projects can be selected by loosening requirements for transfers even more. Now projects are selected without filtering for eligibility, and after that all support from voters with $v_i(p_j) > 0$ is consumed ($v_i'(p_k) = 0$) and required $\text{cost}(p_j)$ is deduced from the budget.

Algorithm 10: Acceptance of Under Supported Projects

```
Data: E = (P, V, cost, b), R, B, f ∈ {f_GS, f_GE, f_GSC}
Result: B' ⊆ P
B' ← B;
b' ← b;
P' ← P;
V' ← V                                              ▷ E' = (P', V', cost, b')
while P ≠ ∅ do
    p ← None;
    x ← −∞;
    for p_j ∈ P' do
        if x < f(E', p_j) then
            x ← f(E', p_j);
            p ← p_j;
        end
    end
    if cost(p) ≤ b' then
        for v_i' ∈ V' do
            if v_i'(p) > 0 then
                for p_j ∈ P' do
                    v_i'(p_j) ← 0;
                end
            end
        end
        b' ← b' − cost(p);
        B' ← B' ∪ {p};
    end
    P' ← P' \ {p};
end
return B'
```

Example 12 *In both examples, minimal transfers finish with a remaining budget insufficient to pick any of the remaining projects, so none of the projects are added to the solution.*

Chapter 3

Analysis methods

To assess the effectiveness of the aggregation methods, it is necessary to define appropriate evaluation metrics. In this thesis, I use five such metrics, each measuring a different aspect of outcome quality.

3.1. Utility score

Utility score is the simplest metric that can measure the aggregation method's effectiveness. Each voter has an assigned utility function that describes how valuable each project is to that voter. The score is generated by summing over the utility of all voters and all selected projects. In the end, the result can be normalized by dividing it by the total number of votes.

This method has several variants, depending on how the utility function is defined. The method used is the most basic, where the utility of project p to a voter v is equal to its number of votes that project received from v: $v(p)$. At the end, I normalize the score by dividing it by the total number of votes.

Algorithm 11: Utility score

Data: $E = (P, V, \text{cost}, b), B$
Result: scaled utility of B in E
$u \leftarrow 0$ ▷ u total utility
for $v \in V$ **do**
  for $p \in B$ **do**
    $u \leftarrow u + v(p)$;
  end
end
return $u / \sum_{v \in V,\, p \in P} v(p)$

3.2. Exclusion ratio

Exclusion ratio is a ratio of voters that didn't have any of their supported projects chosen. This means that if a voter gives 0 votes for all selected projects, then that voter is excluded.

Algorithm 12: Exclusion ratio

Data: $E = (P, V, \text{cost}, b), B$
Result: exclusion ratio of B in E
$R \leftarrow \varnothing$ ▷ R is a set of excluded voters
for $v \in V$ **do**
  if $\sum_{p \in B} v(p) = 0$ **then**
    $R \leftarrow R \cup \{v\}$
  end
end
return $\frac{|R|}{|V|}$

3.3. Power inequality

Power inequality is a metric of how much inequality exists between voters in terms of their power over the budget. The metric is calculated by summing, for all voters, absolute differences between the equal split of the budget and the actual power a voter has, with the voter's power being the sum of splits of selected project's costs between all votes cast on it. At the end, the metric is normalized by dividing it by the number of voters.

Algorithm 13: Power inequality

Data: $E = (P, V, \text{cost}, b), B$
Result: power inequality of B in E
$s \leftarrow 0$ ▷ s is sum of inequality
for $v \in V$ **do**
  $s_v \leftarrow 0$ ▷ s_v is power of v
  for $p \in B$ **do**
    if $\sum_{v_j \in V} v_j(p) > 0$ **then**
      $s_v \leftarrow s_v + \frac{\text{cost}(p)\ v(p)}{\sum_{v_j \in V} v_j(p)}$;
    end
  end
  $s \leftarrow s + \left| s_v - \frac{b}{|V|} \right| \frac{|V|}{b}$;
end
return $\frac{s}{|V|}$

3.4. Improvement margin

Improvement margin is a metric comparing how many voters prefer one project selection over another. I compare CSTV results with corresponding Greedy rules. This metric is obtained by subtracting the number of voters that prefer Greedy rules over CSTV from the number of voters that prefer CSTV over Greedy rules. A voter prefers project selection with higher utility for that voter.

As usual, I normalize this score by dividing it by the number of voters.

Algorithm 14: Improvement margin

Data: $E = (P, V, \text{cost}, b), B_1, B_2$
Result: improvement margin of B_1 over B_2 in E
$m_1 \leftarrow 0$ ▷ m_1 is number of voters that prefer B_1 (CSTV)
$m_2 \leftarrow 0$ ▷ m_2 is number of voters that prefer B_2
for $v \in V$ **do**
  $u_1 \leftarrow \sum_{p \in B_1} v(p)$ ▷ u_1 is utility of B_1 by v
  $u_2 \leftarrow \sum_{p \in B_2} v(p)$ ▷ u_2 is utility of B_2 by v
  if $u_1 > u_2$ **then**
    $m_1 \leftarrow m_1 + 1$
  end
  if $u_2 > u_1$ **then**
    $m_2 \leftarrow m_2 + 1$
  end
end
return $\frac{m_1 - m_2}{|V|}$

3.5. Extended Justified Representation+

This variant of Extended Justified Representation measures the number of projects that have not been selected, but there exists a coalition that is underrepresented in the selection, voted for the project that has not been selected, and has enough power to make that project a good choice to be selected. Such projects are called violations, because they violate the Justified Representation rule. To compare this metric between different aggregation methods, the resulting sets are transformed into sums of their elements' utilities, then normalized by dividing it by the total number of votes.

Algorithm 15: EJR

Data: $E = (P, V, \text{cost}, b), B$
Result: normalized utilities of EJR of B in E
$F \leftarrow \varnothing$ ▷ F is a set of EJR violations
for $p \in P$ **do**
  if $p \notin B$ **then**
    $c \leftarrow 0$ ▷ c is coalition size
    for $v \in V$ **do**
      if $v(p) > 0$ **then**
        $c \leftarrow c + 1$;
        if $\sum_{p_j \in B} v(p_j) < \frac{c}{|V|} b - \text{cost}(p)$ **then**
          $F \leftarrow F \cup \{p\}$;
          break ;
        end
      end
    end
  end
end
return $\frac{\sum_{p \in F,\ v \in V} v(p)}{\sum_{p \in P,\ v \in V} v(p)}$

Chapter 4

Analysis results

Analysis results shown are based on four elections: Katowice 2023, Częstochowa 2024, Warszawa 2024, Łódź 2023. Among those, all except Warszawa 2024 were performed as cumulative elections, as described in the first chapter, while Warszawa 2024 was performed as an approval vote (as cumulative, but with $v(p) \in \{0, 1\}$) and converted into cumulative elections by scaling votes by project costs.

Analyzed are 9 methods: 3 Greedy rules and 6 variants of CSTV rules. CSTV variants are made by combining EWT and MT non-eligible projects resolution methods with the 3 base Greedy rules:

Table 4.1: CSTV variant names based on base method and weighting strategy

	EWT	**MT**
GE	EWT	MT
GSC	EWTC	MTC
GS	EWTS	MTS

The most impact on the results has the type of Greedy rules used. Metrics of CSTV methods are better or, at worst, negligibly worse than Greedy rules, but the improvements are considerably smaller than the difference caused by changing the Greedy rule.

Utility scores - Voter satisfaction gives interesting results. While GSC and GE-based methods give results with similar utility scores, GS-based methods have an outlier. MTS scores significantly outperform GS and EWTS. In smaller instances, namely in Katowice and Częstochowa, GSC-based methods and MTS give results with better utility than other methods, while in larger instances GE methods have results similar or better to GSC-based methods. GS and EWT are consistently the worst results while MTS keeps high utility scores comparable to GE and GSC-based methods.

Exclusion ratio - GSC-based methods give results with a better exclusion ratio than GE-based results in nearly all cases, with the only exception in the Warszawa instance having only a small difference between their ratios. GS-based methods show again an outlier. While GS and EWTS always have much higher ratios than GSC-based methods, in most instances the highest, MTS based methods are on average only slightly worse than the best result: MTC.

Improvement margin - Improvement margin compares CSTV methods to their Greedy rule bases. Smaller instances see less improvement, with the Katowice instance showing negative improvement for multiple rules. GS-base gets the most improvement, with MTS having the highest improvement margin, twice as high as the second-best, or even more in the case

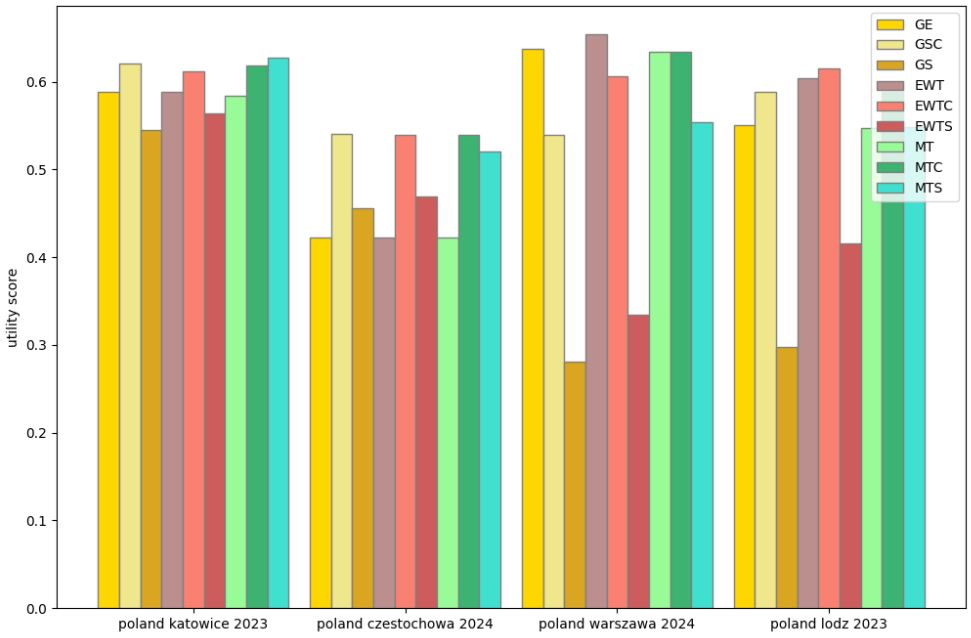

Figure 4.1: utility scores for all methods across the four analyzed elections

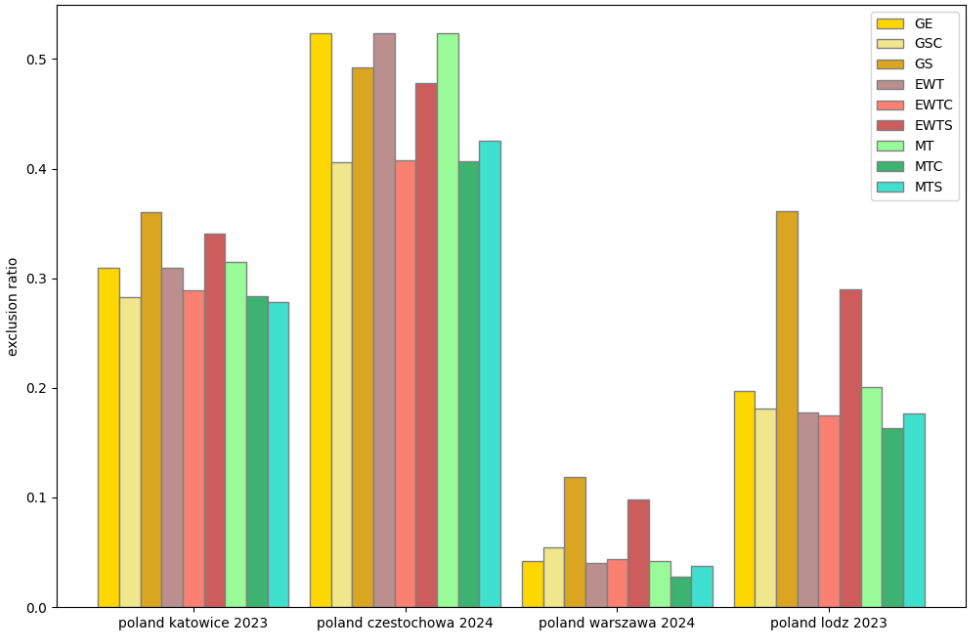

Figure 4.2: exclusion ratio for all methods across the four analyzed elections

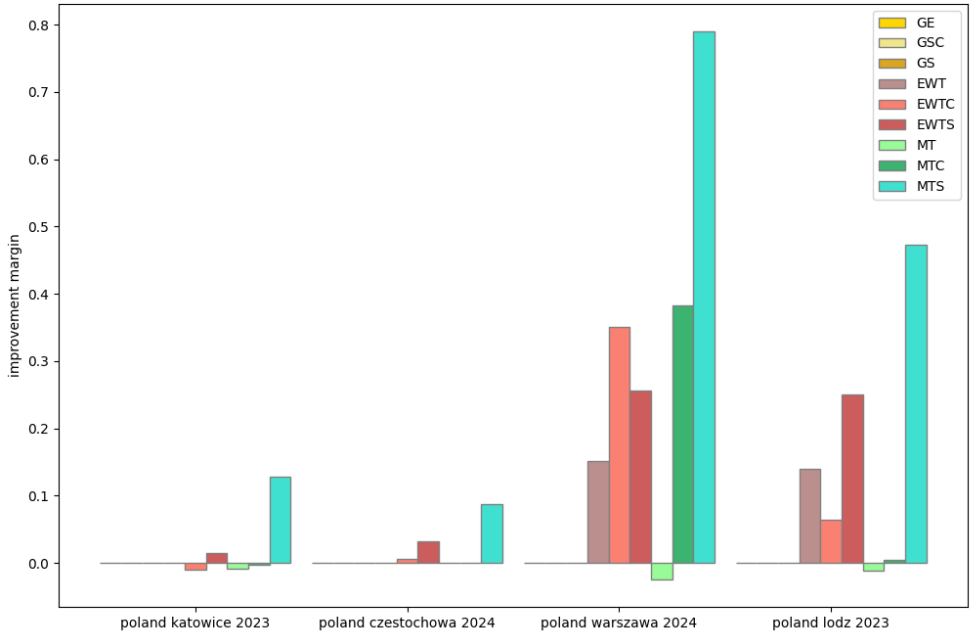

Figure 4.3: improvement margin for all methods across the four analyzed elections

of the Katowice instance. EWTS meanwhile also shows improvement, even in the Katowice instance, where only EWTS and MTS have positive improvement margin. GE-based rules, on the other hand, show the least improvement. MT doesn't have positive improvement margin in any instances when compared to GE and EWT has margin negligibly close to 0 in smaller instances. Lastly GSC-based rules show average improvement, with Warszawa instance having second and third highest improvement margin in MTC and EWTC respectively, but the smallest instance still shows negative improvement.

Power inequality - Power inequality differences are highly dependent on the size of the instance. In small instances this measure is nearly equal to the same base rule, with the exception of MTS that gets better results than all GS and EWTS. MTC is consistently the rule with the lowest power inequality, followed by EWT and EWTC, depending on the instance. GS and EWTS consistently high scores, while GE-based scores are the most volatile, with EWT claiming both nearly the lowest and the highest scores, depending on the instance, with better scores in larger instances.

EJR - The highest number of Extended Justified Representation (EJR) violations occurs in the results of GS and EWTS. The best results are, on average, given by GSC-based results, with GSC being the worst of the three and EWTC being more volatile than MTC that gets consistently low numbers of violations. Among GE-based results EWT is consistently the best rule, with performance similar to EWTC and MTC, while GE is the worst. MTS is, similarly to other analysis methods, significant improvement compared to GS and EWTS. In smaller instances it holds results better than GE-based rules, while in larger instances it has results comparable to the worst non-GS/EWTS rule.

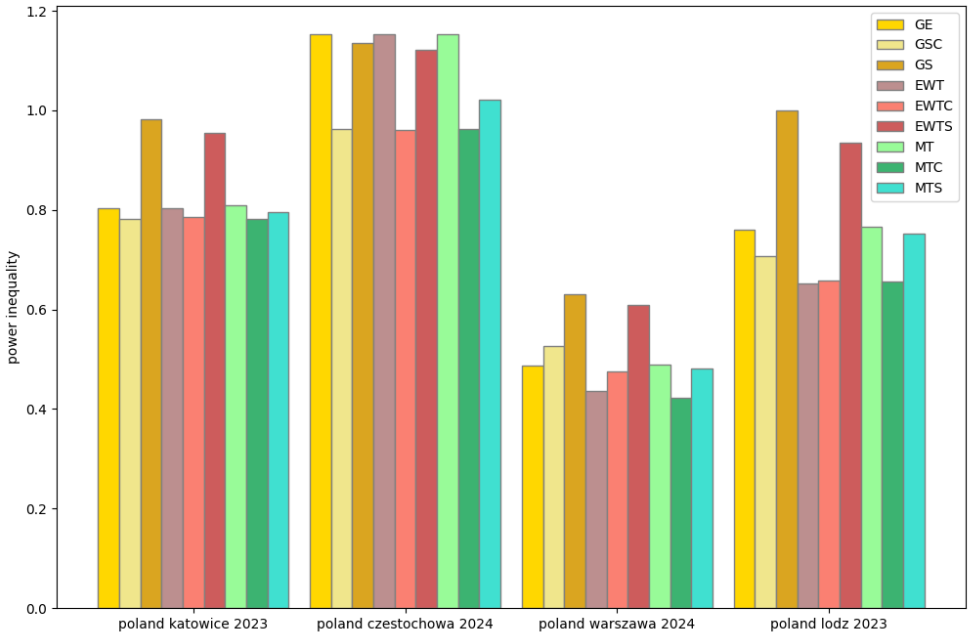

Figure 4.4: power inequality for all methods across the four analyzed elections

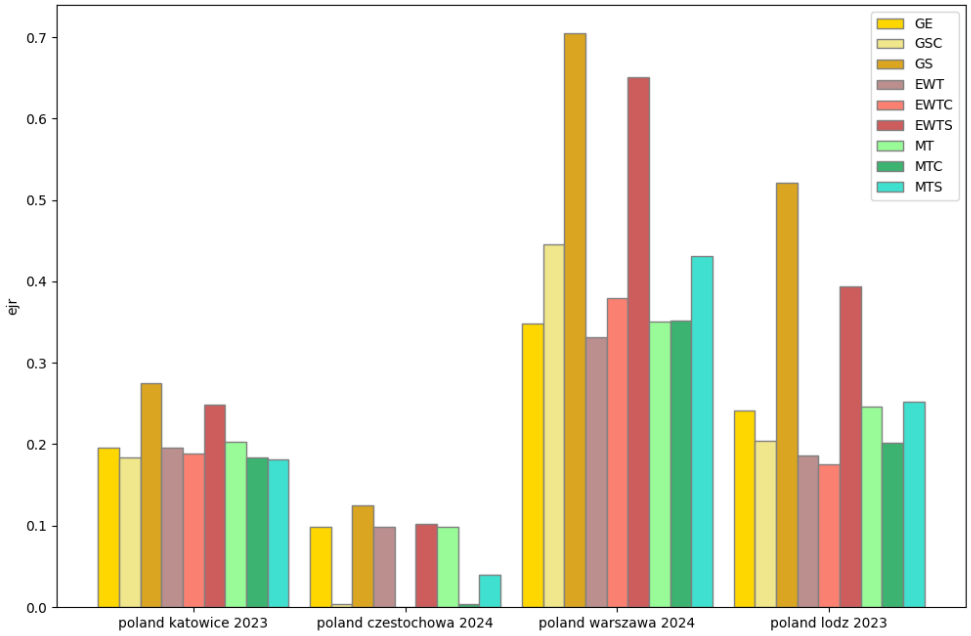

Figure 4.5: ejr for all methods across the four analyzed elections

Chapter 5

Summary

Cumulative voting is not yet a widely used method of participatory budgeting, but thanks to the revised and expanded implementation of CSTV in *pabutools* library, it can become more accessible. Analysis of these methods was performed thanks to the implementation of metrics described in chapter 3.

Based on the real-world data analysis, CSTV methods generated results with consistently better or comparable metrics than Greedy rules, although the improvements were modest. The choice of an appropriate Greedy rule has more impact than switching to CSTV rules or picking between EWT and MT variants.

Among the nine analyzed rules, the EWT and MTC variants achieved the best results on average. MTC performed better in most cases, while EWT achieved better results from a voter's utility perspective, especially in larger instances. The greedy by support rule, often disregarded in prior studies for having significantly worse results than other rules, showed unexpectedly strong results when used as the basis for the MT variant of CSTV. While the GS rule and its EWT variant performed, as expected, significantly worse than other rules, the MTS variant demonstrated consistent improvements over MT and achieved metrics comparable to MTC in a substantial portion of the tests.

Thanks to the source code attached to the thesis and the *pabutools* library, further analysis can be performed on new election results. In addition, the framework enables future comparisons between cumulative and approval voting methods, which could contribute to a deeper understanding of participatory budgeting mechanisms.

Bibliography

[1] Piotr Skowron, Arkadii Slinko, Stanisaw Szufa, Nimrod Talmon, *Participatory Budgeting with Cumulative Votes*, 2020.

[2] Piotr Faliszewski, Jarosław Flis, Dominik Peters, Grzegorz Pierczynski, Piotr Skowron, Dariusz Stolicki, Stanisław Szufa, and Nimrod Talmon *Participatory Budgeting: Data, Tools, and Analysis*, 2023.

[3] Dominik Peters, Piotr Skowron *Proportionality and the Limits of Welfarism*, 2022.