



Visualising and Analysing Participatory Budgeting Elections

WARWICK

The Binary Budgeteers

Kartheyan Sivalingam & Leon Chipchase & James Harvey
& Ahmed Minhas & Taha Mohyuddin

Supervised by Markus Brill

Abstract

Participatory Budgeting (PB) is a democratic process that offers an alternative to conventional budgeting, where elected officials select projects to be funded. It allows community members to directly allocate a portion of the public budget toward projects of their choice. Offering a range of budget allocation rules, each with different fairness, effectiveness and complexity. Through the visual examination and analysis of voting rules, we facilitate the understanding of intricate voting rules for non-mathematical or technical users. This consequently shows the increased fairness and transparency in PB public fund allocation. Specifically, we have developed visual explanations outlining how the Method of Equal Shares (MES) and Greedy Utilitarian Welfare voting rules select projects. They work on elections of any size, ensuring they are applicable to past and future PB elections carried out in various municipalities worldwide.

Our solution is publicly available, building upon the existing Pabutools Python library, a complete set of tools for working with PB instances.

Keywords: *Participatory Budgeting, Pabutools, Visualisation, Method of Equal Shares, Voting Rules, Elections*

Acknowledgements

Our sincere appreciation goes out to Markus Brill, our supervisor, who provided us with invaluable guidance and support throughout the course of this project. We also thank Simon Rey, our customer, for allowing us to contribute to Pabutools, and guiding us for the duration of the project.

Contents

1	Introduction	1
1.1	Introduction to Participatory Budgeting	1
1.2	Voting Rules	3
1.2.1	Utilitarian Welfare	3
1.2.2	Greedy Utilitarian Welfare	4
1.2.3	Method of Equal Shares	4
1.2.4	Additional Rules	7
1.3	Aims and Motivation	8
1.4	Pabutools	9
1.4.1	Pabulib	10
1.5	The Customer	11
1.6	Our Solution	12
1.7	Report Overview	13
2	Literature Review	14
2.1	Origins and Impact of PB	14
2.2	Voting Rules	16
2.2.1	Welfare-Maximisation Rules	16
2.2.2	Sequential Purchase Rules	17
2.3	Visualisation	18
2.4	Existing Solutions	20
2.4.1	Pabustats	20
2.4.2	Pref.tools	21

2.4.3	Pabuviz	22
2.4.4	Limitations of Existing Solutions	23
2.5	Conclusion of Literature Review	24
3	Requirements	26
3.1	Objectives	26
3.2	Requirements	27
3.2.1	Functional Requirements	28
3.2.2	Non Functional Requirements	30
3.3	Testing Objectives	32
4	Design	33
4.1	Visualising Method of Equal Shares	33
4.1.1	Summary Page	33
4.1.2	Round by Round Page	40
4.2	Visualising Greedy Utilitarian Welfare	48
4.2.1	Page Design	48
4.2.2	Satisfaction Measure Chart	49
4.2.3	Round Overview	50
4.2.4	Satisfaction Measure Chart Coloured by Acceptance Status	51
4.3	Software Stack	52
4.4	Architecture Overview	56
4.4.1	Pabutools	56
4.4.2	Data Capture	59
4.4.3	Analysis	62
4.4.4	Generating web pages	63
4.4.5	Web Pages	65
5	Implementation	66
5.1	The Pabutools Library	66
5.1.1	Data Capture Classes	67

5.1.2	MESVisualiser	68
5.1.3	GreedyWelfareVisualiser	72
5.2	Visualising MES	74
5.2.1	Summary Page	74
5.2.2	Round by Round Page	80
5.2.3	Maintaining Efficient Runtime	85
5.2.4	Runtime Analysis	88
5.3	Visualising Greedy	92
5.3.1	Optimising the Greedy Visualisations	94
5.4	Implementing Clustering on .pb Files	96
5.4.1	Data Visualisation	97
5.4.2	Clustering Algorithms	98
5.4.3	Results	99
5.4.4	Conclusion of Clustering	101
6	Evaluation	103
6.1	Testing	103
6.1.1	Unit Testing	103
6.1.2	Integration Testing	106
6.1.3	Performance Testing and Bench Marking	108
6.1.4	Visualisation Time Performance	110
6.2	User Testing	113
6.2.1	Familiarity with Participatory Budgeting	114
6.2.2	Effectiveness of the Greedy Page	115
6.2.3	Effectiveness of the MES Pages	117
6.2.4	Improvements	122
6.3	Testing Objectives	123
6.4	Fulfilment of Aims and Objectives	125
6.5	Limitations	132

7	Project Management	135
7.1	Project Methodology	135
7.2	Planning	137
7.2.1	Feasibility Study	137
7.2.2	Work Breakdown Structure	138
7.2.3	Scheduling	140
7.3	Organisation	141
7.3.1	Roles & Communication	141
7.3.2	Meetings	142
7.3.3	Version Control	144
7.4	Risk Management	144
7.4.1	Original Risk Assessment Form	144
7.4.2	Encountered Risks	147
7.4.3	Revised Risk Assessment Form	148
7.5	Legal, Social, Ethical and Professional Issues	148
7.5.1	Legal Issues	148
7.5.2	Social Issues	149
7.5.3	Ethical Issues	149
7.5.4	Professional Issues	150
7.5.5	Public Good	150
8	Conclusion	151
8.1	Future Work	152
8.1.1	Advanced Visualisations	152
8.1.2	Further Clustering Research	152
8.1.3	Additional Rules	153
8.1.4	Rules Comparison	157
	Bibliography	161
	A Survey	168

B Management	171
C Documentation & User Manual	175

List of Figures

1.1	Comparative Diagram of Conventional Budgeting vs. Participatory Budgeting Processes	2
1.2	An Example Computation of the Effective Vote Count for a Project Which Costs £75	5
1.3	Visual Representation for the Method of Equal Shares Voting Rule [7]	6
1.4	An Example PB Election in .pb Format	11
2.1	Example Displaying the Output After Running the Pabustats Visualisation	21
2.2	The Default View of the pref.tools Site Displaying an Example Election and Its Outcomes	22
2.3	Example Pages From the Pabuviz Site	24
3.1	Testing Objectives	32
4.1	Example of How the Budget Was Distributed Among Nine Selected Projects With and Without a Tooltip	35
4.2	Example of How the Full Overview Table Looks	36
4.3	Expanding a Row in the Summary Table to Reveal a Hidden Row for a Selected Project	37
4.4	Summary Chart Comparing the Project’s Actual Voter Funding with the Initial Voter Funding	38
4.5	Summary Chart with a Tooltip Showcasing the Three Most Popular Previously Selected Projects That Supporters Also Voted For	39

4.6	An Example Round Summary	40
4.7	Example of an Effective Vote Count Bar Graph	41
4.8	Example of a Reduced Effective Vote Count Bar Graph	42
4.9	Effective Vote Count for Project <i>X</i>	44
4.10	Demonstrating the Difference in Effective Vote Count for Project <i>Y</i> Before and After Electing Project <i>X</i>	45
4.11	Demonstrating the Difference in Effective Vote Count for Project <i>W</i> Before and After Electing Project <i>X</i>	45
4.12	A Flow Diagram Visualising the Relation Between Project <i>X</i> and the Other Projects Participants Voted For	45
4.13	Example of a Sankey Diagram	47
4.14	Example of a Chord Diagram	48
4.15	An Example of How the Satisfaction Measure Bar Chart is Displayed on the Greedy Page	50
4.16	An Example of How the Round Analysis Visualisation is Displayed on the Greedy Page	51
4.17	An Example of How the Satisfaction Measure Bar Chart Coloured by Acceptance Status Is Displayed on the Greedy Page	52
4.18	A Basic Example of a Jinja Template Used to Generate a HTML Page	54
4.19	A Partial UML Diagram for Subset of Relevant Classes and a Gen- eralisation of How It Connects to Voting Rule Function (Before Our Changes)	57
4.20	A Partial UML Diagram for Subset of Relevant Classes and How It Connects to the MES Voting Rule Function (Before Our Changes) . .	59
4.21	A UML Diagram for the New Object Returned by Voting Rules En- abling Us to Store and Retrieve Data From Voting Rules (previously, a Voting Rule Would Return Just a List of Projects)	61
4.22	A UML Diagram for the Visualisation Classes and How They Connect to the Classes in Figure 4.21	64

5.1	An Activity Diagram Showing the Responsibilities of the Individual Components of the Projects	68
5.2	Code Structure for MESVisualiser Class	69
5.3	Structure of the rounds List Dictionary That is Passed Into Our MES Templates Using Jinja	71
5.4	Structure of the rounds List Dictionary That is Passed Into Our Greedy Template Using Jinja	73
5.5	Expanded Rows Displaying Dynamic Explanations for Scenarios Involving Rejected Projects	78
5.6	An Example Chord Diagram Without Project Prioritisation	83
5.7	A Demonstration of the Chord Diagrams Not Including an Important Project	84
5.8	Pie Chart Calculations	87
5.9	Runtime Distribution for MES Rule with Analytics Set to True	88
5.10	Percent Increase in Time for MES With Visualisations - Pre Optimisation	89
5.11	Percent Increase in Time for MES with Visualisations - Post Optimisation	90
5.12	Code Structure for Greedy Visualiser Class	92
5.13	Time Increase with Analytics for 605 Elections. Mean: 0.385%, STD: 3.639%	93
5.14	Greedy Visualisation Example	94
5.15	Percentage Increase for Time Taken for Greedy Visualisations. Mean 5.124%, 0.00604s	95
5.16	Percentage Increase for Time Taken for Greedy Visualisations - Post Optimisation. Mean 3.004%, 0.00513s	96
5.17	PCA Versus t-SNE 2D Transformation Results - Poland Wroclaw 2018	98
5.18	Clustering Results Using DBSCAN with $\varepsilon = 0.3, 0.2, 0.1$ (Left, Middle Right)	100
5.19	Time Taken to Run DBSCAN on Varying Size Elections	101

6.1	Code Coverage Report for PR #15 (pabutools/pull/15)	104
6.2	Testing Development Process	107
6.3	Percent Increase in Time for MES with Visualisations. Mean: 152.1%, 2.147s	109
6.4	Percentage Increase for Time Taken for Greedy Visualisations. Mean: 3.004%, 0.00513s	110
6.5	Percent Increase of Runtime With Visualisations Against Voters (Left) and Projects (Right)	111
6.6	Increase of Runtime With Visualisations Against Voters (Left) and Projects (Right)	112
6.7	Percent Increase in Processing Time for Greedy Visualisations	113
6.8	Proportions of Testers Familiar with PB, Greedy, and MES	115
6.9	Comprehension and Understanding of the Greedy Rule	116
6.10	Question: <i>Would you trust a set of projects decided in your local area with this method?</i>	117
6.11	Comprehension and Understanding of the MES Rule. 1: No Under- standing, 10: Perfect Understanding	118
6.12	What Parts of the Pages Provide a Sufficient Understanding of the Rule	119
6.13	How Effective and Clear Are Each of the Figures in Our Visualisations	120
6.14	Understanding of MES Nuances and Specifics	121
6.15	Support and Trust for PB	121
7.1	An Example of how the Kanban Board was used within a Sprint Cycle	136
7.2	The Summary of a Typical Two-Week Sprint Conducted During the Project	137
7.3	Diagram of the Project's Work Breakdown Structure Where Each Colour Corresponds to a Different Deliverable	139
8.1	Example Graph Visualisation for Round-By-Round Analysis of Se- quential Phragmen's Rule	154
8.2	Project Proposal Ideas for Participatory Budgeting in New York City	156
8.3	Mock Example Heatmap of Elected Projects in Warsaw	157

8.4	Comparative Density Plot of Participant Satisfaction for the Greedy Rule and Method of Equal Shares	158
8.5	Visualisation of Projects in PB Elections Using GPS Data	159
8.6	Visualisation of Projects in PB Elections Using Jaccard Distance	160
A.1	User Testing Form Introduction	168
B.1	Example of How the Meeting Notes Were Laid Out For a General Meeting with the Supervisor Exported From Notion	171
B.2	Example of How the Meeting Notes Were Laid Out for a Meeting with the Customer Exported From Notion	172
B.3	Initial Meeting Notes for Requirements Elicitation	173
B.4	Gantt Chart of Our Three-Semester Timeline for the Project	174
C.1	Visualisation Module Documentation Page One	175
C.2	Visualisation Module Documentation Page Two	176
C.3	User Manual to Install and Generate Visualisations Page One	177
C.4	User Manual to Install and Generate Visualisations Page Two	178

List of Tables

3.1	Project Objectives	27
3.2	Functional Requirements	30
3.3	Non-Functional Requirements	31
5.1	Descriptive Statistics for Percent Increase in Time for MES with Visualisations	89
5.2	Descriptive Statistics for Percent Increase in Time for MES with Visualisations - Optimised	91
5.3	Number of Clusters for Varying ϵ Values	100
6.1	Testing Objectives	125
6.2	Achievement of Functional Requirements	127
6.3	Achievement of Non-Functional Requirements	129
6.4	Achievement of Project Objectives	131
7.1	Priority Labels and Their Corresponding Definitions	140
7.2	Team Roles	142
7.3	The initial Risk Assessment Form indicating the Likelihood and Potential impact of Risks	146
7.4	The New Entry to the Original Risk Assessment Form, Indicating the Likelihood and Potential Impact of the Risk Occurring	148
A.1	User Testing Questions	170

Abbreviations

PB - Participatory Budgeting

MES - Method of Equal Shares

Greedy - Greedy Utilitarian Welfare

Chapter 1

Introduction

1.1 Introduction to Participatory Budgeting

Participatory budgeting (PB) is a democratic initiative where community members directly influence how a public budget is allocated. This contrasts the more popular and widely adopted process of electing representatives, who then choose how to allocate the public budget. The diagram in Figure 1.1 illustrates this distinction: in representative systems, the public elects officials who select projects, potentially leading to outcomes that do not reflect the community's interests. In PB, citizens vote directly on projects, ensuring alignment with community needs. This innovative approach empowers citizens by engaging them in the decision-making process that impacts their lives and communities. Originating in Porto Alegre, Brazil, in 1989, PB has since spread globally, with the latest estimates suggesting over 1500 implementations of PB, finding diverse applications in cities, schools, and public agencies across different cultures and government structures [1].

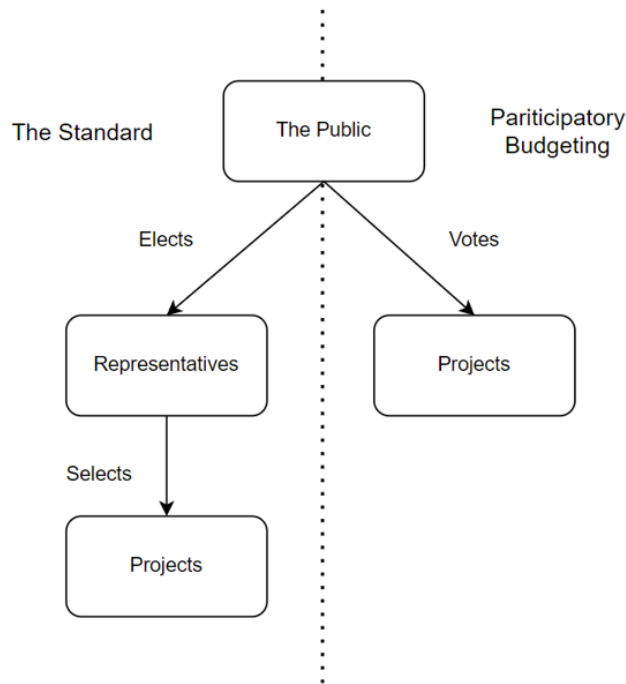


Figure 1.1: Comparative Diagram of Conventional Budgeting vs. Participatory Budgeting Processes

The core of PB involves community members taking on the role of decision-makers tasked with identifying, discussing, and prioritising public spending projects. The process typically follows several stages: ideation, where community needs are identified; proposal development, where ideas are transformed into feasible projects; voting, where citizens choose which projects to fund; and implementation, followed by monitoring and evaluation of the funded projects.

Following its inception in Porto Alegre, PB has been adapted to various global contexts with notable successes. For instance, in Tower Hamlets, London, the ‘You Decide!’ initiative engaged local residents in allocating resources for community projects, enhancing local participation and satisfaction [2, 3]. Similarly, in Govanhill, Glasgow, PB has been instrumental in addressing local priorities through direct community involvement, showcasing the adaptability of PB across different urban settings [4].

A key benefits of PB is its direct impact on community engagement. By involving community members in decision-making, PB helps demystify budgetary processes, making government operations more transparent and accountable. It often leads to more equitable public spending, with funds more likely to be directed towards high-priority community needs, which may be overlooked in the traditional budgeting process. Moreover, PB can significantly enhance local infrastructure and services by aligning them more closely with the community's actual needs and desires. For instance, in Porto Alegre, citizens used PB to fund the construction of new schools and healthcare facilities, as well as the improvement of public transportation, where citizens increased the health and education budget from 13% to almost 40% of the total city budget from 1985 to 1996 [5]. This alignment can increase public trust in government and improve participation rates, as individuals see tangible results from their involvement in government decision-making.

1.2 Voting Rules

Voting rules in PB elections determine how participants' preferences are translated into decisions about which projects get funded. The choice of voting rule significantly impacts the fairness and outcomes of the budget allocation, influencing how effectively community needs are met. The following subsection will explore various voting rules employed in PB, detailing how each method functions and their implications for project selection and community engagement.

1.2.1 Utilitarian Welfare

Utilitarian welfare refers to a framework where the budget allocation decisions aim to maximize the community's overall happiness or utility. This approach prioritises funding projects that provide the greatest total benefit to all participants, thus optimising collective satisfaction.

1.2.2 Greedy Utilitarian Welfare

Utilitarian welfare maximisation is NP-hard [6]. A solution to this issue is the Greedy Utilitarian Welfare algorithm. The Greedy Utilitarian Welfare is an approximation of the utilitarian welfare. It selects projects in rounds, each time selecting a project that leads to the highest increase in total satisfaction divided by the project's cost.

1.2.3 Method of Equal Shares

Methodology

The main idea of the Method of Equal Shares (MES) voting rule is that each voter is assigned an equal part of the total budget. The voter's budget can only fund projects for which they have voted. The method iterates through all project proposals, starting with the projects with the highest number of votes. If a project can be funded using the budget share of those who voted for it, it is selected. The method divides the project's costs evenly among those who support it. The method can be used with two distinct types of inputs:

- **Approval Voting:** Where each participant votes for some of the projects with the same strength.
- **Utilities:** Where each voter can designate the amount from their personal budget that they want to allocate to each project.

Effective Vote Count

When identifying the winning projects of MES, it is essential to calculate what's known as the effective vote count. A key principle here is the exclusion of voters who have exhausted their budget share from the count. This rule operates on the principle that if a voter has already spent their entire budget share, then they have already been satisfied by the projects that were selected. Consequently, priority is given to funding projects that appeal to the remaining voters.

Voters with remaining funds but not enough to finance the project when its cost

is equally divided will count as a partial contribution. The effective vote count for a participant is calculated as their remaining budget divided by the maximum funds a participant will have to pay to fund the project. Figure 1.2 demonstrates how the effective vote count is calculated for a project. Suppose we have five project supporters, each with a different remaining budget (10, 10, 10, 20, 30 respectively). To fund a project that costs 75, the first four voters must pledge their entire remaining budget, whilst the fifth voter must allocate 25. The effective vote count of each participant is then the amount they will spend to fund this project divided by 25. In the case in Figure 1.2, the effective vote counts are 0.4, 0.4, 0.4, 0.8, 1, respectively. The effective vote count for the project is calculated as the sum of all effective votes of supporters; in our case, the effective vote count is three.

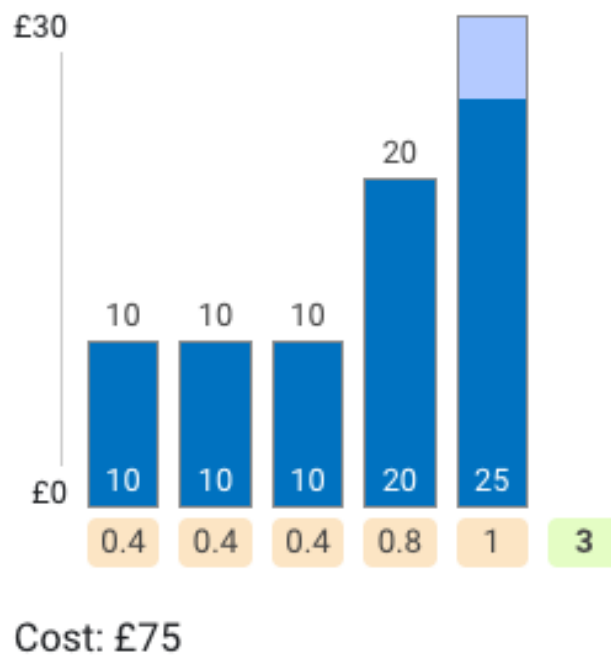


Figure 1.2: An Example Computation of the Effective Vote Count for a Project Which Costs £75

Figure 1.3 illustrates a simple example of a PB election outcome employing the MES voting rule. Initially, the total budget is distributed evenly among all participating voters. In this scenario, each voter receives an equal stake of €150 to allocate. Voters assign their allocated funds to various projects of their choosing. The projects are then funded with the shares of those who voted for them. This visualisation exemplifies the PB process, where the collective input of community members directly influences the financial support of community projects.

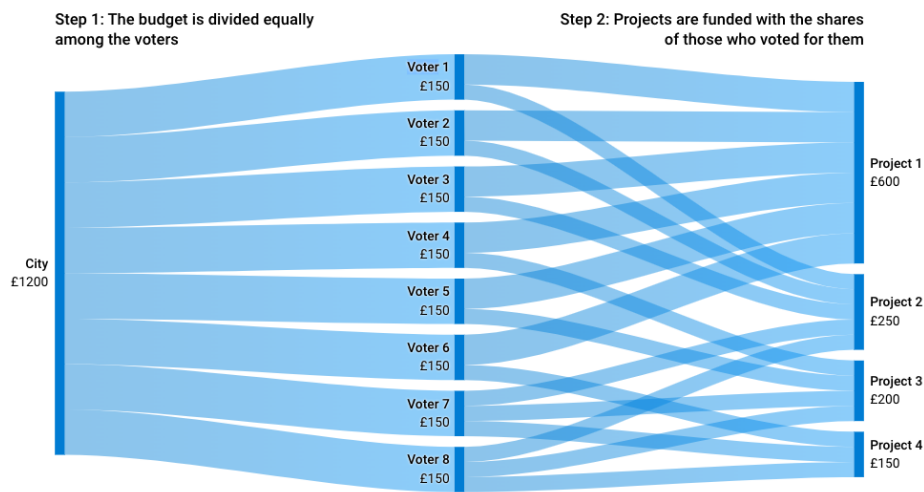


Figure 1.3: Visual Representation for the Method of Equal Shares Voting Rule [7]

Benefits

The MES voting rule amplifies the democratic aspect by ensuring each vote carries equal weight in budget allocation decisions. Consider, for example, that 51% of the population supports ten educational projects, 49% supports ten community projects, and the money suffices only for ten projects. The classical election method will choose the ten educational projects supported by 51% and ignore the 49% altogether. In contrast, MES would pick five of each of the project types. The MES voting rule has the following benefits:

- **Increased and More Equitable Voter Satisfaction:** Simulations indicate that this method results in higher approval rates for winning projects, and a

more evenly distributed satisfaction across the voter base.

- **Reduced Bias:** Unlike traditional methods, which tend to favour projects in specific categories at the expense of others, the MES voting rule minimises these biases. This ensures a representation that more accurately reflects the preferences of the entire population.

1.2.4 Additional Rules

Completion Rules

Since not all rules return exhaustive budget allocations, further methods are required to render the outcome exhaustive. Completion methods further increase satisfaction as a larger proportion of the budget is assigned.

- **Exhaustion by Budget Increase:** One method is to increase the budget limit in the hope that the rule returns an exhaustive budget allocation for the original instance. If, at any point, the rule returns a budget allocation that is not feasible for the original budget limit, then the previously returned budget allocation is returned.
- **Exhaustion by Rule Combination:** Another strategy involves applying a sequence of voting rules until an exhaustive budget allocation is achieved. The rules are implemented in a specific order, and the process continues until an exhaustive budget allocation is reached, or all rules have been applied.

Other solutions to resolve remaining funds include the flexible option of saving the money to be included in the available budget for the following year. Another option is to spend the money on an amicable backup solution, for example, to fund the maintenance and improvement of existing infrastructure or public amenities such as road works.

Tie-Breaking

In executing the MES voting rule, a tie can occur when two or more projects have the same effective vote count. In isolation, ties are unlikely to occur in large elections. An experiment on election data on Pabulib, a PB library [8], found that only 0.6% of elections resulted in a tie-breaking rule [9, 10]. One method of breaking ties is as follows:

- The project with the lowest cost is selected.
- If two or more tied projects have the same lowest cost, then the project with the highest initial vote count is selected.
- If two or more projects have the lowest cost and the same initial vote count, the tie is broken uniformly randomly.

Rule Composition

Another method is composing the previously mentioned methods by computing the outcome of several rules and returning the most preferred by the most extensive set of voters, according to a given satisfaction measure.

1.3 Aims and Motivation

For almost all cities' PB elections, the Greedy rule is the standard rule used [11]. This is because, compared to modern voting rules such as MES, the Greedy rule is simpler and more transparent. These properties make it easier to understand how Greedy works and why this voting rule chooses to accept or reject different projects. Because of this, modern voting rules are less likely to be trusted and used. This can happen even when these voting rules provide benefits that may be highly desirable for specific elections. For example, MES is able to guarantee proportional representation to groups of voters with common interests [12], which can help with ensuring that minority groups are properly represented in an election's outcome, encouraging them

to participate in further elections.

In analysing PB elections, the use of visual tools and data analysis is instrumental not just for providing insights but also for enhancing public understanding and trust in the process. Effective visualisation conveys complex data in a straightforward, accessible manner, allowing community members to see how decisions are made and how funds are allocated. This transparency helps demystify the budgeting process, building trust and confidence among participants by showcasing the direct impact of their contributions. This transparency extends to the various voting rules that can be used in these elections, meaning that visualisations could be used to better inform voters about how different voting rules work, as well as why these voting rules may choose to accept and fund different sets of projects.

Taking the reasoning above into account, the three aims of this project were as follows:

- Create a product that visually explains the outcomes of different PB voting rules.
- Build this product on the existing work implemented in the Pabutools library (to be discussed in the next section) - a package owned by the customer.
- Offer varying levels of explanation depth for each voting rule, suitable for the general public.

1.4 Pabutools

Pabutools is a Python library that provides tools to handle various kinds of PB instances and a variety of voting rules to simulate the outcome of elections and provide analysis. This toolkit enables users to parse Pabulib files, an extensive library for PB data [8], and apply selected rules for participatory budgeting. The library is publicly available via PyPI [13], making it accessible for anyone working with PB budgeting data.

1.4.1 Pabulib

Pabulib (PARTicipatory BUDgeting LIBrary) is a publically accessible library for Participatory Budgeting [8] election. This website is dedicated to gathering data on participatory budgeting elections from various global sources. The library uses a universal data format to store the files with the **.pb** extension [14].

Information regarding a single instance of participatory budgeting election should be stored in a UTF-8 encoded text file with the **.pb** extension. This file is organised into three distinct sections:

- **META** section containing essential metadata such as the nation, budget allocation, and tally of votes.
- **PROJECTS** section detailing the financial aspects of projects and other potential metadata, including project category, objectives, and location.
- **VOTES** section recording the voting data, which may include various types of voting. This section may also encompass voter metadata, such as demographic information, including gender and age.


```

META
key;value
description;Experimental PB on Mechanical Turk | Knapsack_7
country;Artificial
unit;Mechanical Turk
instance;Knapsack_7
num_projects;20
num_votes;69
budget;500000
vote_type;approval
language;english
currency;USD
PROJECTS
project_id;cost;name;category;votes;description
3;27000;Computers for the community learning;Culture & community;48;
14;25000;Benches for a Walkable City;Streets, Sidewalks & Transit;41;
13;24000;Real-Time Bus Arrival Monitors in bus stations;Streets, Sidewalks & Transit;40;
12;50000;Separate Bike Lanes from Traffic;Streets, Sidewalks & Transit;39;
2;13000;Little (book exchange) libraries;Culture & community;39;
31;8000;Fire Hydrant Markers;Environment, public health & safety;38;
VOTES
voter_id;age;sex;vote;education
512;21;F;5,13,31,33,50;College
513;49;M;41,42,47;College
514;36;M;13,14,21,22,31;Graduate degree
515;24;M;2,3,12,13,14,21,23,33,41;College
2176;26;M;3,13,31,41,47;High School/GED
2306;49;F;3,5,12,13,14,15,21,23,31,41;College
2442;33;M;3,8,15,21,31,33;Graduate degree
909;28;M;2,3,12,13,14,21,23,31,33,41;Graduate degree
2062;55;F;8,15,21,41;Graduate degree
1041;38;F;3,12,13,14,15,21,23,31,33,41;College
404;24;F;41,42,50;College

```

Figure 1.4: An Example PB Election in .pb Format

1.5 The Customer

Simon Rey is a recent PhD graduate from the University of Amsterdam who has done extensive work and research in PB [15]. Our supervisor, Markus Brill, who is also active in PB research, introduced us to Simon. Both Simon and Markus have expressed the need for a visualisation tool to explain voting rules to the general public. Simon is a key contributor and owner of the Pabutools Python package that we discussed previously. Having Simon as a customer has been very beneficial because we do not have to worry about implementing functionality that already exists within Pabutools, such as parsing of .pb files or simulating the PB elections with different rules. This allows us to focus on generating the visualisations which truly bring value. Contributing to an already existing package gives our work more visibility within the wider PB research community. However, contributing to an open source

package also comes with additional requirements, such as a higher bar for code quality, writing external user documentation on usage and changes, and an external PR (Pull-Request) approval process. Fortunately, since our customer is also the package owner, we have had the opportunity for more direct communication outside of just the PRs.

1.6 Our Solution

Our solution builds on the Pabutools library by creating a visualisation subpackage on top of the existing rule implementations. It provides a visual explanation for the outcome of PB elections decided using either MES or Greedy as the deciding rule. More precisely, our solution focuses on the explanation for how the election outcome was achieved rather than an in-depth analysis of the outcome itself.

For MES, we produce both a high-level overview and an in-depth analysis of the election. The summary page displays the budget broken down into the elected projects, as well as the outcome of the election and dynamically generated explanations for each part of the rule. This links to the in-depth analysis, allowing users to gain a better understanding of any stage of the election they may have been interested in or confused about. Our in-depth analysis includes a wide range of statistics and visualisations explaining various intricacies of MES to the user. For example, the relationship of votes between the elected project and the other projects not yet selected by the rule, giving insight as to why other popular projects may never be selected.

For Greedy, we display the selected projects each round and an ongoing showcase of the change in the remaining budget over time. Additionally, this displays the cases where a rejected project is very popular but is too expensive to be funded. Moreover, we give an overview of all projects with filtering and sorting, allowing users to view the election in depth.

Our solution ensures that the original implementations are unaffected, requiring only

the addition of a class storing auxiliary information required for our visualisations. Furthermore, they provide a basis for future work, ensuring other contributors can build on the implementations and expand to a complete set of PB rules, giving comprehensive explanations for all scenarios.

1.7 Report Overview

The remainder of this paper's structure is as follows: Chapter 2 explores the relevant literature surrounding PB. Chapter 3 covers the set of requirements, aims and objectives for this project. Chapter 4 discusses the overall design of the project. Chapter 5 analyses the implementation process of the visualisations and explanations. Chapter 6 examines how the system was tested against our requirements to produce a successful product. Chapter 7 reviews how the project was managed as a whole before Chapter 8 concludes with a short summary and discusses the prospects of future projects.

Chapter 2

Literature Review

In this chapter, we explore the relevant literature surrounding PB. Firstly, we discuss the inception and impact of PB before moving on to briefly discuss the various voting rules that have been developed for PB. The rest of the chapter makes up the bulk and involves exploring the existing solutions for visualising PB elections, before ending the chapter with a conclusion. Further, it is worth mentioning that the topic of PB and its various voting rules are still relatively new within this field. Therefore, as we explore the relevant literature, we make sure to recognise the ongoing contributions to this topic.

2.1 Origins and Impact of PB

While the topic of PB within this field is comparatively new, the origins actually date back to the late 1980s within the city of Porto Alegre in Brazil. As described by Cabannes [16], the historical analysis of PB comes in three stages. The first being the experimental phase between 1989-1997, where the initial trial of PB took place within Porto Alegre and a few other cities. The second was classed as the 'Brazilian spread' and took place between 1997-2000, this saw rapid adoption of the PB model by more than 130 municipalities within the country. Finally, the third stage, from 2000-present marks the expansion of PB beyond Brazil, where European and numerous Latin American cities have adopted similar models. As of now, PB has

spread across the globe and has been adopted in most countries, including the United Kingdom [17, 18].

Since its invention within Porto Alegre, PB has shown to have a positive impact to communities worldwide. This is demonstrated by Rathore et al. [19] who found that using PB within Porto Alegre has resulted in improved facilities for the local residents. For instance, increasing the number of sewer and water connections within households as well as increasing the number of schools four-fold. Further, Wampler et al. [20] found evidence that PB positively impacts the well-being of under-served communities in Brazil, Peru, and South Korea. Finally, closer to home with London's Tower Hamlets, 'You Decide!' project [2], in this case study, £5.06 million was allocated over two years for residents to decide on local service funding within the borough of Tower Hamlets. The project aimed to improve local services and increase community participation by allowing the local residents to design and choose services through the process. In the end, the project demonstrated positive impacts on empowerment and decision-making processes for the participants. Most of this has already been briefly mentioned within Chapter 1.

However, despite its praise and rapid adoption, the process still received a considerable amount of criticism. For instance, in the same paper by Rathore et al. [19], they attributed one of PB's shortcomings to a lack of representation of citizens who come from poorer backgrounds in the process. Similar findings were found with the 'You Decide!' project in Tower Hamlets [2]. While the project was considered a success (as it showed a noticeable impact on participants and their decision-making), many participants found that the process failed to reflect their preferences, indicating that the voting system may not have proportionally represented every group of voters. In addition, PB has largely been criticised for its low participation rate, as shown by Zepic et al. [21], where in Germany, public participation rates in PB projects were often below expectations, with reports of the rate as low as 0.1%. Similarly, from across the ocean, Stewart et al. [22] found the participation rate in Chicago to be

between 1%-3%.

From these papers mentioned, it is clear to see that while PB has rapidly been expanding and has showcased some positive results, there is still room for improvement to address the criticisms laid out earlier, such as satisfying proportional representation. The next section of the literature will briefly explore these ideas.

2.2 Voting Rules

Up to this point in the literature, we have discussed the origins, impact as well as some criticisms of PB. Moving forward, we briefly discuss the various rules and methods created to select a project within PB.

2.2.1 Welfare-Maximisation Rules

The first class of rules we discuss are the Welfare-Maximisation rules. These rules aim to optimise the overall 'welfare' of voters by returning the groups of projects which maximise a chosen utility function for the group of voters [23].

As described within Chapter 1, the utilitarian welfare maximisation is an NP-hard problem as it requires solving the Knapsack problem. Therefore, the Greedy Utilitarian Welfare is introduced which is an approximation of the utilitarian welfare. This is the most common rule within most cities that employ PB elections [11]. However, from the criticisms of PB discussed earlier, this rule does not provide a proportionally representative system.

An additional rule for Welfare-Maximisation to discuss briefly is known as proportional approval voting (PAV). This rule is a special case of Thiele's voting rule, which is a multi-winner voting rule proposed by Thorvalds N. Thiele [24]. The multi-winner voting rule is an electoral system explored by Edith et al. [25] in which multiple candidates can be elected. Later, this rule was adapted to PB by Pierczynski et al. [12]

and attempts to choose a rule that maximises the total score. Nevertheless, it fails to satisfy the strong proportionality guarantees that was observed with the multi-winner voting rule by Los et al. [26].

2.2.2 Sequential Purchase Rules

Another class of rules is known as sequential purchase rules, where voters receive some virtual currency which they use to 'buy' the projects. These rules aim to satisfy proportional representation, a criticism mentioned within Rathore et al.'s paper [19] and the Tower Hamlet's 'You Decide!' project [2]. Proportional representation ensures fair representation for all voters by having projects selected in proportion to the number of votes they receive.

The first of these rules is the Sequential Phragmén's Rule [27] - which has already been explained within Chapter 1 - where each voter starts with a budget of zero that continually increases. When a group of supporters has enough virtual currency to buy a project they all approve of, the project is bought. The rule stops when a project can be bought, but only by violating the budget constraint. This rule can be computed in polynomial time and was initially investigated by Brill et al. [28] within the concept of the multi-winner voting rule who found that it satisfies proportional justified representation, this rule was then later adapted to PB by Los et al. [26]. The proportional justified representation property was initially proposed by Sánchez-Fernández et al. [29]. This property implies the rule fairly distributes resources among voters based on their preferences and needs.

Following this is an adaptation of the previous Phragmén rule labelled the Maximin support rule and developed by Sánchez-Fernández et al. [30]. This rule allows a redistribution of the loads in each round and was introduced as a multi-winner voting rule by Sánchez-Fernández et al. Similarly, with the Phragmén rule, the rule was then later adapted to PB by Aziz et al. [31].

The final sequential rule to discuss is MES, this has already been discussed in detail within Chapter 1. This rule was first developed by Dominik Peters and Piotr Skowron [32] for multi-winner voting and then later adapted to PB by Peters et al. [12]. Comparable to the Phragmén rule, MES can also be computed in polynomial time and satisfy the proportional justified representation.

This section provides a brief overview on the various rules and the properties they satisfy, and as evident, there are voting rules which can resolve the criticisms of PB mentioned in the previous section, most notably the issue with proportional representation. However, these rules are still regrettably not being adopted into PB elections, one reason being the difficult nature of understanding these complex rules for both the general public and organisers of PB elections. One way to combat this is to employ the use of effective visualisations to convey how these voting rules work, which will be explored in the next section. Note that for additional reading on this topic, refer to Simon Rey and Jan Maly’s recent paper [33] where they provide an in-depth overview of all the different rules within PB, along with proofs showcasing the properties they satisfy.

2.3 Visualisation

The aim of this section is to explore justifications for visualisations (as opposed to just using text-based explanations) and the benefits it can provide, before ending it off by presenting key design choices that we can utilise to construct effective visualisations.

The use of visualisations has had a rapid expansion across the fields of various elections and social choice. This rise is due to the recognition of the benefits of visualisation, therefore, it becomes vital we discuss the justifications for including these visualisation. Firstly, a paper by Elena Long [34] described one important advantage of visualisation, which is its ability to enable humans to interpret and comprehend large amounts of data within a small amount of time. A further study by Paul Lester

[35] highlighted psychologist Jerome Bruner's findings, indicating that people only remember 10% and 20% of what they hear and read respectively, yet they remember 80% of what they see. These points combined together show why visualisations can be used as an effective tool.

Following this, we briefly discuss the key elements that should be incorporated within good visualisation design. Stephen Few [36] wrote in his book - Show Me the Numbers: Designing Tables and Graphs to Enlighten - the practical rules for using colour within charts. There are nine rules and the key ones we focus on are as follows:

1. If you want different objects of the same colour in a table or graph to look the same, make sure that the background is consistent.
2. If you want objects in a table or graph to be easily seen, use a background colour that contrasts sufficiently with the object.
3. Use different colours only when they correspond to differences of meaning in the data.
4. Use soft, natural colours to display most information and bright and/or dark colours to highlight information that requires greater attention.

In addition, Shneiderman in his paper [37] introduced a list of tasks that visualisations should support, these tasks led to what is now known as the Shneiderman mantra. This mantra showcases the key principles in producing effective visualisations and emphasises beginning with an overview, allowing for zooming and filtering, and then finally enabling details-on-demand. This methodology, as discussed by Elena Long [34], has been used for many years by web designers as a guiding framework to construct effective visualisation.

Through adhering to these key elements, we are able to construct more effective visualisations - ones that can simplify complex processes and allow individuals to quickly interpret large amounts of data. Note that these are just a few examples of

the benefits and justifications of visualisations, as well as what makes a visualisation effective. For additional reading on this topic, please refer to Stephen Few’s design book [36], Shneiderman’s paper [37] and Elena Long’s paper [34].

2.4 Existing Solutions

Having covered the justifications for why visualisations can be an effective tool, we now investigate existing solutions for visualising PB elections. However, given the recent emergence of this topic, solutions are regrettably few; despite this, there are still three solutions we can discuss.

2.4.1 Pabustats

The first known application that attempts to visualise PB elections comes from Faliszewski et al. [38] who also unveiled the Pabutools package and Pabulib library. This web-application is known as Pabustats [39] and is used to compare different voting rules in PB based on the data from Pabulib. The web-application features a straightforward text-based interface, enabling users to select a base rule and multiple other rules to compare it with. These rules include: Utilitarian Welfare, Phragmén and MES, each rule can additionally be considered in four variants. Upon selecting an election - whether by uploading their own or selecting from the provided list - and choosing the rules for comparison, the user is presented with a table displaying each candidate project. These projects are accompanied by a checkmark or dash to indicate whether that project was selected or rejected under each respective rule. In addition, general statistics of the election as well as the projects are presented, for example, cost, mean and standard deviation. While this page can be useful in comparing different rules, the analysis provides no visual information, or more importantly, it fails to give explanations for how these voting rules arrived at these outcomes. Further, since Pabustats features text-based visualisations, it can be less engaging for the user to use and retain the information provided. See Figure 2.1 for an example of how Pabustats displays the selected or rejected projects of each rule.

Pabulib: Artificial, Mechanical Turk, Knapsack_7

[← go back to form](#)

Information about the instance:

- Number of voters: 69
- Number of projects: 20
- Budget: 500 000

Projects table

- [click on a table header to sort the rows](#)
- [click here to go to the statistics below](#)

Subunit	Project Name	Cost	Score	Utilitarian Greedy (citywide, cost utilities)	Equal Shares (no completion) (citywide, cost utilities)	Equal Shares (no completion) (citywide, point utilities)
	Computers for the community learning center	27 000	48	✓	✓	✓
	Benches for a Walkable City	25 000	41	✓	✓	✓
	Real-Time Bus Arrival Monitors in bus stations	24 000	40	✓	✓	✓
	Separate Bike Lanes from Traffic	50 000	39	✓	✓	✓
	Little (book exchange) libraries	13 000	39	✓	✓	✓
	Fire Hydrant Markers	8 000	38	✓	✓	✓
	Planting trees in the city	119 400	37	✓	✓	–
	5 Water Bottle Refill Stations	40 000	36	✓	✓	✓
	Outdoor Fitness Equipment in the public park	65 000	33	✓	–	–
	Protect the Health and Safety of our Firefighters	165 000	32	–	–	–
	Urban Bicycle Wash Stations	20 000	21	✓	✓	✓
	New Playground for public school	200 000	14	–	–	–
	Invention and Production of Music	150 000	13	–	–	–
	New Chairs for Public Schools	190 000	13	–	–	–
	Upgraded Water Fountains for Public Schools	200 000	11	–	–	–
	Volleyball Court in the public park	61 000	9	–	–	–
	24H public toilet	320 000	9	–	–	–
	Digital Sign at City Hall in Multiple Languages	75 000	9	✓	–	–
	Inclusive Playground for All Kids	305 000	3	–	–	–
	Meeting Room Upgrade for libraries	250 000	3	–	–	–
GENERAL STATISTICS						
Total cost		466 400			326 400	272 000
Total utility		381			339	335
METRIC: SCORE UTILITY						
Mean:		5.52			4.91	4.86
Standard deviation:		2.64			2.41	2.44
Comparison with Utilitarian Greedy (citywide, cost utilities):		-			for 0.0% better, for 53.62% worse	for 0.0% better, for 62.32% worse
METRIC: COST UTILITY						
Mean:		219 417.39			178 547.83	145 608.7
Standard deviation:		111 238.45			96 294.08	78 341.68
Comparison with Utilitarian Greedy (citywide, cost utilities):		-			for 0.0% better, for 53.62% worse	for 0.0% better, for 62.32% worse
METRIC: POWER INEQUALITY						
Total distance:		0.44			0.39	0.46

Figure 2.1: Example Displaying the Output After Running the Pabustats Visualisation

2.4.2 Pref.tools

The next application we discuss is another interactive web-application known as Pref.tools [40]. Similar to Pabustats, this site is built using Pabutools and allows the user to compare different voting methods; however, unlike Pabustats, it is not designed to compare the outcomes of real elections. Instead, this page allows the user to interactively add voters and projects as well as assign voters to projects, and then compare how different voting rules select different outcomes. This is shown by a table where users can click on a cell to assign a voter to a project and observe the table below, which displays whether each project was selected or not under each re-

spective rule. On top of this, when compared to the text-based visualisations present within Pabustats, the page features clear and effective visualisations. Nonetheless, the site still fails to provide explanations for how these voting rules arrived at these outcomes, further, the site fails to load with medium-large elections such as those featuring 1,000+ voters or 100+ projects and struggles handling smaller elections such as those featuring 200+ voters or 20+ projects. See Figure 2.2 for a default view of the Pref.tools site which features a default example which users can modify.

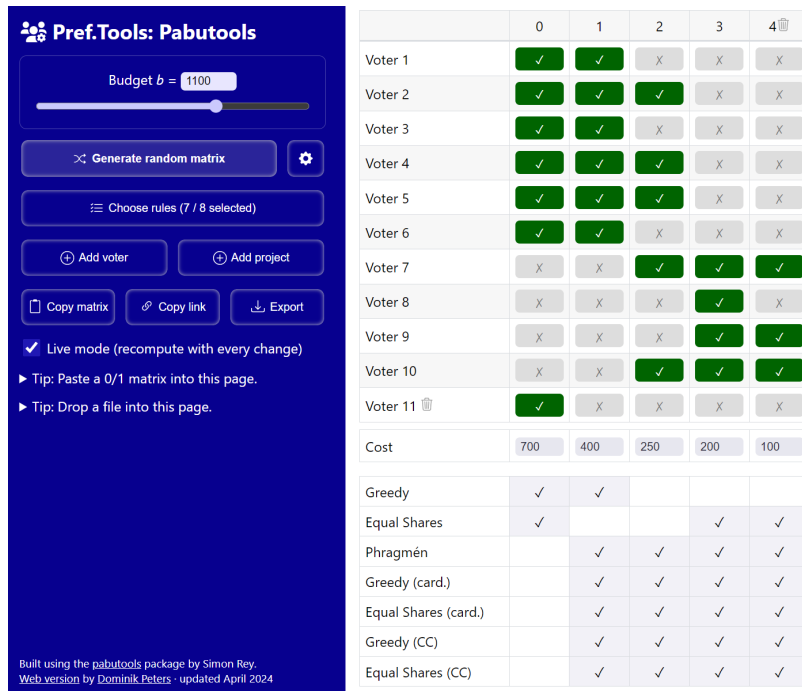


Figure 2.2: The Default View of the pref.tools Site Displaying an Example Election and Its Outcomes

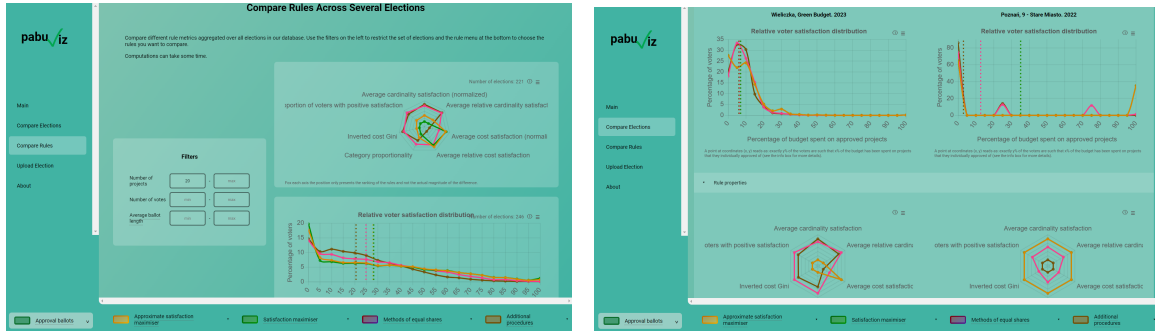
2.4.3 Pabuviz

The final application we discuss is the newest addition, known as Pabuviz [41]. Once again, similar to Pabustats, this page allows the user to compare different voting methods and akin to Pref.tools, features similar engaging visualisations. On the other hand, what separates this application apart is that it can also be used to compare outcomes from two different elections and showcase useful visualisations.

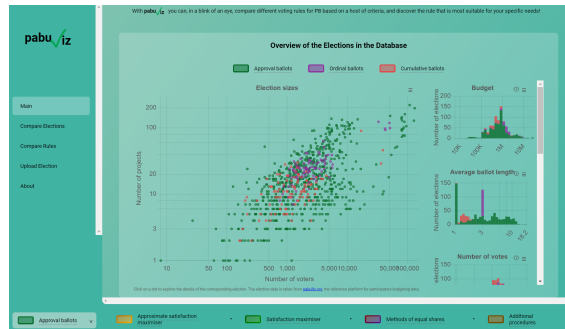
Various examples include a histogram displaying the distribution of the relative voter satisfaction and the average satisfaction chosen by different rules, this can be seen in Figure 2.3b. The purpose of this application is to provide tools for organisations of PB elections to make better-informed decisions, as they can quickly determine at a glance what rule might suit their election more. As a result, a lot of the terms used within the site are more suited to someone who understands PB very well and its various rules, such as an electoral officer. Further, as with the previous two sites, the site still does not provide explanations nor justify how these voting rules arrived at these outcomes. Therefore, a member of the general public may not fully grasp the data shown across the various visualisations. In addition, this site can only allow elections that have up to 20 projects and 5,000 voters to be uploaded, this is somewhat unsatisfactory as there are many elections that can feature up to 100 projects and 10,000 voters, or even higher in some cases that cannot be visualised using Pabuviz. See Figure 2.3 for various visualisations that Pabuviz provides.

2.4.4 Limitations of Existing Solutions

Whilst the solutions we have described above do offer promising approaches. They are not without limitations, for example, all three solutions fail at providing explanations for why a project got selected or rejected under any given voting rule. This is critical as members of the general public without any knowledge of PB will find it difficult to comprehend the complex voting rules without any accompanying explanation for the outcomes. This is further exacerbated by some of the solutions, most notably Pabuviz, where they make large use of research terms and a regrettably lack of definitions, which may make it difficult for the average user to comprehend the subtle nuances within each visualisation. Finally, Pref.tools and Pabuviz are unable to handle large elections, with the former site becoming unstable with large elections and the latter preventing users from uploading any elections that feature more than 5,000 voters and 20 projects.



(a) Pabuviz Page Comparing Rules Across Several Elections (b) Pabuviz Page Comparing Statistics of Two Elections Across Several Rules



(c) Main Page for Pabuviz Displaying the Overview of all the Elections in the Database

Figure 2.3: Example Pages From the Pabuviz Site

2.5 Conclusion of Literature Review

As is evident from the literature, there are voting rules which resolve the issues regarding the criticisms of PB. However, they are not typically utilised in elections, this can be attributed to the fact that the Greedy rule can be easier to understand due to its transparency and therefore, is often picked up by election organisers. As mentioned in Chapter 1, the Greedy rule is used by almost every city that conducts PB elections [11]. As a result, creating effective visualisations for PB elections aimed at the general public could help them understand rules such as MES more since the public is not likely to adopt a rule they do not understand. However, none of the existing solutions provide visualisations for voting rules, such as MES, aimed

at the general public (as explored in the previous section). For this reason, one of our primary aims in this project is to provide visualisations that not only explain the outcomes of different PB voting rules but also offer varying levels of explanation depth for each voting rule.

Chapter 3

Requirements

To ensure the success of the project, a set of requirements, aims, and objectives was created from the inception. These requirements ensure the product that is produced matches the customer's desires and achieves the broader aims and motivations we set out to achieve.

3.1 Objectives

Alongside the aims defined in Section 1.3, we have also established objectives we wish to achieve. They are more specific than our broader aims and define project-specific goals that this project aims to address. These objectives are SMART (Specific, Measurable, Achievable, Relevant, Time-bound) [42]. Each is time-bound by the project deadline - 30th April 2024 and must be completed by or before that point. The objectives are measured with a set of tests designed to numerically determine how well we achieved them. The objectives can be seen in Table 3.1.

Code	Objective
O1	Build a set of tools that visually explains the outcomes of different PB voting rules. This tool will display all relevant information required to understand how the final selection of projects has been made.
O2	This tool will improve the explainability of the PB voting methods, ensuring that with ten minutes of use on each PB voting rule, the user will be able to explain effectively in their own words how the rule works.
O3	The increased understanding of voting rules will ensure voters have complete confidence in the election results.
O4	The tool will be made publicly available via the Pabutools Python library.
O5	This tool will decrease the complexity of explanations by using methods such as abstractions and clustering to reduce the apparent quantity of data.
O6	The tools will offer varying levels of explanation depth for each voting rule. Since some elections will have a significant number of votes, simplicity is integral to ensuring the voters have a strong understanding of the underlying processes.

Table 3.1: Project Objectives

3.2 Requirements

Defining a set of requirements ensures the project has a work plan we can follow. We prioritise the changes we implement to ensure that we produce a minimum viable product (MVP) as quickly as possible. In this subsection, we present the initial requirements that we created for our project specification. Whilst we followed these closely throughout the project, some requirements were added, removed, or adapted depending on the views of the customer or the supervisor. Our requirements were prioritised using MoSCoW (Must, Should, Could, Won't). Ensuring we can strate-

gically plan what we do at which stage of the development. The customer approved the set of requirements before beginning project development.

3.2.1 Functional Requirements

First, we consider functional requirements. These “define what a product must do and what its features and functions are” [43]. The team developed the requirements based on meetings with the customer and supervisor. The meeting notes are shown in Figure B.3 and provide a basis for the requirements we have created in Table 3.2.

Code	Requirement	MoSCoW
R1	C1: Develop a set of tools to visualise outcomes of PB elections.	Must
R2	C2: The tool will be embedded into Pabutools. D2: Submit the changes to the tool as a pull request to the GitHub repository.	Must
R3	C3: The implementation of PB rules within Pabutools will need to be tweaked so they can provide explanations as well as the result. D3: Add a parameter to the existing PB rules implementation, which runs the explanation tool if selected.	Must
R4	C4: The modification made to the rules should not significantly modify the running times of rules when no explanation is requested. D4: Adapt PB rules to include an additional parameter which ensures any explanations are only run if this is True.	Must
R5	C5: A separate module will be created from scratch for the generation of the explanations. D5: Create a new set of Python classes to be integrated into the Pabutools library.	Must

R6	C6: The module will allow the user to provide an explained result as output by the Pabutools rules and will automatically generate visualisations for the explanation. D6: While the results of the PB rules are being calculated, generate visualisations as an HTML file.	Must
R7	C7: This tool will implement the visualisations for MES. D7: Initially, only adapt the Equal Shares implementation in Pabutools.	Must
R8	C8: This tool will implement the visualisations for Greedy Utilitarian Welfare, and Phragmen’s Voting Rules.	Should
R9	C9: This tool will implement the visualisations for the Welfare Maximisation method. This method is very complex; therefore, implementation of simple explanations may be very difficult. D9: Develop an abstraction of the Knapsack problem, to explain the problem to a non-technical user.	Could
R10	C10.1: Add visualisations to show the outcomes of elections and allocations of funds. D10.1: Include a separate module for the generation of visualisations of results. D10.2: As with R4 , this should not increase the runtime of the existing PB rule implementation.	Could
R11	C11: Visualisations will be interactive, allowing users to dynamically change parameters to see differences in real-time. D11: On top of the created HTML file, include additional features allowing for an interactive environment where the user can engage with visualisation features.	Could

R12	C12: The tool will include additional features allowing for the comparison between different voting rules. D12.1: Include a parameter allowing the user to select an additional voting rule to be computed. D12.2: Run the computation of the additional rule in parallel, ensuring minimal impact on running time.	Could
R13	C13: Include abstractions grouping similar voters into one to increase the simplicity of explanations. D13: Use various clustering methods to group voters into separate clusters.	Should

Table 3.2: Functional Requirements

3.2.2 Non Functional Requirements

Conversely, non-functional requirements “describe the general properties of a system” [43]. We include these since they provide additional clarity into the design of the system. Take, for example, requirement **R14**, defining how quickly the visualisations should run. Whilst this is not a feature we implement directly, it is a requirement we must achieve to make the user experience as seamless as possible. Table 3.3 shows each of these with the accompanying code and MoSCoW rating.

Code	Requirement	MoSCoW
R14	The running time of the PB rules with the explanations request should not exceed 300% of the running time of the initial rules without explanations.	Should
R15	The tools' runtime should scale proportionally to the number of voters and projects in the election, ensuring that when used in real-life elections with 100,000 voters, the tool works effectively.	Must
R16	The output of the explanation and accompanying visualisations will be intuitive to use and require no more than 5 minutes of explanation for a new user to learn how to use them.	Must
R17	All visualisations and interactive features should be accessible on both desktop and mobile devices.	Could
R18	All visualisations should represent the data accurately, with no errors or discrepancies.	Must
R19	The system should be designed to accommodate additional voting rules in future, with adaptation to the system being as minimal as possible.	Could
R20	The tool and its visualisations should be accessible to users with disabilities complying with relevant accessibility standards.	Must

Table 3.3: Non-Functional Requirements

As with our functional requirements, these were based off the meeting notes from the customer meeting, as seen in Figure B.3 within the Appendix.

3.3 Testing Objectives

We created a set of testing objectives in advance of project development. These objectives allow the team members to clearly understand the required quality and functional benchmarks. Since many of our objectives require user testing to ensure they have been achieved appropriately, we set a plan to measure the objectives against this. These testing objectives are shown in Figure 3.1, which are evaluated in Chapter 6.

- **TO1:** Verify a set of tools that are built to visualise explanations of outcomes for PB rules and ensure all relevant information required to understand the project selection is present.
- **TO2:** Evaluate if the tool improves the explainability of the PB voting methods. Ask the reader to explain the before and after the use of the tool after ten minutes of use.
- **TO3:** Assess if the voter trusts the election result with 100% confidence after using the tool on each of the PB rules.
- **TO4:** Assess if the tool has been successfully integrated into the Pabutools library.
- **TO5:** Assess if the tool decreases the complexity of the evaluation after using a range of abstraction methods.

Figure 3.1: Testing Objectives

Chapter 4

Design

In this chapter, we discuss the overall design of our project. There are many moving parts involved in our system, and therefore, we discuss each component separately before attempting to see how things fit together. We will first look into understanding the design of the web pages we want to generate. We will then look at our software stack and see what technologies and tools we want to use, and finally, we will go over the technical architecture of our solution; in particular, we will look at the different classes that will be created and how everything will be organised within the package. Given that our work contributes to the existing Pabutools open source library, we will also look at the current state of Pabutools and understand how it works.

4.1 Visualising Method of Equal Shares

4.1.1 Summary Page

The summary page provides users with a straightforward, easy-to-understand overview of the results of a given election calculated using the MES voting rule. This page is typically the first page users access before examining more details with the round-by-round page, as understanding can aid them in interpreting the detailed information more effectively. Further, this ensures that users will focus on the most critical aspects of the election results and can subsequently motivate them to explore the round-by-

round page to discover any additional details.

Page Design

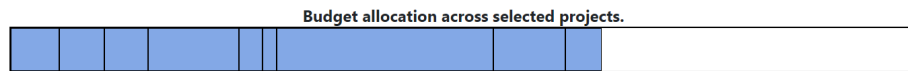
Having described the general idea of the summary page, we can now proceed with the design of the summary page. When making important design decisions for our pages, we kept in mind the 9 rules of visualisation listed in Election Data Visualisation (2013)[44]. Firstly, we include text at the top of the screen describing what the page is and what it displays. This helps the users understand the purpose of the page and the visualisations within them. Further, we include necessary features and details pertaining to the election itself, this includes the name of the election, number of participants, number of selected and rejected projects, total budget and budget spent.

Below this, the first visualisation appears presenting how the budget was allocated across selected projects in a chart. This is then followed by the second visualisation which displays a summary table where each row represents a round of the election, each row illustrates what project got selected or rejected in that round with details pertaining to the project such as, cost, effective vote count and funding lost. Moreover, each row can be expanded on to reveal a hidden section containing additional details of the specific project from the expanded row. After describing the visualisations used very briefly, we will now go through each visualisation in detail starting with the budget allocation chart.

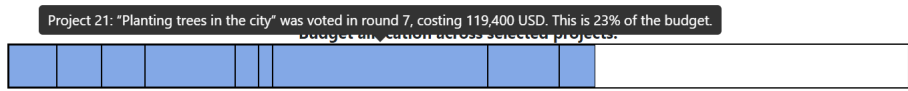
Budget Allocation Chart

This chart is used to present how an election's budget was distributed amongst the projects selected this election. The length of the chart represents the budget, and each bar represents the cost of the selected project. In addition to this, a tooltip is displayed when hovering the mouse over a bar, containing details on what the chart represents, as well as the project name and how much of the overall budget was deducted from the project cost. This chart enables the users to quickly identify how much of the overall budget was spent and the relative cost of the selected project. See

Figure 4.1 for an example of how the budget chart looks like in the summary page.



(a) Budget Chart Without Tooltip Displaying How the Budget Was Distributed Among Nine Selected Projects



(b) Budget Chart with a Tooltip Displaying the Project Name, Cost and How Much of the Overall Budget It Represents

Figure 4.1: Example of How the Budget Was Distributed Among Nine Selected Projects With and Without a Tooltip

Overview Table

Following the budget chart, an overview table is laid out where each row represents each round of the election and contains details of a specific project. For example, the project's name, cost and the number of votes it received are all provided. Further, the effective vote count is included which takes into account fractional votes in cases where projects receive fewer votes due to voters also supporting more popular alternatives. Moreover, a chart is included on the right-most column which displays how much the supporters of each project initially had at their disposal compared with what they actually have at the start of the round, further details on this chart is provided in a subsequent section. Finally, the projects that were selected are marked in green whereas the rejected projects are marked in white, allowing the user to easily distinguish between the two. See Figure 4.2 for an example of how the overview table is displayed within the summary page.

Round	ID	Project Name	Cost	Number of Votes	Effective Support	Chart	
>	1	3	Computers for the community learning center	27,000	48	48	
>	2	14	Benches for a Walkable City	25,000	41	41	
>	3	13	Real-Time Bus Arrival Monitors in bus stations	24,000	40	40	
>	4	12	Separate Bike Lanes from Traffic	50,000	39	39	
>	5	2	Little (book exchange) libraries	13,000	39	39	
>	6	31	Fire Hydrant Markers	8,000	38	38	
>	7	21	Planting trees in the city	119,400	37	37	
>	8	23	5 Water Bottle Refill Stations	40,000	36	29	
>		41	Protect the Health and Safety of our Firefighters	165,000	32	32	

Figure 4.2: Example of How the Full Overview Table Looks

In addition having a row for each project, the user is able to click on a row to expand it and reveal additional information pertaining to the project. This includes the project's description, categories and any other details. Furthermore, for projects that were selected, a direct link to the round-by-round page is provided, which sends the user to the specific round corresponding to the row. The addition of this hyperlink allows users to glimpse at a more thorough analysis of the specific round. When a user clicks on a row, it implies they are already looking for more information. Hence, it make sense to include the link within the row, as opposed to placing the link on the table itself.

Moreover, within the expanded row, we generate a dynamic explanation. This dynamic explanation provides an insight as to why a project was selected or rejected, using an appropriate amount of information. Finally, if applicable, we list all the projects that were also voted by supporters of the given project and how much funding was lost to each of them. See Figure 4.3 for an example of what the expanded row looks like for a project that was selected.

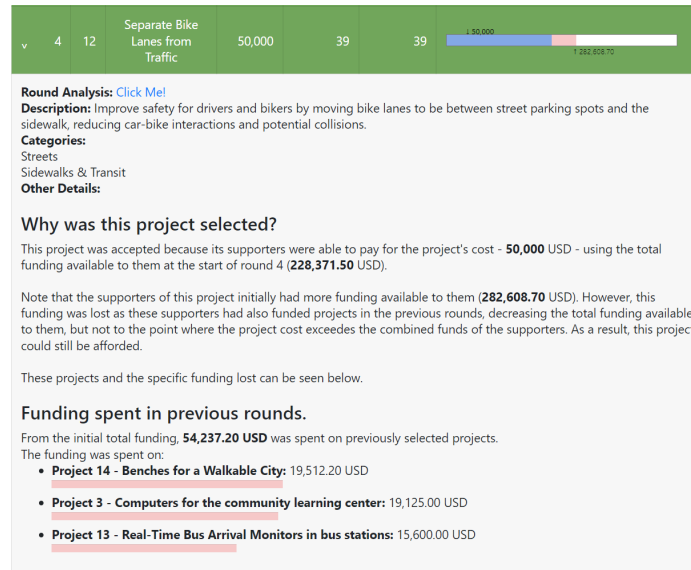


Figure 4.3: Expanding a Row in the Summary Table to Reveal a Hidden Row for a Selected Project

Finally, we can dive deeper into the chart as mentioned earlier, exploring its features and justification. The idea for this chart - along with designs for the un-expanded rows in the table - originally came from the customer who provided us with an HTML file example containing this chart. After playing around with it, we decided to incorporate it within the summary page. As previously stated, the general idea of the chart is to display how much the supporters of each project initially had at their disposal compared with what they actually have at the start of the round. More specifically, as observed in Figure 4.4, the amount marked as the lower arrow indicates the funding the supporters were initially entitled to, this can also be interpreted as the number of votes for a project multiplied with the amount distributed to each voter at the very start of the election. However, this amount can decrease as it becomes partially allocated to previously selected projects that these voters have also supported, the total amount used for previously selected projects is then represented by the pink bar within the chart. Therefore, the amount left over for voters or in other words, what they actually have at the start of the round is represented by the blue bar. Lastly, the amount marked as the upper arrow simply indicates the cost of the project. See Figure 4.4 for an example of how a chart looks like for a project that was rejected,

as can be observed, the amount for the blue bar is less than the cost of the project, therefore, the project was not selected. However, it's also worth mentioning in Figure 4.4 that the amount for the pink bar exceeds the cost of the project, this implies that supporters initially had enough to select the project but unfortunately lost too much funding from previously selected projects and as a result could no longer afford the project.

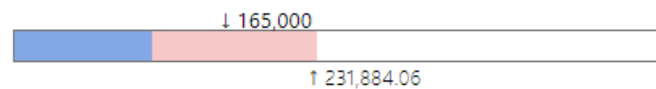


Figure 4.4: Summary Chart Comparing the Project's Actual Voter Funding with the Initial Voter Funding

In addition to the chart, ideas for the tooltips on the chart were also recovered from the previously-mentioned HTML file. They were incorporated into the page to provide users clarity on what each component of the chart represented and in the case of the pink bar, provide additional details about aspects of the round. As observed in Figure 4.5, upon hovering over the pink bar, information regarding the three previously-selected projects that were most popular with supporters of the project in consideration is shown. This includes the project's name, how much funding was spent on the project and a bar representing the funding spent. To avoid the tooltip being too bloated, only three projects was chosen to be shown. Displaying only three projects also allows the user to easily make comparisons between the most relevant projects in terms of funding lost, without being overwhelmed with information.

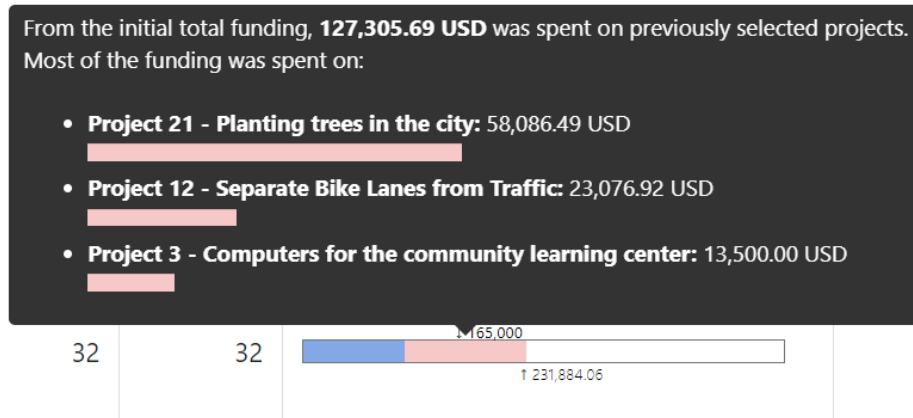


Figure 4.5: Summary Chart with a Tooltip Showcasing the Three Most Popular Previously Selected Projects That Supporters Also Voted For

There are a couple of reasons why we pushed to use this chart visualisation as opposed to simply text. One such reason is because it provides an intuitive way of conveying detailed information for the user. Another reason is that it can make it easier for a user to identify relationships between projects and spot at a glance if for example, a rejected project was close to being selected, or if a rejected project initially had enough funding to be selected but lost some of it due to previously selected projects. Further, the colours of blue and pink were used for this visualisation to clearly differentiate between funding available and funding lost, a darker colour such as blue contrasts with a lighter shade of pink to distinguish between those two elements - satisfying Few's third rule for using colour within charts mentioned within Section 2.3.

Finally, we aimed to keep high-level details about the chart within the tooltip to ensure the chart remains at a high level of abstraction. This is beneficial as it keeps the visualisation simpler and more intuitive for the user to understand at a glance, by ensuring that users who desire more information can access it simply by using the tooltips without overwhelming those who would prefer a simpler visualisation.

4.1.2 Round by Round Page

The round-by-round page is a comprehensive resource that allows the user to gain a deeper understanding of each stage of the election, why the project was elected at that round. The first page displays the project that was elected in the first round, while the visualisations provide further insights into why this project was elected and how it impacts the other projects in future rounds.

The page has navigation so the user can proceed through each round as the election unfolds. The page navigation comes in the form of a button to go to the next or previous page and a drop-down button to go to any of the rounds.

Round Overview

The round overview text includes information on the project that was elected at the current round, including the project name, description, and cost. Other information regarding the election, including the remaining budget and round number, is also displayed. Navigation between rounds can be reached from this part of the website to allow for easy and intuitive navigation for the user.

It was essential to include the project's name in the drop-down menu as well as the round number so that the user can identify a specific project of interest without iterating through each round. For example, if a user wanted to see at which stage a project they voted for was elected.

Round 1
Winner - Computers for the community learning
Description: Funding 20 laptops including mice and keyboards, giving students a place to study.
Project Cost: 27000
Election Budget Remaining: 473000.0

[Previous Round](#) [Next Round](#)

Figure 4.6: An Example Round Summary

Effective Vote Count

In each round of the MES voting rule, the project with the greatest effective vote count is selected. The cost of the project should be split as evenly as possible among the project voters. Each voter's share of the project cost is taken away from their budget, which can result in a change in the effective vote count of the remaining projects.

The effective vote count bar chart displays the effective vote count for each project remaining at a current round. The project with the greatest effective vote count in this graph is the project that was elected in this round by the definition of the MES voting rule. This graph is informative as it indicates which projects were close to being selected and the projects that are likely to be selected in the coming rounds should their effective vote count remain a similar value.

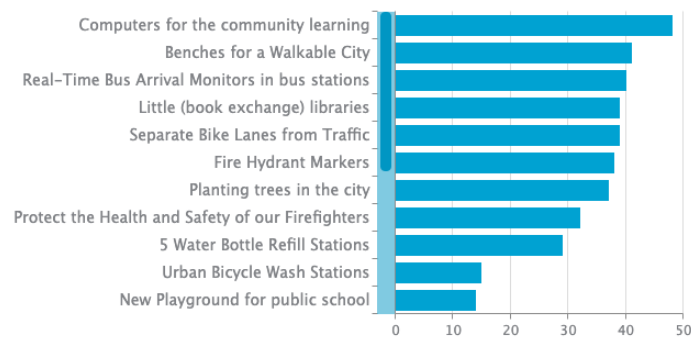


Figure 4.7: Example of an Effective Vote Count Bar Graph

Pie Charts

The voter flow for the project elected at the current stage is explored in the previously mentioned diagrams. Pie charts are utilised to see how voters for all other projects voted for the project elected at the current round. For each of the remaining projects that have not been rejected, the proportion of how many people voted for the current project is also displayed. This allows the user to get a complete picture of the relationship between voter flows. Each pie chart is accompanied by dynamically

generated text describing the information contained within the pie chart. The text also explains the average budget decrease of the voters who voted for this project after the project at the current round has been elected.

Reduced Effective Vote Count

Similar to the effective vote count bar graph, the reduced effective vote count bar graph explores the effective vote count of each project after the current round has been elected. This is necessary as the effective vote count for each project changes round to round as the budget a voter had previously pledged to a project has been spent on another project.

The graph displays a bar for each of the remaining projects. A proportion of the bar represents the new effective vote count (blue), and the other proportion of the bar represents how much the effective vote count for that project has been reduced compared to the previous round (red). The combination of the two proportions sums up to the effective vote count of the project before the project at the current round has been elected.

A key advantage of this graph is its ability to highlight the projects that have been most affected by the project at the current round. This is evident in the bars that show the largest proportion of reduced effective vote count.

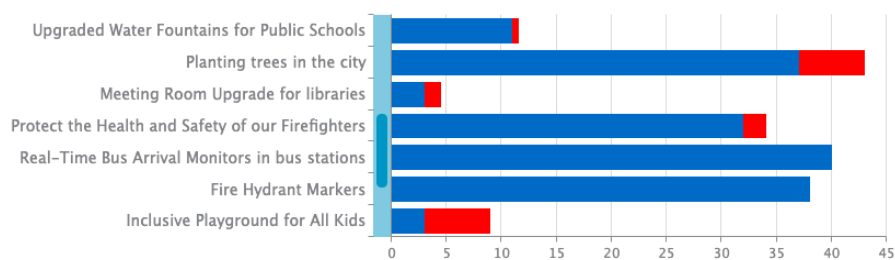


Figure 4.8: Example of a Reduced Effective Vote Count Bar Graph

Figure 4.8 gives an example of a reduced effective vote count bar chart. The red proportion of the bar represents how much the project's effective vote count has decreased in the current round. In Figure 4.8, we see that before the round, 'planting trees in the city' had the highest effective vote count; after calculating the new effective vote count for each of the projects, 'Real-Time Bus Arrival Monitors in bus stations' had the highest effective vote count. Therefore, in the next round, if all remaining projects were represented in the bar graph, then this project would be elected in the next round.

Flow Diagrams

As mentioned in the introduction section, the effective vote count of a project is equal to the sum of its supporters' individual effective vote counts. As previously stated, if a participant cannot contribute an equal share of the project, then their respective effective vote count is reduced, affecting the effective vote count of the entire project. The flow diagrams show the voters' relationships between projects. This is useful as it demonstrates how electing a project will likely affect another project. Consider electing project A for the first round; each supporter spends nearly their entire virtual budget. It is crucial to analyse the other projects these participants voted for as it is likely that they will no longer be able to fund their equal share of the remaining projects they voted for, reducing the effective vote count of these projects, hence making them less likely to be elected in following round. The flow diagrams capture the voters' relationship between projects. The two flow diagrams we use are the Sankey diagram and the chord diagram, which are explained in detail in the following section.

Example

We will discuss an example to highlight the importance of flow diagrams. Consider an election with six participants. Participants *A*, *B*, *C*, and *D* voted for Project *X*, which cost £80, and participants *B*, *C*, *D*, and *E* voted for Project *Y*, which cost

£40. In the first round of the election, project X is chosen as it has the highest effective vote count, as seen in Figure 4.9; project X has an effective vote count of four as the equal share of the project is within all four participants budget. The effects of electing project X on project Y can be seen in Figure 4.10. Figure 4.10a calculates the effective vote count before project X was elected, and Figure 4.10b shows the effective vote count after project X is calculated. The vote counts are 4 and 1.6, respectively. We see a significant decrease in the effective vote count after project X has been elected. This is because they share a large proportion of voters. This relation is captured in the Sankey diagram in Figure 4.12 where the majority of the participants who voted for project X voted for project Y . Figure 4.11 shows how the effective vote count was unaffected after electing project X as they share no common voters. This example shows the importance of capturing the flows and relations between projects. We dive deeper into the type of flow diagrams used in the following section.

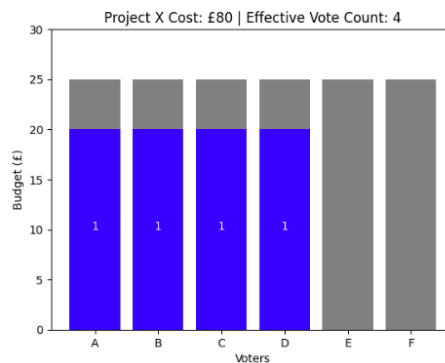
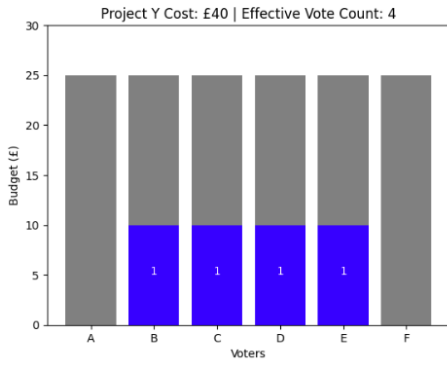
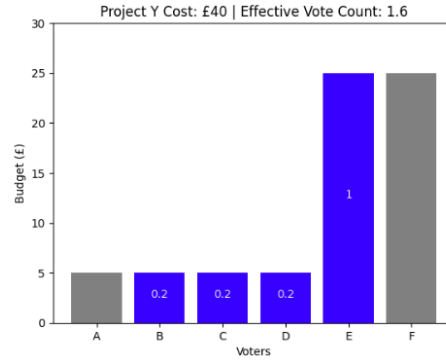


Figure 4.9: Effective Vote Count for Project X

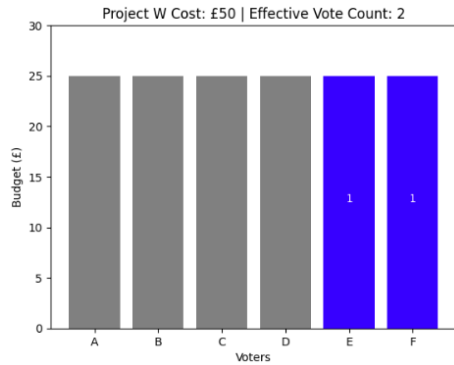


(a) Effective Vote Count for Project Y Before Electing Project X

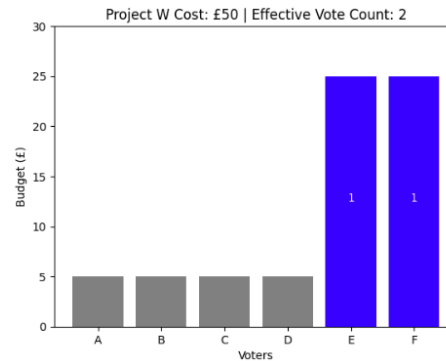


(b) Effective Vote Count for Project Y After Electing Project X

Figure 4.10: Demonstrating the Difference in Effective Vote Count for Project Y Before and After Electing Project X



(a) Effective Vote Count for Project W Before Electing Project X



(b) Effective Vote Count for Project W After Electing Project X

Figure 4.11: Demonstrating the Difference in Effective Vote Count for Project W Before and After Electing Project X



Figure 4.12: A Flow Diagram Visualising the Relation Between Project X and the Other Projects Participants Voted For

Sankey Diagram

A Sankey diagram was chosen to visually represent voting patterns among participants who could vote for multiple projects. A Sankey diagram is a flow diagram characterised by using thick arrows or bands to show the flow volumes between different nodes. The width of the arrows or bands is proportional to the flow quantity they represent, effectively visualising the distribution and transfer of a specific quantity among various parts of a system [45].

On the interactive round-by-round page, a dynamic Sankey diagram is used to show how voters for the project elected at the current round (left node) vote for the other projects (right nodes). This feature allows the user to explore the voting patterns in real time, enhancing their understanding of the data. The individual bands flowing from the singular left node to various nodes on the right-hand side illustrate the distribution of voters who have also voted for other projects. Each band's thickness corresponds to the number of voters who voted for both the project on the left and the project on the right. The graph is designed to show the overlap in voting behaviour, making it clear which projects share a common voter base with the project elected at the current round. An example Sankey diagram can be seen in Figure 4.13 visualising how voters of 'Computers for community learning' voted for other projects. The diagram provides visual clarity in flow representation, illustrating flows and their quantities in proportion to their size.

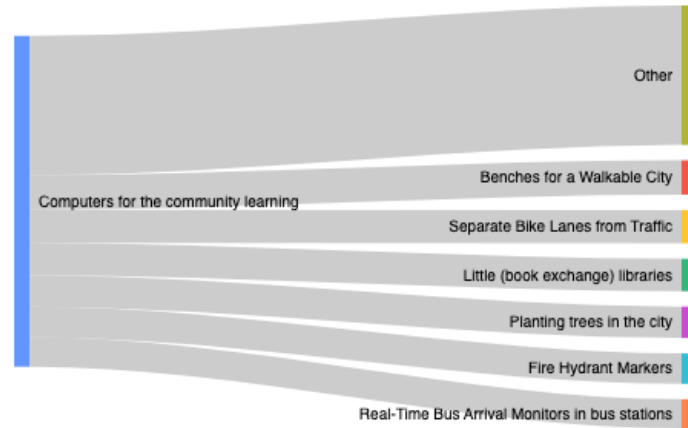


Figure 4.13: Example of a Sankey Diagram

Chord Diagram

Building upon the concepts explored in the Sankey diagram, a chord diagram was used to show the voter flows between multiple projects. A chord diagram shows the flows or connections between several entities (nodes) [46]. An arc along the circumference of a circle represents each entity. Chord diagrams are helpful in showing relationships between entities and their relative magnitudes compared to alternative arcs.

As explored in the Sankey diagram, we have seen the one-way relationship in how people who voted for the project were elected at the current round and elected for other projects. The chord diagram builds upon this by exploring how the voters of the other projects voted.

Figure 4.14 shows how voters for seven different projects voted. Chords between the arcs represent participants who voted for both projects.

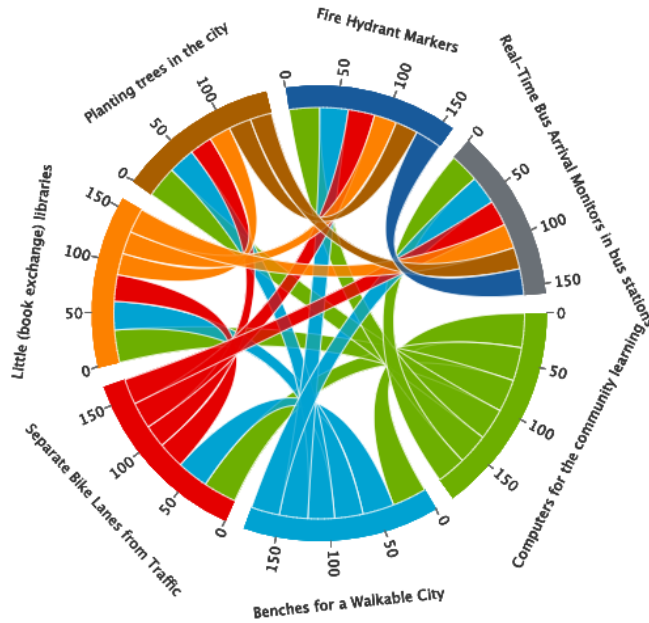


Figure 4.14: Example of a Chord Diagram

4.2 Visualising Greedy Utilitarian Welfare

Unlike with visualising MES, due to the simple nature of the Greedy rule, a single page was used to provide users with both a straight-forward overview of the results and a brief analysis of each round.

4.2.1 Page Design

The design of the page follows on from both pages of MES by first introducing the user to text at the top of the screen describing what the page is and what it displays. This includes the name of the election, number of participants, total budget and budget spent. As mentioned earlier with the summary page of MES, this is to aid the user in understanding the visualisations within the page, as well as the purpose of the page as a whole.

Following this, the budget allocation chart from the summary page appears as the first visualisation. This was included in this page as well as the summary page for MES to provide consistency for users who go to visit both pages, giving a sense of

similarity for the user and having to avoid to learn an entirely new visualisation. For a more detailed discussion on this chart, refer to Section 4.1.1 as the design decisions and justification have already been explained within the summary page of MES.

Following the budget allocation chart, the second visualisation emerges and displays a bar chart which showcases the satisfaction measures for each project. The satisfaction measure of a project defines how important a project is to the election voters as a whole (usually, this corresponds to the number of votes a project receives). Below this, a round overview visualisation appears which displays the remaining budget, selected project and any rejected projects in that specific round, the user is able to click through each round using the buttons provided. For the final visualisation, another satisfaction bar chart is displayed except each bar is now coloured by acceptance status, giving an overview of the outcome of the election.

The decision not to include the summary table from the MES page was intended, this is because many of the visualisations and explanations present within the summary page are not necessary for the Greedy page. For example, rejected projects do not require their own explanations when using the Greedy rule, as a result, this means further abstraction of the summary table is possible. Therefore, a round overview and the final satisfactory measure chart are included as replacements. The round overview can handle both accepted and rejected projects in a round, and the final chart handles giving an overview of the election. After describing the visualisations used, the subsequent sections will describe each visualisation mentioned in detail, starting with the first satisfaction bar chart.

4.2.2 Satisfaction Measure Chart

This chart is used to display the satisfaction measures for each project, informing the user on what projects were the most important to the voters in the election. The projects are displayed as horizontal bar charts as it can accommodate longer labels and allow for easier comparison between projects. Further, the user is able to

utilise the drop-downs provided to filter the bar chart by selected or rejected projects, and sort by name, cost or satisfaction score. The usage of filtering and sorting were included to allow the user to focus on the information that is relevant to them and be able to explore the data more efficiently in the case of very large elections. In addition, it is one of the key principles present in Shneiderman’s mantra as described in Section 2.3, to allow for filtering and data manipulation. Finally, the chart is accompanied with text describing what the visualisation is and a floating info-button, in which users are able to hover to receive additional details regarding the visualisation. See Figure 4.15 for an example of how this chart is displayed on the Greedy page.

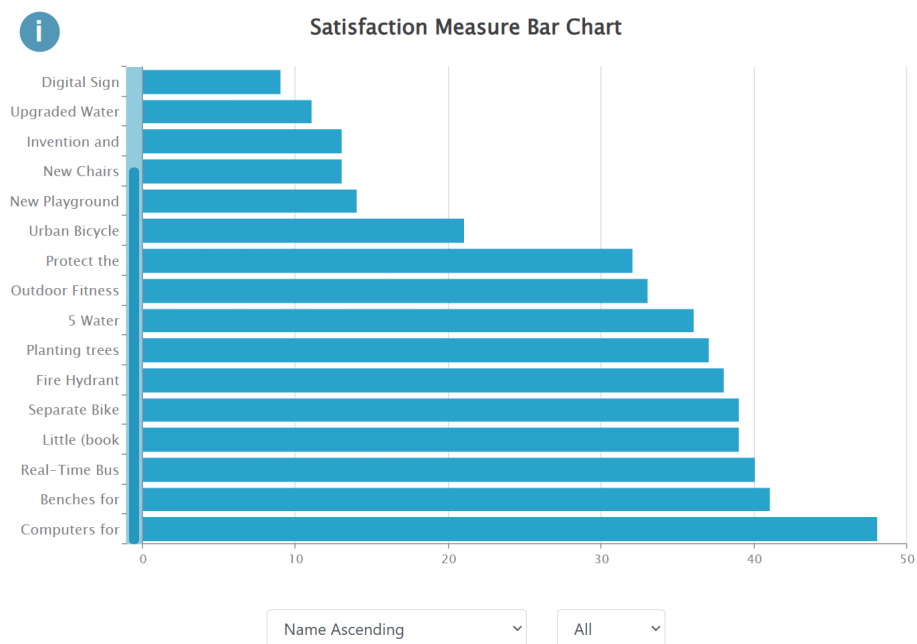


Figure 4.15: An Example of How the Satisfaction Measure Bar Chart is Displayed on the Greedy Page

4.2.3 Round Overview

Within this section of the page, a visualisation delving into each round is displayed. This visualisation allows the user to click through 'round by round' of the election and at each round displays the remaining budget, and the projects with the highest satisfaction down to the selected project. The selected project is shown in green,

any any rejected projects are shown in red accompanied with a cross. A project will be shown in red if it is has the next highest satisfaction score, however, there is not enough budget remaining to select the project. As a result, the next project with the highest satisfaction score is considered - and so on until a project is found that can be selected with the remaining budget or until no more projects can be considered. The user is able to use the previous and next round buttons provided to move through the different rounds of the election. This chart is useful as it can inform the user what project got rejected in each round and how close it was to being selected, which is done in a straight-forward manner for a user as they can simply examine the bar containing their project and compare with the top bar containing the remaining budget. See Figure 4.16 for an example of how this analysis is displayed on the Greedy page.

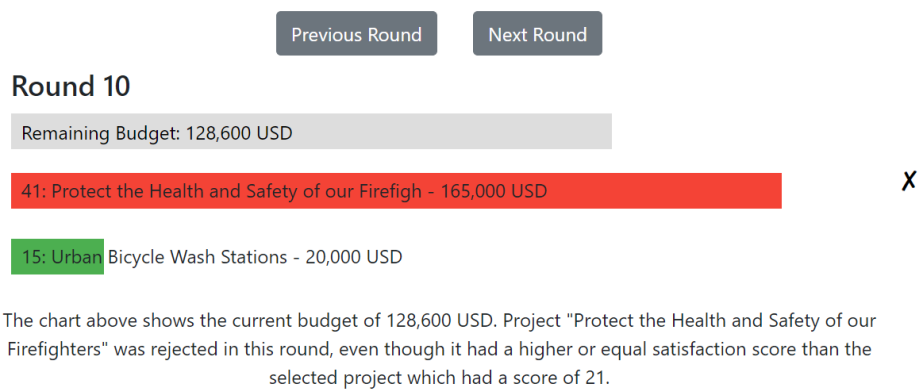


Figure 4.16: An Example of How the Round Analysis Visualisation is Displayed on the Greedy Page

4.2.4 Satisfaction Measure Chart Coloured by Acceptance Status

The final visualisation shown in the Greedy page is a copy of the satisfaction measure chart from the beginning of the page. However, the bars are now colour coded to display what project got selected with red indicating a rejected project and green indicating a selected project. Similarly, with the previous chart, an explanation is

provided as well as a floating info-button to reveal additional details about the visualisation. This chart is included to provide a quick overview for the users to illustrate that it is not always the most popular project which is selected, but the most popular projects within budget. See Figure 4.17 for an example of how this final visualisation is displayed on the Greedy page.

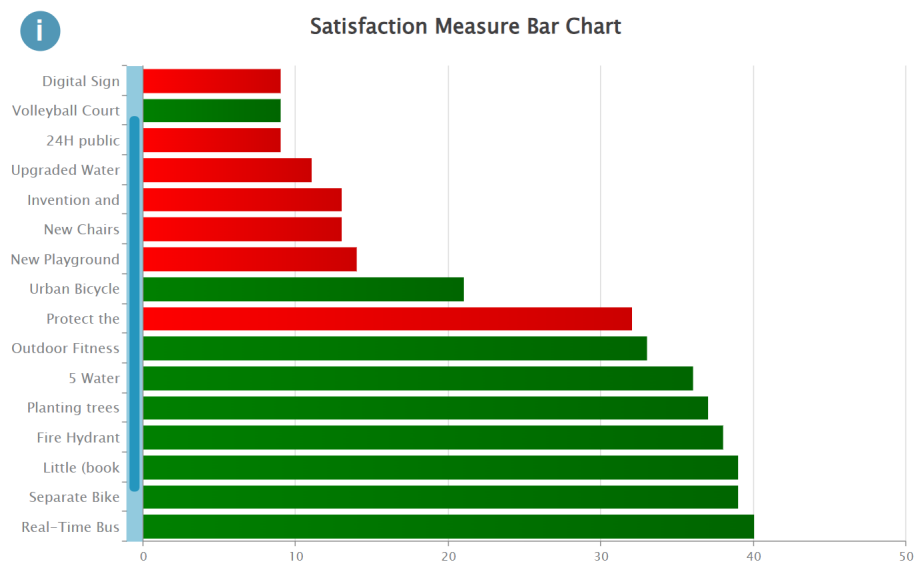


Figure 4.17: An Example of How the Satisfaction Measure Bar Chart Coloured by Acceptance Status Is Displayed on the Greedy Page

4.3 Software Stack

Jinja

Jinja is a powerful templating engine widely used within Python, primarily used to generate dynamic HTML content for web applications. It facilitates the seamless integration of data into HTML templates, enabling developers to create flexible, readable, and maintainable code. Key features of Jinja include:

- **Template Inheritance:** Jinja allows the development of a base template that holds the common layout and structure of the website. Specific blocks within

this base template can be overridden by child templates, promoting reuse and consistency across web pages.

- **Control Structures:** Incorporating loops and conditional statements within templates is straightforward in Jinja, enabling the dynamic generation of web pages based on varying conditions and data sets.
- **Optimised Performance:** Jinja converts templates into optimised Python code, which can significantly enhance performance and is especially important in high-load environments.

Jinja makes it possible to render web pages with complex, dynamic content efficiently and with minimal code. Figure 4.18 demonstrates a basic example of a Jinja template used to generate an HTML page. In this template, variables such as '`{{ page_title }}`', '`{{ heading }}`', and '`{{ user_name }}`' are placeholders that Jinja replaces with actual data when the template is rendered. Control structures such as '`{% if %}`' and '`{% for %}`' allow the template to dynamically alter the content based on the provided data, making web pages more interactive and user-responsive. With respect to the solution, standard HTML templates will be created for each of the visualisation pages. Relevant information specific to each individual election will be parsed into the template to dynamically render a web page unique to the queried PB election.

```
<html>
<head>
  <title>{{ page_title }}</title>
</head>
<body>
  <h1>{{ heading }}</h1>
  {% if user_logged_in %}
    <p>Welcome back, {{ user_name }}!</p>
  {% else %}
    <p>Please log in to see this page.</p>
  {% endif %}
  <ul>
    {% for item in item_list %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
</body>
</html>
```

Figure 4.18: A Basic Example of a Jinja Template Used to Generate a HTML Page

ZingChart

ZingChart is a powerful and versatile JavaScript library designed for creating interactive and responsive charts and visualisations [47]. It is widely used in web development to render data visually in various formats, including bar charts, pie charts, scatter plots, and more. The library supports HTML5 and can be easily integrated into web applications, providing developers with a robust toolset to create highly customisable charts.

ZingChart was chosen as one of the main visualisation libraries due to its wide variety of chart types and high level of customisability and styling. It supports the specific chart types that aid the round-by-round analysis of the MES voting rule, including chord diagrams.

Google Charts

Google Charts is a versatile web-based toolset to create interactive, rich charts and data visualisations within web applications [47]. ZingChart was chosen as the primary toolset for creating interactive graphs; however, some graphics were absent in ZingChart, such as the Sankey diagram. Therefore, Google Charts was chosen as the JavaScript library to create the diagrams that ZingChart could not.

Bootstrap

Bootstrap is an open source front-end framework that simplifies the design and development of web pages and applications [48]. Its core features include a responsive grid system, pre-designed UI elements (like buttons, forms, dropdowns, and alerts), and JavaScript plugins that add dynamic behaviour to static HTML pages.

Bootstrap was used for its ease of use with ready-to-use components, which enabled rapid development. Developers can easily incorporate complex UI elements without the need to write extensive CSS and JavaScript code from scratch. While Bootstrap

offers a default theme, it is very adaptable, allowing developers to override existing styles to create a custom look that fits the specific needs of their projects [48].

4.4 Architecture Overview

4.4.1 Pabutools

As discussed previously, Pabutools is a Python library that allows us to simulate the outcome of different elections using various supported voting rules. It takes an Object-Oriented approach to this problem and as such implements many classes across many files and heavily utilises features such as inheritance and composition to model and evaluate elections. These include classes to model entities such as Voters, Projects, Ballots etc as classes as well as a range of sub-classes for more specialised instance. However, interestingly the different voting rules are not modelled using a class hierarchy but are instead implemented as a series of different functions, although different voting rules functions do tend to share a common interface, no rules were programmatically enforced.

We have created a diagram to visualise this common interface for voting rules in Figure 4.19, it is important to note that not all classes are shown in this diagram but only the subset that are directly passed into a voting rule. There are also exist sub-classes (as well as abstract base classes) for the classes that are shown in the diagram, however we have omitted any inheritance relationships for the sake of simplicity.

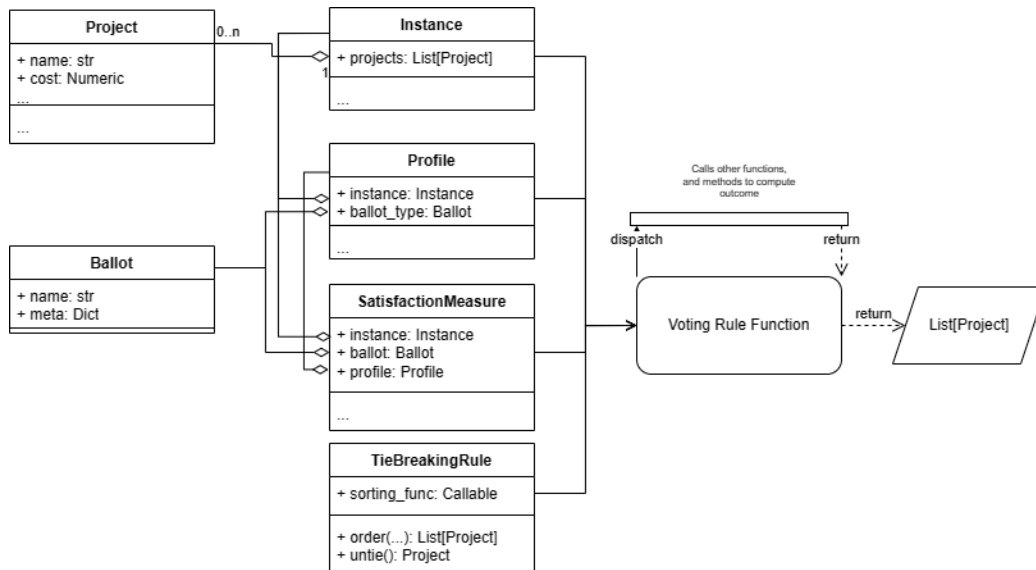


Figure 4.19: A Partial UML Diagram for Subset of Relevant Classes and a Generalisation of How It Connects to Voting Rule Function (Before Our Changes)

Below is a summary of all the main classes (from Figure 4.19) within Pabutools:

- **Instance:** An **Instance** contains the projects that are voted on, together with other information about the election such as the budget limit. Importantly, the ballots (votes) submitted by the voters is not part of the instance.
- **Project:** A **Project** represents projects that are voted upon by the voters. It does not store any data related to votes but gives you access to data such as cost and other metadata parsed from the .pb file
- **Ballot:** A **Ballot** is the class that represents an individual voter and stores their votes/scores for the different projects. It also stores general metadata related to specific voters, e.g. gender, location, etc. There are many different sub-classes of **Ballot** for all the different types of voting schemes. These include: **CardinalBallot** (voter assigns scores to projects), **CumulativeBallot** (voter distributes a given amount of points to the projects), **OrdinalBallot** (voter orders some projects according to their preferences), and **ApprovalBallot** (voter indicates the projects that they approve of). Approval voting is the most com-

mon and simplest voting scheme used in PB elections and as such our visualisations specifically target these type of elections.

- **Profile:** A `Profile` is simply a collection of ballots that all correspond to one ballot type. As such there is a `Profile` subclass for each of the ballot types we have listed above (e.g. `CardinalProfile`, `ApprovalProfile`, etc).
- **SatisfactionMeasure:** `SatisfactionMeasure` is a class representing the satisfaction score of a particular `Ballot`. There are many different types of satisfaction functions that all compute satisfaction in different ways which are all implemented across various sub-classes
- **TieBreakingRule:** A `TieBreakingRule` implements different tie breaking schemes that can break ties amongst projects. It takes a sorting function within its constructor which is what it uses to determine how to reorder the projects. There are a few different tie breaking rules provided by default in `Pabutools`, such as: `lexico_tie_breaking` (project name ordered alphabetically) or `max_cost_tie_breaking` (ties are broken in favour of the project with the highest cost).

As mentioned before, all voting rules tend to follow a similar pattern to the diagram shown in Figure 4.19. However there are differences, in particular with MES, as it involves additional classes that abstract some additional complexity that is involved in computing the outcomes of MES. These involve the construction of two new classes `MESVoter` and `MESProject`, see Figure 4.20.

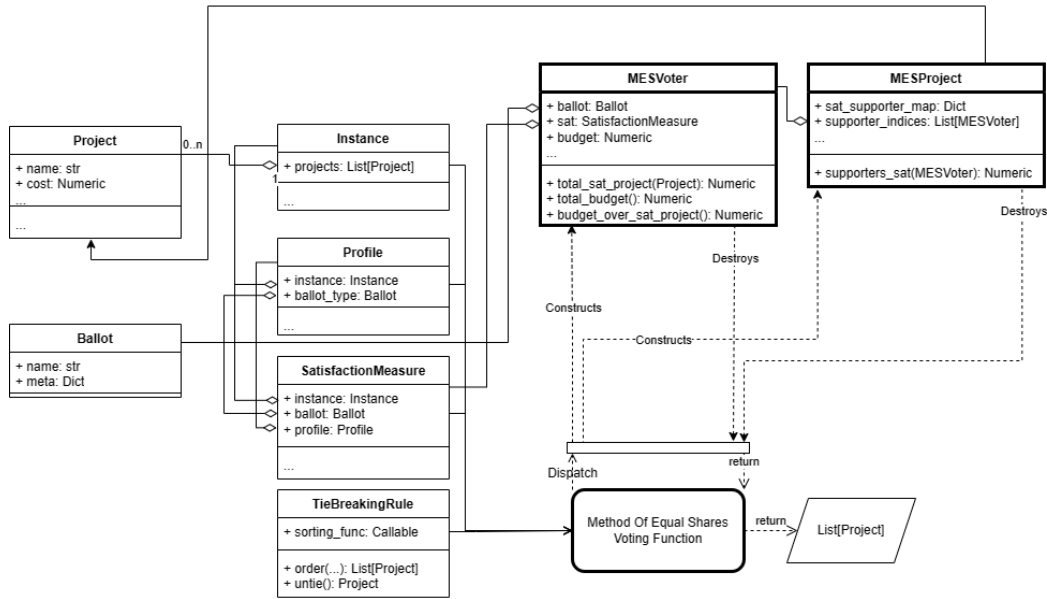


Figure 4.20: A Partial UML Diagram for Subset of Relevant Classes and How It Connects to the MES Voting Rule Function (Before Our Changes)

4.4.2 Data Capture

In the previous section we have seen how the main classes within Pabutools are related, and specifically how they connect to a voting rule function. One of our major tasks is to implement a way to store/retrieve the data to perform analysis and generate our visualisations. Some of this data can be retrieved directly from the Project, Instance and Profile objects that were initialised using data directly from the `.pb` file (so we also avoid having to parse the `.pb` files ourselves). Data from these objects will be easy to retrieve and do not require any significant effort. However, the key data to retrieve for our visualisations is data that is computed during the calculation of our voting rule. There are many ways we could achieve this; the simplest would be to recompute using data from the same objects that are passed into the voting rule, however this would result in duplicate code and also would not meet the customer requirement of being easily extendable to all supported (and any future) voting rules. It has to be designed in a way that is general, so that it can easily fit into any voting rule but is also specific enough, such that we can store data that is specific for a particular voting rule. Since such a solution will involve modifying the existing voting

rule functions, we also have to ensure that we minimise any impact on performance.

Together with the customer, we explored two potential methods in which we could achieve this. One method was to create a new object to store details and pass that as an argument into a voting rule function. This argument would be optional, and the function could then call methods on this object to store any required data. Since the caller of the voting rule would be responsible for initialising this object, after the function finishes computing we would have access to this object and can use the data to generate our visualisations.

```
def voting_rule(..., details=None):
    ...
    details.store(x)
    ...
    details.store(y)
    ...
details = Details()
outcome = voting_rule(..., details)
# Use 'details' to generate visualisations
```

The alternative was to modify the return value of voting rules from a list of projects to a new wrapper class that also has the option to store details.

```
def voting_rule(...):
    outcome = Outcome()
    ...
    outcome.details.store(x)
    ...
    outcome.details.store(y)
    ...
    return outcome
outcome = voting_rule(...)
# Use 'outcome.details' to generate visualisations
```

The customer preferred the latter mostly because it involved less significant changes to the existing API, and a change similar to this had already been planned as part of

future refactoring. In Figure 4.21, we can see the new classes that have been created to implement the proposed solution.

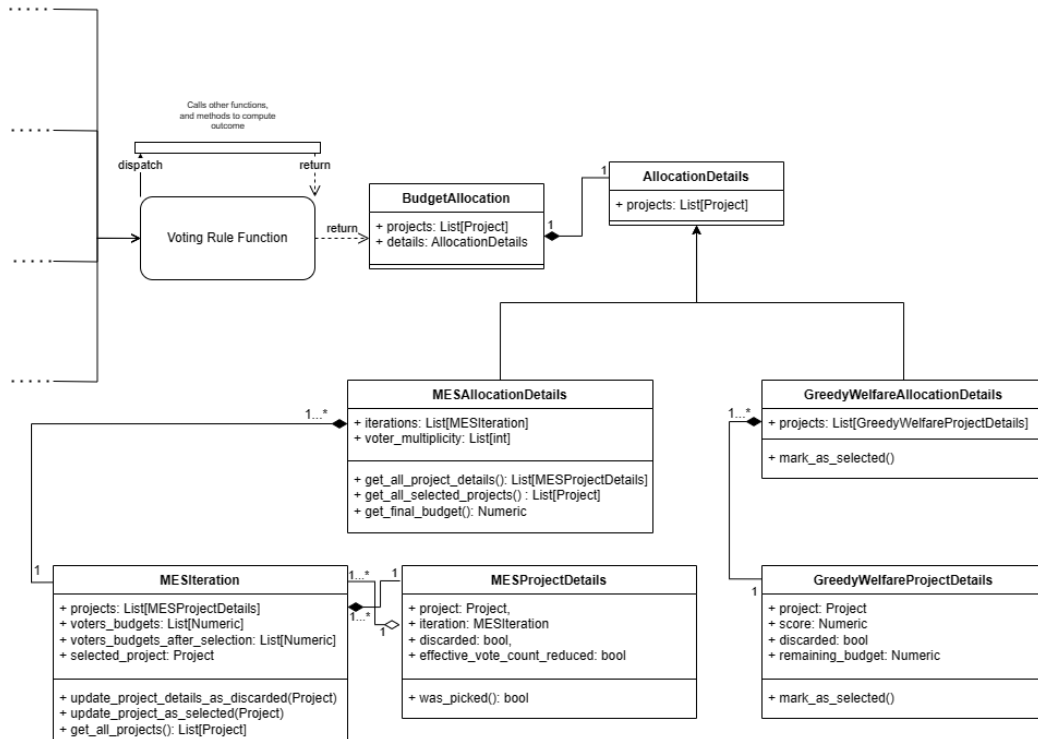


Figure 4.21: A UML Diagram for the New Object Returned by Voting Rules Enabling Us to Store and Retrieve Data From Voting Rules (previously, a Voting Rule Would Return Just a List of Projects)

This design went through numerous iterations, evolving alongside the changing requirements of our visualisations until we arrived at this version. Below, we briefly discuss each of these new classes.

- **BudgetAllocation:** A budget allocation is the new return value for voting rules (replacing what was previously just a list of projects). It stores a list of projects as well as additional information (via a `AllocationDetails` object) for explanation purposes.
- **AllocationDetails:** This is used to store additional information related to a specific run of a voting rule. It is a property of `BudgetAllocation` and exposes

methods that allow the voting rule to store data. Currently, there are two supported voting rules: `MESAllocationDetails` and `GreedyWelfareAllocationDetails`. Both have different sets of methods and store their details in different ways (See Figure 4.21 for list of methods).

- **MESIteration:** The complexity of the MES rule meant it made sense to create a further abstraction to store data related to an execution of MES. This class corresponds to one iteration of MES (i.e. one project being selected), the main idea behind this is that we should be able to entirely reconstruct an entire run of MES using all the iterations. Other properties, such as voter budgets are also stored as within this class. A list of `MESIterations` is stored as part of the `MESAllocationDetails`
- **MESProjectDetails:** Within each `MESIteration`, there are also multiple `MESProjectDetails`; one for each project that was considered during this iteration. They will be marked as either discarded or selected, and also store other information about the project within a specific iteration. In addition, it is important to note that the same project can appear in different iterations with different instances of `MESProjectDetails`.
- **GreedyWelfareProjectDetails:** This stores information about the project related to a Greedy run. A list of `GreedyWelfareProjectDetails` is part of one `GreedyWelfareAllocationDetails`. There is a one to one correspondence with projects in the election and instances of this class (unlike `MESProjectDetails`).

4.4.3 Analysis

After capturing and storing the data from the run of our voting rule, our next step would be to develop analysis functions that can use this data (as well as data that was already stored directly as part of objects initialised during the parsing of the .pb files) to calculate more useful metrics that can form part of our visualisations and explanations. Some of our analysis functions were implemented as 'standalone'

functions within an analysis submodule (following the existing convention), however some were directly part of our Visualisation class (we discuss this in detail in the next section). Analysis that is more specific to our visualisations we kept within the Visualisation class and ones we felt had more use even outside the context of visualisations we added to the analysis submodule.

One such analysis function was computing voter flows, a two-dimensional dictionary where $\text{flow}[a][b]$ is the number of voters for 'a' who also voted for 'b'. This dictionary can then be used in Sankey diagrams and chord diagrams. These graphs are designed to show the overlap in voting behaviour, making it clear which projects share a common voter base with the project elected at the current round. The pseudo-code for this algorithm can be seen below. In an election with n voters and m projects, the worst-case run time would be $O(nm^2)$, where each participant votes for all projects.

```
for each vote in profile do
    for i from 0 to length(vote) - 1 do
        for j from i + 1 to length(vote) - 1 do
            flow[vote[i]][vote[j]] = flow[vote[i]][vote[j]] + 1
            flow[vote[j]][vote[i]] = flow[vote[j]][vote[i]] + 1
```

4.4.4 Generating web pages

The next big component of our changes to Pabutools would be implementing the API that would enable users to generate the visualisations. When considering the design of this API, we wanted it to be simple to use and not require many additional steps on top of already running the election. Below is an extract from our usage guide on how we can generate visualisations.

```
# from pabutools ...
from pabutools.visualisation.visualisation import MESVisualiser

instance, profile = election.parse_pabulib("sample_election.pb")
outcome = method_of_equal_shares(instance, profile, sat_class=
    Cost_Sat, \
```

```

analytics=True)
vis = MESVisualiser(profile, instance, outcome)
vis.render("./output_folder", name="demo")

```

Although, the above example is for MES - it works in exactly the same way for Greedy Welfare but we would instead use the `GreedyVisualiser` and pass in the outcome returned by running the Greedy rule. If we look at Figure 4.22, we can see how all the classes are related and how such an API could be implemented.

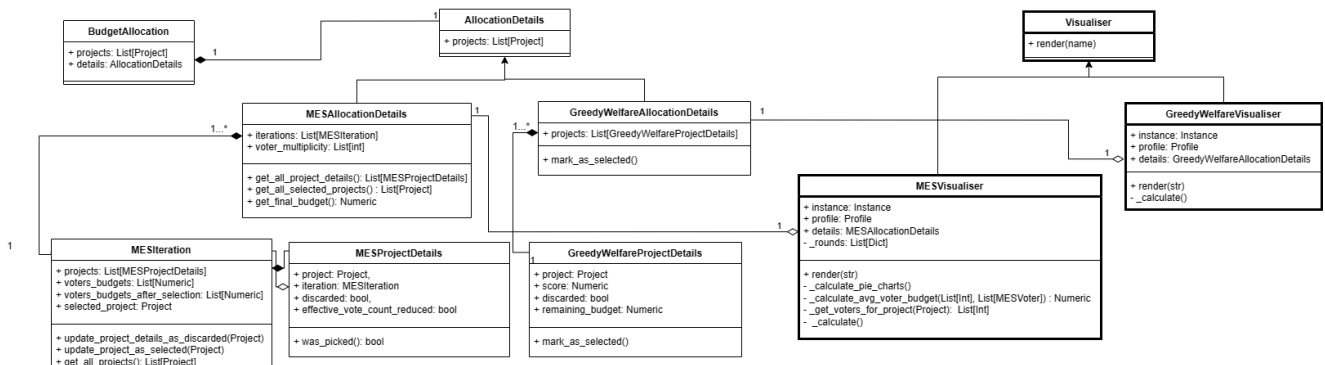


Figure 4.22: A UML Diagram for the Visualisation Classes and How They Connect to the Classes in Figure 4.21

The sole public function available within the `Visualiser` class would be the `render` method; this is what generates the HTML file(s) within the user's specified output folder and also with the specified name. It calls the `_calculate` method which is responsible for performing some analysis but mostly restructures all the data into a format that is ready to be injected into our predefined templates. This injection is performed using the Jinja library (discussed in detail later). Almost all the data that we inject into the template goes into JavaScript objects, where it is then accessed by our different JS graphing libraries. We also use some of the data directly within the HTML, e.g. in our dropdowns and title texts.

4.4.5 Web Pages

It will be difficult to talk in detail regarding the architecture/design of our web pages since they vary a lot from page to page. However, all the web pages do tend to follow the SPA-like (Single Page Application) structure, where the content on the page is dynamically reloaded using JavaScript as opposed to having multiple separate pages. There are also a few interesting points we can mention here specifically related to how we have optimised our pages for performance. Given the size of some elections, some of which will have within the range of 100+ projects, 10,000s voters, 50+ rounds, etc we had to consider how we render all our visualisations/charts in JavaScript. For example, we found that rendering all our visualisations all at once during page load would result in a web page that will never load because it will eventually crash or will take way too long (5+ minutes) for some large elections. This problem was really only prevalent in the MES round-by-round analysis page because of the number and variety of visualisations that are presented in each round. Thankfully, the design of the MES round-by-round page doesn't require all the visualisations to be generated at once and we can instead only render the round that is currently being displayed to the user. The exact approach we took to implement the solution to this problem will be discussed in the later Implementation chapter (Chapter 5).

Chapter 5

Implementation

This chapter discusses the implementation process of our visualisations and explanations. This includes various steps we encountered throughout our development and any challenges we faced and overcame to deliver our final solution. As stated in Chapter 1, our work is part of the Pabutools library; for this reason, as well as the specification of our customer, our implementation had to follow a strict set of rules to conform with the customer needs as well as the format of code within that library.

5.1 The Pabutools Library

Before examining our implementation in detail, we discuss the pre-existing Pabutools architecture. This includes introducing relevant classes we build on, introduce, or adapt to produce our final solution. Most notably, we discuss implementations of the classes integral to developing our solution. First of all, it should be noted that our customer implemented the initial class structure for the `BudgetAllocation` class. This was to ensure that the class fit the customer's specifications exactly and the team could implement changes directly.

5.1.1 Data Capture Classes

The implementation of the `BudgetAllocation` and the other data store classes was something that was fairly simple. The biggest challenge was designing the solution itself. This was a very collaborative process since it involved changes to the existing API of how voting rules were executed. Although the solution was designed to minimise any significant changes - it was a balancing act between that and still having a flexible and easy-to-use system that enabled data capture within any voting rules. The design process and the alternatives we considered are discussed in Section 4.4.2.

Both `BudgetAllocation` and the base `AllocationDetails` had simple implementations. The `BudgetAllocation` has an initialiser which takes in a list of `Project(s)`; the list of projects that are passed into this initialiser would be the same exact list of projects that were previously just returned by voting rule methods. This will then be set as a property of the class. Additionally, the initialiser also takes in an object of type `AllocationDetails`, this is where we can pass a specific instance of either `MESAllocationDetails` or `GreedyWelfareAllocationDetails` (or one for any future supported rule). This argument is optional since it will not always be required to store details, and doing so in every run would unnecessarily impact performance

Not storing details in every run was one simple measure we took in our effort to minimise any impact on performance. However even in the case that the user opts to store details, we still had to design the system in a way that there is minimal impact to the performance. The way we achieved this is by deferring any additional calculation needed for our visualisations to render time rather than during computation of the rule. In other words, the methods used by the voting rule within any of our `AllocationDetails` classes are only responsible for storing the raw data (and that only). This means we only store variables that have already been computed as part of the regular voting rule run, and so the impact on runtime will only be this additional storing step. See Figure 5.1 for a clearer picture of how this works.

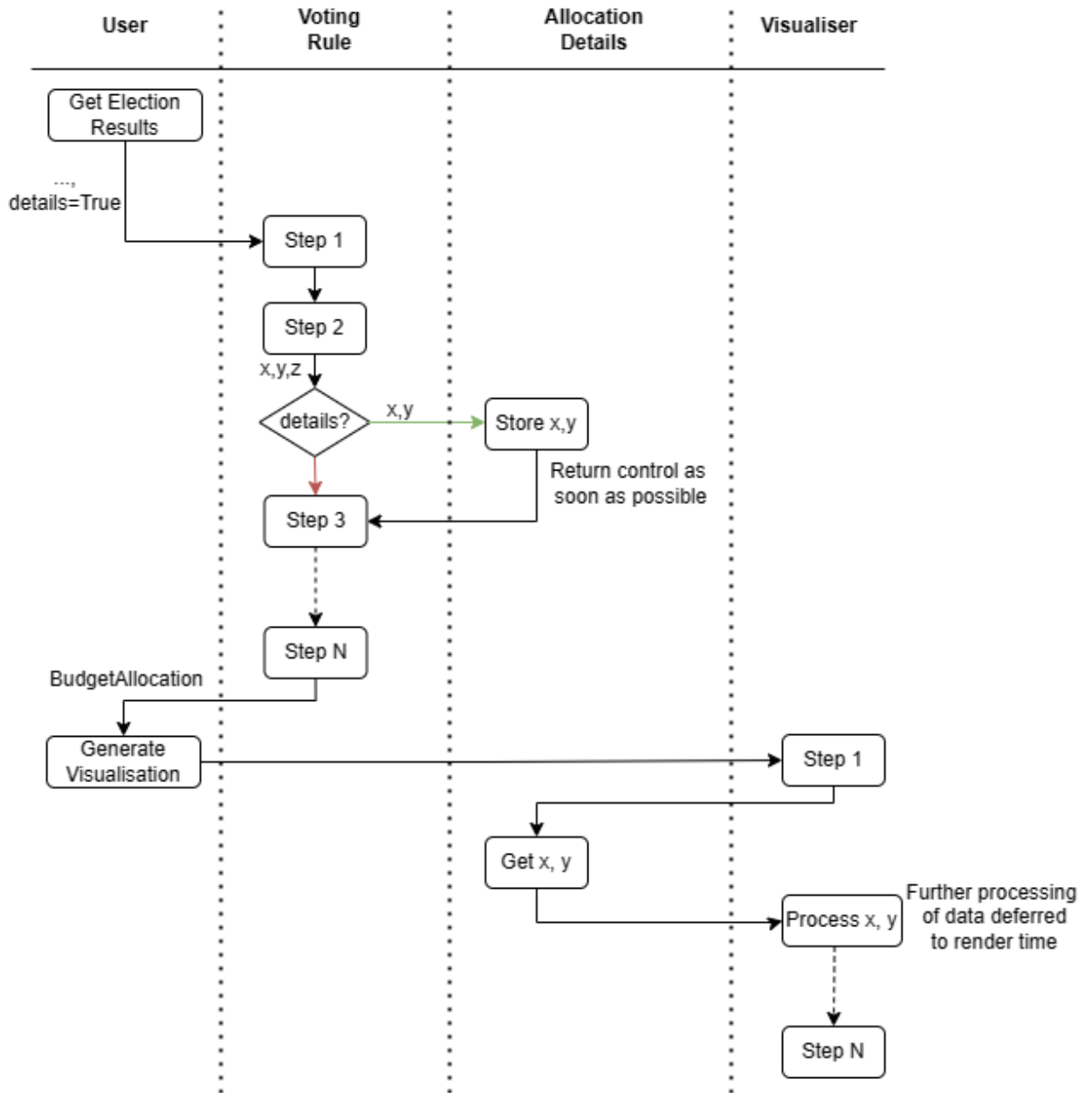


Figure 5.1: An Activity Diagram Showing the Responsibilities of the Individual Components of the Projects

5.1.2 MESVisualiser

The `MESVisualiser` class is used for generating the visualisations web pages. It is a child of the base class `Visualiser`. As it stands now, the base class does not implement any methods shared across the different visualiser classes. This is be-

cause there was no potential for further abstraction between `MESVisualiser` and `GreedyWelfareVisualiser`. However, it is likely that, as more voting rules are supported in the future, this base class could implement some shared functionality. In this section, we will be going over the implementation details of some of the more interesting methods that are part of `MESVisualiser`. A list of all the methods that are part of this class can be seen in Figure 4.22 (in Section 4.4).

```
class MESVisualiser(Visualiser):
    def __init__(self, profile, instance, mes_details, verbose):
    def _calculate_rounds_dictionary(self):
    def _calculate_pie_charts(self, project_votes):
    def _calculate_avg_voter_budget(self, voters_budget, supporters):
    def _get_voters_for_project(self, project):
    def _calculate(self):
    def _render(self, outcome, output_folder_path):
```

Figure 5.2: Code Structure for `MESVisualiser` Class

Calculating Pie Chart Data

The `_calculate_pie_charts()` method calculates all the data necessary for the pie chart visualisations on the round-by-round page. The function returns a dictionary containing the pie chart information for each round. For each round, all projects yet to be elected are represented in pie charts. The pie charts for these projects show how the voters for that project voted for the current project, hence having two sides (those who voted for it and those who did not).

The dynamically generated text accompanying each diagram is another key part of the pie chart visualisation. One crucial part of the text is the average budget that has decreased among all the participants who voted for the project, as represented in the pie chart. The `_calculate_avg_voter_budget` calculates the average budget of the supporters of a project. With the budgets for all participants before and after

the current round was elected, the average budget decrease between supporters of the other projects can be calculated.

Calculating Final Rounds Data

As we have mentioned before, the `render()` method is the main method of this class, and it is what generates the web pages and also saves them to disk. The main data that is passed into our HTML templates via Jinja is the `rounds` property, which is a list of dictionaries containing data used to generate the visualisations. There is a dictionary for each `MESIteration` ('iteration' and 'round' are used interchangeably within the code but refer to the same thing). See Figure 5.3 for all the items that form part of the dictionary. There is a main `_calculate` method that is called by `render` and this method orchestrates the construction of this `rounds` object by calling the other smaller `_calculate_...` methods; the main part of this is the `_calculate_rounds_dictionary` method. This method works by looping through each `MESIteration` that is attached to the `MESProjectDetails` and creates a new dictionary for each one. We populate the dictionary by accessing information that is stored within `MESIteration` and we calculate some further data by also looping through each project that is part of each iteration. For example, one such metric we need to calculate is the funding that is lost per project in each round; we do this by looping through each project in an iteration and computing the difference between the total voter funding for each project in subsequent iterations. All the required data has already been stored within `MESProjectDetails` and `MESIterations` and it is the role of these `_calculate_...` methods to simply further process them and store the relevant details in rounds.

```
[
  {
    "name": "Sample Project",
    "effective_vote_count": {
      "Project 1": 32.2, "Project 2": 20.1, ...
    },
    "effective_vote_count_reduction": {
      "Project 1": 10.2, "Project 2": 0, ...
    },
    "cost": 200,
    "totalvotes": 33,
    "initial_voter_funding": [12, 0, 13, ...],
    "funding_lost_per_round": {"Round 2": 3, "Round 3": 10},
    "final_voter_funding": 130,
    "dropped_projects": ["Project 4", "Project 5"],
    pie_chart_items: [...]
  },
  ...
]
```

Figure 5.3: Structure of the `rounds` List Dictionary That is Passed Into Our MES Templates Using Jinja

Render

The `render` method is the only ‘public’ method (Python has no strict sense of public/private methods, but private methods are denoted with an ‘_’) available in the `MESVisualiser` class and it is what users would use to generate visualisations. It takes in two arguments: `output_folder_path` and an optional `name`. The first is self-explanatory and is the argument where the user can specify where to save the generated HTML files would be stored. The second argument allows the user to specify the prefix of the file names that are generated, e.g. if `name=‘parks’`; the files that are generated would be ‘`parks_round_analysis.html`’ and ‘`parks_summary.html`’. This argument was introduced at the suggestion of the customer as he thought it would be

useful to control the name of the files to avoid overriding existing files that have been generated in the same folder. The implementation of this method is fairly simple and just uses Jinja's `Template.render` to pass our data into our HTML templates. We load the `Template` object as a class variable using `ENV.get_template` and pass in the path to our templates. The `Template.render` function actually just returns a string so the final step is to write the returned string to an output file. It is important to note that we do not just pass in our `rounds` property which contains data related to each iteration of MES, but we also pass in other properties related to the election itself, e.g. total number of votes, budget, projects metadata, etc.

5.1.3 GreedyWelfareVisualiser

The structure of `GreedyWelfareVisualiser` is very similar to the structure for MES. For this rule, there is very little additional information that needs to be required outside the run of the rule. This time we also don't need any significant additional computation on the data and so unlike `MESVisualiser`, we only have one `_calculate()` function that creates the rounds dictionary - using the data from the `GreedyWelfareAllocationDetails` class.

Calculating Rounds Data

Similar to the `MESVisualiser`, we have a `rounds` property which is a list of dictionaries which stores all the data required for our visualisations. We generate this list within the `_calculate()` method which is called by `render`. If we see Figure 5.4 we can see the data that is passed into our Greedy template.

```

[
  {
    "selected_project": {
      "id": "...",
      "name": "...",
      "cost": "...",
      "votes": "...",
    }
    "rejected_projects": [
      {"id": "...", "name": "...", "cost": "...", "votes": "
..."},
      ...
    ],
    "max_cost": 100
  },
  ...
]

```

Figure 5.4: Structure of the **rounds** List Dictionary That is Passed Into Our Greedy Template Using Jinja

From Figure 5.4, we can see that there is not as much data that is passed into the template compared to MES. As such the calculating of this data is also relatively simple. Unlike MES, we are not storing data for each iteration of Greedy but rather we store details related to the individual projects themselves, their score (usually number of votes), and which projects were selected and discarded. Every dictionary in the **rounds** list corresponds to one selected project. We use the score to order the projects and then loop through each project. We create a new dictionary if we encounter a project that was selected; whilst we are looping, we are also keeping track of all the projects that have been previously rejected; this is what we store in **rejected_projects**. We also have another property **max_cost** to store the maximum cost of all the rejected projects.

Render

We do not need to spend too much time discussing the `render` method since it works very similarly to `MESVisualiser`, which we have already previously discussed. The only main difference is where `MESVisualiser` generated two files using two different templates; the `GreedyWelfareVisualiser` only has one template and generates one file. There were not any significant challenges in implementing this method since we already went through a thorough refinement process when we worked on `MESVisualiser`.

5.2 Visualising MES

As stated in Chapter 4, creating the visualisations for MES was a two-step process, giving an overview for each election, as well as an in-depth round-by-round analysis. Whilst each section of this came with its own challenges, some critical underlying issues were the complexity of the elections, with many rounds and, consequently, pages required for the election. With these issues came some key concerns and considerations needed to maintain the efficiency of running the rule on large-scale elections.

5.2.1 Summary Page

For the first step, we discuss how the summary page was developed to be consistent with our design as well as any challenges that we have faced.

Reconstructing an existing election's results page

When originally designing the summary page, we were provided with example Python files from the customer, which we utilised as a starting point for our development process. These files were created by Piotr Skowron and his research group, Piotr is credited as one of the authors who first proposed the MES rule [12]. They included a Python file to generate a simple overview table, similar to the one described in Section 4.1.1 as well as a Python file to retrieve all the necessary data from the `.pb`

file to be used within the table. As a result, we deconstructed and analysed the files to extract the necessary components to be able to create an HTML template and have it function within our current system.

By using this approach, we were able to leverage existing code provided by the customer while adjusting it to be consistent with our design. This proved to be beneficial when utilising the first Python file to implement the overview table skeleton to contain dummy data, as it was implemented seamlessly. Unfortunately, unlike the first Python file, not all the existing code could be leveraged as simply within the second Python file, owing to the fact the Python file used to retrieve data utilised a much older version of Pabutools, the version being 0.12 whilst the current release version is now at 1.1.7. Therefore, the file contained now depreciated methods for running the MES algorithm of the election and contained incompatible variables, further, there were no longer any documentation for this version of Pabutools and comments were regrettably lacking. As a result, this required modifying the existing Pabutools package to retrieve the data we require or compute it ourselves with the variables already present. This setback especially brought about challenges when handling rejected projects, an issue that is addressed in the following section.

Dealing with Rejected Projects

The overview table was recreated using the example files, including most of its functionality and the data being dynamically inserted using Jinja placeholders. However, no rejected projects were included as well as the tooltips referencing them. This was due to the necessary data, such as the rejected rounds, being unavailable at the time. This was rectified by adjusting the MESIteration class to include a function that updates the project details of a given project as discarded if it has been rejected. As a result, when calculating the round by round dictionary, we simply check if the project in the current iteration has been rejected and if so, retrieve the necessary details of the project and append the list of rejected projects to the dictionary. Hence, allowing us to simply inject the rejected projects into the HTML web page through Jinja as

normal.

Expanded Row

Moreover, we implement an additional details section for each row when it is clicked. This involved utilising the hidden attribute on a `<tr>` tag to hide a table row and a function to remove the attribute on the specific row that was clicked on. Initially, this section only contained the hyperlink to the relevant round-by-round page; however, additional content was subsequently included once the data from Pabutools was available. This content included the project metadata - such as the project's description, categories and any additional notes for the project. Additional content added also included a list of the funding lost to each other project in the election and a dynamically generated explanation for why the project was accepted or rejected. Implementation for displaying the project metadata and the funding lost was straightforward: The project's metadata gathered in the visualiser class was retrieved and displayed line-by-line, while code previously used for the summary chart tooltips (see Figure 4.5) was reused to create the charts for the complete list of funding taken by other projects.

However, implementation for the dynamically generated explanations for each project was much more complicated. This was because the explanation required for each project could vary significantly, depending on details such as the acceptance status of the project as well as the amount of funding lost to previous projects. While one project could have been rejected simply because its voters were not able to afford it even when using their total budget, another project might be rejected because voter funds were used to afford other projects accepted in previous rounds (leaving an insufficient amount of funds for this project). The former scenario requires a short sentence or two comparing the initial total budget of the project's against the project's cost, while the latter scenario requires a short paragraph describing how the voter's available funds changed from the start of the project selection process until the round that the project is being considered in.

Simply using just one template for every dynamically-generated explanation was not a viable solution - this would have led to either too much unnecessary information being included in the explanation for the former scenario defined above (potentially causing information overload), or insufficient information being included in the explanation for the latter scenario (leading to a poor explanation that users may find hard to understand due to the lack of details). To implement a solution that avoided these potential issues, four templates were created for the dynamic explanations, where each template corresponded to a specific scenario for accepting/rejecting a project:

1. The project was **accepted** and had lost **none** of its initial funding. For these projects, this meant that the projects voters used the funds initially available to them to pay for the project's cost. Note that this is always the case for the first accepted project in the selection process.
2. The project was **accepted**, and had lost **some** of its initial funding. While projects in this scenario did lose some of their initial funding to previously accepted projects, the voters for these projects had enough funds available during this round to afford the project's cost.
3. The project was **rejected**, and the initial voter funds available to it were **less than** its cost. Projects falling into this scenario were rejected because they were not fundable at any point in time.
4. The project was **rejected**, and the initial voter funds available to it were **equal to or greater than** its cost. These projects were initially fundable, but their voters lost too much funding to previously-accepted projects to be able to afford these projects during the round that they are evaluated.

Why was this project not selected?

This project was rejected because its supporters were not able to pay for the project's cost - **250,000 USD** - using the total funding available to them at the start of round 9 (**12,923.99 USD**).

Note that this project would have been rejected, even before taking any funding lost into account. This is because the project's cost exceeded the total funds initially allocated to the supporters of this project (**21,739.13 USD**).

Funding spent in previous rounds.

From the initial total funding, **8,815.14 USD** was spent on previously selected projects.

The funding was spent on:

- **Project 21 - Planting trees in the city:** 6,454.05 USD
- **Project 23 - 5 Water Bottle Refill Stations:** 1,377.53 USD
- **Project 3 - Computers for the community learning center:** 562.50 USD
- **Project 31 - Fire Hydrant Markers:** 421.05 USD

(a) An Example of the Dynamic Explanation Displayed for the Third Scenario - When a Project is Rejected and the Initial Voter Funds was Less than the Cost

Why was this project not selected?

This project was rejected because its supporters were not able to pay for the project's cost - **165,000 USD** - using the total funding available to them at the start of round 8 (**104,578.37 USD**).

Note that the supporters of this project initially had more funding available to them (**231,884.06 USD**), enough to cover the cost of project.

However, this funding was lost as these supporters had also funded projects in the previous rounds, decreasing the total funding available to them to the point where the project cost exceeds the combined funds of the supporters. The top projects that can be seen in the list below were the main cause for the decrease in funding, these were regarded as more important as they had a higher effective vote count, implying they were selected earlier than this project.

Funding spent in previous rounds.

From the initial total funding, **127,305.69 USD** was spent on previously selected projects.

The funding was spent on:

- **Project 21 - Planting trees in the city:** 58,086.49 USD
- **Project 12 - Separate Bike Lanes from Traffic:** 23,076.92 USD
- **Project 3 - Computers for the community learning center:** 13,500.00 USD
- **Project 13 - Real-Time Bus Arrival Monitors in bus stations:** 12,000.00 USD
- **Project 14 - Benches for a Walkable City:** 10,975.61 USD
- **Project 2 - Little (book exchange) libraries:** 5,666.67 USD
- **Project 31 - Fire Hydrant Markers:** 4,000.00 USD

(b) An Example of the Dynamic Explanation Displayed for the Fourth Scenario - When a Project is Rejected and the Initial Voter Funds was Greater than the Cost. Note How This Explanation Refers to the "Funding Spent" Section Below it to Further Support its Argument

Figure 5.5: Expanded Rows Displaying Dynamic Explanations for Scenarios Involving Rejected Projects

Template explanations for projects described by the first and third scenarios were only a few sentences long, as the results in these situations could be explained by comparing the project's initial funds with its cost. On the other hand, template explanations for projects belonging to the second and fourth scenarios were much longer, as they required descriptions of how the funds available to these projects changed over time, on top of a comparison between the funds available to the project during its evaluation against the project's cost. See Figure 5.5 for how explanations varied depending on which scenario applied to the project. Varying the lengths of each explanation depending on the scenario helps ensure explanations contain an appropriate amount of detail - an amount which avoids both ambiguity and information overload. Alongside this, information such as available funds and project costs are shown in bold, highlighting them and guiding the user's focus towards the most important information in the explanations.

Dynamic explanations were implemented into the visualisations by inserting Jinja conditional statements into the summary page's template file. For accepted projects, a conditional statement checks whether or not the project's funding lost is equal to 0 - selecting either the second or first template explanation depending on the result. For rejected projects, the statement instead checks whether or not the project's initial voter funds are less than its cost - using this information to either select the third or fourth template explanation. The Visualiser class provides the information needed to resolve these conditional statements, and the code is executed when Jinja populates the HTML template with data.

Page Efficiency

The summary page provides an overview table as its stand-out feature which includes rounds in each row, this was achieved by iterating through each round and dynamically inserting data into each row through Jinja. This meant the HTML for the web page would contain the HTML needed for all the individual rows, which on the surface did not appear problematic. However, in large elections, this proved to be a

slight issue as the generated HTML files could reach to upwards of 60,000 lines and 5 MB in size. This meant on lower-end devices, the page was slow to load on a browser and can become frustrating to use for a user.

Through examining the contents of the generated HTML files, we found that a CSS style block was being repeated again and again. This CSS style comes from the tooltip associated with each row's chart that features the dynamically generated diagram displaying the most popular projects where funding was lost. This tooltip utilised an inner HTML block and CSS to display the visualisation, however, this also required iterating through a list of popular projects to generate the visualisation through Jinja. As a result, this meant the inner HTML block was repeated for each popular project displayed and again for each row in the election, resulting in large amount of repeated blocks of code. One method of addressing this issue that was attempted was to create a CSS class in the page's main HTML block and factorise it - similar to a fix for a related problem on the round-by-round page (explained in Section 5.2.2). However, this method proved to be infeasible, as tags within the inner HTML block were not able to access the CSS classes available in the main HTML block. As a result, we instead collapse the style attribute of the CSS within the inner HTML block to one line as opposed to a block of code. While these changes heavily reduce the readability of the code involving the tooltip, they result in the file size being reduced by 60% - 65% and the number of lines being reduced by 70% - 80% in large elections.

5.2.2 Round by Round Page

As mentioned in the technologies section, Jinja is used as a templating engine to build the web page dynamically. As the round-by-round page explains the decision-making behind why a project was elected at a given round, there must be a variables for each of the individual rounds. At each round, variables describe the metadata of the project elected at the current round and information regarding how the other projects were affected after this project was elected, for example, the effective vote

count of the remaining projects. These variables were stored in an array, with each index representing a round. The information at index zero of the array represents the first round, and so on. The Jinja templating was used to create a section element containing the web page for each round. Each section element had an id equivalent to the round number. This page is a round-by-round view, so only one section element can be displayed at a time. Intuitively, the page opens with the first round. This is done by making the first round the active section whilst the rest of the sections remain hidden. The new page is made active upon page navigation, previous, next, or selection, and the previously shown page is hidden with the rest of the rounds.

After implementation, the round-by-round page contained all the graphs discussed in the design chapter. The diagrams were created using different software tools. The chord diagram, pie charts and bar charts were all made using the ZingChart JavaScript library, whilst the Sankey diagram was created using Chart.js.

Page Efficiency

The round-by-round page gives a step-by-step run-through of the election, explaining why each project was elected at each round. The page updates for each individual round; therefore, there are the same number of pages as there are elected projects. Initially, the HTML for the web page included the HTML needed for all the individual rounds, and section elements contained the HTML relevant to that specific round. Only one section element was shown at a time, and the rest were hidden. When the user requests the next round, the current section is changed to hidden, and the new round section is active.

An advantage of this method is that once the page has loaded, navigation between pages is extremely quick, and there is no waiting time for the new explanations to load. However, this method has provided an issue for large elections. In an election where many projects are elected, more section elements must be generated for each round. This resulted in HTML files of 600,000 lines, which were never rendered when

loaded into a browser. Therefore, a new method of generating the HTML was required.

To address this issue, the HTML would have to be a similar size for all elections. This would involve dynamically changing the elements rather than generating all elements and only showing the relevant parts whilst hiding the rest. A JavaScript array containing all of the pertinent information for each round was created with the idea that when loading the next round, the respective information could be drawn from the array and the HTML elements reloaded.

As each diagram is accompanied by dynamically generated text, changing this at each round was also required. However, storing the whole text for each round was wasteful. To fix this issue, span elements with class names equivalent to the data stored were used to ensure the text was still specific to the given round while not storing duplicate text for each round. For example, there is a span element for the project's name elected at the current round. Upon navigating to the next round, all instances where this span element was declared within the text will be replaced with the newly elected project name.

Project Prioritisation

It quickly became apparent that it would not be possible to display all of the projects in some of the diagrams. This was especially the case in the Sankey diagram and the chord diagram. When more than six projects were included in these diagrams, the information was difficult to understand, and the relationship between projects became increasingly complex.

It became crucial to address this problem by prioritising the projects. Only the top projects were listed in the diagrams, which required careful consideration. Establishing a metric for importance was a necessary step. Initially, the projects that shared the most common voter bases were chosen, as these were the projects that

would be most affected by the current round.

In the Sankey diagram, an additional ‘other’ node represents the projects not included in the most critical projects. This node helps the user understand the proportion of votes the most relevant projects take up relative to the rest.

An example of how the diagrams become uninterpretable if all of the projects are included can be seen in Figure 5.6. In this election, there are 20 projects. Therefore, each arc can lead to at most 19 different arcs, resulting in 190 individual chords. As we see in the figure, although the underlying patterns are captured, the graphic is difficult to understand and draw any inference from.

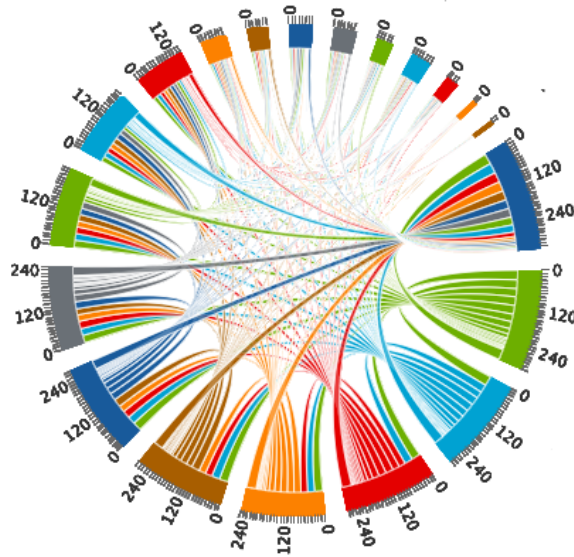
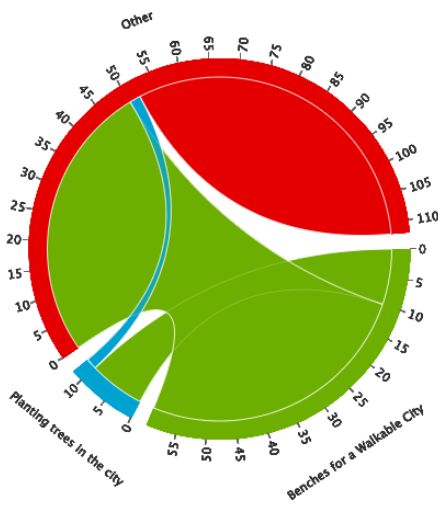


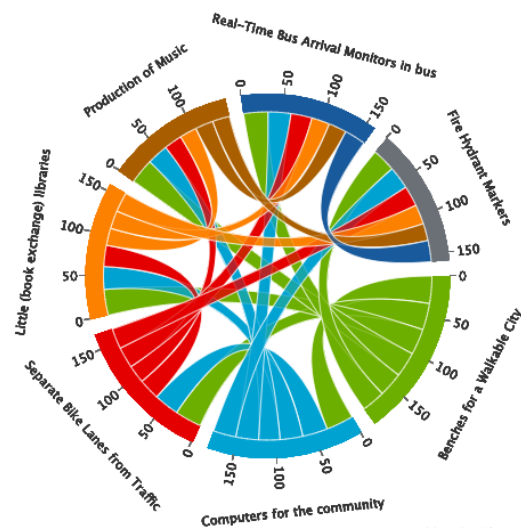
Figure 5.6: An Example Chord Diagram Without Project Prioritisation

Initially, the prioritisation of projects was ranked by the amount of flow they take up; this was chosen as it captures the majority of the flow of users’ votes. However, using this method might not capture the most essential underlying patterns. Consider project ‘*A*’ that has been elected at the current round if we also have project ‘*B*’ such that all voters for project ‘*A*’ also voted for project ‘*B*’. In this case, we expect the effective vote count for this project to decrease significantly. However, if the amount

of people who voted for ‘*B*’ is a small proportion of those who voted for ‘*A*’, then it would not be included in the prioritisation. Therefore, other methods need to be explored to capture all the patterns. Figure 5.7 demonstrates this issue. The chord diagram on the left shows that the majority of people who voted for ‘Planting Trees in the City’ also voted for ‘Benches for a Walkable City’; however, this relationship is not captured in the chord diagram displayed on the webpage (diagram on the right). This is an issue as this is a significant relationship to capture as electing ‘Benches for a Walkable City’ is likely to affect the effective vote count of ‘Planting Trees in the City’.



(a) Flow from ‘Bench for a Walkable City’, ‘Planting Trees in the City’ and All Other Projects



(b) Chord Diagram Showing the Top Seven Ranked Projects (Not Showing ‘Planting Trees in the City’)

Figure 5.7: A Demonstration of the Chord Diagrams Not Including an Important Project

As an extension to this, we examined the issue from a different perspective. Since the round-by-round analysis should show the user exactly why a voter’s budget may have been deducted because a project they had voted for was selected, we needed to adapt this approach slightly. Say project x_w wins this round. The budgets which this

will most impact will correspond to the projects which had the highest overlap in the vote count. Say we have a matrix X corresponding to the pairwise interactions of vote counts per project (x_{ij} corresponds to the number of voters who voted for i and also voted for j). Say project i was selected; from here, we take the 6 highest values in the row x_i , which corresponds to the highest overlap of votes for that project (call these values c_1, \dots, c_6).

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix} \Rightarrow X_{i,c_1,\dots,c_6} = \begin{pmatrix} x_{ii} & x_{ic_1} & \cdots & x_{ic_6} \\ x_{c_1i} & x_{c_1c_1} & \cdots & x_{c_1c_6} \\ \vdots & \vdots & \ddots & \vdots \\ x_{c_6i} & x_{c_6c_1} & \cdots & x_{c_6c_6} \end{pmatrix}$$

Whilst in lots of cases, this method will yield the same results as the previous, this captures more nuances in the data - say a project (b) which in general was unpopular. However everyone who voted for b also voted for the project which won (a), this highlights the reason why the budget voter budget was deducted for b and why it may not have been selected.

In the PB research field, the exploration of which projects are relevant to an election based on the proportionality of the voters they represent and satisfy is still a very open question. Our approach is one which we felt was an effective balance between simplicity and ensuring the relevant information is passed to the user for the explanation results.

5.2.3 Maintaining Efficient Runtime

Two key requirements **R14** and **R15** stated that the runtime of our visualisations should scale with the number of votes and that only minimal adjustments should be made to the base class. Since many elections have 100,000's of votes and 100s of projects, ensuring our visualisations scale effectively is vital. For the following rea-

sons, we opted for two main approaches.

Firstly, ensure we do not calculate any additional information that is already calculated by the rule. We achieved this by passing an additional parameter to the `method_of_equal_shares_scheme` called `analytics`. This means we keep the remainder of the base class the same. The updated parameters are as follows:

```
def method_of_equal_shares_scheme(  
    # Additional Parameters  
    # ...  
    # ...  
    analytics: bool = False, # Our added parameter  
) -> BudgetAllocation | list[BudgetAllocation]:
```

Resultantly, we only marginally edit the base class and fulfil our customer requirement **C4**. While we ensured we maintained a focus on efficiency as much as possible, there were still some prevalent issues. Most notably, this was the calculation of the budget reduction in the pie charts. The reason this is significant is because of the calculation required for the reduction in budget. For each round, we must iterate through all of the projects, excluding the winner. From here, we calculate the round voters and the non-round voters, taking the budget before and after the project selection. Figure 5.8 shows the code used to calculate the pie charts.

```
# Function used for the voter budget calculation  
def _calculate_avg_voter_budget(self, voters_budget, supporters):  
    if not supporters:  
        return 0  
    return sum(voters_budget[s] for s in supporters) / len(  
supporters)
```

```

# Calculate the information for the pie chart
# This is done per project per round
round_voters = round["voter_flow"][project.name][selected]
non_round_voters = projectVotes[project.name] - round_voters
reduction = self._calculate_avg_voter_budget(
    round["_current_iteration"].voters_budget,
    self._get_voters_for_project(project),
) - self._calculate_avg_voter_budget(
    round["_current_iteration"].voters_budget_after_selection,
    self._get_voters_for_project(project),
)
# Append the information as one of the pie chart items
pie_chart_item = {
    "project": project.name,
    "roundVoters": round_voters,
    "nonRoundVoters": non_round_voters,
    "reduction": reduction,
}

```

Figure 5.8: Pie Chart Calculations

Given that in many cases, there may be up to 100 projects present in the elections where the number of rounds is a similar magnitude to that, this means the calculations need to be performed 1000's of times - taking a significant time to calculate. As with the chord and Sankey diagrams, it was vital to use project prioritisation to ensure that the first few relevant projects are shown and that we can 'discard' the later ones. This optimisation ensures three things. First, the page size is decreased - since for each round, we only calculate this information for (or up to) the first nine projects with the highest vote count relative to the selected project, this significantly reduces the page size. Second, this ensures the explanation remains as simple as possible "overload of information reduces a user's overall comprehension of the information expressed to them" [49]. This approach ensures only relevant information is expressed to the user. Finally and most importantly, this significantly reduces the runtime required

for calculating and displaying the pie charts - ensuring we align with our customer requirement **R4**. We explore the decrease in processing time achieved in Section 5.2.4.

5.2.4 Runtime Analysis

As stated earlier, the requirement of the runtime being unaffected where the visualisations are not running, we ensured that the only addition to the rule run is storing data during the run. To verify this, we run the rule on 700 elections to determine the change in runtime with analytics set to true.

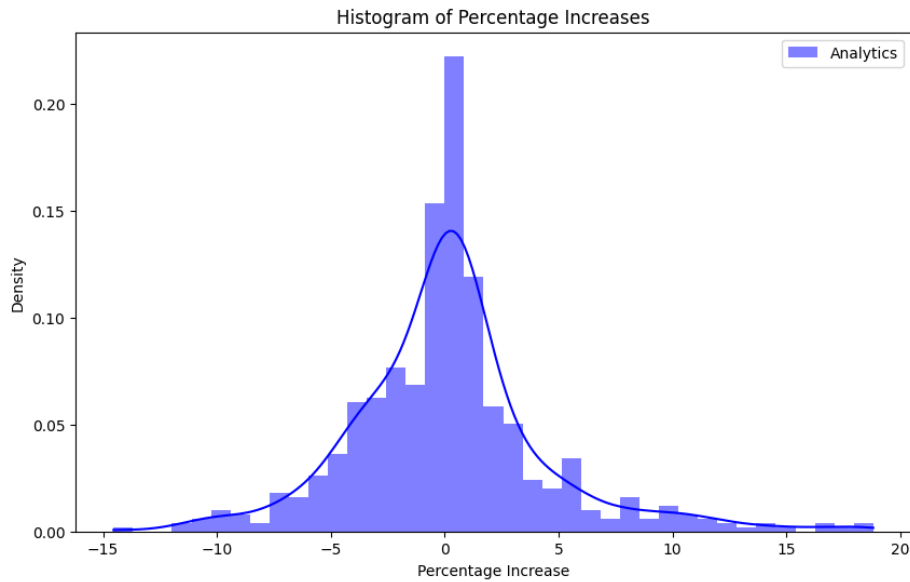


Figure 5.9: Runtime Distribution for MES Rule with Analytics Set to True

Evidently, the distribution of runtimes indicates that, on average, the rule’s runtime remains unchanged even with analytics. This is because we successfully ensure that the only additional computation required when using the rule is to store the information needed in our visualisation class. Resultantly, this means the computation time is transferred to the other class.

Now, we explore the increase in calculation efficiency when only considering the top

nine projects based on the vote count overlap. Figure 5.10 shows the distribution for the percentage increase in processing time before our optimisations, with the descriptive statistics in Table 5.1.

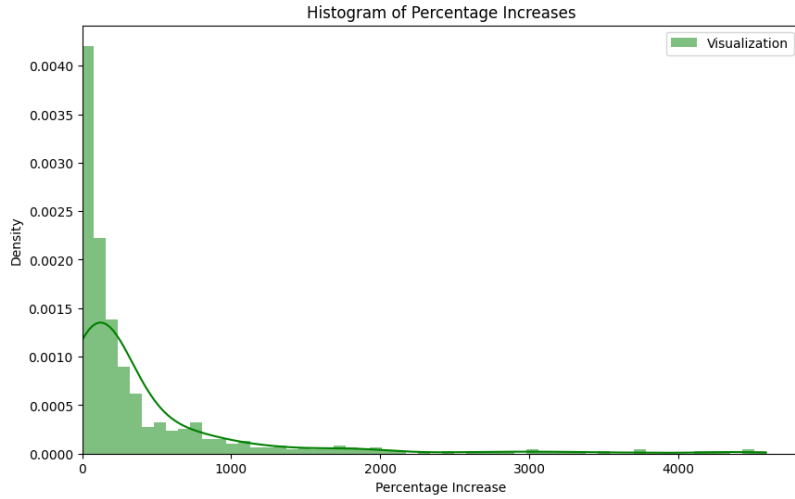


Figure 5.10: Percent Increase in Time for MES With Visualisations - Pre Optimisation

Statistic	% Increase (Visualisation Time)	Visualisation Time (s)
mean	417.8	13.28
std	718.2	80.89
min	0.0	0.0000
25%	58.4	0.0344
50%	146.9	0.2099
75%	394.6	1.3916
max	4587.8	1095.36

Table 5.1: Descriptive Statistics for Percent Increase in Time for MES with Visualisations

Whilst it is clear that, in general, the increase in visualisation time is minimal - with a median value of a 147% increase in processing time, the mean indicates the issue that in some cases, there are exceedingly large increases in processing time. The maximum of which is over 4,000% corresponding to over 1,000 seconds (or 20 minutes). This is also not an isolated issue - as shown by Figure 5.10, there are a large number of elections which significantly bring the mean processing time up. Each of these has a common single factor - many projects highlight the issue with our current approach. Given the issues displayed in the previous figures, we now introduce our optimisation - only considering the top nine projects per round for pie charts. Figure 5.11 the updated distribution for the times taken with our implemented optimisation. Most notable, is the decrease for the times taken as a whole - there is a much higher mass in the distribution towards the lower end, as well as a much higher mass within our ‘acceptable’ increase in visualisation time ($\leq 300\%$ corresponding to 84% of the visualisations being within that acceptable region which is a far more acceptable metric).

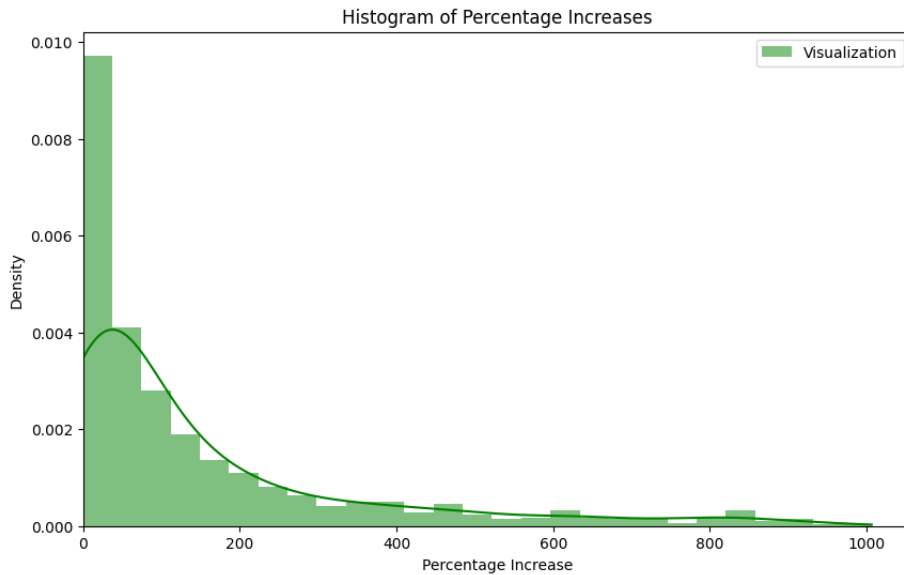


Figure 5.11: Percent Increase in Time for MES with Visualisations - Post Optimisation

Whilst the plot still shows some elections where the processing time is far larger, this significant increase in time is still far smaller and within the bounds of acceptability. In Table 5.2, we abbreviate Visualisation Time with (VT) and Percentage Decrease with Optimisation with (PDWO). Most notably, the significant decrease in our maximum processing time - for the percentage increase as well as the time taken (78% and 77% respectively). This means the longest visualisation now takes under five minutes, which is a reasonable time to wait for the explanation of a large and complex election.

Statistic	% Increase (VT ¹)	PDWO ² (%)	VT (s)	PDWO (%)
mean	152.1	64.6	2.137	84.2
std	203.8	72.3	17.51	78.4
min	0.0	0.0	0.000	0
25%	18.7	68.0	0.0103	70.1
50%	69.5	53.7	0.05951	72.7
75%	189.2	52.1	0.3444	75.3
max	1006.6	78.1	259.7	76.3
Mean Decrease	-	55.6	-	77.3

Table 5.2: Descriptive Statistics for Percent Increase in Time for MES with Visualisations - Optimised

Whilst there were additional optimisation in reducing the time taken for processing such as reducing loops in our templating, reducing the line count of the files and populating the page dynamically, this was the most significant optimisation we performed.

```

class GreedyWelfareVisualiser(Visualiser):
    def __init__(self, profile, instance, outcome, verbose):
    def _calculate(self):
    def _render(self, output_folder_path, output_filename):

```

Figure 5.12: Code Structure for Greedy Visualiser Class

5.3 Visualising Greedy

Another of our key deliverables was to develop an approach which can effectively explain the results of the Greedy Utilitarian Welfare Rule - as referenced in Chapter 1. In our case, we focus on additive satisfaction rules. The default example this includes is Cost Sat, which defines satisfaction as the total cost of the approved and selected projects. Say we have two sets of projects $S_1, S_2 \wedge S_1 \cap S_2 = \emptyset$ with satisfaction cs_1, cs_2 respectively. Then the satisfaction for $S_1 \cup S_2 = cs_1 + cs_2$.

Since the majority of our previous work was spent on developing the framework and visualisations for the MES visualisations, this meant we already identified a structured approach to developing the next set of visualisations. One of our key focuses was ensuring the code’s modularity and reproducibility to extend visualisations to different scenarios. First and foremost, this includes the structure of the class used to generate our visualisations. Figure 5.2 shows the generated structure of the MESVisualisation class. If we compare this to Figure 5.12, we see the similarities between the classes we implemented. Evidently, the Greedy class contains a subset of the functionality required for the MES visualisations. This is because - as we discovered in Chapter 1, the MES rule requires far more information to be calculated in order to generate the explanations for elections.

Additionally, we continue our same approach of implementing very few changes to the base implementations of the rule themselves. As with previously, we add an additional parameter to the original implementation of the rule as shown in the code

block below:

```
def greedy_utilitarian_welfare(  
    # Original Parameters  
    # ...  
    # ...  
    analytics: bool = False, # Our added parameter  
) -> BudgetAllocation | list[BudgetAllocation]:
```

The only additional code added to the base rule is storing additional information in a BudgetAllocation object. This ensures we fulfil customers requirement **R4**. Moreover, this results in two things, firstly, the runtime of the rule even with analytics is impacted only marginally. Figure 5.13 shows the distribution for the increase in processing time for elections with analytics set to True.

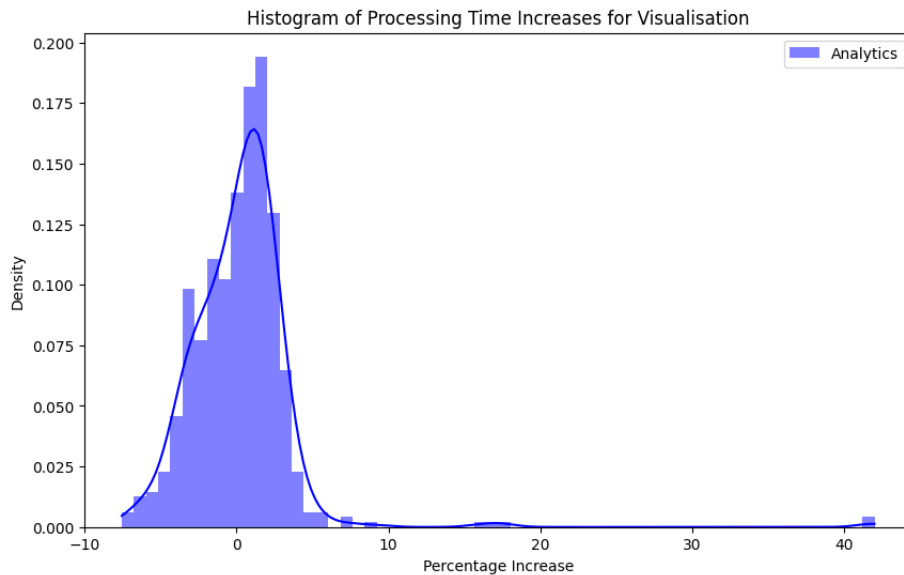


Figure 5.13: Time Increase with Analytics for 605 Elections. Mean: 0.385%, STD: 3.639%

The incredibly low mean increase in processing time, along with the high majority of the mass being centred within $\pm 3\%$ of the origin time taken, implies that the processing time differences are due to fluctuations in the load on the computer at the time across different runs. However, there are a few outlier values that could be

attributed to elections with a large number of projects compared to the number of voters or significantly large loads on the machine used for testing. Further analysis into these metrics will be continued in Chapter 6.

5.3.1 Optimising the Greedy Visualisations

As with the visualisations of MES, due to the size of elections, a key priority was to ensure the file size remained minimal and the added processing time was as small as possible. First of all, we took a naive approach - bulk rendering all HTML without consideration for the large files generated. Take the 2018 Municipal PB In Warsaw shown in Figure 5.14. In this case, there are five projects which were rejected despite their higher satisfaction measure score. This is a simple example for explicative purposes, however in some cases there may be 20 – 30 projects rejected before the selected one. The HTML code to generate each bar at this point took at least 6-8 lines of code and additional CSS and text, generated below the graph shown. Considering this is round 27 of the election, and each round consists of upwards of 30 lines of HTML, this increases the time required for the templating engine to generate the visualisations.



Figure 5.14: Greedy Visualisation Example

Through optimisations on the number of lines through our templating and populating

HTML by using JavaScript to dynamically visualise populate the page, we can reduce the size of the files, the required work for the templating engine, and consequently the time taken to produce the visualisations. Figure 5.15 shows the distribution for the percentage increase in time taken when including analytics and visualisations to the rule compared to the original time. Whilst these increases in time are very acceptable considering the incredibly small mean processing time, the optimisations can still improve our results.

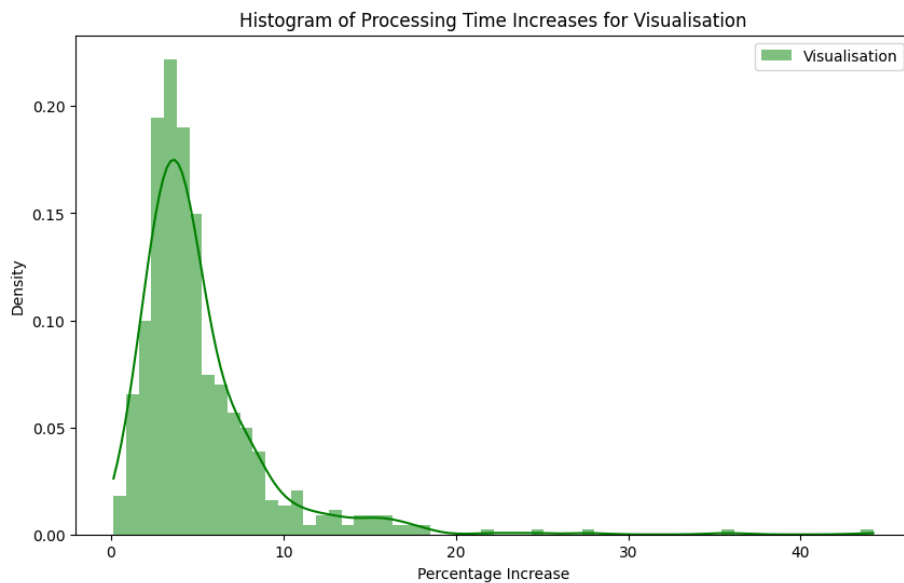


Figure 5.15: Percentage Increase for Time Taken for Greedy Visualisations. Mean 5.124%, 0.00604s

Figure 5.16 shows our times taken for visualisation post optimisation. Evidently, there is a significant shift in the mass of the distribution towards a zero percent increase, indicating an improvement in the time taken to generate the visualisations. Whilst the percentage difference in this case is significant, the mean time taken is only 0.00513 seconds around a 15% improvement.

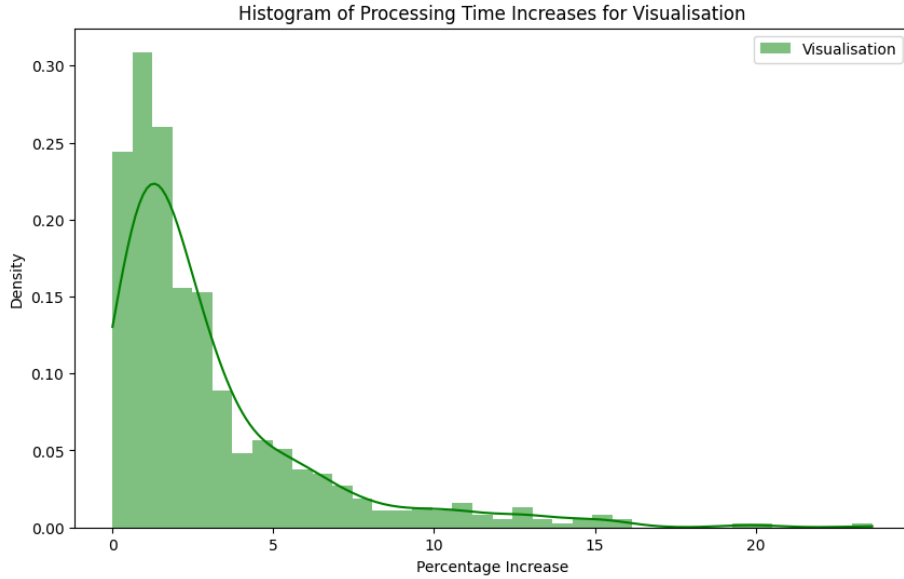


Figure 5.16: Percentage Increase for Time Taken for Greedy Visualisations - Post Optimisation. Mean 3.004%, 0.00513s

5.4 Implementing Clustering on .pb Files

Given the background information from Chapter 2, we know the size of an election (the number of voters) can exceed 100,000 and consist of over 100 projects. Given the magnitude of the data to analyse, this highlights a key issue when producing visualisations which abstract large numbers of voters into one. For this task, we consider clustering - “the process of grouping similar objects”. With the goal of merging groups of similar voters into one. This would allow visualisations to be created for representative groups of voters and consequently, contain far simpler visual explanations.

First, we define our dataset; in our case, this is a set of voters who vote for a set of projects. Whilst in some cases, each voter has additional contextual information such as age, voting method, and gender, in some cases, we have none of this. Therefore, the information available to us is the set of the projects which the voter votes for. This introduces us to the construction of our dataset X . For each voter i , we assign the value 1 at x_{ij} if i votes for project j - one hot encoding the vector for the voters. Before we begin clustering, however, we need to consider what a cluster of voters may

represent and visualise the results of the clustered data. Due to the dataset’s high dimensionality (d), the visualisation of clusters is non-trivial.

5.4.1 Data Visualisation

For visualisation of the data in two dimensions, we considered two approaches first Principal Component Analysis (PCA) and t-distributed Stochastic Neighbour Embedding (t-SNE). PCA is the process of transforming data linearly onto a new coordinate set such that the directions preserve the maximum variance [50]. Given our dataset X , PCA transforms the dataset as follows:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix} \Rightarrow X_{\text{PCA}} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{pmatrix}$$

Now, we are able to plot our first against our second principal components and visualise our data. Below we show some of our transformed datasets (elections) using PCA - note that each of the datasets presented below are elections taken from Pabulib [8].

t-SNE is a non-linear technique which focuses on preserving the pairwise variance between data points at a lower dimension [51]. t-SNE stands out particularly when visualising very high dimensionality data while maintaining the data points’ relative locality. For our use this is particularly beneficial, we want to ensure that in two dimensions, voters who have voted for a large group of similar projects appear together locally.

This algorithm can be summarised by a four-step process:

1. Compute the pairwise similarity in the high-dimensional feature space.
2. Compute the pairwise similarity in the low dimensional space.
3. Minimise the Kulback Leiber (KL) divergence.

4. Iterate until convergence.

Once again, we can plot the data in two dimensions upon completing our data transformation. Below, we show some examples of elections which have been transformed using PCA versus t-SNE.

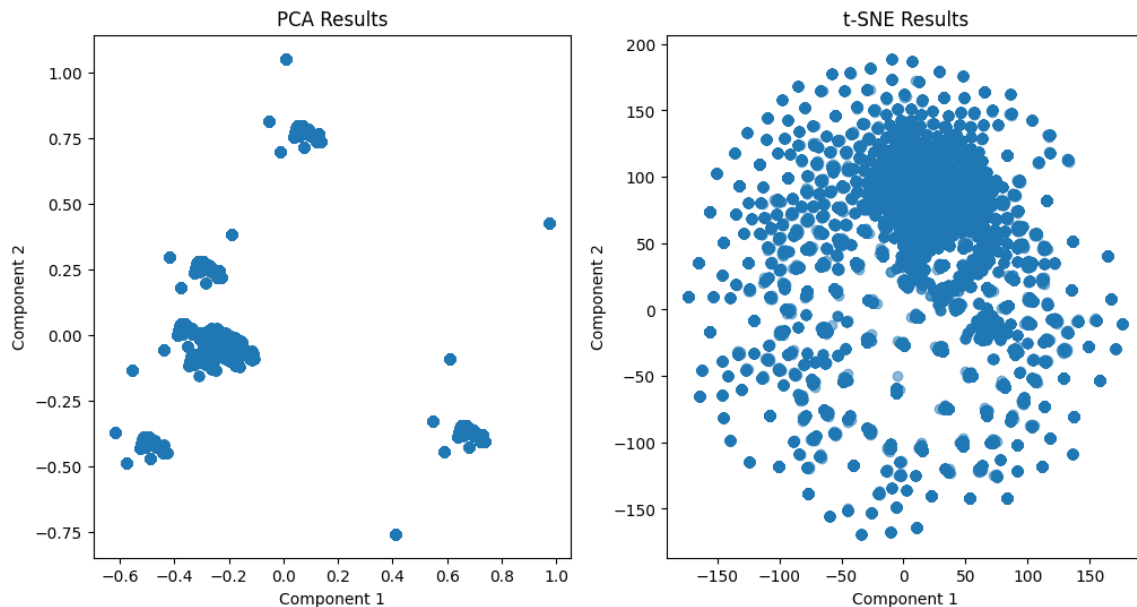


Figure 5.17: PCA Versus t-SNE 2D Transformation Results - Poland Wroclaw 2018

As shown in Figure 5.17, in this case, PCA appears to show some clear groups of clusters in the voter set, whereas t-SNE performs far worse on this. However, these results clearly show that there are clusters in the data for this election result. This is, of course, a promising result indicating that an election of over 50,000 voters has large sets of similarities within its voters.

5.4.2 Clustering Algorithms

As with visualisation, we have various options available to us. Since, in our case, the number of clusters present in each election will vary significantly, we require an adaptive algorithm for the dataset. Therefore, an algorithm such as K-Means is not suitable since the number of clusters is a hyperparameter which we set. This is likely

wildly different based on the number of voters and projects. However, there are some assumptions we can reasonably make:

1. There will be some projects which are inherently the most popular.
2. Some subgroups will likely vote for a similar set of projects - those in a similar geographical area or families with children who likely have the interests of their children at heart.
3. Some projects will be very unpopular; therefore, we can likely discard these projects when considering the visualisations later on.

Based on the assumptions above, we chose Density-Based Spatial Clustering of Applications with Noise (DBSCAN) as our algorithm of choice.

DBSCAN groups together closely packed points, marking points that lie alone in low-density regions as outliers. This relies on two key hyperparameters: ϵ - the radius around the point, and *min_points* - the minimum number of points required to form a dense region [52]. This algorithm works in the following steps:

1. **Identify Core Points:** - a point if at least *min_points* are in distance ϵ .
2. **Expand Clusters:** - from each core point, if another point is reachable, it is part of the cluster else it is labelled as noise.
3. **Merge Clusters:** - if a point is reachable from two core points, the clusters are merged.

This has key advantages—most notably, the number of clusters is automatically identified based on the chosen hyperparameters.

5.4.3 Results

Now, we examine the results of the clustering. This explores the effectiveness of clustering on election datasets. This includes the results of varying hyperparameters and extraction of a ‘typical voter’ within a cluster.

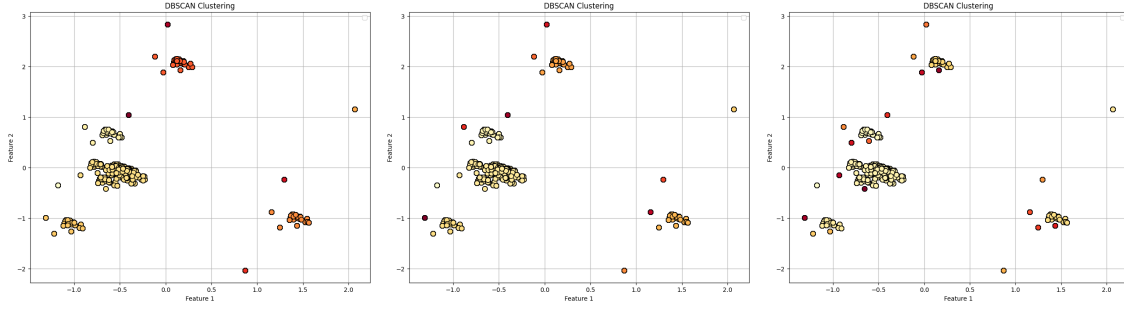


Figure 5.18: Clustering Results Using DBSCAN with $\varepsilon = 0.3, 0.2, 0.1$ (Left, Middle Right)

For this example, Table 5.3 shows the number of clusters. In this case, even though there appears to be comparatively few clusters, the variation in the hyperparameter values shows a fundamental issue of this method - for each election, we need to adapt the hyperparameters to a suitable number of clusters for which to create our visualisations.

ε	Election	Number of Clusters
0.3	Poland Wroclaw 2018	10
0.3	Poland Wroclaw 2017	18
0.2	Poland Wroclaw 2018	14
0.2	Poland Wroclaw 2017	23
0.1	Poland Wroclaw 2018	24
0.1	Poland Wroclaw 2017	35

Table 5.3: Number of Clusters for Varying ε Values

The second fundamental issue is the time taken to run the clustering algorithms. First of all, the dataset must be transformed - we need to do this since, given the high dimensionality of an election where d typically exceeds 30, to ensure DBSCAN is efficient, we first transform the dataset using PCA, then run the clustering algorithm on the transformed data. Figure 5.19 shows the time taken to run for DBSCAN on varying election sizes (post-PCA) - each was run 10 times to calculate an average,

with the green bars showing the minimum and maximum times.

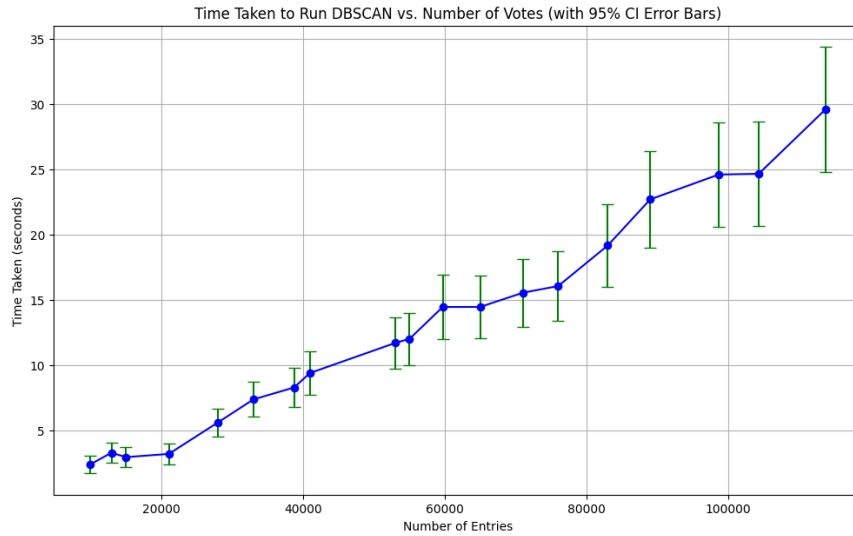


Figure 5.19: Time Taken to Run DBSCAN on Varying Size Elections

Whilst the time taken to run would be reasonable given that the algorithm had to be run only once, since we would have to re-run the algorithm over a wide range of hyperparameters for a grid search, this could very quickly turn into significant lengths of time. Particularly if we were iterating over five values of ϵ and five values of *min_samples* (a comparatively low estimate). Therefore, this quickly becomes unsuitable for larger-scale elections, particularly when a key customer requirement is having visual explanations producible for any election regardless of size.

5.4.4 Conclusion of Clustering

In conclusion, whilst clustering showed some very promising results, we decided to drop this as a primary focus of generating visualisations for the elections. Most significantly, there is high variability when adapting an approach that works for all-size elections. The significant difference from election to election, which would generate our dataset, means that finding a suitable set of ‘majority groups’ or groups of voters which are similar is a very involved process with significant hyperparameter optimisation required for each different election. This is unsuitable for this project since it doesn’t align with requirement **R14** that the running time of the PB rules

with explanations should not exceed 300% of the runtime of the initial rules without explanations.

Whilst we are excluding this from our project in this case. We feel that further research in this area is very valuable. Identifying which projects or groups are most relevant has been a key issue in the PB research community; therefore, finding a method that adaptively identifies ‘significant’ groups of an election is key to furthering interesting explanations and results for the user. Particular areas of interest would be analysing the representation of voters using MES versus the representation of voters using Greedy Utilitarian Welfare. For example, taking the result of the election and finding groups or clusters of voters who were proportionally represented in the outcomes of the result. The incorporation of visual explanations, including this information, would be invaluable in providing further insight and analysis of elections for users.

Chapter 6

Evaluation

In this chapter, we write the evaluation of our project. We conducted multiple types of testing, including unit, integration and user testing, and use all the information to review the success of the product as a whole.

6.1 Testing

To evaluate the success and performance of the project we use a wide range of tests throughout the entirety of the project development lifecycle. This means on a rolling basis we are able to adapt the project, and add any backlog onto our later sprints. Consequently, the iterative improvement of the project and the tools developed as a whole occurs. Finally, in our original specification, we created a set of Testing Objectives available in Figure 3.1. At the end of our testing, we conclude this with a set of testing objectives to determine the extent of the success of the project.

6.1.1 Unit Testing

As an open source library that is continuously being published to PyPI (Python Package Index), unit tests were an essential part of the package. We had to write new unit tests for our new visualisation subpackage as well as modify existing unit tests to account for the changes we made to existing voting rules to store data. Pabutools

uses the built in Python `unittest` library to create and run unit tests. There are also automatic hooks setup on GitHub so that each PR runs all unit tests across all the supported Python versions. There are also automatic reports of code coverage with each PR, this allows our customer to easily check whether we have sufficient coverage for every new line that we are introducing. See Figure 6.1 for one such report.

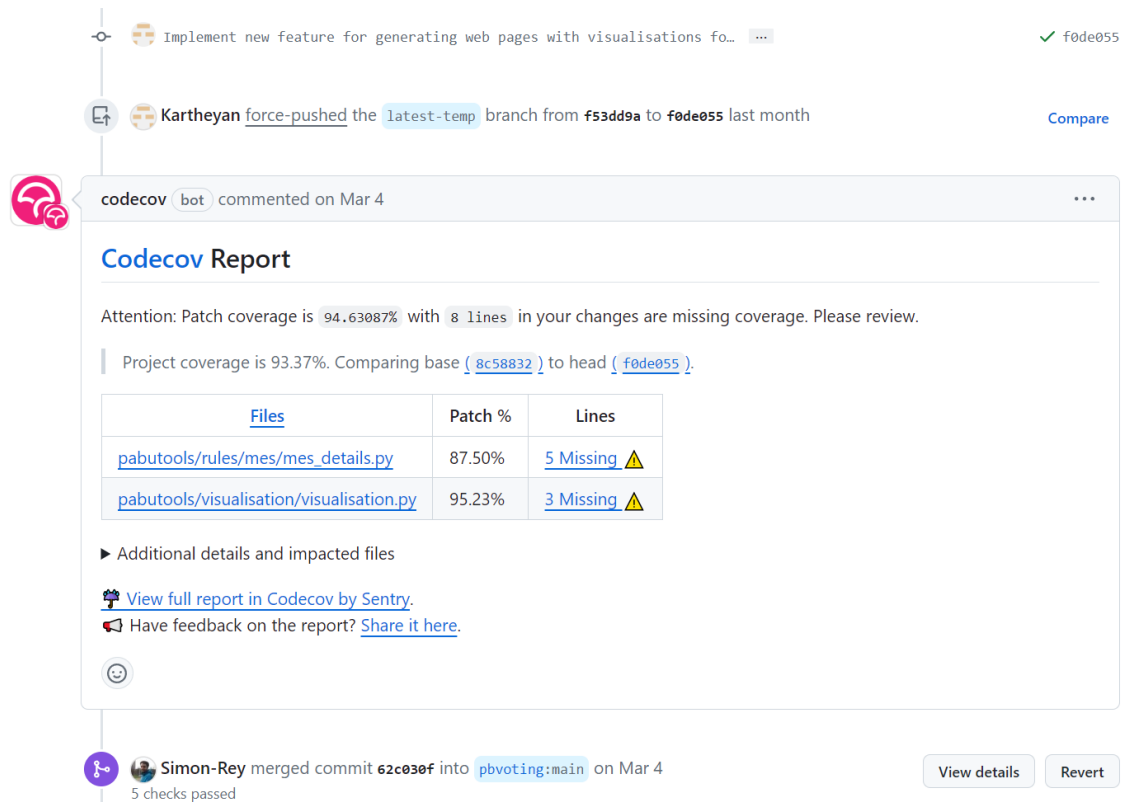


Figure 6.1: Code Coverage Report for PR #15 (pabutools/pull/15)

One of the new unit tests we wrote, tested the generation of our web pages end-to-end; this was the `test_mes_visualisation` unit test. It simulates how a real user would use our visualisation subpackage. It starts with loading one of the sample test `.pb` files that already exist in the package, runs `method_of_equal_shares` on this election with the `analytics` flag set to true (so we store the details of the run), and then generate the visualisations using `MESVisualiser`. The test is successful if the following all hold true:

- No errors/exceptions are thrown.

- Two files are created in the expected directory path.
- Both files contain at least a `<!DOCTYPE html>` tag.
- The length of the `MESVisualiser.rounds` is correct and is the same as the number of projects that were selected.

Looking at the above list, it is possible to spot limitations and potential errors that could leak and pass through this test. For example, we do not actually check the correctness of the HTML files that are generated; this could have improved the robustness of the test. The reason we did not prioritise this is because since we rely on Jinja to generate our HTML files there are certain built in mechanisms within Jinja itself that will throw errors if we violate certain constraints. Another potential future improvement could have been checking specific items within the `rounds` list not just checking the length. The reason we chose not to prioritise this is because we implemented separate unit tests for all of our analysis functions, but eventually it would also be a good idea to test within this unit test as well to ensure the structure of the `rounds` list is correct.

As mentioned previously, we also wrote unit tests for testing individual analysis functions. These would not use real data from `.pb` files (like we did for `test_mes_visualisation`) but instead we would create dummy data to run the analysis functions. This was following the existing convention that was already in place and it ensured the unit test is more easily interpretable and also acts as a good reference for the analysis function itself. There were quite a lot of unit tests for testing the execution of the different voting rules under variety of different conditions and arguments. We did not have to modify all of them but only the subset of conditions/arguments for which our visualisation subpackage is designed to support. These changes mostly involved, setting the `analytics` flag to true. and asserting whether the expected details are correctly being stored.

6.1.2 Integration Testing

Due to the nature of this project and this being designed as part of the Pabutools library. This means two things; firstly, that the library already has tests which must be passed each time there is an update to the repository, as well as, us working with the customer who is very familiar with the repository since they are the maintainer and designer of it. Therefore, each time we would make a pull request, the customer would go through the updates in detail and ensure that every time we made a pull request, they are happy with what we had created.

To view our development and integration testing process we present Figure 6.2. First of all, when we develop a new feature or update, we first create a new branch. From here this means we are able to develop additional system components without impacting the rest of the groups work or the code as a whole. When this feature has been implemented, we first test the changes along. Does each additional component work as intended, and does the requirement fit the specifications of our customer and supervisor? From here, we test the system as a whole - ensure other rules or methods work as specified with and without the visualisation options set to true. From here, we are able to present our changes to our customers and have a meeting with our supervisor, where we discuss the next steps, any changes or removals, and any additional tasks. From here, we create a pull request allowing our changes to be added to the Pabutools library and become publicly accessible.

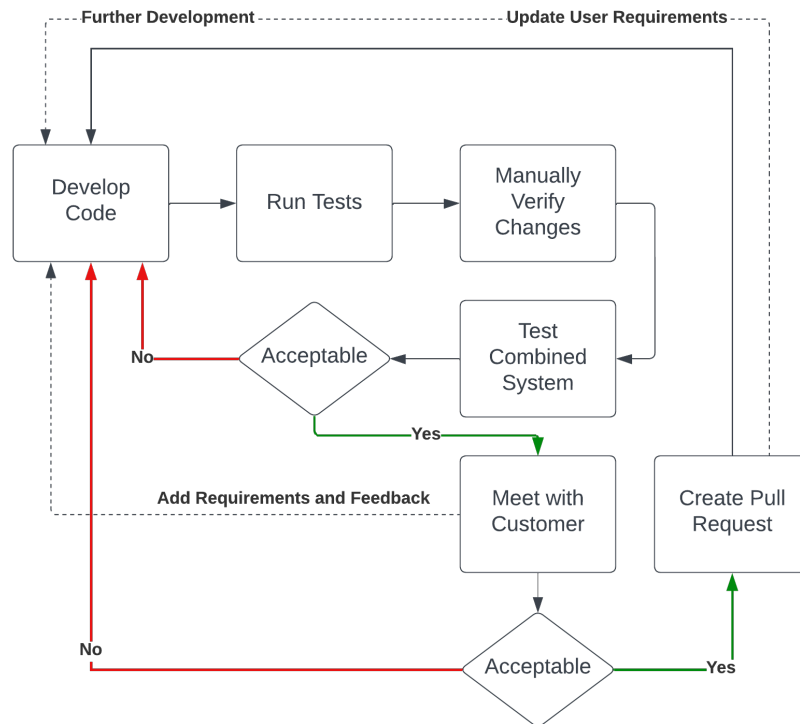


Figure 6.2: Testing Development Process

Through this process, we ensure that all of our new features and changes work together and on the system as a whole. Since this is a public library, it is essential to ensure that any changes we make to the repository as a whole will not affect the system’s functionality elsewhere. Particularly since there may be users using the library, and if they were to update it, it would mean they are unable to use it. Finally, an additional check is that before making a pull request, we run the unit tests for the repository; if all pass, then this is one final indication that this may be acceptable to merge.

It is important to note that passing tests alone do not necessarily indicate that the system is working as intended. This means that we have to follow a rigorous testing process to ensure that the system is functioning correctly. Much of our code is located at the front end of the system, so any changes we make must be tested on a variety of elections with varying numbers of projects and voters. It is essential to test each

round of visualisations thoroughly, as there are many cases where visualisations may differ or scenarios where they may not work correctly. Therefore, we had to test each update thoroughly before implementing any changes to the main repository.

Another key point is that since we are a group, each working on different branches and features, we must ensure that this works as intended with each other's changes. Therefore, each time we joined branches, we would either merge or rebase the changes depending on the group's requirements. However, in some cases, if two different features had been implemented on the same file, we would have to thoroughly test one more time before pushing these changes. We did this in a similar manner to the process shown in Figure 6.2. However, these tests would be done locally until we were sure that the merged branches worked as desired; they could then be pushed to the remote repository so the group as a whole could test the system. This constant testing of each individual component and part of the system meant that we would minimise the cases where we submit a pull request to the customer where the changes are not functional or working as specified.

Finally, after each pull request, our customer would additionally test the system and interact with the visualisations. This means they can provide us with any additional issues or bugs in the system. Furthermore, since they have not developed the system, they may more easily identify any issues since they may not know the intended purpose of the visualisation changes.

6.1.3 Performance Testing and Bench Marking

In Chapter 5, we included benchmarking results for the decrease in the time taken due to our optimisations. Whilst these tests tell us the general performance of our system and consequently the achievement of the customer-facing requirements **R14**. To gain a more in-depth understanding of the complexity of the elections compared to the runtime of our algorithms and visualisations, we ran the test set of the elections - off of the Pabutools library. This gives us results for almost 1,000 elections, all of

varying complexity. Therefore, we are able to create a comprehensive idea of where any performance bottlenecks are created.

Increase in Processing Time

For testing this, we consider the optimised solutions to our implementation - as we discussed in Chapter 5; therefore, we will only briefly touch on this subject with a quick review of the summary. Figure 6.3 shows the distribution for our results post optimisation, the mean time taken for the rule with the visualisations applied was 152.1%. This achieves user requirement **R13** and verifies that the visualisations were run on time. Whilst there were, of course, some visualisations where the runtime was far higher than the original - at around 1,000% increase. Over 86% of the runtime increases were within an acceptable range. Finally, the maximum time taken to generate the visualisations was under 300 seconds, and being under five minutes long, this is a reasonable time to wait for a very large and complex election.

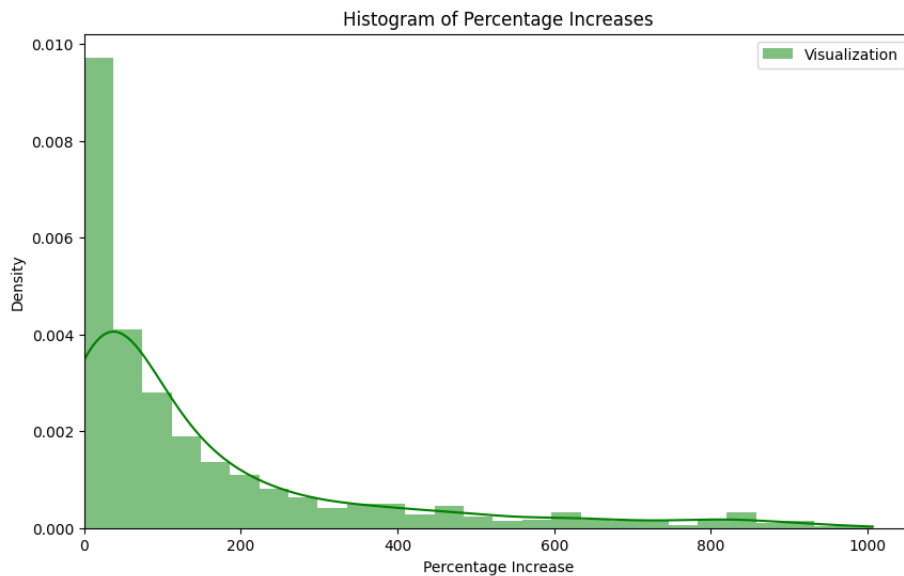


Figure 6.3: Percent Increase in Time for MES with Visualisations. Mean:
152.1%, 2.147s

On the other hand, with Greedy, our results both pre and post-optimisation were well into the realm of acceptability. As shown in Figure 6.4, the mean increase in time

with our visualisations in 3.004%, well into our acceptable time range. Even without our optimisation, this was still the case. However, both we and our customer felt it was prudent to ensure that, in all cases, the solutions were as optimal as possible.

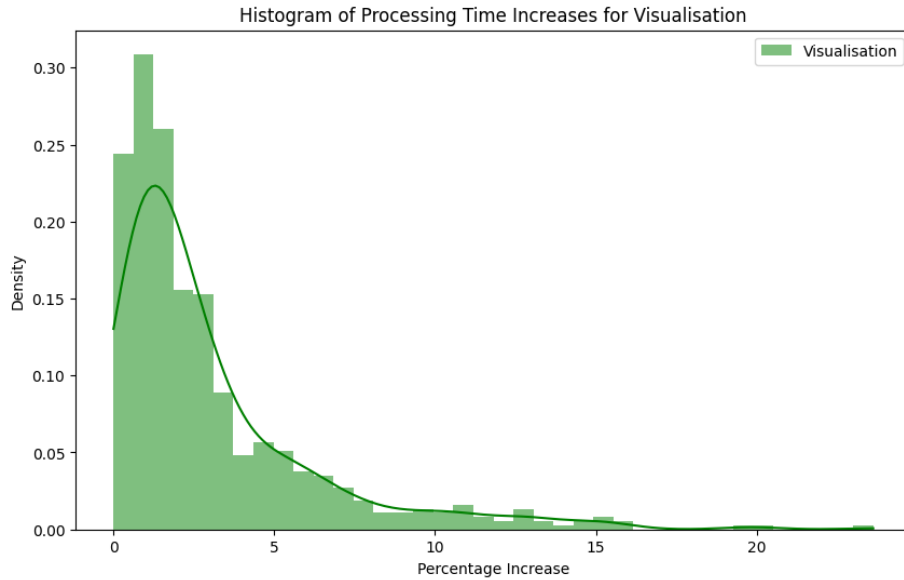


Figure 6.4: Percentage Increase for Time Taken for Greedy Visualisations. Mean: 3.004%, 0.00513s

6.1.4 Visualisation Time Performance

To further assess the performance of visualisation time and the increase in time taken due to visualisations, we plot these metrics against each other and conclude some comparisons on what dictates visualisation speed runtime. Additionally, we perform linear regression and calculate the correlation coefficients with the R^2 value and the P-value. This will give us an indication on the relationships between the different variables. Moreover, we will perform linear regression on the two variables giving us further insight on this.

MES Runtime Analysis

A reasonable assumption for the increase in runtime due to visualisations would be that both a larger number of voters and a larger number of projects would cause

a significant increase in the runtime compared to the original running of the rules. However, Figure 6.6 indicates this is untrue. Evidently, the number of voters in the election does not significantly increase the time taken to generate the visualisations when compared to the original runtime. This is indicated by the incredibly low R^2 and correlation coefficient values, as well as the p value being very close to 1. While surprising, this could be explained by the fact that computing the voter budget reduction was the most time-consuming part of the visualisation generations. On the other hand, the number of projects had a clear relationship with the increase in visualisations; moreover, the R^2 value for the increase is over 0.136; whilst this is a small value, this still indicates there is a relationship between the two. Moreover, considering that the vast majority of elections involve fewer projects, it is harder to get an exact indication. As we have stated previously, this aligns with our expectations since our performance bottleneck is in the computation of the pie charts.

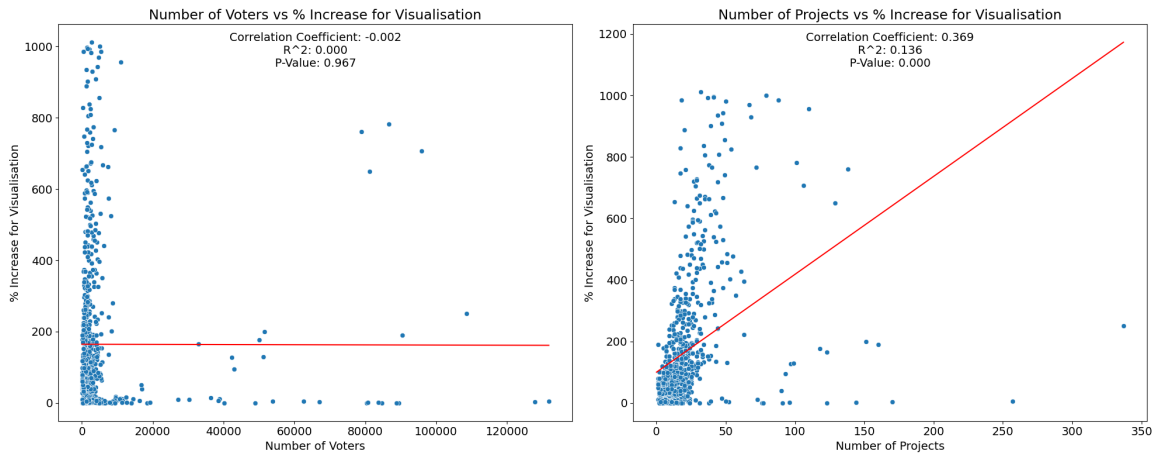


Figure 6.5: Percent Increase of Runtime With Visualisations Against Voters (Left) and Projects (Right)

In the case of the raw visualisation time, this follows more closely with our expectations. In general, it appears there is an increase of processing time for both the increase in the number of voters and the number of projects in the elections. Figure 6.6 shows the correlations between the two variables and the increased runtime with visualisations. In this case, both variables have an R^2 value and correlation coef-

ficients of over 0.5. When taking the 10 projects with the maximum visualisation times, they tended to have both a large number of voters and many projects. The median number of voters was 95,025 and the median number of projects was 162 - only 0.52% of the elections in our dataset have a number of votes and projects larger than this.

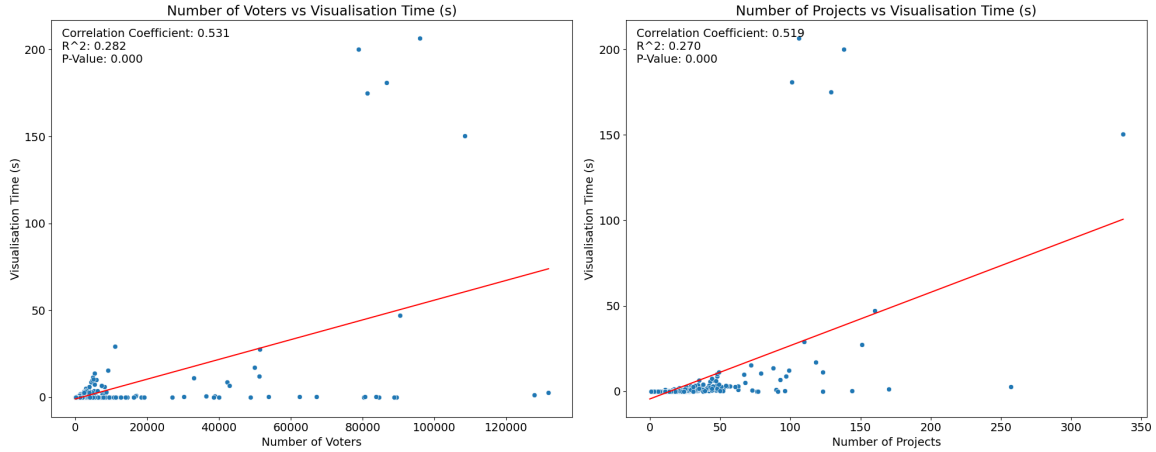


Figure 6.6: Increase of Runtime With Visualisations Against Voters (Left) and Projects (Right)

In conclusion, it appears that both the number of voters and the number of projects correspond to an increase in the time taken to generate the visualisations. However, it appears that only the number of projects significantly increases the time taken for visualisations relative to the time taken to run the base election rule. Implying that the number of projects is the performance bottleneck for larger elections. And as shown by the maximum visualisation times the effects caused by the larger number of projects are exacerbated when the number of voters is high. Therefore, more time should be allocated to produce visualisations when generating visualisations for elections used in larger municipalities.

Greedy Runtime Analysis

As with MES, we run each election with the visualisations to calculate the visualisation time correlations. Across all our elections, the increase in visualisation is incred-

ibly minimal at a mean of 3% increase processing time. However, we still performed an analysis similar to the one we did for the MES visualisations. This additionally yielded surprising results; as shown in Figure 6.7, there is zero correlation between either the number of voters or the number of projects.

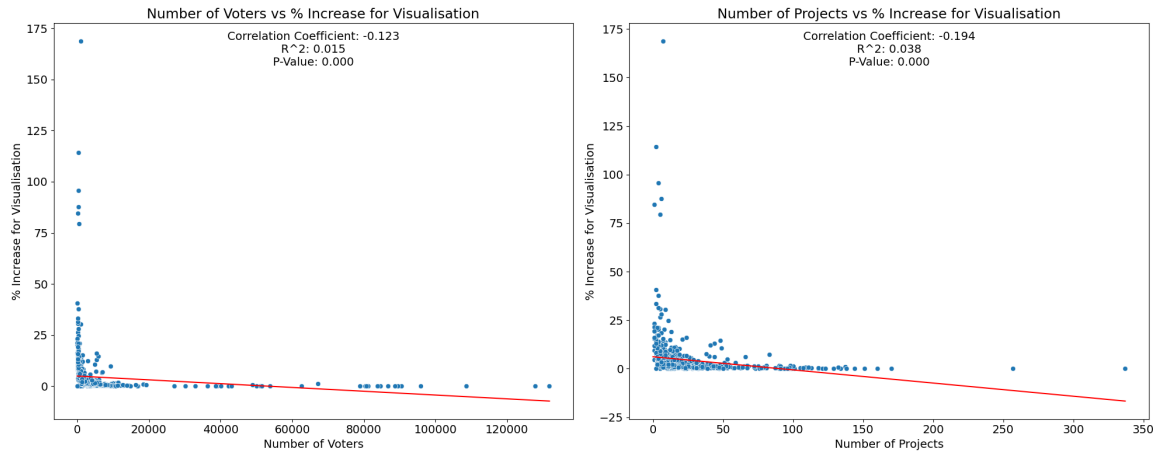


Figure 6.7: Percent Increase in Processing Time for Greedy Visualisations

On the other hand, as with MES, there was a stronger relationship between the raw visualisation times and the number of projects

6.2 User Testing

For our user testing, we primarily used close friends and family members to test our system. Most of these users had never heard of or were familiar with PB or the rules of Greedy or MES. The process for this is as follows:

1. State the definition of the rules to the user. As well as a piece of accompanying information from the corresponding Wikipedia page [53].
2. Ask them a series of questions about their understanding of the rules.
3. Allow the user to have 5 minutes with the Greedy Page, then ask them a set of questions.

4. Allow the user to have 15 minutes with the MES Page, then ask them a set of questions.
5. Finally ask a set of summary questions on the PB rules to gather their understanding.

The questions and the introduction to the form are available in Table A.1 and Figure A.1. Key objectives of this project are to increase the interpretability and explainability of PB election results. Given that MES, in particular, is far more complex than Greedy, we allow the user up to 15 minutes with the MES page to gather additional insight and understanding of the rule. The main basis on the success of the explanations is the increase of understanding to the rules to see if the user correctly understood the rule and understands the outcome of the election.

Mainly, we use numerical data - asking the user to give a number between 1 to 10, giving us a sense of the tool's effectiveness numerically. However, we also ask more difficult questions, such as the user explaining the rule in their own words. This tests the user's understanding of the topic since the user has to express this directly. Finally, we ask four technical questions to assess the users' understanding of the differing representation of voters and groups in MES compared to Greedy. The form for the full set of questions is available in the appendix.

Additionally, to verify we achieved our objectives defined previously, we ask the user a question such as "to what extent would you trust an election result given this tool as an aid?". This allows us to determine if the trust, understanding, and comprehension of a rule and the result is understood. Furthermore, we allowed users to address any complaints or issues with the page for future work.

6.2.1 Familiarity with Participatory Budgeting

First of all, we found it was necessary to check if users were previously familiar with PB. Whilst this was not the case in the majority of cases, some of our family members

who had heard about our project had looked up PB. This meant that, in some cases, they had a limited understanding of PB as well as some of the associated rules. Figure 6.8 shows the proportions for which voters are familiar with each rule and PB as a whole. Whilst it appears in some cases the users have heard of PB, they are not familiar with the rules beforehand.

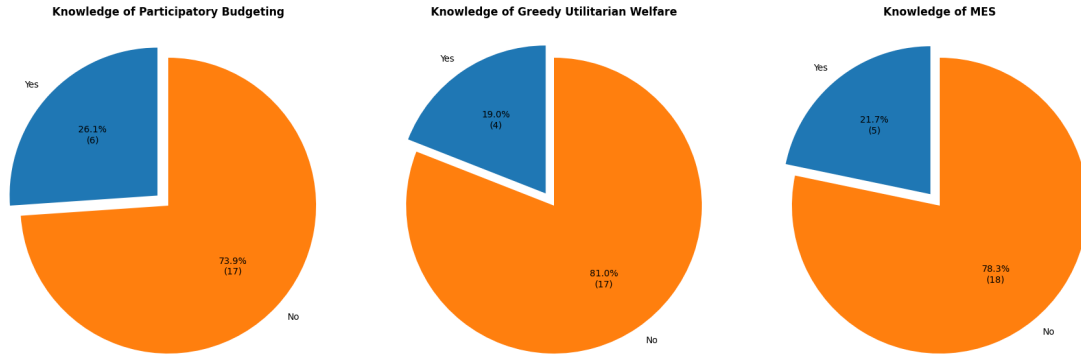


Figure 6.8: Proportions of Testers Familiar with PB, Greedy, and MES

6.2.2 Effectiveness of the Greedy Page

For the Greedy page, the main way we assess the quality of it, is by comparing the understanding of the rule before and after using our explanations page. Moreover, we ask the user if they would trust the outcome of the election if was it given to them by their local authority.

Firstly, Figure 6.9 shows the user’s understanding of the rule before and after using our explanation page. Note that the user was given a few minutes on the Wikipedia page to read through the information and gain a base level of understanding. From here, the user was able to view the election result for five minutes, going through the round-by-round and interacting with the page before continuing to answer the additional questions. Evidently, there is a significant increase in the understanding of the rule from all users and the mean score is now 8.88 rather than 5.04. This gives us a numerical indication that the visualisations allow the user to understand how

the rule works specifically.

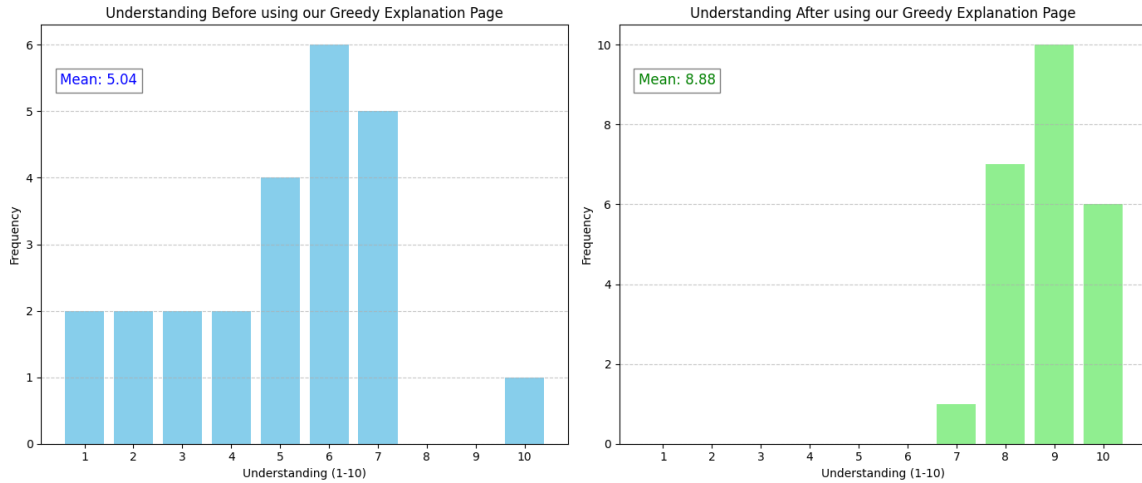


Figure 6.9: Comprehension and Understanding of the Greedy Rule

It should be noted that a large number of users did understand the rule initially, at least to a good level, when accompanied with explanations for the initial definitions of the rule. An example of a user with good comprehension was: *“The rule takes each of the projects available and sorts them by the number of people who voted for the project. These are then selected in order of vote count.”*. Whilst this is almost correct, it does not take into account that the rule is run on a satisfaction measure rather than purely the number of votes for the project. This was a typical example of the responses we received, with the majority not understanding the idea of the satisfaction measure.

Figure 6.10 shows the proportion of the testers who stated that if a set of projects were decided like this in their local area they would trust the outcome of the election given their understanding of this rule. This is very positive, since it shows the users trust the outcome of the rule, and therefore likely understand the rule sufficiently. Especially since they would be happy to support the result of an election such as this.

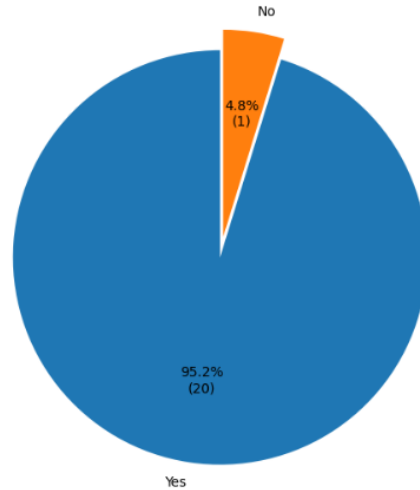


Figure 6.10: Question: *Would you trust a set of projects decided in your local area with this method?*

6.2.3 Effectiveness of the MES Pages

To ascertain the effectiveness of the MES pages, we need to consider more information than just their understanding. There are a wide range of aspects to MES that the user must understand to properly view the election outcome; consequently, we must test if our explanations and visualisations provide sufficient insight into these parts of the rule. First of all, Figure 6.11 shows the increase in understanding of the rule before and after using our visualisations when compared to the Wikipedia page. This clearly shows there is a significant increase in understanding of the rules when using our explanations. Even though this is still 5 out of 10, this corresponds to a user whose original understanding was a 1, therefore, this is still a significant increase.

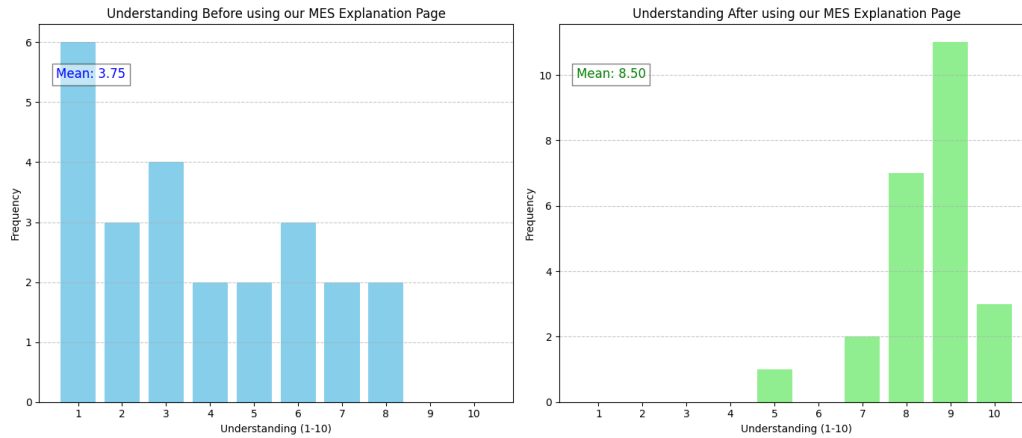


Figure 6.11: Comprehension and Understanding of the MES Rule. 1: No Understanding, 10: Perfect Understanding

Additionally, we check which parts of the page show a sufficient understanding of the MES rule and the election. For this, we checked whether the summary statistics alone, the summary page for the election, and the combination of a summary page and the round by round analysis contained sufficient information. Evidently, as shown in Figure 6.12, as expected, the summary statistics of the election do not provide enough information. However, there is a significant increase of support for the page summary. This shows the budget allocation as well as the projects selected and their order. This gives an overview of the results of the election as well as some simpler explanation of the results. Finally, it is clear that users felt the page summary, in combination with the round-by-round analysis, did provide sufficient insight into the details of the rule. Since this was intended in our design to allow more details where necessary, this shows the choice of a summary and round-by-round was a correct decision.

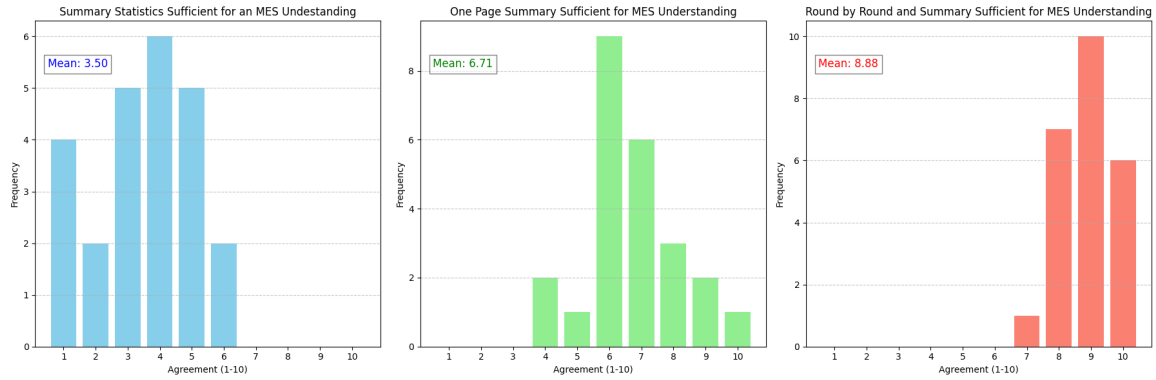


Figure 6.12: What Parts of the Pages Provide a Sufficient Understanding of the Rule

Since we included a range of visualisations for the rule. Particularly in explaining the voter flow, the effective vote count (with the reduction), and the reduction of the voter budget. We felt it was necessary to determine how successful these were at achieving our goals. Figure 6.13 shows the information for these figures. First of all, indicating how effectively the voter flow was shown in the chord diagram and the Sankey diagram. This aimed to show how many voters also voted for different projects. Evidently, on average the users found this very effective and they were able to see clearly what was happening at this stage of the rule. Additionally, the effective vote count charts were also very effective, they showed the users why each of the projects at this point may have been elected, and how other projects that were popular may have been discarded. Finally, we see the pie charts. Whilst this was not ranked as highly as the other two, in general the feedback was still positive. Particularly when the users had a better understanding of the MES rule. This aligns with expectations, since the pie charts showcases the budget reduction for each of the voters. This is a more technical part of the system, therefore the users who understood the nuances in MES better also found the information in the pie charts clearer.

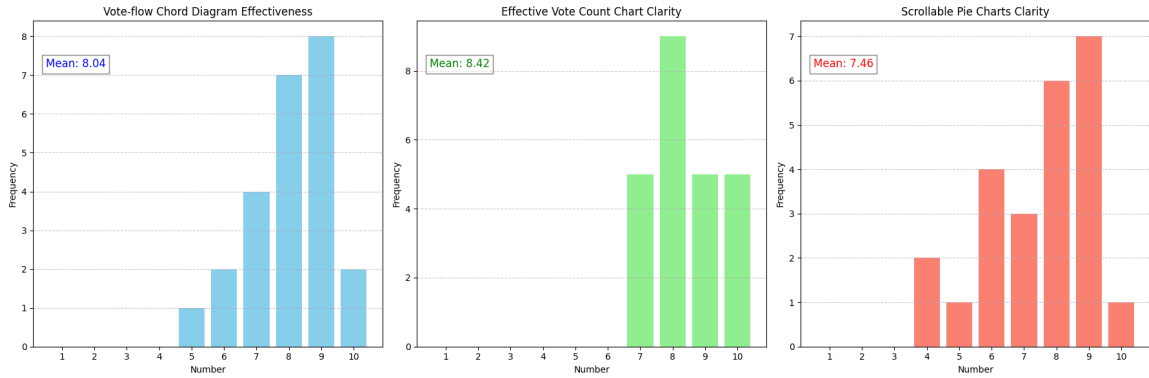


Figure 6.13: How Effective and Clear Are Each of the Figures in Our Visualisations

Following from these charts, we ask the users if they understand the more technical concepts within the rule. Figure 6.14 shows the result of this. Firstly, we consider the concept of an equal share - the idea that every voter has an equal budget available to them. This is an integral part of the MES rule, and ensures that each voter has a proportional representation in the system. Overall, the understanding of this is positive, however there are some shortfalls. This indicates that for some users a further explanation about the equal share that every voter has would improve the system. Second, we see the effective vote count chart, this indicates that each project has a vote count which adapts when the voters budget is reduced. Once again, whilst in general there is a good comprehension of this, there is one user who did not understand this at all, which is negative; they evidently also appeared to have struggled to understand the rule as a whole. What was positive however, was that users in general understood better that the support for a project may reduce as time goes on. This implies they understand the basis of the rule even if they do not understand the specific concept of effective vote count.

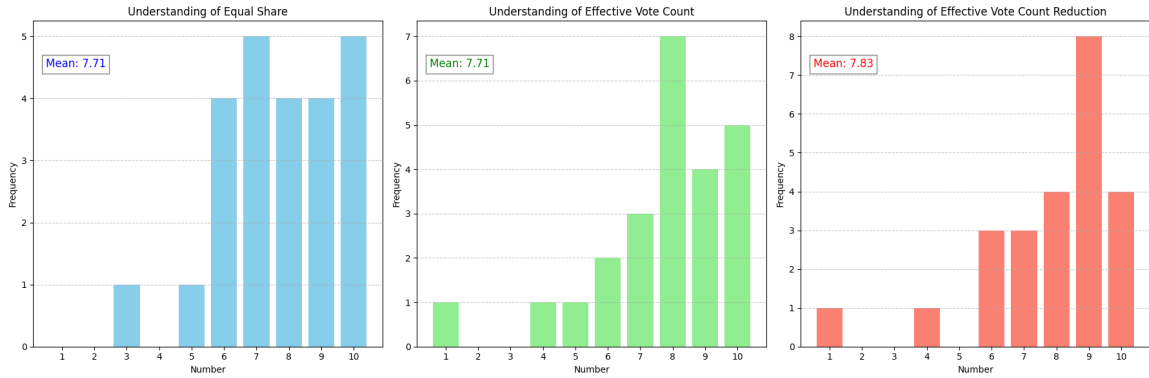


Figure 6.14: Understanding of MES Nuances and Specifics

Finally, since a key objective of our project is to ensure that people support PB and also trust the results of the rule. We checked that users supported this, and evidently this is very strongly the case. This shows that, overall, the project achieved this objective, particularly since there is no one who would not support PB in their local constituency and no one who would distrust an election result decided by this.

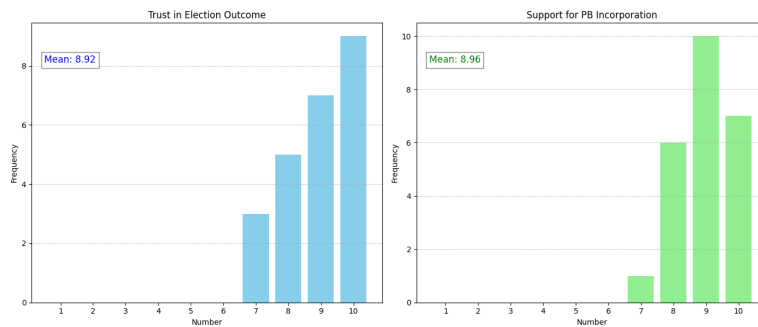


Figure 6.15: Support and Trust for PB

In conclusion, the feedback from users for this report was very positive. Whilst there were some users who did not understand some of the specific intricacies of the rule, they all said that their understanding of the rules improved - both for MES and Greedy. Moreover, they felt that our visualisations and the ability to go through the steps of the rule were very effective, particularly seeing how the budget available to voters was reduced as time went on. We established that having a summary page with a round-by-round analysis was very effective and that, in general, we provided the

information necessary for users to explain the election result properly. This feedback shows where our project achieved its goals and the way it could have achieved them better.

6.2.4 Improvements

While the vast majority of the users stated that the tools were very effective and helped them understand the rule and the reason for a specific outcome, users suggested a set of improvements.

- **Allowing Comparisons:** Whilst the users were able to view the elections side by side to get a comparison of the election. This was something they had to do manually. A common note was that a side-by-side view within the visualisations themselves would be ideal. Particularly if the user were able to toggle these on and off. Since we only have two rules implemented at the time, this is still a relatively simple change and could easily be done with a budget allocation bar - this would show the difference between the rules directly.
- **Allowing more Interactivity:** Despite the fact the users are able to hover over the charts and view more information. Many users stated it would be interesting to be able to edit the metrics about different projects. Say for example, they could edit the cost of a project, this would mean a higher number of voters would need to support it (in MES) for it to be elected. This gives users further insight into the rules and their specifics or nuances. Note, however, that this would require access to a 'backend system'. However, for future use, this library could be used to create visualisations, and then the rule could run in the background while there is an update.
- **Additional explanations about specifics:** There are, of course, some metrics that are not trivial to understand, particularly for our everyday users. As shown by the graphs in Figure 6.14, some users appeared to struggle with the specifics of the rules for vote count reduction and effective vote count. If we

were able to link to specific examples of the rule or articles in depth, this may assist in understanding here.

- **Being able to create elections:** Some users stated they would like the ability to make their own elections. For example, creating an election with 10 voters, and 5 projects would show how the rules work at a very low level.
- **Ensuring different languages:** Since PB is currently implemented in countries outside of English-speaking countries, having the option to change the language allows users from those nationalities to read the pages without any additional work.

The feedback from users is invaluable for future work and advice. Ensuring that the system is tested by users from all backgrounds is vital for further development and ensuring that our work is effective. From a combination of the results of our user testing in our graphs and the specific feedback taken from users, we were able to deduce any strengths and weaknesses within our implementation accurately.

6.3 Testing Objectives

As part of the success criteria for the project, we created a set of test objectives. These were created as an additional way to measure the project's success. To say we have successfully achieved this, we require verification from user testing or from having fulfilled a requirement from Table 3.2. These are shown in Table 6.1.

Code	Achieved	Description
TO1	Y	We successfully built a set of tools that visualise the explanations of outcomes for PB rules. We ensure all relevant information required to understand the project selection is present. In the case of MES, we added a summary page which allows the user to view the overview of the election. In case more detail is required, the user can now select a project on the summary page and go to the round-by-round analysis for the election. Moreover, we verify that we achieved this, as shown in Section 6.2. Evidently, the vast majority of users significantly increased their understanding of the rule and were provided with sufficient information to understand the election outcomes.
TO2	Y	As shown by the graphs in our user testing section. Our tool improves the explainability of the PB voting methods. Particularly in the case of MES, there is a more significant increase here since most users were far less able to understand MES initially than Greedy.
TO3	Y	To ascertain whether we successfully achieved this objective, we asked the users a question - to what extent would the user trust the results of the election were it presented to them. The average score here was 8.96, meaning we can conclude that this testing objective was also achieved.
TO4	Y	The tool has been successfully (and continuously) integrated into the Pabutools library. This was achieved with a series of pull requests approved by our customer and now available to the public using the Pabutools library. This additionally means that any contributor can further the work on both the visualisation side and the general research for PB.

TO5	Y	This objective was partially achieved. We originally looked into using clustering to look into methods to abstract groups of voters into a single one. Whilst we did not adopt this for our final solution, we did adopt a different implementation. When we had to consider only a subset of the projects, we took the projects with the largest overlap of voters. This means we ensure only the most relevant information is maintained for the user.
------------	---	--

Table 6.1: Testing Objectives

6.4 Fulfilment of Aims and Objectives

In this section, we consider the original set of requirements that we gave for the project and compare how successfully we achieved them at the project termination. Using this, we can directly measure the success of the project compared to our initial goals.

Our requirements were prioritised using the MoSCoW (Must, Should, Could, Won't) method. This very easily allowed us to create SMART objectives (Specific, Measurable, Achievable, Relevant, Timebound), which we assess in this section. For this, we take the requirement codes from Tables 3.3 and 3.1 and outline the extent to how we have achieved this: Y (Yes), N (No), P (Partially). From here, we can assess what proportion of deliverables were achieved and any adaptations to the original work plan.

Code	Achieved	MoSCoW	Description
R1	Y	M	We developed a set of rules which help visualise the outcomes for PB rules.
R2	Y	M	The tool was embedded into Pabutools. And integrated with frequent pull requests.
R3	Y	M	We added an additional parameter for visualisations, which otherwise do not adapt the base class.
R4	Y	M	The runtime of the rules is unchanged without the visualisations. Additionally, even with the visualisation set to true, the runtime of the rules is unaffected.
R5	Y	M	A new set of classes were created from scratch which allow the storing of of additional information required for the visualisations.
R6	Y	M	We adapted this approach since we established that we require the rule to complete and provide us with all information before creating the visualisations. Therefore, we store all required information during the rule run and then begin the visualisations.
R7	Y	M	We explain the outcome of the MES rule with a round-by-round analysis and a summary page. This allows the user to have multiple levels of details, ensuring the page is both

R8	P	S	We implemented the Greedy Utilitarian Welfare rule visualisations. Additionally, we allow for the extension to further rules very simply. Implementation of Phragmen's Voting would be further work for the project.
R9	N	C	That was not implemented.
R10	Y	C	To show the budget allocation, we included a summary page. This gives a project-by-project breakdown of the budget allocation. And outlines where projects were selected or rejected.
R11	Y	C	For our MES implementation, the user can go through round by round and view each section of the election. From here, they are able to hover over bars in a bar chart or flow on the chord diagram. This shows the additional information to the user and allows for interactivity. In the case of the summary page, the user is able to hover over the budget allocation and view where the budget has been allocated.
R12	N	C	Whilst this was not implemented directly, the two visualisations can be run simply after one another. These can then be displayed side by side to view the election results.
R13	N	S	After testing a wide range of different methods for clustering, we decided not to adopt this for our final solution due to the time taken to successfully calculate clusters in elections.

Table 6.2: Achievement of Functional Requirements

As is evident by the table above, in the majority of cases, our requirements were achieved. In particular, in all cases, our Must objectives were achieved. These were the basic requirements to ensure that this project was successful, all achieved to a high standard. Moreover, the majority of our Should and Could objectives were achieved; however, not for all cases, including **R9,R12,R13**. The most notable objective we did not implement is **R9**, since that would have been an additional rule that would have been implemented and therefore explained more effectively, and consequently furthered the explainability of PB rules.

Now, we consider the non-functional requirements of the project to gain further insight into what we have and have not achieved in our end results. For the following table, when we refer to **R14.M**, **R14.G**, we mean for MES and Greedy, respectively.

Code	Achieved	MoSCoW	Description
R14.M	Y	S	This requirement was achieved for 86% of elections tested on. There are, of course, cases where this is not possible, where there is a combination of lots of projects and lots of voters where voters vote for larger numbers of projects at a time. The mean runtime of the visualisations is around 150% of the runtime of the rule.
R14.G	Y	S	This requirement was achieved with 100% of elections tested on. There was a mean runtime increase of around 3% when using our visualisations.
R15	Y	M	As shown by the plots in the section 6.1.4, the runtime scales very well for larger elections both regarding the number of projects and the number of voters.

R16	Y	M	Our visualisations pages are very intuitive to use. Using common UI principles, and an intuitive UI design, the created visualisations are trivially navigable. Additionally, we have created additional documentation for the Pabutools library meaning a user who wants to generate the visualisations can easily do so.
R17	P	C	Whilst the visualisations are not designed for mobile devices, we decided that the visualisations should be designed for computer use since there are a large number of graphs and accompanying text which require the user to have a large screen for a clear overview. Moreover, the pages are interactive, allowing the user to hover over the charts and tooltips and view further information.
R18	Y	M	All data is accurately presented. We pass information directly from the rule that was run, and verify through testing that the calculations are accurate.
R19	Y	C	The classes we created follow a clear structure that can be extended to different types of rules easily. Additionally, the way templating is done can be inferred very easily from the existing implementations for each rule. Much of the code can be re-used from rule to rule - for example, voter flows or project vote counts.
R20	M	N	This was not implemented.

Table 6.3: Achievement of Non-Functional Requirements

For our non-functional requirements, we implemented all but one - ensuring that the visualisations are accessible for disabilities. Whilst this was a must requirement, this was also not integral to the effective use and running of the tool. Therefore, we prioritised additional features and rules rather than implementing this change. This is, of course, something that should be extended for future work. Finally, we will compare the objectives from Table 3.1 with to the extent that we have achieved them, to view our projects success as a combination of the features implemented as well as the objectives achieved.

Code	Achieved	Description
O1	Y	Our user testing indicated that the tool explains and displays all relevant information and ensures that the user can view everything they need - for both Greedy and MES visualisations.
O2	Y	The visualisations for both MES and Greedy ensure that they improve their understanding of the rule. The user feedback indicated that they understood the rules very well after using the page, as well as understood why the outcome happened.
O3	Y	The users stated that they trusted an election's results and would support PB elections in their local area. They did, however, mention they preferred that MES be the rule that dictates the selection of projects due to the better representation of the voters.
O4	Y	The tool is available on the Pabutools library with accompanying documentation. This means any person with access to Python is able to use the resulting visualisations. Most notably, they provide some part of a process that allows the explanation of why a result occurred and closes the gap between the additional explainability.

O5	P	The tool does limit the amount of data available to the user as to ensure they are not overwhelmed and the visualisations look effective. Additionally, we filter out further projects by taking the others with the relative vote count (the highest number of votes for projects other than the one selected that round that the voter also voted for).
O6	Y	For our MES implementation, we have a summary page which gives a higher-level overview to the user. This includes the breakdown of the budget allocation and the order in which projects were chosen. Then, they can view the round-by-round analysis of the rule run. Moreover, for the Greedy visualisations, the user can view the graphs in an overview and filter through them in ways they like for more detail. Finally, there is a round-by-round analysis, which goes through cases in detail where the user may not understand why a particular project was chosen.

Table 6.4: Achievement of Project Objectives

Robustness, Accuracy and Correctness Testing

We use various methods to measure the robustness, accuracy, and correctness of the visualisations we generate. First of all, to ensure that the visualisations work on any type of election in the ‘.pb’ file, we ran the visualisations on all of the elections from the Pabulib site. This ensures that regardless of the outcome or type, the visualisation works correctly in these scenarios. Moreover, when creating sample elections within Pabutools, we can test lots of scenarios - the projects may not have any description or be missing data that a typical election file would have. This ensures once again that the system works regardless of input. Since we need to verify that the visualisations work correctly additionally (the visualisations can generate without causing an error

within Python), we are required to manually go through the pages and ensure that there are no issues in any of the visualisations or parts of the page. In the case of an election created within Pabutools in Python, the lack of information will mean only the available information will be displayed.

Conclusion

In conclusion, when comparing the requirements we originally set to the ones we achieved, we can conclude that the project has been a success. Whilst there are a few objectives we did not implement, these are non-integral to the success of the project and were typical could or should objectives. Moreover, the project still achieves our goals. We can clearly see through our user testing that our project objectives were achieved for the rules we implemented. They allow for a range of details, simplify the PB voting explanations, and ensure that users trust and understand the outcome of an election. Whilst this is the case for MES and Greedy, this project would have been more of a success if we had produced visual explanations for a wider set of rules. Unfortunately, time did not permit this. However, we did lay the foundation for providing further visual explanations and a framework to do this repeatedly for the Pabutools library.

6.5 Limitations

Alongside the goals reached in the development of the project, there are limitations. We outline these in the following list, alongside reasons for their relevance.

1. **Comparison Between Voting Rules** - Our visualisations do not directly allow the comparisons of voting rules on the same page. This was a key bit of feedback provided to us by the user testing. Outcome comparisons would help users improve their understanding in areas where different rules select different projects. Key examples where this would work effectively are on a summary page, showing the budget breakdown for the different rules. This might illustrate the benefits of MES versus Greedy, showing that voters from

different groups are better supported or that the voters are better satisfied in the election. While this was something that we considered implementing, the customer - later on in the project's timeline - preferred that we not focus on this directly but rather on the implementation of explanations for the Greedy rule. This is, however, future work that should be considered and developed, as discussed in the following chapter.

2. **Visualisations for Other Voting Rules** - Visualisations are currently only available for the MES and Greedy voting rules. To ensure the completeness of the package, the visualisations should be built in the future to cover all of the rules implemented on the Pabutools site.
3. **Support for Non-additive Utility Functions** - Our Greedy visualisations currently only work when using an additive utility function. This means we can only generate the elections for some utility functions - Cost Sat, for example. Future work should cover this for all sorts of utility rules, as with works for our MES explanations.
4. **Labels in the Budget Allocation Chart** - From user feedback, our budget allocation chart could contain numerical labels that display important information in the graph. This would improve the chart's clarity, as well as help users absorb the information contained in the visualisation with just a glance. Examples of numerical labels that could be added to the graph include the cumulative cost of the projects at the end of the election and the election's budget.
5. **Visualisations of Voter Satisfaction** - Our visualisations are used to explain the outcome of the election rather than visualise statistics about the results - the cumulative satisfaction of the rule. Further visualisations include the location of the projects, the different groups, and the locations of voters who had the highest satisfaction. Having an all-inclusive resource that provides outcome visualisations and statistics as well as an explanation for the outcome would be valuable, as it would provide a single resource for all visualisations.

6. **Construction of Dummy Elections** - Being able to construct ‘mini-elections’ within the pages directly - such as the ones from Pref.tools [40] - would be very useful. This would allow users to update the information given by a sample election and create their own toy example (e.g. an election containing 10 voters and 5 projects). Allowing the construction and execution of custom mini-elections on various voting rules enables users to test what decisions these rules make in different scenarios, potentially aiding them in understanding of how these rules work.

Chapter 7

Project Management

The aim of this chapter is to discuss how the project was managed as a whole. This includes any issues and difficulties that were faced during the project. At the end of this section, a discussion on the legal, social, ethical and professional issues is included - demonstrating that our project was conducted responsibly.

7.1 Project Methodology

Throughout this project, an Agile methodology [54] was utilised as it was anticipated that plans and ideas would change over the course of the project. This is especially the case given the nature of this project due to the high possibility of requirements, priorities and scope changing unexpectedly. The Agile methodology used contained a mixture of elements from the Kanban and Scrum methodologies, taking advantage of the features that were most suitable for the project from each framework.

Kanban is an Agile methodology aimed at visualising tasks, centered around using a Kanban board to keep track of these. From this methodology, a Kanban board was used to assist in distributing tasks and keep track of who was assigned what. This approach has been known to improve visibility and productivity [55]. The Kanban board used throughout this project was included in Notion [56], an application that was used extensively to keep track of the project. See Figure 7.1 for an example of

how a Kanban board was utilised in one of our sprints.

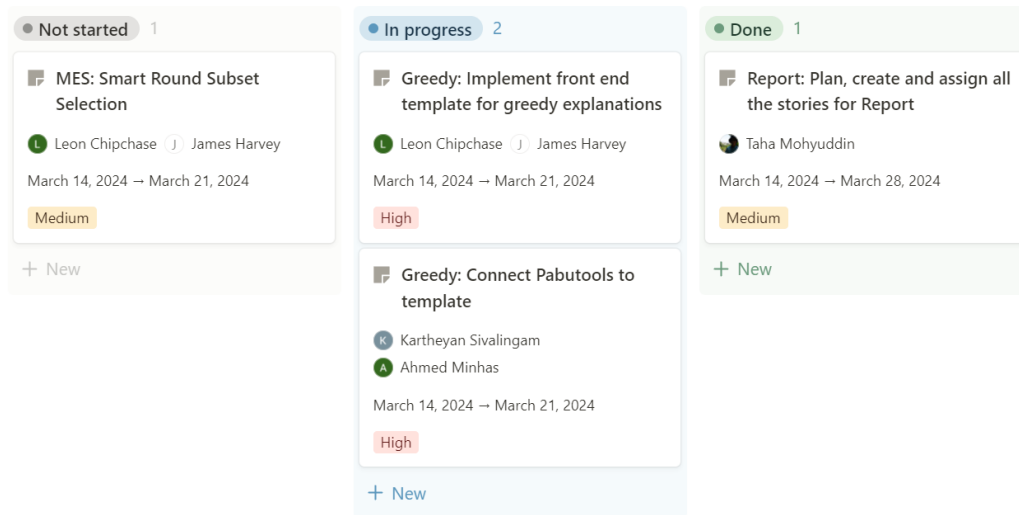


Figure 7.1: An Example of how the Kanban Board was used within a Sprint Cycle

In contrast to Kanban, Scrum is an Agile methodology where incremental versions of a product are delivered in short timelines through the use of sprint cycles. This approach has been linked to higher productivity, faster delivery and higher quality work [57]. A two-week sprint cycle was used for the project in order to deliver a product that satisfied the requirements set. See Figure 7.2 for an example of how a two-week sprint cycle was conducted. As the diagram shows, the sprint cycle's first phase involves sprint planning. This took place at the end of the stakeholder meeting from the previous sprint and involved picking features to implement during the upcoming cycle and assigning people to complete these tasks. The next phase of the sprint involved implementing the new features, with weekly stand-up meetings conducted during this part of the sprint to discuss the progress made and tackle any issues affecting productivity. Following this, a meeting with the stakeholder (in this case, the customer) took place, where current changes were presented to the supervisor and customer. These stakeholders would then provide feedback, and the backlog would be adjusted accordingly. The penultimate phase evaluated the completed sprint's effectiveness and made changes to future sprints if necessary. On

top of this, any feedback from the stakeholders during the stakeholder meeting were addressed. If it were required, a pull request would be submitted to the Pabutools repository once the customer was satisfied with our current implementation. Finally, planning for the next sprint would begin at the end of the stakeholder meeting.

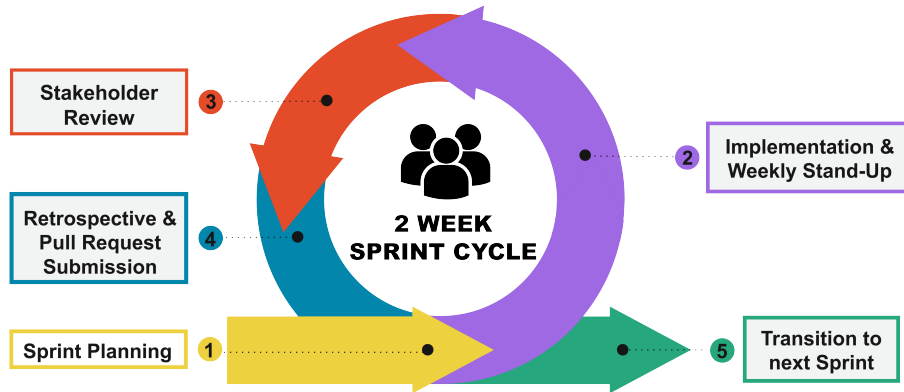


Figure 7.2: The Summary of a Typical Two-Week Sprint Conducted During the Project

7.2 Planning

In this section, we discuss how the project was planned in order to meet the requirements and deliverable deadlines set.

7.2.1 Feasibility Study

A feasibility study “is an assessment of the practicality of a proposed plan or project. [It] analyzes the viability of a project to determine whether the project or venture is likely to succeed.”[58]. We considered various aspects of the project to determine if it could be done, with some of the main topics including:

- **Technical Issues:** We determined we would not need any specialist software or code to do this. Our work was to be implemented onto Pabutools; therefore, ensuring there were minimal dependencies or additional libraries was key. As a result, we determined that we had all the required software available to us.

- **Project Scope:** Before the project specification, we defined a scope. Agreed upon by the team, our supervisor and our customer. This scope, along with our Objectives to define it, was SMART, and our requirements were prioritised using MoSCoW. Through our prioritisation, we additionally ensured that even in the case of significant delays or development issues, we could deliver a minimum viable product (MVP), ensuring the successful delivery of a working system to the customer.
- **Legal Issues:** While there were countless legal and social issues to consider (since we were building on top of Pabutools, which follows EU GDPR rules), the data we were working with adhered to these rules, meaning that legal issues should not cause any disruption to the project. Legal issues are further discussed later in this chapter.

Based on the considerations made in this section and the identified risks for which we had planned mitigation strategies, we determined the potential issues related to these aspects would not make this project infeasible. Therefore, after delivering and reviewing our project specifications, we determined that we could begin the system's implementation, design, and development.

7.2.2 Work Breakdown Structure

During the project's inception, a Work Breakdown Structure (WBS) [42] was produced to identify the boundary and scope of the project. This diagram broke down the work in the project into manageable deliverables, with each deliverable being further broken down into tasks and sub-tasks. These tasks and sub-tasks were then converted into stories, which were assigned to members of the team. By doing this, the project was made much more manageable. Figure 7.3 displays the full overview of the project's WBS. As can be observed from the diagram, deliverables 1, 2 and 3 are related to the application itself, while deliverable 4 includes non-software products (such as the project's report). The three application-related deliverables acted as milestones that marked significant points in the project, and they are as follows:

1. **Designs of the Visualisation Pages** - Involved researching how each voting rules worked, designing visualisations for each page, identifying generated by Pabutools to be saved and creating stories for deliverables 2 and 3.
2. **Framework for Saving Data in Pabutools** - Where new data retrieval classes (such as `BudgetAllocation` and `AllocationDetails`) were designed, documented and implemented.
3. **Implementation of the visualisation page generators** - Here, the visualiser classes were defined, the stories from deliverable 1 were implemented and different types of tests (including integration and user testing) were constantly conducted.

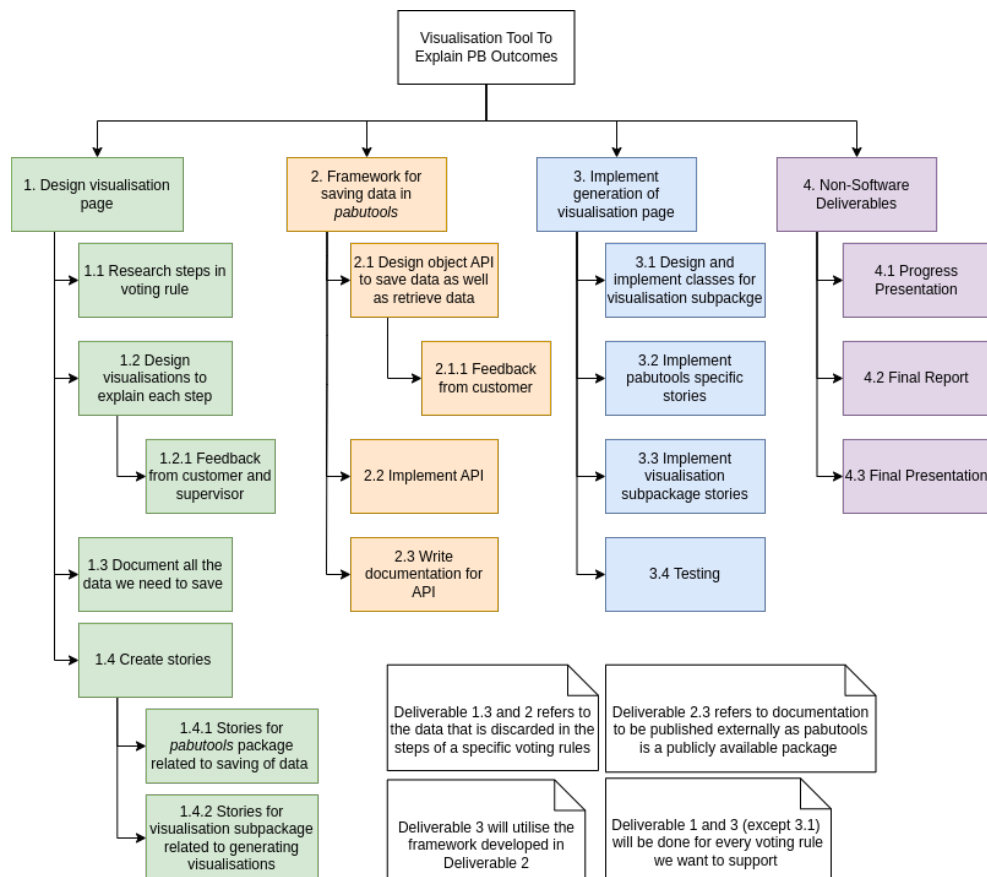


Figure 7.3: Diagram of the Project's Work Breakdown Structure Where Each Colour Corresponds to a Different Deliverable

7.2.3 Scheduling

Alongside the WBS, a Gantt chart - one integrated with Notion - was initially produced to schedule the project over the three university terms. Using a Gantt chart provided the team with a visual representation of the deliverables from the WBS in a useful timeline, making tracking the project's progress much easier. Furthermore, the timeline features dependencies between tasks of deliverables, allowing for better planning in terms of what tasks should be included in a sprint cycle. The full Gantt chart can be viewed in Appendix B.4.

As mentioned above, the Gantt chart enabled us to decide what tasks should be included in a sprint cycle. Once the tasks were decided, they were added to the Kanban board on Notion. In addition to this, each of these tasks was characterised with labels to indicate their priorities. The labels, along with a short description of what they cover, can be observed below in Table 7.1.

Label	Description of the Issue
High priority	An issue of high importance that must be dealt with immediately, otherwise it will halt the project's progress and may render the project faulty.
Medium priority	An issue of some importance but may not be as urgent as high-priority tasks. These issues should still be dealt with promptly, although there may be more flexibility in regard to deadlines or the consequences of delay.
Low priority	An issue that will not affect the project's progress and may not be required for the end product.

Table 7.1: Priority Labels and Their Corresponding Definitions

7.3 Organisation

This section explores how the project was organised, describing the various roles team members had as well as how the meetings were conducted before discussing the virtual control application used within the project.

7.3.1 Roles & Communication

Formally, the project consists of a team with five members who were given responsibility for different roles at the start of the project. The roles consist of:

Role	Name	Description
Project Manager	Taha	Responsible for keeping track of the project and ensuring that the project goals are achieved.
Admin Contact	Taha	Liaises with the Module Organiser and Project Supervisor, and informs them of any issues that may arise within the project.
Customer Contact	Leon	Liaises with the Customer to inform them of any updates, ask any questions, and schedule meetings with them.
Scrum Master	Kartheyan	Leads the weekly stand-up meetings and is responsible for managing the exchange of information between team members.

Documentation Manager	James	Responsible for managing and maintaining the documentation used throughout the project (deliverables, meeting notes, software documentation)
Meetings Coordinator	Ahmed	Responsible for finding and scheduling meetings between the team, the project supervisor and/or the customer. This is done by coordinating with the Customer and Admin contacts to organise meetings that were suitable for all parties involved.
Developers	All	In charge of writing the code and developing the application.

Table 7.2: Team Roles

For internal communication, Discord [59] was chosen as the group’s main platform, as the majority of the group already possessed previous experience with the many tools and features on here. These features include text channels (for organising discussions into sections such as research, development and report writing), team voice/video calls (to conduct online internal meetings), and screen-sharing (to support any pair programming done online) - features that helped us coordinate and communicate with each other throughout the project.

7.3.2 Meetings

Various meetings were conducted during the project in order to facilitate communication and collaboration between project stakeholders. Three types of meetings were held - each with a different purpose:

- **Weekly Stand-Up Meetings** - To discuss the current tasks being imple-

mented and resolve any issues encountered. These meetings are led by the Scrum master.

- **Bi-weekly Meetings with Supervisor** - To keep track of the project's progress and general direction.
- **Monthly Meetings with Customer** - Early on in the project's timeline, this was to bounce ideas off of the customer and discuss a realistic setup for the project. Later into the project, these meetings also involved presenting the latest version of the product to the stakeholders, in order for them to give constructive feedback.

Whilst some external meetings took place in person, most meetings were conducted online via Microsoft Teams [60]. This was because the customer lived abroad and hence, regrettably, was unable to meet in person.

Before meetings, the meeting agenda would be discussed as a team and then recorded into the meeting notes for reference. These topics - which included presenting the latest changes made to the visualisations, discussing updates to the Pabutools repository and addressing administrative matters - would then be discussed during these meetings. Alongside the meeting agenda, relevant resources for topics to be discussed would also be prepared in order to help facilitate discussions of these topics. For example, for meetings where the current version of the product was reviewed, the relevant visualisation pages would be generated in advance. During meetings, screen-sharing would then be used to display these pages to the stakeholders, allowing them to evaluate the latest changes and provide feedback easily.

During the meetings, rough meeting notes were written up - usually by hand. After the meeting, these notes were then cleaned up and recorded online into Notion. Having two separate copies of the meeting notes located in different locations introduced a recovery strategy for the records. If the Notion servers were down or the notes were inaccessible for any reason, having a separate - usually physical - copy of the

meeting minutes meant that we would be able to recover these notes and recreate the cleaned-up versions if needed. For examples of the online meeting minutes, refer to Appendix B.

7.3.3 Version Control

For the version control system for this project, GitHub [61] was used due to it allowing us to manage and keep track of our source code history. Furthermore, the repository for Pabutools was stored on GitHub, therefore, through using the same system we were able to fork the repository and easily submit pull requests from our repository into Pabutools. In addition to this, through using GitHub, we were able to maintain current, as well as older versions of the solution, enabling us to quickly restore code to a previous working state in cases of unexpected errors or other problems. This minimised any downtime caused by these problems and gave us a recovery strategy for recovering the source code of the project in the event that some code is lost.

7.4 Risk Management

This section aims to discuss how the team proactively prepared for different risks in the project, the risks that were encountered throughout the project and how these risks were managed to mitigate their negative impacts.

7.4.1 Original Risk Assessment Form

During the project's inception, a Risk Assessment Form was completed. This contains ratings for the likelihood and potential impacts of different risks that could occur during the project, as well as a mitigation plan to handle and deal with issues caused by these risks. This form can be seen in Table 7.3. The potential impact of a risk occurring was rated using a 3-point high-medium-low scale, while the likelihood of a risk occurring was rated using the 5-point Likert scale [62] shown below:

- Very Likely
- Likely
- Neutral
- Unlikely
- Very Unlikely

Description of Risk	Likelihood	Potential Impact	Risk Mitigation Plan
A team member is ill.	Likely	Medium	Assess the new situation, adjust plans accordingly and coordinate work to fit the new situation.
A team member leaves the project.	Very unlikely	High	Assess the new situation, adjust plans accordingly, coordinate work to fit the new situation and contact module organiser immediately.
Slow email replies from Supervisor or Customer.	Unlikely	Medium	Make sure to email ahead of time and contact module organiser if necessary.

A team member does not contribute to the progress of the project.	Unlikely	Medium	PM discusses the situation with the team member to find alternate arrangements in order for them to contribute effectively to the project. Otherwise, contact module organiser.
The voting rule is more complicated than expected.	Neutral	Medium	Lower the priority of that voting rule and implement the next voting rule from the priority list instead.
Being slowed down by Customer's PR approvals.	Unlikely	Low	Email customer or set up meeting with them beforehand to let them know the upcoming PR's are urgent.
Unable to design good enough visualisations to explain a voting rule.	Neutral	High	Gather feedback from customer and users to improve visualisations.
Under or overestimating the duration of implementing stories.	Neutral	Medium	Discuss every story estimate with all developers to come to a consensus.

Table 7.3: The initial Risk Assessment Form indicating the Likelihood and Potential impact of Risks

7.4.2 Encountered Risks

Throughout the project, some risks were unfortunately encountered. An example of this was when one of our group members, unfortunately, became unwell very close to the deadline of our first non-software deliverable - the specification. They were assigned with another group member to work on various sections of the specification and were unable to complete their tasks. As a result, through consulting the risk assessment form, the team decided to carry out the mitigation plan for team members falling ill. First, an emergency meeting was conducted to assess the new situation and plan accordingly. It was decided as a group to divide the work up again. Subsequently, the deadline was still met on time by preparing plans for potential risks at the start of the project and then quickly applying the relevant risk mitigation plan when an issue occurred.

Another risk that was encountered occurred mid-way through the project, when changes were lost due to a merge conflict. However, this was not foreseen from the original risk assessment form; thus, we did not have a mitigation plan in place to address the risk. The first step taken was consulting the group to determine if there was a working branch within the repository - if there weren't any working branches, the changes would have to be rolled back to a previous working state. Unfortunately, there were no working branches with the latest changes, and thus, the changes were lost and had to be re-implemented while the repository was rolled back. As a result of this event, the risk assessment form was revised to account for the new risk, which is described in the following section.

Outside of the two issues mentioned above, no other risks were encountered, and no interpersonal conflicts had occurred throughout the project. The risk assessment form did not require any more revisions after the merge conflict, as there were no additional risks that required planning and no additional risks that came up during the project.

7.4.3 Revised Risk Assessment Form

Due to the merge conflict that occurred, the risk assessment form was adapted to include this new risk, as well as a mitigation plan for it should it happen again. This additional risk can be seen below in Table 7.4.

Description of Risk	Likelihood	Potential Impact	Risk Mitigation Plan
A Github merge conflict occurs during development.	Neutral	Medium	Minimise the number of branches created, and only create them when required. On top of this, minimise the number of stories that cause such conflicts within the same sprint.

Table 7.4: The New Entry to the Original Risk Assessment Form, Indicating the Likelihood and Potential Impact of the Risk Occurring

7.5 Legal, Social, Ethical and Professional Issues

7.5.1 Legal Issues

- **Data Privacy and Protection:** Compliance with data protection regulations is crucial when handling the personal data of voters or participants. Ensuring all personal data is collected, stored, and processed legally is critical. Election data collected from Pabulib complies with European Union privacy law [63] by addressing data sets, including personally identifiable voter information. By asking the contributors to remove all personally identifiable voter information before submission and reserving the right to remove all personally identifiable

voter information [64]. In our user testing form, we ask users a series of questions and then collect and process this data. Before they fill out the form, we inform them the information they enter will be used by us and may be included in this report. Additionally, we ask that they do not enter any personal information into the form. This is shown in Figure A.1.

- **Accessibility Standards:** As our project aims to engage the entire community, ensuring that the visualisations and web presence comply with legal accessibility standards is crucial. We have ensured that the visualisations are accessible by providing textual descriptions summarising the data. The webpages also feature a standard HTML layout with well-structured headers and footers, complemented by intuitive navigation.

7.5.2 Social Issues

- **Representation and Bias:** Ensuring that the visualisations do not misrepresent or bias the data in ways that could mislead users or skew public understanding. This includes being mindful of how data is presented and avoiding visual elements that could lead to misinterpretation. To address this, we ensure transparency in the design and functioning of our algorithms. Additionally, we provide a link to educational resources that explain how to interpret visualisations to minimise misinterpretations and ensure that users better understand what the visualisations represent [7].

7.5.3 Ethical Issues

- **Transparency and Accountability:** Maintaining transparency about how data is collected, analysed, and visualised. This includes clear documentation of methodologies and any assumptions made during analysis. As the Pabutools library is open source, it is available for everyone to see the methodologies behind the analysis [13]. Clear documentation is also provided alongside the library to ensure transparency and trust in the user [65].

- **Consent and Anonymity:** Ensuring that data used in the project is either anonymised or collected with informed consent, particularly when dealing with sensitive information about individuals' voting behaviours or preferences. As mentioned in the section above, the data used from Pabulib adheres to European Union privacy law [63].

7.5.4 Professional Issues

- **Accuracy and Rigor:** It is paramount to maintain high standards of accuracy and scientific rigour in data analysis and visualisation. We have ensured that we have met professional standards through thorough planning and research, rigorous testing, regular reviews, and detailed documentation.

7.5.5 Public Good

- **Enhancing Democratic Engagement:** This project directly contributes to the public good by enhancing democratic engagement and transparency. By visualising PB results, the project makes it easier for community members to understand budgeting decisions, fostering greater trust and participation in democratic processes.

Chapter 8

Conclusion

In this project, we designed a tool that allows users to generate interactive web-pages that explain the outcomes of PB elections for different voting rules. This tool was built on top of the existing work implemented in the Pabutools Python library. The system was designed in a way to provide suitable explanations that can be understood by the general public. This informs them about the benefits of various voting rules and aids them in making better-informed decisions about which voting rules are more suitable for the elections they partake in. Throughout the project, visualisations for MES and the Greedy rule were produced, which were then organised and shown across three pages - two for MES and one for Greedy. These visualisations and features include simple overview charts, dynamic explanations and more detailed graphs used for in-depth round-by-round analysis. Thorough user testing demonstrated that understanding of the two rules improved after interacting with our web-pages and indicated that our visualisations and the ability to go through the steps of each rule were very effective. Most importantly, all of our aims were met and the customer was “happy with the end result, and definitely grateful for the time and investment [we] dedicated to this project.” As of right now, our solution is available to use within the Pabutools Python library.

8.1 Future Work

During the development of our project, we encountered both anticipated and unexpected software limitations. These limitations, though outside our initial focus, have led to the identification of interesting prospects for future work. The following discussion will delve into some of the most intriguing and feasible ideas identified for further investigation.

8.1.1 Advanced Visualisations

Advanced visualisations go beyond the fundamental visual tools we have implemented, which include integrating geographical representations and complex diagrams. These enhanced analytical capabilities could help users make more informed, strategic decisions.

8.1.2 Further Clustering Research

As explored in the investigation as to how clustering can be employed to abstract the data into manageable subsets, the fundamental issue with the time taken to run the clustering algorithms stopped our effort as it did not align with requirement **R14**. Further research into clustering would aim to identify significant groups of an election adaptively. The DBSCAN algorithm falls as the hyperparameters need to be tuned for each election instance to have meaningful results. Therefore, further research could go into algorithms where the hyperparameters do not have to be tuned whilst finding a dynamic number of clusters.

The HDBSCAN algorithm is an area of interest as it addresses the issue of using a single density threshold not properly characterising common data sets [66]. This algorithm provides a density-based clustering hierarchy representing all possible DBSCAN-like solutions for an infinite range of density thresholds and from which a simplified tree of significant clusters can be extracted [66]; this novel approach would address the issue of tuning hyperparameters.

Additional research could use information regarding each participant, such as age, voting method, and gender. With this information passed into the clustering algorithms, it could be determined whether or not there is a relationship between demographics. The clustering research aims to obtain clusters that could be used in the diagrams, minimising the amount of information that has to be cropped out and maximising the amount of information exposed to the user.

8.1.3 Additional Rules

Sequential Phragmen's

As seen in Chapter 1, Sequential Phragmen's is another voting rule where all participants start with a zero budget, which continuously increases. When a group of supporters has enough virtual currency to buy a project they all approve of, the project is bought. The rule stops when a project can be bought, but only by violating the budget constraint. The sequential Phragmen's rule has already been implemented in the Pabutools library. Therefore, creating a visualisation page for this rule would take a similar approach as the method of equal shares visualisation.

Envisioned visualisations for Sequential Phragmen's rule encompass a summary page akin to the MES page. This page would present all the chosen and rejected projects, offering additional insights into the selection process. Similarly, a round-by-round page, similar to MES, could guide the reader through the election, illustrating each round for every new project that can be funded. These visualisations would enhance the reader's understanding of the rule's application and outcomes.

For each round, a bar chart could be incorporated, displaying the funds available to the project's voters, the total voter's funds (excluding 'purchased' projects), and the project's cost. This graphic would enable the user to identify projects that were close to being funded and understand why some projects were prioritised over others.

It would serve as a visual aid, enhancing the reader’s understanding of the election process.

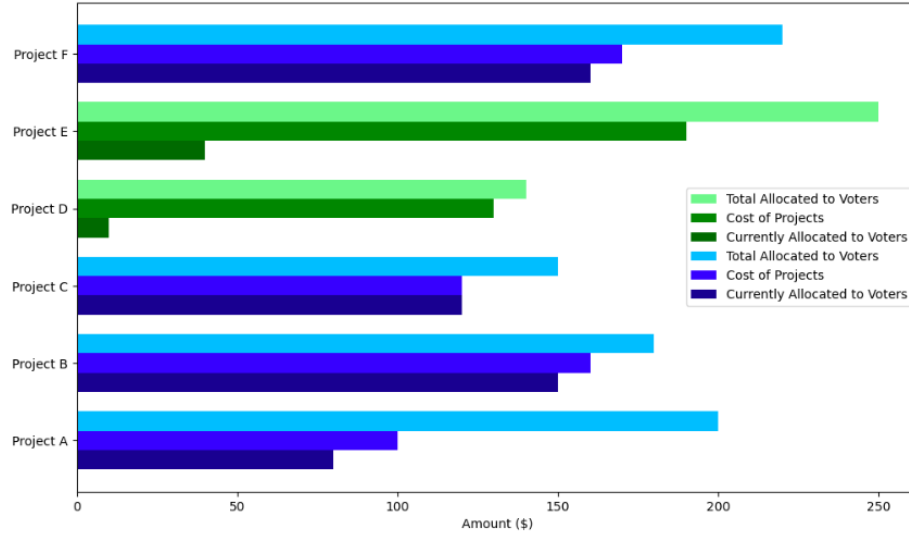


Figure 8.1: Example Graph Visualisation for Round-By-Round Analysis of Sequential Phragmen’s Rule

Figure 8.1 shows how this graph could look. Each project is associated with three bars. The top bar shows the total accumulation of virtual money voters of a project have received; the middle bar represents the project’s cost, and the bottom bar represents the sum of the available money between voters for the money. The green bars represent projects already selected for funding, and the blue bars represent projects still yet to be funded. In Figure 8.1, we see that ‘Project D’ and ‘Project E’ have been selected for funding. The next project to be funded will be ‘Project C’ as the available budget between its voters is equal to the project’s cost; therefore, it can be funded as long as it does not violate the budget of the election.

Welfare Maximisation

Pabutools has a voting rule implementation to maximise utilitarian social welfare. The outcome is computed via an integer linear program solver [67]. Visualising the process behind project selection in PB using the utilitarian welfare maximisation ap-

proach is challenging due to the inherent complexity of integer linear programming (ILP) used to determine optimal outcomes, particularly for those without a background in operations research or mathematics. This page's visualisations would be similar to the Greedy page, which summarises the selected projects. This rule can only be used for additive satisfaction measures, as there is no general solution for non-additive satisfaction measures. Therefore, this page would only be available for some elections.

Exhaustion Methods

As seen in Chapter 1, some voting rules do not return exhaustive budget allocations. There are different methods to render the outcome exhaustive. Different completion methods select different projects. Therefore, it is helpful to visualise the differences between the different exhaustion methods to see how the completion techniques vary.

Heatmaps

The Participatory Budgeting in New York City (PBNYC) initiative sees the community directly decide how to spend \$1,000,000 in participating council districts [68]. One website allows participants to submit ideas about how things could work better in their community by sharing them on a map [69]. The ideas are shared with volunteers who work with the council to turn ideas into proposals, with input from city agencies. The resulting map can be seen in Figure 8.2, with different proposals tagged under different categories across New York. If all elections use a similar approach, then the geographical information regarding projects will be available. From this, it would be possible to visualise projects that were chosen for funding using a heatmap. One method to geographically visualise the projects is by a heatmap.



Figure 8.2: Project Proposal Ideas for Participatory Budgeting in New York City

A heatmap is a geographical representation that uses varying colours to indicate the density or intensity of elected projects across different areas. Each point on the map reflects the location and budget of a project, with more intense colours signifying higher budget allocations. Thus, the heatmap provides a visual summary of where resources are concentrated within the city. An example visualisation can be seen in Figure 8.3, with the dark red areas representing areas where projects have been elected and the lighter colours representing areas where fewer projects were elected. Using heatmaps to visualise the distribution of funded projects, particularly under the MES voting rule, provides clear insights into spatial equity and resource allocation. This idea will be explored further in the following comparison section.

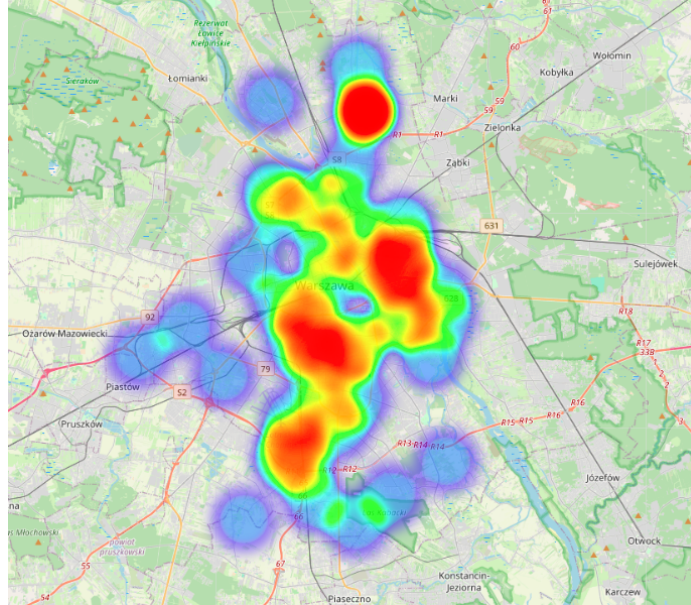


Figure 8.3: Mock Example Heatmap of Elected Projects in Warsaw

8.1.4 Rules Comparison

Currently, our solution provides in-depth analysis into one rule at a time. For users to compare different rules, they would have to manually alternate between different pages to analyse the differences. Further development could see a cohesive page directly comparing multiple voting rules. Some potential ideas for this page’s visualisations and diagrams could include the following:

Density Plot of Participant Satisfaction Across Voting Rules

The density plot in Figure 8.4 visually compares participant satisfaction between two voting rules: the Greedy rule and MES. The diagram shows the spread and concentration of satisfaction levels, which helps stakeholders quickly assess which voting rule generally yields higher participant satisfaction and how varied the responses are under each rule. Moreover, the graph highlights the diversity in satisfaction for the Greedy rule, with a broader spread of satisfaction scores compared to the more concentrated, generally favourable scores for MES. This helps users understand the consistency and predictability of participant reactions to each voting method.

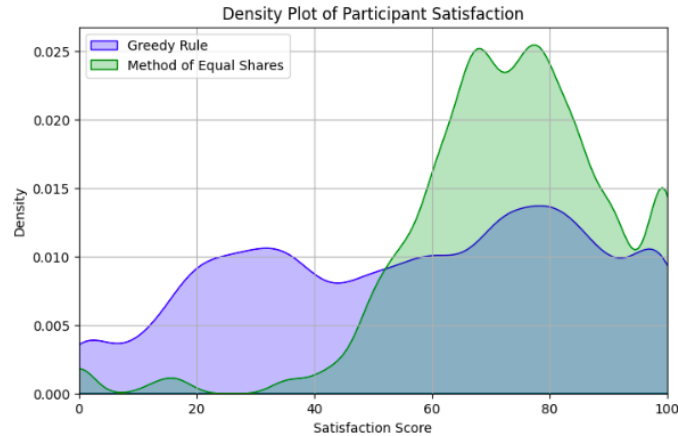


Figure 8.4: Comparative Density Plot of Participant Satisfaction for the Greedy Rule and Method of Equal Shares

Visualising of Project using Geographic Comparison

Figure 8.5 shows visualisations of projects in PB elections using GPS data. Each project is represented by two glued-together half-discs. The size of the left half is proportional to the project's cost, whereas the size of the right half is proportional to the total number of votes the project received. The figures compare the outcomes of the cost-utility variant of MES with the outcomes of the Greedy rule. Specifically, grey projects were not selected by either of the rules, green projects were selected by both, blue projects were selected only by MES, and red projects were selected only by the Greedy rule [70].



Figure 8.5: Visualisation of Projects in PB Elections Using GPS Data

Further development could see a comparison similar to the ones seen in Figure 8.5 between the Greedy rule and MES. This will give the user further insight into the difference in projects elected between each rule. From Figure 8.5, we observe that the MES algorithm selects more diverse and more representative sets of projects in terms of their geographic locations and supporters. This could aid the user’s understanding of why the MES is often chosen over Greedy.

We already have the required information to implement this feature; however, the geographical locations of each project are required, which are not always present in election data.

Visualisation of Projects using the Jaccard distance

Another approach offers significant benefits. By creating a map that illustrates voters’ preferences rather than the geographic locations of the projects [70]. For a given approval PB election, we first compute the Jaccard distances between all pairs of projects. Next, we use these distances to create a two-dimensional embedding.

An example for 16 elections can be seen in Figure 8.6 with the same colour key mentioned in the previous section. Once again, it is evident that MES picks a broader and more representative collection of projects in terms of their supporters. This provides another visualisation that strongly emphasises the fairness behind the MES voting rule.

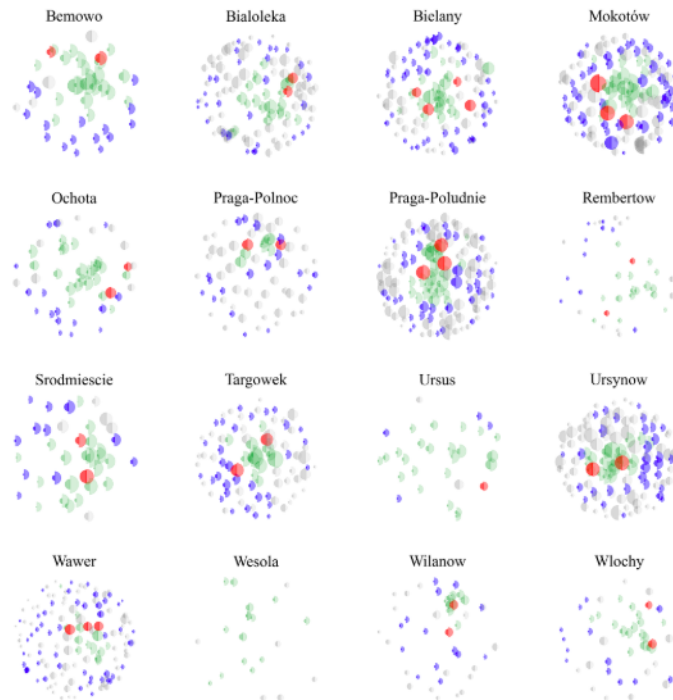


Figure 8.6: Visualisation of Projects in PB Elections Using Jaccard Distance

Bibliography

- [1] Anwar Shah. *Participatory Budgeting*. Public Sector Governance and Accountability. <http://hdl.handle.net/10986/6640>, License: CC BY 3.0 IGO. Washington, DC: World Bank, 2007.
- [2] Gareth Young. *Participatory Budgeting (Tower Hamlets, London, UK)*. Accessed: 20/04/2024. 2010. URL: <https://participedia.net/case/26>.
- [3] *Tower Hamlets, 'You Decide!'* Accessed: 21/04/2024. 2016. URL: <https://www.local.gov.uk/case-studies/tower-hamlets-you-decide>.
- [4] *Govanhill, Glasgow*. Accessed: 21/04/2024. 2016. URL: <https://www.local.gov.uk/case-studies/govanhill-glasgow>.
- [5] *Participatory budgeting in Brazil*. https://archive.epa.gov/international/jius/web/pdf/14657_partic-budg-brazil-web.pdf. [Accessed 25-10-2023].
- [6] Benny Lehmann, Daniel Lehmann, and Noam Nisan. “Combinatorial auctions with decreasing marginal utilities”. In: *Games and Economic Behavior* 55.2 (2006). Mini Special Issue: Electronic Market Design, pp. 270–296. ISSN: 0899-8256. DOI: <https://doi.org/10.1016/j.geb.2005.02.006>. URL: <https://www.sciencedirect.com/science/article/pii/S089982560500028X>.
- [7] Dominik Peters and Piotr Skowron. *Method of Equal Shares for Participatory Budgeting*. 2024. URL: <https://equalshares.net>.
- [8] Stanisław Szufa Dariusz Stolicki and Nimrod Talmon. *Pabulib*. 2022. URL: <http://pabulib.org/>.

- [9] Piotr Skowron Dominik Peters. *Tie breaking — Method of Equal Shares*. URL: <https://equalshares.net/implementation/tie-breaking>.
- [10] Dominik Peters. *Tie Simulation*. Accessed: [20/04/2024]. 2023. URL: <https://gist.github.com/DominikPeters/2208ca4c7c1464bc1d3956829195f20a>.
- [11] Dominik Peters and Piotr Skowron. *Comparison with other voting systems for participatory budgeting*. 2024. URL: <https://equalshares.net/benefits/comparisons>.
- [12] Dominik Peters, Grzegorz Pierczyński, and Piotr Skowron. *Proportional Participatory Budgeting with Additive Utilities*. 2022. arXiv: 2008.13276 [cs.GT].
- [13] Simon Rey. *pabutools - PyPI*. 2022. URL: <https://pypi.org/project/pabutools/>.
- [14] Dariusz Stolicki, Stanisław Szufa, and Nimrod Talmon. *Pabulib: A Participatory Budgeting Library*. 2020. arXiv: 2012.06539 [cs.DC].
- [15] Simon Rey. *Simon Rey — PhD candidate in Computational Social Choice at ILLC*. 2023. URL: <https://simonrey.fr/en>.
- [16] Yves Cabannes. “Participatory budgeting: a significant contribution to participatory democracy”. In: *Environment and Urbanization* 16.1 (2004), pp. 27–46. DOI: 10.1177/095624780401600104. URL: <https://doi.org/10.1177/095624780401600104>.
- [17] Nelson Dias. “Hope for democracy. 30 years of participatory budgeting worldwide”. In: *Epopeia Records* (2018).
- [18] Sahsil Enríquez Nelson Dias and Simone Júlio (Eds). “The World Atlas of Participatory Budgeting”. In: Oct. 2019.
- [19] Animesh Singh Rathore et al. “Participatory Budgeting in Brazil”. In: 2003. URL: <https://api.semanticscholar.org/CorpusID:154256664>.
- [20] Stephanie McNulty Brian Wampler Michael Touchton. “Participatory budgeting and well-being: governance and sustainability in comparative perspective”. In: *Journal of Public Budgeting, Accounting & Financial Management* 36 (2024).

- [21] Robert Zepic, Marcus Dapp, and Helmut Krcmar. “Participatory Budgeting without Participants: Identifying Barriers on Accessibility and Usage of German Participatory Budgeting”. In: *2017 Conference for E-Democracy and Open Government (CeDEM)*. 2017, pp. 26–35. DOI: 10.1109/CeDEM.2017.24.
- [22] R. W. Hildreth LaShonda M. Stewart Steven A. Miller and Maja V. Wright-Phillips. “Participatory Budgeting in the United States: A Preliminary Analysis of Chicago’s 49th Ward Experiment”. In: *New Political Science* 36.2 (2014), pp. 193–218. DOI: 10.1080/07393148.2014.894695.
- [23] Nimrod Talmon and Piotr Faliszewski. “A Framework for Approval-Based Budgeting Methods”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 2181–2188. DOI: 10.1609/aaai.v33i01.33012181. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/4052>.
- [24] Thorvald N Thiele. “Om Flerfoldsvvalg”. In: *Oversigt over Det Kongelige Danske Videnskabernes Selskabs Forhandlinger* (1895), pp. 415–441.
- [25] Edith Elkind et al. “Properties of multiwinner voting rules”. In: *Social Choice and Welfare* (2017).
- [26] Maaïke Los, Zoé Christoff, and Davide Grossi. *Proportional Budget Allocations: Towards a Systematization*. 2022. arXiv: 2203.12324 [cs.GT].
- [27] Svante Janson. *Phragmén’s and Thiele’s election methods*. 2018. arXiv: 1611.08826 [math.HO].
- [28] Markus Brill et al. *Phragmén’s Voting Methods and Justified Representation*. 2023. arXiv: 2102.12305 [cs.GT].
- [29] Luis Sánchez-Fernández et al. *Proportional Justified Representation*. 2016. arXiv: 1611.09928 [cs.GT].
- [30] Jesús A. Fiseus Luis Sánchez-Fernándezz Norberto Fernández-García and Markus Brill. “The maximin support method: an extension of the D’Hondt method to approval-based multiwinner elections”. In: *Mathematical Programming* (2024).

- [31] Haris Aziz, Barton Lee, and Nimrod Talmon. *Proportionally Representative Participatory Budgeting: Axioms and Algorithms*. 2017. arXiv: 1711.08226 [cs.GT].
- [32] Dominik Peters and Piotr Skowron. *Proportionality and the Limits of Welfareism*. 2022. arXiv: 1911.11747 [cs.GT].
- [33] Simon Rey and Jan Maly. *The (Computational) Social Choice Take on Indivisible Participatory Budgeting*. 2023. arXiv: 2303.00621 [cs.GT].
- [34] Elena Long. “Visualisations of Election Data”. PhD thesis. University of Plymouth, 2013.
- [35] Paul Martin Lester. “Syntactic Theory of Visual Communication”. In: (2011).
- [36] Stephen Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten*. 2nd. Oakland, CA, USA: Analytics Press, 2012. ISBN: 0970601972.
- [37] B. Shneiderman. “The eyes have it: a task by data type taxonomy for information visualizations”. In: *Proceedings 1996 IEEE Symposium on Visual Languages*. 1996, pp. 336–343.
- [38] Piotr Faliszewski et al. *Participatory Budgeting: Data, Tools, and Analysis*. 2023.
- [39] Stanisław Szufa Dariusz Stolicki and Nimrod Talmon. *PABUSTATS*. 2022. URL: <http://pabulib.org/wsgi/analysis/menu>.
- [40] Dominik Peters. *Pref.Tools: ABC Voting*. 2023. URL: <https://pref.tools/abcvoting/>.
- [41] Markus Utke. *Welcome to pabuviz*. 2024. URL: <https://pabuviz.org/>.
- [42] *A guide to the Project Management Body of Knowledge: (PMBOK Guide)*. Project Management Institute, 2017.
- [43] AltexSoft. *Functional and nonfunctional requirements specification*. Accessed: [20/04/2024]. Nov. 2023. URL: <https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/>.

- [44] Elena Long. *Election Data Visualisation*. 2013.
- [45] P. Riehmman, M. Hanfler, and B. Froehlich. “Interactive Sankey diagrams”. In: *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*. 2005, pp. 233–240. DOI: 10.1109/INFVIS.2005.1532152.
- [46] Nava Tintarev, Shahin Rostami, and Barry Smyth. “Knowing the unknown: visualising consumption blind-spots in recommender systems”. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing. SAC ’18*. Pau, France: Association for Computing Machinery, 2018, pp. 1396–1399. ISBN: 9781450351911. DOI: 10.1145/3167132.3167419. URL: <https://doi.org/10.1145/3167132.3167419>.
- [47] *ZingChart: A Comprehensive Charting Library for Data Visualization*. <https://www.zingchart.com/>. Accessed: 2024-04-12.
- [48] Jacob Thornton Mark Otto and Bootstrap contributors. *Bootstrap — getbootstrap.com*. <https://getbootstrap.com/>. [Accessed 25-10-2023].
- [49] Mingyue Fan et al. “Effects of Information Overload, Communication Overload, and Inequality on Digital Distrust: A Cyber-Violence Behavior Mechanism”. In: *Frontiers in Psychology* 12 (2021). DOI: 10.3389/fpsyg.2021.643981. URL: <https://doi.org/10.3389/fpsyg.2021.643981>.
- [50] Avcontentteam. *PCA: What is Principal Component Analysis & How It Works? (updated 2024)*. Apr. 2024. URL: <https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python/>.
- [51] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605.
- [52] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise”. In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*. 1996, pp. 226–231.



- [53] Wikipedia. *Combinatorial participatory budgeting*. Accessed: [02/04/2024]. Apr. 2024. URL: https://en.wikipedia.org/wiki/Combinatorial_participatory_budgeting.
- [54] Kent Beck et al. *Manifesto for Agile Software Development*. 2001. URL: <http://www.agilemanifesto.org/>.
- [55] Kanban University. “2022 State of Kanban Report”. In: (2022). URL: <https://kanban.university/wp-content/uploads/2022/10/State-of-Kanban-Report-2022.pdf>. (visited on 10/23/2023).
- [56] Inc. Notion Labs. *Notion*. 2023. URL: <https://www.notion.so/>.
- [57] Valpadasu Hema et al. “Scrum: An Effective Software Development Agile Tool”. In: *IOP Conference Series: Materials Science and Engineering* 981.2 (Dec. 2020), p. 022060. DOI: 10.1088/1757-899X/981/2/022060. URL: <https://dx.doi.org/10.1088/1757-899X/981/2/022060>.
- [58] Investopedia. *Feasibility study*. Accessed: [22/04/2024]. 2024. URL: <https://www.investopedia.com/terms/f/feasibility-study.asp>.
- [59] Discord. *IMAGINE A PLACE...* 2023. URL: <https://discord.com>.
- [60] Microsoft 2023. *Online Meetings*. 2023. URL: <https://www.microsoft.com/en-gb/microsoft-teams/online-meetings>.
- [61] GitHub Inc. *GitHub*. 2023. URL: <https://github.com>.
- [62] R. Likert. *A Technique for the Measurement of Attitudes*. A Technique for the Measurement of Attitudes nos. 136-165. Archives of Psychology, 1932. URL: <https://books.google.co.uk/books?id=9rotAAAAYAAJ>.
- [63] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation)*. Official Journal of the European Union. 2016. URL: <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.

- [64] Stanisław Szufa Dariusz Stolicki and Nimrod Talmon. *About*. URL: <https://pabulib.org/about>.
- [65] Simon Rey et al. *Pabutools: PB as easy as ABC*. URL: <https://pbvoting.github.io/pabutools/>.
- [66] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. “Density-Based Clustering Based on Hierarchical Density Estimates”. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Jian Pei et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 160–172. ISBN: 978-3-642-37456-2.
- [67] Simon Rey et al. *Rules — Pabutools*. URL: <https://pbvoting.github.io/pabutools/usage/rules.html>.
- [68] *Participatory Budgeting*. URL: <https://council.nyc.gov/pb/>.
- [69] *Participatory Budgeting NYC*. URL: <https://shareabouts-pbnyc-2018.herokuapp.com/page/about>.
- [70] Piotr Faliszewski et al. “Participatory budgeting: Data, tools and analysis”. In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence* (Aug. 2023). DOI: 10.24963/ijcai.2023/297.

Appendix A

Survey

User Testing for Election Visualisation

B I U  

This form is a feedback form for the *binary budgeteers*. This aims to gather whether the developed visualisations achieve their goals and improve the understanding of participatory budgeting.

The test is as follows

1. Ask the user whether they have heard of Participatory Budgeting (PB), as well as the rules
2. State the definition of the rules to the user.
3. Ask them questions about their understanding of the rules
4. Allow the user 5 minutes with the Greedy Utilitarian Welfare (GUW/Greedy) Visualisation Page
5. Allow the user 15 minutes with the Method of Equal Shares (MES) Page
6. Ask the user summary questions based on the pages

From here, we collect this data to assess the quality of our development against our fundamental objectives.

By submitting this form you agree that we have the permission to use the data you enter in our report - therefore, responses may be seen by the University of Warwick.

We do not collect any personal details, nor the email addresses when filling out this form. Please do not submit any personal information.

Figure A.1: User Testing Form Introduction

	Question
1	Have you ever heard of Participatory Budgeting?
2	Have you ever heard of Greedy Utilitarian Welfare (Greedy)?
3	Have you ever heard of the Method of Equal Shares (MES)?
4	Explain in your own words the PB rule - Greedy.

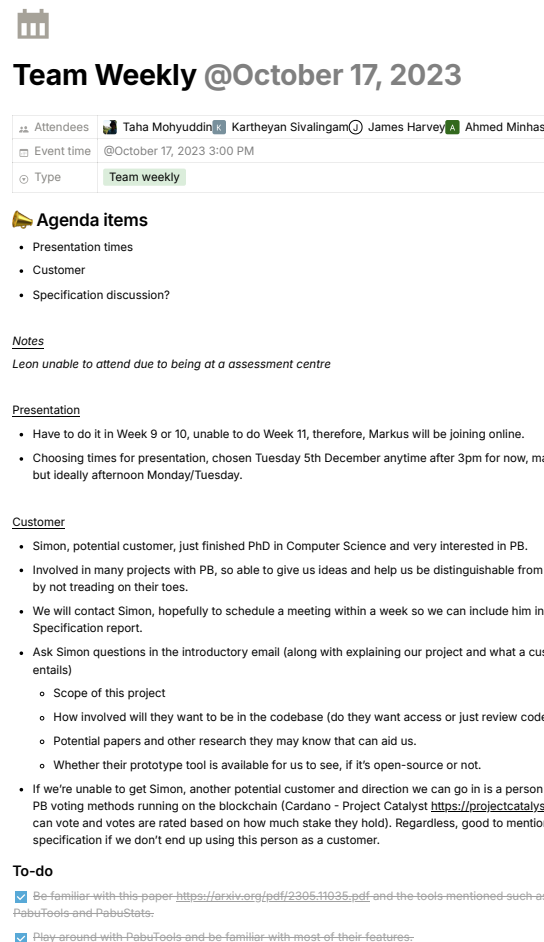
5	Explain in your own words the MES rule.
6	Give a number between 1 and 10 indicating your understanding of the Greedy rule.
7	Give a number between 1 and 10 indicating your understanding of the MES rule.
8	After you have explored both pages, to what extent do you understand the difference between MES and Greedy (1-10)?
9	To what extent do you understand the concept of an 'Equal Share' in MES (1-10)?
10	To what extent do you understand the concept of 'Effective Vote Count' in MES (1-10)?
11	To what extent do you understand the concept of an 'Effective Vote Count Reduction' in MES (1-10)?
12	Will the outcome of MES always be the same as Greedy?
13	Explain in your own words the MES Rule.
14	Explain in your own words the Greedy Rule.
15	Explain in your own words the benefits of MES over Greedy.
16	How well do you now feel you understand the Greedy rule (1-10)?
17	Would you trust a set of projects decided in your local area with Greedy?
18	How well do you now feel you understand the MES rule?
19	To what extent does the summary statistics at the beginning of the visualisation page give a sufficient understanding of the rule (1-10)?
20	To what extent does the summary page give a sufficient understanding of the rule (1-10)?
21	To what extent, using the round-by-round analysis and the summary page, do you completely understand the election and the rule (1-10)?
22	To what extent did the vote flow chord diagram and Sankey diagram show effectively the interaction between projects (1-10)?

23	To what extent do the effective vote count and effective vote count reduction bar charts show clearly why some projects may have been skipped despite higher vote counts (1-10)?
24	To what extent do the scrollable pie charts show clearly how each project being selected changes the funding available to the next projects (1-10)?
25	Were you provided the results of the election (the selected projects) and the provided visualisations, to what extent would you trust the outcome of the election were it provided to you by your local authority (1-10)?
26	The visualisations provided to you improved your understanding of the MES rule (Y/N)?
27	To what extent would you support the incorporation of PB at your local authority?
28	You would prefer MES to be implemented over Greedy at your local authority?

Table A.1: User Testing Questions

Appendix B

Management



Team Weekly @October 17, 2023

Attendees	Taha Mohyuddin, Kartheyan Sivalingam, James Harvey, Ahmed Minhas
Event time	@October 17, 2023 3:00 PM
Type	Team weekly

Agenda items

- Presentation times
- Customer
- Specification discussion?

Notes

Leon unable to attend due to being at a assessment centre

Presentation

- Have to do it in Week 9 or 10, unable to do Week 11, therefore, Markus will be joining online.
- Choosing times for presentation, chosen Tuesday 5th December anytime after 3pm for now, may be change but ideally afternoon Monday/Tuesday.

Customer

- Simon, potential customer, just finished PhD in Computer Science and very interested in PB.
- Involved in many projects with PB, so able to give us ideas and help us be distinguishable from other groups by not treading on their toes.
- We will contact Simon, hopefully to schedule a meeting within a week so we can include him in our Specification report.
- Ask Simon questions in the introductory email (along with explaining our project and what a customer role entails)
 - Scope of this project
 - How involved will they want to be in the codebase (do they want access or just review code?)
 - Potential papers and other research they may know that can aid us.
 - Whether their prototype tool is available for us to see, if it's open-source or not.
- If we're unable to get Simon, another potential customer and direction we can go in is a person involved with PB voting methods running on the blockchain (Cardano - Project Catalyst <https://projectcatalyst.io>), anyone can vote and votes are rated based on how much stake they hold). Regardless, good to mention in specification if we don't end up using this person as a customer.

To-do

- ☑ Be familiar with this paper <https://arxiv.org/pdf/2305.11035.pdf> and the tools mentioned such as PabuLib, PabuTools and PabuStates.
- ☑ Play around with PabuTools and be familiar with most of their features.

Figure B.1: Example of How the Meeting Notes Were Laid Out For a General Meeting with the Supervisor Exported From Notion



Team Weekly /w Customer @February 29, 2024

Attendees	Taha Mohyuddin, Kartheyan Sivalingam, Leon Chipchase, James Harvey, Ahmed Minhas
Event time	@February 29, 2024 2:00 PM
Type	Customer

Pre-read

- PR other group made

<https://github.com/pbvoting/pabutools/pull/11>

Agenda items

- Discuss new BudgetAllocation class that another group has implemented.
- Discuss progress made.

James and Leon - Round analysis page

- Added annotations to the graphs and included dynamic explanations for them.
- New dynamic visualisations.
- Graphs are now fixed in that the values on the graph now correspond to the correct values.
- The larger charts such as the chord diagram now show the 5 most common projects, so the chart can be a bit cleaner for a large election.

Moved on to progress Kartheyan made whilst they load up the new page

Kartheyan - BudgetAllocation class

- Working on following Simon's email as well as the other group's recent PR
- Changed MESIteration to MESProductDetails.
 - Call the iteration for each project being selected.
 - Add all projects to be considered and budget at the start of the round call.
 - When we go to the next project, get the selected project and append this iteration to all other iterations.
- Simon's not entirely happy with the implementation as there can be some elections with projects that never get considered until a very long time down the iteration, therefore, if we store all projects at the start, we are storing a large amount of unnecessary data, hence, why the other group (<https://github.com/pbvoting/pabutools/pull/11>) only adds projects that actually got selected or discarded.
 - Computing affordability is very expensive, so we want to minimise it as much as possible.
- However, within our visualisations, we require all projects to be stored so we can retrieve the effective vote counts. Simon's fine with this.
- We can still try and use the same logic as the previous groups by involving a flag if a project was discarded.
- Asked Simon if we should have unit tests that test each value of the iteration or simply check if the length of the iteration equals the number of selected projects.
 - Simon said the latter would be preferable.

Back to James and Leon

Now sharing screen

- If we hover our graphs, we can see tool tips and more information regarding the graph.
- Sankey chart now has an other selection to see the other projects not part of the 5 most common projects.

Taha and Ahmed - Page summary

- Connected page summary to the round analysis page.
 - So within the summary table, we can expand a row which includes a hyperlink to the round page and will eventually include other useful metrics that wouldn't fit in the table (such as average, max, min, etc).
 - The hyperlink opens the round analysis page on the specific round that was picked by the user on the summary page.

Next tasks

- We are able to push our visualisations whenever to pabutools, the most important PR is the MESIteration stuff.
- Kartheyan will modify MESIteration and submit a PR to Simon within the week.
- Leon will plan on getting the Greedy rule working.
- James will continue working on the round analysis page.
- Taha and Ahmed will try and add dynamic explanations and more useful stats and information on the page summary.
- Tentatively plan for a meeting from two weeks on - depending on if the PR gets accepted or not, if there are any issues with the PR, we will have a meeting next Thursday with Simon and Markus.
 - No issues, then internal meeting next Thursday.
- Fixing graphs on round analysis page

Figure B.2: Example of How the Meeting Notes Were Laid Out for a Meeting with the Customer Exported From Notion

Team Weekly /w Customer @October 24, 2023

Attendees: Taha Mohyuddin, Kartheyan Sivalingam, Leon Chipchase, James Harvey, Ahmed Minhas

Event time: October 24, 2023 3:00 PM → 3:40 PM

Type: Customer

4 more properties

Add a comment...

Pre-read

▼ Docs

Requirements set by Customer

The main idea for this project is to develop tools to visualise explanations for the outcomes of participatory budgeting (PB) rules. The tool will display all kinds of relevant information for anyone to understand how the final selection of projects has been made. In the following, I provide a (potentially partial) specification for the tool:

- The tool is to be embedded into the pabutools, a Python library for PB (<https://github.com/pbvoting/pabutools>)
- The implementation of PB rules within the pabutools will need to be tweaked so that they can output explanations on top of the result
- The modification made to the rules should not increase significantly the running time of the rules when only the outcome is requested (i.e., no explanation)
- A separate module will be created from scratch for the generation of the explanations
- This module will allow the user to provide an explained result as output by the pabutools rules and will automatically generate visualisations for the explanation (typically as an HTML file)
- The method of equal shares is the most important rule to focus on, though it would be good to have this tool working for all rules currently provided by the pabutools
- The end product will be integrated within the pabutools to be made available to all
- The end product needs to adopt the code specification of the pabutools (comments, code style, etc...)

Figure B.3: Initial Meeting Notes for Requirements Elicitation

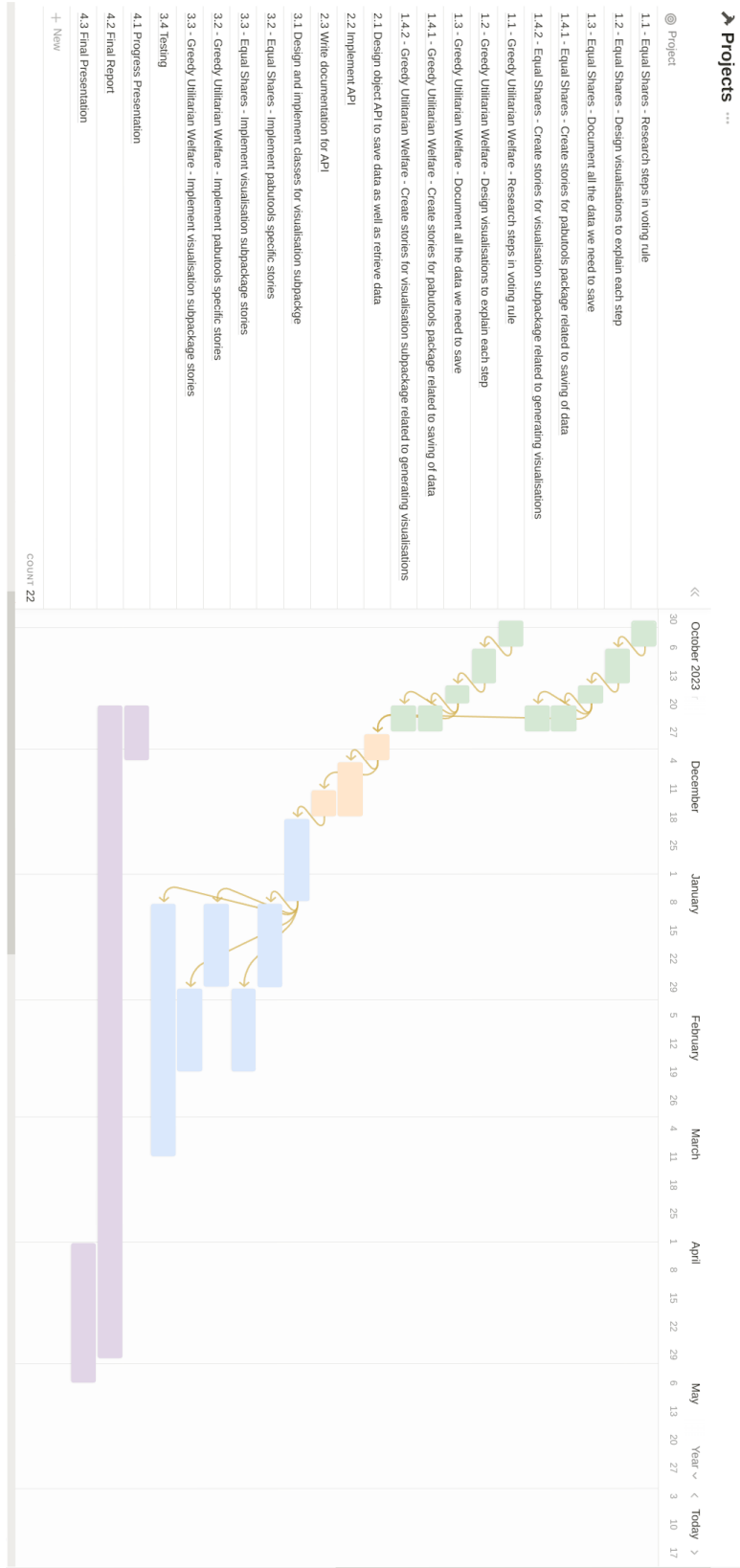


Figure B.4: Gantt Chart of Our Three-Semester Timeline for the Project

Appendix C

Documentation & User Manual

Visualisation Module

Module for the generation of visualisation for explaining rules.

`class Visualiser #` [\[source\]](#)

`class MESVisualiser(profile, instance, outcome, verbose=False)` [\[source\]](#)

Class used to visualise the results of a MES election. The visualisation result consists of two pages: a summary page called 'summary.html' and a round by round analysis page called 'round_analysis.html'.

Parameters:

- **profile** (`AbstractProfile`) – The profile.
- **instance** (`Instance`) – The election instance.
- **outcome** (`AllocationDetails`) – The outcome of the election.
- **verbose** (`bool, optional`) – Whether to print the results to the console. The default is `False`.

Return type:

`None`

`render(output_folder_path, name='')` [\[source\]](#)

Render the visualisation.

Parameters:

- **output_folder_path** (`str`) – The path to the folder where the visualisation will be saved.
- **name** (`str, optional`) – The prefix of the output files. The default is `""`.

Return type:

`None`

Figure C.1: Visualisation Module Documentation Page One

class GreedyWelfareVisualiser(*profile, instance, outcome, verbose=False*)

Class used to visualise the results of a Greedy Welfare election. The visualisation result [\[source\]](#) consists of a round by round analysis page called 'round_analysis.html'.

Parameters:

- **profile** ([AbstractProfile](#)) – The profile.
- **instance** ([Instance](#)) – The election instance.
- **outcome** ([AllocationDetails](#)) – The outcome of the election.
- **verbose** (*bool, optional*) – Whether to print the results to the console. The default is False.

Return type:

None

render(*output_folder_path, name=''*) [\[source\]](#)

Render the visualisation.

Parameters:

- **output_folder_path** (*str*) – The path to the folder where the visualisation will be saved.
- **name** (*str, optional*) – The prefix of the output files. The default is "".

Return type:

None

Figure C.2: Visualisation Module Documentation Page Two

CS407 - Group Project - User Manual and Submission Demo

Visualising and Analysing Participatory Budgeting Elections

This notebook gives a demo for running the visualisation code we have created. We show you how to create the visualisations for any file (located in the tests directory), as well as specific examples:

- A small election (small.pb) - An artificial election used for testing (from PaBulib).
- A medium election (medium.pb) - An election in France Toulouse 2019.
- A large election (large.pb) - An election from Centrum Begroot - a great example of high numbers of voters and projects.

We have located the example visualisations in `demo-visualisations/(mes/greedy)/(small/medium/large)/` for your ease of viewing results quickly.

The documentation page to the visualisations specifically is located [here](#). Additionally, Jinja may need to be installed if testing outside of this notebook (the installation is included below):

NOTE: This package requires Python \geq 3.9.

Additional Installation

```
In [ ]: %pip install jinja2
```

Method of Equal Shares (MES)

First, the markdown cell below shows the general format for visualisations, this is just an outline if you had a specific file to test.

```
from pabutools.visualisation.visualisation import MESVisualiser
from pabutools.rules.mes import method_of_equal_shares
from pabutools import election
from pabutools.election import Cost_Sat

# General format
instance, profile = election.parse_pabulib("./{path_to_election_file}.pb")
outcome = method_of_equal_shares(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = MESVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render("./{path_to_output_file}/", name="{name}")
```

Code Example

For the following four cells, we show the required imports, then the running of the code for the three election files provided.

```
In [ ]: # Setup
from pabutools.visualisation.visualisation import MESVisualiser
from pabutools.rules.mes import method_of_equal_shares
from pabutools import election
from pabutools.election import Cost_Sat # You can use any satisfaction function you like for this rule
# Changing this function will not change our visualisations,
# But may change the results of the election
```

```
In [ ]: # Parse the election and get the outcome
instance, profile = election.parse_pabulib("./demo-pb-files/small.pb")
outcome = method_of_equal_shares(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = MESVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render("demo-visualisations/small/mes", name="small")
```

```
In [ ]: # Parse the election and get the outcome
instance, profile = election.parse_pabulib("./demo-pb-files/medium.pb")
outcome = method_of_equal_shares(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = MESVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render("demo-visualisations/medium/mes", name="medium")
```

```
In [ ]: # Parse the election and get the outcome
instance, profile = election.parse_pabulib("./demo-pb-files/large.pb")
outcome = method_of_equal_shares(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = MESVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render("demo-visualisations/large/mes", name="large")
```

Figure C.3: User Manual to Install and Generate Visualisations Page One

Greedy Utilitarian Welfare

As previously, we show the general format for the visualisations.

```
from pabutools.visualisation.visualisation import GreedyWelfareVisualiser
from pabutools.rules.greedywelfare import greedy_utilitarian_welfare
from pabutools import election
from pabutools.election import Cost_Sat

instance, profile = election.parse_pabulib("./{path_to_election_file}.pb")
outcome = greedy_utilitarian_welfare(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = GreedyWelfareVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render("./{path_to_output_file}/", name="{name}")

In [ ]: from pabutools.visualisation.visualisation import GreedyWelfareVisualiser
from pabutools.rules.greedywelfare import greedy_utilitarian_welfare
from pabutools import election
from pabutools.election import Cost_Sat # This can be changed to a different ADDITIVE satisfaction function

In [ ]: # Parse the election and get the outcome
instance, profile = election.parse_pabulib("./demo-pb-files/small.pb")
outcome = greedy_utilitarian_welfare(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = GreedyWelfareVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render("./demo-visualisations/small/greedy", name="small")

In [ ]: # Parse the election and get the outcome
instance, profile = election.parse_pabulib("./demo-pb-files/medium.pb")
outcome = greedy_utilitarian_welfare(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = GreedyWelfareVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render("./demo-visualisations/medium/greedy", name="medium")

In [ ]: # Parse the election and get the outcome
instance, profile = election.parse_pabulib("./demo-pb-files/large.pb")
outcome = greedy_utilitarian_welfare(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = GreedyWelfareVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render("./demo-visualisations/large/greedy", name="large")
```

Testing on Additional Files

The path for the rest of the files that the visualisations can be tested on are located in: tests/PaBulib/All/ The following cell shows you how to run this code.

```
In [ ]: # The path to the test folder
input_path = "./tests/PaBulib/All/"

# Adjust these as needed
pb_file = "france_toulouse_2019_.pb"
output_path = "./demo-visualisations/further-tests/"
name = "france_toulouse_2019_"

# Create the path
file = input_path + pb_file

# ===== Method of Equal Shares ===== #

# Parse the election and get the outcome
instance, profile = election.parse_pabulib(file)
outcome = method_of_equal_shares(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = MESVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render(output_path, name=name + "MES")

# ===== Greedy Welfare ===== #

# Parse the election and get the outcome
instance, profile = election.parse_pabulib(file)
outcome = greedy_utilitarian_welfare(instance, profile, sat_class=Cost_Sat, analytics=True)

# The visualiser takes the profile, instance, and outcome as arguments
visualiser = GreedyWelfareVisualiser(profile, instance, outcome)

# name is optional and defaults to the empty string
visualiser.render(output_path, name=name + "Greedy")
```

Figure C.4: User Manual to Install and Generate Visualisations Page Two