Implementation of Computational Social Choice Algorithms for Distortion

Natsuhiko Ozawa

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2024

Abstract

We present *socialchoicekit*, a software library that provides out-of-the-box implementations for major algorithms in computational social choice with a focus on the metric of distortion. Our goal is to make implementations for key distortion-related algorithms in the voting, one-sided matching, and two-sided matching settings to enable researchers and software engineers to easily access research in distortion. In doing so, we propose Double λ-TSF, an application of the binary search-based elicitation mechanism to stable matching using Irving's algorithm. We also prove a theorem related to generating distortion-maximizing cardinal profiles for one-sided matching.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Natsuhiko Ozawa*)

Acknowledgements

I want to express my gratitude to Dr. Aris Filos-Ratsikas, who tirelessly and patiently advised me with his knowledge and wisdom. My gratitude extends to the Japan Student Services Organization, without whose continuous financial support this endeavor would not have been possible. Lastly, I am thankful to my friends and family for their encouragement and support throughout my studies.

Table of Contents

Chapter 1

Introduction

Social choice theory [Sen, 1986] is the study of aggregating individual preferences into a collective decision. Social choice theory has been used to formulate, analyze, and evaluate decision-making processes in a number of settings. In this work, we consider three settings. One of the most traditional settings is *voting* [Arrow, 1951], where the goal is to select an *alternative* that best represents the preferences of *agents* [Zwicker, 2016]. The second setting is *one-sided matching* [Hylland and Zeckhauser, 1979], where we want to match agents with preferences to items. The final setting is stable matching, a classical problem introduced by Gale and Shapley [1962]. The goal of stable matching is twofold. First, we want to match agents from one group to agents from a second group while respecting the preferences of agents from both groups [Klaus et al., 2016]. We also want to make a matching where no two agents are incentivized to form their own pair. These settings each have a diverse array of applications, from policy and budget planning [Adler, 2011] to university admissions [Gale and Shapley, 1962], and marriage partner matching [Roth and Sotomayor, 1992]. Social choice theory is an extremely interdisciplinary field, with origins in economics and incorporating elements from philosophy, political science, mathematics, computer science, and biology [List, 2022]. Computational social choice theory is an active research area that examines the application of computational techniques and paradigms to social choice theory and the application of social choice theoretical concepts to computational environments [Brandt et al., 2016].

Generally, social choice theory has two main approaches: the axiomatic approach and the quantitative approach. The axiomatic approach originates from the first papers on social choice and is the dominant approach in the traditional areas of social choice. This approach studies the trade-offs of satisfying different axioms. A fundamental result [Arrow, 1951] in social choice theory is that there is no aggregation method for two or more agents and more than two alternatives that can satisfy five basic axioms desirable in a social choice procedure: *universal domain* (can accept any set of individual preferences), *ordering* (aggregated preference is well-ordered), *weak Pareto principle* (if everyone's preferences match for some pair of alternatives, this is reflected in the aggregated preference), *independence of irrelevant alternatives*, and *non-dictatorship*. The quantitative approach is more popular in computational social

choice, although the two approaches are often taken together. Most often, this approach involves optimizing a metric, such as the asymptotic space and time complexity of a social choice algorithm or other computational properties that describe the ”quality” of the aggregated preference.

The key computational property that we study is *distortion*. In social choice theory, preferences are generally assumed to be cardinal information [Neumann and Morgenstern, 1953]. This means that an agent's preference towards different alternatives can be expressed as numerical values by a utility function. However, many algorithms only require ordinal information (total or partial orders that express the individual's relative preferences between alternatives). This is because cardinal information is usually more difficult to obtain, requiring a heavier cognitive load on agents. However, making a choice based on incomplete information will often result in lower aggregate utility compared to if we were to optimize a utility function based on known agent utilities. In such cases, a key desire is to measure, analyze, and minimize this decrease, which is named by Procaccia and Rosenschein [2006] as distortion. Elicitation is a technique that achieves a better distortion by performing additional queries to obtain the cardinal values for a small subset of alternatives. New applications in computational environments make it easier to gain information about individuals' cardinal preferences, even if partially.

With the advent of the Internet and computational technologies, applied social choice settings are changing. While applying results from social choice theory is difficult in high-stakes political environments, they are far more feasible in the low-stakes, dynamic, and easily calculable environments that computers provide [Brandt et al., 2016]. There is a developing ecosystem of libraries and applications that enable researchers to empirically experiment with their ideas and developers to push out social choice prototypes to end users. PrefLib [Mattei and Walsh, 2013] is a comprehensive suite of preference datasets relevant to social choice. PabuLib [Stolicki et al., 2020] is a library of datasets specific to participatory budgeting primarily aimed to aid computational social choice research. Researchers have also developed web applications like Pnyx [Brandt et al., 2015] and Whale4 [Bouveret and Natete, 2015] which enable non-technical users to run well-known aggregation methods studied in social choice theory. Similar developments are seen in related areas; for example, Polis [Small et al., 2021] is an advanced system for public discourse that maps non-quantitative individual inputs to high-dimensional opinion spaces using statistics and machine learning. However, we found that the computational social choice research and development ecosystem lacks a software library for social choice algorithms oriented around distortion.

We present socialchoicekit, a Python library that aims to be a comprehensive implementation of the major algorithms in computational social choice with a focus on distortion. We hope that socialchoicekit will help computational social choice researchers test their algorithms empirically and help developers prototype software products using social choice algorithms more easily. For this purpose, socialchoicekit provides an easily extensible interface and out-of-the-box integration with datasets from PrefLib.

1.1 Our Contributions

- We present socialchoicekit, which offers a comprehensive implementation of distortion-related computational social choice algorithms. (Chapter 3)
- To our knowledge, we are the first to implement Irving's algorithm, which is the only known cardinal algorithm for stable matching. (Section 3.7.2)
- We propose and implement Double λ-TSF, a new elicitation-based mechanism for stable matching. (Section 4.2)
- We name two types of distortion-maximizing cardinal profiles: pseudo-distortion-maximizing cardinal profiles and rule-specific distortion-maximizing profiles. (Chapter 5)
- We prove a theorem related to generating distortion-maximizing cardinal profiles for one-sided matching. (Chapter 5)

We also fix a few minor mistakes and oversights in the related work. Namely, we impose an additional requirement for constructing a graph used in a subroutine of Irving's algorithm. Furthermore, we make clarifications about an argument used in proving a theorem related to generating a distortion-maximizing cardinal profile presented in a previous paper.

1.2 Outline of Work

In Section 2.1, we formally introduce the voting, one-sided matching, and stable settings, along with the definition of distortion in each of those settings. In Section 2.2, we provide a literature review of previous works that studied distortion. We defer the literature review of studies focusing on elicitation to Chapter 4.

In Chapter 3, we discuss the general implementation details of socialchoicekit, including the representation of ordinal and cardinal profiles in Section 3.1, integration with PrefLib in Section 3.2, and random data generation in Section 3.3. Furthermore, we discuss the implementation of baseline algorithms for each of the three settings. Section 3.5 for voting covers three classes of baseline ordinal algorithms as well as a very simple cardinal algorithm. Section 3.6 for one-sided matching discusses two randomized ordinal algorithms and a graph-based cardinal algorithm. Finally, Section 3.7 for stable matching starts with the famous Gale-Shapley algorithm (ordinal) and dives into the details of Irving's algorithm (cardinal).

Chapter 4 focuses on elicitation. Section 4.1 describes implementation details specific to elicitation. We also introduce two classes of elicitation-based methods: binary search-based and two query-based. Section 4.2 focuses on the new Double λ-TSF mechanism for stable matching. Chapter 5 discusses distortion-maximizing cardinal profile generation. In particular, we provide proofs to show that a simple and easily generated type of cardinal profile maximizes distortion given any ordinal profile and algorithm. Chapter 6 concludes the work and discusses limitations and areas for future work.

Chapter 2

Background

2.1 The Model

Throughout this work, we consider the voting, one-sided matching, and stable matching settings of social choice.

In the *voting* setting, also referred to as the *general social choice* setting, we consider a set N of n agents labeled $\{1,2,\ldots,n\}$ and a set A of m alternatives labeled $\{a_1,a_2,\ldots,a_m\}$. Our goal is to select a single alternative based on the preferences of agents in N. We consider two types of preferences. An *ordinal profile* σ represents the agents' preferences through ordered comparisons. It is expressed as a family of *preference rankings* $(\succ_i)_{i\in N}$, where each preference ranking of agent i is a linear ordering $\succ_i$ over the alternatives, and $a_j \succ_i a_{j'}$ holds if and only if agent i strictly prefers alternative a_j to $a_{j'}$. Note that a linear ordering is a binary relation that is transitive, anti-symmetric, and complete. In practice, preference rankings may not be complete, as datasets often contain missing data; we will introduce methods in Section 3.2 to handle this case. A *cardinal profile* v, also known as a valuation profile, represents the agents' preferences with non-negative numeric values. By using a cardinal profile, we assume that each agent i has a cardinal value or numeric utility $v_{ij} \in \mathbb{R}_{\geq 0}$ for each alternative a_j. We represent v as an (n,m) matrix. We can also view v as a family of valuation functions $v_i : A \to \mathbb{R}_{\geq 0}$. A cardinal profile contains strictly more information than an ordinal profile, and it induces a unique ordinal profile given a tie-breaker.

Example 2.1. *Example voting problem with* 3 *agents and* 4 *alternatives.*

Table 2.1: Ordinal Profile

Agents	*Preference rankings*
1	$a_4 \succ_1 a_2 \succ_1 a_3 \succ_1 a_1$
2	$a_4 \succ_2 a_3 \succ_2 a_2 \succ_2 a_1$
3	$a_1 \succ_3 a_4 \succ_3 a_2 \succ_3 a_3$

Table 2.2: Cardinal Profile

	a_1	a_2	a_3	a_4
1	0.1	0.3	0.2	0.4
2	0.25	0.25	0.25	0.25
3	1	0	0	0

We refer to the alternatives, items, or agents being ranked in an ordinal or cardinal profile as *choices*. An ordinal profile is said to be *complete* if no values are missing, i.e. if its preference rankings are complete. A cardinal profile is complete if for every

agent i and every choice j, j is in the domain of the valuation function v_i. An ordinal profile is said to be *strict* if, for every agent i, there are no ties in the preference ranking. Our definition above does not admit any ordinal or cardinal profiles with ties or missing values. In some parts of our implementation, we allow those for increased generality. In Example 2.1, agent 3 strictly orders $a_4 \succ_3 a_2 \succ_3 a_3$ although the cardinal utilities for each alternative are all 0. This is coherent as we can assume there is a negligibly small difference between the utilities.

The second setting is the *one-sided matching* setting. We consider a set N of agents. Instead of alternatives, we consider the set A of items to allocate to agents. One-sided matching is a special case of a more general setting called resource allocation, where we require that the number of agents and items are equal, i.e. $|N| = |A| = n$, and every agent receives 1 item. We let the preference ranking $\succ_i$ be a linear ordering over the items and the ordinal profile σ be the family of preference rankings for each agent. Note that the existence of a solution to the one-sided matching problem can be guaranteed if and only if the profiles are complete. However, in our implementations, we extend our algorithm to accept incomplete profiles for increased flexibility. We interpret that a_j is not related by $\succ_i$ to any other item if an agent i finds an item a_j unacceptable. We similarly consider a cardinal profile v where $v_{ij} \in \mathbb{R}_{\geq 0}$ is the cardinal value agent i has for item a_j, with all properties of v from the voting setting holding for one-sided matching. A matching M is a set comprising of pairs (i, a_j) such that each agent i is matched to exactly one item and no two agents are matched to the same item. Our goal is to produce such a matching based on the agents' preferences.

Example 2.2. *Example one-sided matching problem with* 3 *agents and* 3 *items.*

Table 2.3: Ordinal Profile

Agents	*Preference rankings*
1	$a_3 \succ_1 a_2 \succ_1 a_1$
2	$a_3 \succ_2 a_2 \succ_2 a_1$
3	$a_1 \succ_3 a_3 \succ_3 a_2$

Table 2.4: Cardinal Profile

	a_1	a_2	a_3
1	0.1	0.3	0.6
2	0.33	0.33	0.34
3	1	0	0

The *stable matching* setting works with two sets of agents, $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$. There are two ordinal profiles: σ_X where agents from X rank their preference over agents in Y, and σ_Y where agents from Y rank their preference over agents in X. Each ordinal profile is a family of preference rankings over agents $(\succ_{X,i})_{x_i \in X}, (\succ_{Y,j})_{y_j \in Y}$. We assume that ordinal profiles are strict and complete. Similarly, we also consider two complete cardinal profiles v_X, v_Y. Our goal is to produce a stable matching, which is a matching that satisfies the following stability property.

Definition 2.3. *A matching M is **stable** if and only if it does not admit a **blocking pair**. A blocking pair is a pair of agents not in the matching M where both prefer each other over the agent they are matched to. More formally, it is a pair $(x_i, y_j) \notin M$ such that both of the following holds.*

1. *$y_j \succ_{X,i} y_{j'}$, where $y_{j'}$ is the agent x_i is matched to in M*
2. *$x_i \succ_{Y,j} x_{i'}$, where $x_{i'}$ is the agent y_j is matched to in M*

The stable matching setting is a special case of the two-sided matching setting, which does not have the stability requirement.

Example 2.4. *Example stable matching problem with* $X = \{x_1, x_2, x_3\}$ *and* $Y = \{y_1, y_2, y_3\}$.

Table 2.5: Ordinal Profile for X

X	*Rankings (Y)*
x_1	$y_3 \succ_{X,1} y_2 \succ_{X,1} y_1$
x_2	$y_1 \succ_{X,2} y_3 \succ_{X,2} y_2$
x_3	$y_1 \succ_{X,3} y_3 \succ_{X,3} y_2$

Table 2.6: Cardinal Profile for X

	y_1	y_2	y_3
x_1	*0.1*	*0.3*	*0.6*
x_2	*0.4*	*0.3*	*0.3*
x_3	*0.8*	*0*	*0.2*

Table 2.7: Ordinal Profile for Y

Y	*Rankings (X)*
y_1	$x_2 \succ_{Y,1} x_3 \succ_{Y,1} x_1$
y_2	$x_3 \succ_{Y,2} x_1 \succ_{Y,2} x_2$
y_3	$x_3 \succ_{Y,3} x_2 \succ_{Y,3} x_1$

Table 2.8: Cardinal Profile for Y

	x_1	x_2	x_3
y_1	*0.1*	*0.7*	*0.2*
y_2	*0.4*	*0.1*	*0.5*
y_3	*0*	*0*	*1*

A matching $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ *is not stable because there are blocking pairs, for instance* (x_2, y_1).

We now explain terminology commonly used in all three settings. A cardinal profile v is consistent with an ordinal profile $\sigma = (\succ_i)_{i \in N}$ if $a_j \succ_i a_{j'}$ implies $v_i(a_j) \geq v_i(a_{j'})$ for any agent i and two choices $a_j, a_{j'}$. In all three settings, the provided cardinal profile(s) must be consistent with the provided ordinal profile(s). An *ordinal algorithm* f, also known as a social choice function (SCF), is a process that takes as input ordinal profile(s) and produces an outcome. This is in contrast to a *cardinal algorithm*, which takes as input cardinal profile(s) and produces an optimal outcome. *Deterministic algorithms* produce the same outcome given the same profile. *Randomized algorithms* produce a discrete probability distribution over the outcomes, and will randomly select an outcome according to the probability distribution each time. Unless explicitly specified, we assume that algorithms are deterministic. We also refer to algorithms as rules.

Definition 2.5. *The* ***social welfare*** SW *of an outcome measures the total utility of the agents with respect to cardinal profile(s).*

1. *For voting:* $SW(a_j \mid v) = \sum_{i \in N} v_i(a_j)$
2. *For one-sided matching: Let* $M = ((i, a_i))_{i \in N}$. *Then,* $SW(M \mid v) = \sum_{i \in N} v_i(a_i)$
3. *For stable matching: Let* $M = ((x_i, y_i))_{i \in N}$. *Then,*
$$SW(M \mid v_X, v_Y) = \sum_{i \in N} (v_X)_i(y_i) + \sum_{i \in N} (v_Y)_i(x_i)$$
4. *For randomized algorithms, let* $\Delta(M)$ *be the probability distribution outputted. Then,* $SW(\Delta(M)) = \sum_{M \in \Delta(M)} P(M) \cdot SW(M)$ *where the v are omitted.*

An ***optimal outcome*** *is an outcome that maximizes the social welfare.*

An ordinal algorithm works with inherently limited information, which may result in a suboptimal outcome. The concept of distortion captures this discrepancy in the worst-case scenario.

Definition 2.6. *The **distortion** $dist(f)$ of an algorithm f is the worst-case ratio between the social welfare of an optimal outcome and the social welfare of the outcome produced by the algorithm.*

1. *For voting: Let Σ be the set of all ordinal profiles possible. Let $f : \Sigma \to A$. Then,*

$$dist(f) = \sup_{N,A,v} \frac{\max_{a^* \in A} SW(a^* \mid v)}{SW(f(\sigma) \mid v)}$$

2. *For one-sided matching: Let $\mathbf{M}$ be the set of all matchings possible. Let Σ be the set of all ordinal profiles possible. Let $f : \Sigma \to \mathbf{M}$. Then,*

$$dist(f) = \sup_{N,A,v} \frac{\max_{M^* \in \mathbf{M}} SW(M^* \mid v)}{SW(f(\sigma) \mid v)}$$

3. *For stable matching: Let $\mathbf{M}$ be the set of all stable matchings possible. Let Σ_X, Σ_Y be sets of all ordinal profiles possible for X, Y. Let $f : \Sigma_X \times \Sigma_Y \to \mathbf{M}$ Then,*

$$dist(f) = \sup_{X,Y,v_X,v_Y} \frac{\max_{M^* \in \mathbf{M}} SW(M^* \mid v_X, v_Y)}{SW(f(\sigma_X, \sigma_Y) \mid v_X, v_Y)}$$

2.2 Related Work

Since distortion has been defined by Procaccia and Rosenschein [2006], it has been mainly explored in the voting setting, with two major versions of utility. The first version is *normalized social choice*, where we assume that each agent's cardinal utility for the alternatives sum to 1. Complexity analyses have found that the best achievable distortion by deterministic voting rules is $\Theta(m^2)$ [Caragiannis and Procaccia, 2011], where m is the number of alternatives. This means that a general lower bound of $\Omega(m^2)$ was proven, and there exists an algorithm where the distortion with an upper bound of $O(m^2)$. For randomized rules, distortion is calculated using the expected social welfare. Randomization allows for significantly lower distortion, with the best possible distortion of $\Theta(\sqrt{m})$ proven by Boutilier et al. [2015] and Ebadian et al. [2022].

The second version of utility, metric social choice, represents agents and alternatives as points in a metric space and aims to minimize the distance of an agent against the selected alternative. Under metric social choice, distortion can be lowered to a constant factor. For deterministic rules, a lower bound of 3 can be shown by a simple example. Anshelevich et al. [2018] conjectured that this lower bound of 3 is tight, and Munagala and Wang [2019] described properties that would be sufficient to achieve this bound. The plurality matching voting rule, recently proposed by Gkatzelis et al. [2020], finally achieved the lower bound of 3. The lower bound for randomized rules is an open area of research. Along a line of work started by Anshelevich and Postl [2017], Charikar and Ramakrishnan [2022] most recently showed a lower bound of 2.112, and Charikar et al. [2024] showed that there is an algorithm that achieves distortion of at most 2.753. For a more comprehensive survey of distortion literature, see Anshelevich et al. [2021].

In the one-sided matching setting, Amanatidis et al. [2022] proved a tight bound of $\Theta(n^2)$ for deterministic rules, and Filos-Ratsikas et al. [2014] proved a tight bound of $\Theta(\sqrt{n})$ for randomized rules. Distortion in stable matching has not been explored.

Chapter 3

Implementation of the Library

3.1 Profile Representation

When implementing a library for social choice, choosing a way to represent ordinal and cardinal profiles is a crucial design decision. The inputs to an algorithm for a social choice problem include the number of agents, the number of choices, and the ordinal or cardinal profile(s) required by this algorithm. In practice, the input information can be entirely represented by the ordinal or cardinal profile(s). All algorithms operate on the given profile(s) to create intermediate data structures as needed and produce an outcome. Poor representations impact the space and runtime efficiency of the implementations.

We chose to represent ordinal profiles as NumPy matrices [Harris et al., 2020] such that the rows correspond to agents whose preferences are being represented, and the columns correspond to choices. If agent i ranks a choice j as kth from the top, then the NumPy matrix entry at $(i-1, j-1)$ is set to $m-k+1$, where m is the number of columns. Note that the subtraction is a conversion from 1-index to 0-index. We also represent cardinal profiles as NumPy matrices where rows and columns correspond similarly to the representation of ordinal profiles. The entry at $(i-1, j-1)$ contains v_{ij}.

Example 3.1. *A representation of the voting problem presented in Example 2.1.*

Figure 3.1: Ordinal Profile

```
import numpy as np
sigma = Profile.of(
  np.array([
    [4, 2, 3, 1],
    [4, 3, 2, 1],
    [1, 3, 4, 2]
  ])
)
```

Figure 3.2: Cardinal Profile

```
import numpy as np
v = ValuationProfile.of(
  np.array([
    [0.1, 0.3, 0.2, 0.4],
    [0.25, 0.25, 0.25, 0.25],
    [1, 0, 0, 0]
  ])
)
```

The `Profile` and `ValuationProfile` classes are subclasses of the NumPy matrix class, `np.ndarray`, equipped with the `of` method to check the required profile properties before construction. Missing entries in incomplete profiles will be set to NaN.

Using NumPy matrices has the major advantage that efficient implementations of many required operations on profiles, such as summation, reordering, conditional replacement, range indexing, and argsort, are available by default. NumPy features are stable, have good debugging support, and many users will likely have prior experience with NumPy. A major disadvantage of our representation is that it is space-inefficient. Alternative representations, such as that of PrefLib-Tools introduced in Section 3.2, have lower memory requirements because they store a *multiplicity* value alongside each preference ranking $\succ_i$ in the ordinal profile. This approach does not admit any duplications, even if two agents have the same $\succ_i$. This disadvantage becomes critical in settings where the number of agents is large and the number of choices in the preference ranking is small. However, we decided to not adopt this alternative approach because the multiplicity and the lack of ordering of agents complicate the implementation of many algorithms. Moreover, many practical use cases of this library assume that users are working with data small enough to fit in memory. We leave supporting large data sizes as an extension for future work. An alternative approach to representing cardinal profiles is to use sparse matrices, such as the one provided by SciPy [Virtanen et al., 2020]. This is more memory efficient if most entries of the valuation profile are 0. While we expect that the cardinal utilities of most choices for most agents will not be 0 in practice, this is not the case in empirical distortion-related experiments. We give as an example the dichotomous cardinal profile discussed in Chapter 5, which maximizes distortion with respect to some ordinal profile. Nevertheless, we dismiss the relative importance of memory efficiency in our use cases as there is no practical need to run distortion-related experiments with extremely large dataset sizes.

3.2 PrefLib Integration

As mentioned in the introduction, PrefLib [Mattei and Walsh, 2013] is a standard source of data for researchers conducting empirical social choice experiments and simulations. We offer out-of-the-box integration with PrefLib. Normally, users must download PrefLib's custom-formatted data files and parse them. For Python users, PrefLib provides PrefLib-Tools [Mattei and Walsh, 2013], which can download the data files programmatically and parse them into a usable representation. Our integration provides wrappers on top of PrefLib-Tools to automatically download the PrefLib-formatted file and convert it to our NumPy-based representation. Datasets on PrefLib are classified into several different categories, based on completeness and strictness of profiles. For each category, we provide a subclass of `Profile` to allow for smooth conversion. For instance, the `StrictCompleteProfile` corresponds to the SOC (Strict Orders - Complete List) format in PrefLib. We also provide helpers to convert profiles from one type to another. For example, socialchoicekit can convert incomplete profiles to complete profiles by assuming choices with missing rank are least preferred.

3.3 Data Generation

In many cases, especially when using datasets from PrefLib, cardinal profiles are not readily available. As a social choice library with a focus on distortion, providing access

to cardinal profiles is a key concern of socialchoicekit. As a baseline, we provide random cardinal utility generation with the uniform and normal probability distributions to create a cardinal profile consistent with a given ordinal profile. These methods work by simply generating random numbers according to some probability distribution and reordering them using the preference rankings. If the user can accurately model the distribution of cardinal values as a probability distribution, then this approach would be effective at calculating the expected distortion in an empirical experiment.

However, users may want to test their data or algorithm with cardinal profiles that result near the highest distortion. This would help them understand how their algorithm or data performs in the worst scenarios. In these cases, random generation is not appropriate. This is a difficult problem that we further explore in Chapter 5.

3.4 Testing

We use pytest [Krekel et al., 2004] to unit test our implementations. Testing socialchoicekit is challenging because while the library requires stability, there are often no other implementations available to generate test cases. Hence, we hand-crafted test cases for most tests and used examples presented in other works for other tests. We automatically test our library on versions 3.8, 3.9, 3.10, and 3.11 of Python.

3.5 Voting Algorithms

In socialchoicekit, we implemented the ordinal voting algorithms indicated in Table 3.1. The cardinal algorithm we implemented for voting is simple; it selects the choice a_j with the highest value of $SW(a_j \mid v)$, which can be computed with a simple sum over v. We follow the classification by Brandt et al. [2016] of ordinal voting algorithms into three classes: scoring, tournament, and multiround.

Scoring algorithms assign a numeric score to each agent-alternative combination. Then, they select an alternative that has the highest total score across the agents. Randomized scoring algorithms use the total score for each alternative, normalized by the total score across all agents and alternatives, to create the discrete probability distribution. Borda is the classic example of a scoring rule, where scores are assigned linearly with respect to an alternative's rank in the preference rankings. Plurality is widely used in elections, where for each alternative, the number of agents who voted it as their top choice is counted. This is equivalent to assigning a score of 1 for the top choice and 0 for all others. See Table 3.1 for the rest of the scoring algorithms we implemented. Due to their simplicity, scoring algorithms are often used as baseline algorithms to test new concepts empirically. As such, they are critical to implement for a library like socialchoicekit. Scoring algorithms are also used as subroutines by other algorithms.

Tournament algorithms, also known as *Condorcet-extensions*, compare for pairs of alternatives which alternative has the majority vote.

Definition 3.2. *The net preference of a profile σ for an alternative a_j over another alternative $a_{j'}$ is the net difference of the number of voters that prefer a_j over $a_{j'}$ to*

Table 3.1: Voting Algorithms Implemented in socialchoicekit

Name	**Type**	**Description**	**Randomization**
Plurality	Scoring	Choose the alternative with the highest number of top-rank votes.	Both
K-Approval	Scoring	Choose alternative with the highest number of votes that are in top K. (e.g. assign a score of 1 to the top-K ranked choices and 0 to others, for each preference ranking)	Both
Veto	Scoring	Choose the alternative with the lowest number of bottom-rank votes.	Both
Borda	Scoring	Assign score of $m-k+1$ to the kth-ranked choice for each preference ranking.	Both
Harmonic	Scoring	Assign score of $\frac{1}{k}$ to the kth-ranked choice for each preference ranking.	Both
Copeland	Tournament	Choose the alternative with the highest Copeland score.	Deterministic
Single Transferable Vote	Multiround	Iterate over rounds dropping an alternative with the lowest Plurality score.	Deterministic

those that prefer $a_{j'}$ over a_j.

$$Net_\sigma(a_j \succ a_{j'}) = |\{i \in N \mid a_j \succ_i a_{j'}\}| - |\{i \in N \mid a_{j'} \succ_i a_j\}|$$

*Let the **pairwise majority** relation $a_j >^\mu a_{j'}$ hold if and only if $Net_\sigma(a_j \succ a_{j'}) > 0$.*

Definition 3.3. *A **Condorcet winner** is an alternative a_j is an alternative for which the pairwise majority relation $a_j >^\mu a_{j'}$ holds for every other alternative $a_{j'}$.*

It is easy to see that a Condorcet winner does not always exist. A Condorcet-extension is a voting algorithm that always outputs a Condorcet-winner if it exists. Condorcet-extensions are useful because they exhibit desirable properties including strategyproofness [Campbell and Kelly, 2003]. We implemented the Copeland algorithm, which is the prime example of a Condorcet-extension. It works by selecting the alternative with the highest Copeland score as defined below.

Definition 3.4. *The Copeland score for an alternative a_j counts the net difference between the number of alternatives that have a_j has a pairwise majority and those that a_j have a "pairwise minority".*

$$Copeland(a_j) = |\{a_{j'} \in A \mid a_j >^\mu a_{j'}\}| - |\{a_{j'} \in A \mid a_{j'} >^\mu a_j\}|$$

A final class of voting algorithms is known as *multiround algorithms*, which iteratively drop alternatives according to some score until there is only one left. We implemented the Single Transferable Vote (STV) algorithm first introduced by Hare [1859], which uses the plurality scoring rule introduced above to determine which alternative to drop.

3.6 One-sided Matching Algorithms

For one-sided matching, we implemented the three ordinal algorithms and one cardinal algorithm as indicated in Table 3.2.

Table 3.2: One-sided Matching Algorithms Implemented in socialchoicekit

Name	Description	Randomization
Maximum Weight Matching	Assign a matching that maximizes the total weight of the edges in a weighted bipartite graph.	Deterministic
Random Serial Dictatorship	Randomly generate an ordering of the agents and have agents select an item in that order.	Randomized
Probabilistic Serial	Treat items as divisible and have agents greedily take fractional items until they run out.	Randomized
Simultaneous Eating	Probabilistic Serial with possibly different speeds at which agents take items.	Randomized

The *maximum weight matching* problem is a general problem that finds a matching in a graph with weighted edges such that the total weight of the edges in the matching is maximized. We work with a special case of maximum weight matching, called the assignment problem, in which we restrict the input weighted graph to bipartite graphs. The following theorem shows that maximum weight matching can be used as a cardinal algorithm for one-sided matching.

Theorem 3.5. *Consider the following weighted undirected graph $G = (V, E)$.*

- *Define a node for each agent or alternative, i.e. let $V = N \cup A$.*
- *Let there be an edge $(i, a_j) \in E$ between agent $i \in N$ and item $a_j \in A$ if and only if i finds a_j acceptable and attach a weight of v_{ij} to this edge, i.e. $w\big((i, a_j)\big) = v_{ij}$.*

Then for complete cardinal profiles:

1. *There always exists a perfect matching M' whose weight is the maximum among all matchings in G.*
2. *The perfect matching corresponds to a matching M in the one-sided matching problem which maximizes the social welfare.*

Proof. Notice that in the case of a complete cardinal profile, every agent will be connected to every item. Now, suppose that the solution to the maximum weight matching problem on G is not a perfect matching. Then, we can create a perfect

matching by iteratively adding (i, a_j) to the imperfect matching for any agent i and item a_j not in the matching until all agents and items are matched. Adding an edge will not decrease the weight of the matching because all values of v_{ij} are non-negative. Hence, this perfect matching also has maximum weight, and the first claim is proven.

For the second point, since each agent is only paired with one item in a perfect matching M' of G, it is easy to see the correspondence with M. For social welfare maximization

$$\sum_{(i,a_j)\in M'} w\big((i,a_j)\big) = \sum_{(i,a_j)\in M} v_{ij} = SW(M \mid v)$$

follows by definition with a slight abuse of notation in the second summation. □

We use SciPy's `sparse_graph.min_weight_full_bipartite_matching` function with the maximize option to find the maximum weight matching in our implementation.

Next, we discuss randomized algorithms. *Probabilistic serial* and *simultaneous eating* are both algorithms which produce a bistochastic matrix deterministically. A *bistochastic matrix* is a square matrix with non-negative real entries such that its columns and rows sum to 1. Through an algorithm called the Birkhoff-von Neumann algorithm [Birkhoff, 1946], which we implemented, we can convert any bistochastic matrix into a discrete probability distribution of permutation matrices such that the expectation of this distribution is equivalent to the bistochastic matrix. At runtime, the algorithm will randomly select a matching from the distribution. From the perspective of an agent, if she is allocated some fraction $0 \leq k \leq 1$ of an item a_j, she can expect to receive a_j with probability k.

We now informally elaborate on how the two algorithms create the bistochastic matrix. Suppose that the items are divisible, i.e. each whole item can be split into fractional parts and be allocated to different agents. To fairly allocate items, we assume that agents who receive fractional items will receive multiple such fractional items, such that the sum of the fractions of items is 1 $(*)$. We represent the result of this allocation as an (n,n) matrix, with rows corresponding to agents and columns corresponding to items. Every row will sum to 1 because of $(*)$. Every column will sum to 1 because only 1 of each item is split. Every entry in the matrix is trivially non-negative.

In both algorithms, we let agents increase their share of any remaining fractions of items continuously until the sum of fractions they claimed reaches 1. Items that are preferred by multiple agents will be consumed by multiple agents simultaneously. In simultaneous eating, the speeds at which agents take the items are parameters. Probabilistic serial is a special case of simultaneous eating where the eating speeds are equal across agents. Hence, we implement the more general simultaneous eating algorithm and call it from probabilistic serial. In implementing this, we use a greedy approach where we calculate the "time" until an item next becomes fully taken. Then we calculate the changes in this "time" period and redirect any agents who still can receive items to claim their next available preferred item.

In contrast, *random serial dictatorship* is a simple algorithm. It works by picking a random order of agents and having agents pick their favorite remaining item in that order. We have not implemented any deterministic algorithms for one-sided matching,

but we can think of a *deterministic serial dictatorship*, where agent $1 \in N$ picks first, then $2 \in N$, and so forth, and $n \in N$ picks last.

3.7 Stable Matching Algorithms

Gale and Shapley [1962] proved that there is at least one stable matching for all instances of the stable matching problem. The paper also provided an efficient ordinal algorithm to compute it, which we call the Gale-Shapley algorithm. We start by discussing the Gale-Shapley algorithm and its implementation. We then discuss Irving's algorithm, which is a cardinal algorithm for stable matching proposed by Irving et al. [1987]. While its implementation is involved and not comfortably efficient, we included Irving's algorithm in our library because it was the only cardinal algorithm we found in the literature.

3.7.1 Gale-Shapley

The Gale-Shapley algorithm can produce a stable matching that maximizes the utilities of agents in X and minimizes the utilities of agents in Y, or vice versa. In this work, we call the former an X-optimal stable matching and the latter Y-optimal. While we have implemented both the algorithm that produces the X-optimal stable matching and the algorithm that produces the Y-optimal matching, we will describe the former as the two are structurally similar. This is an efficient algorithm that runs in $O(n^2)$ time [Gusfield and Irving, 1989].

Let the *X-oriented Gale-Shapley* algorithm proceed as follows. First, start with an initial state where no agents are matched. Then, iteratively perform rounds of the following procedure. For each agent $x_i \in X$ who is currently not matched to an agent, x_i *proposes* to an agent y_j who is x_i's most preferred agent that has so far not rejected x_i. If y_j is not currently matched, then y_j accepts the proposal. If y_j is currently matched, then y_j accepts the proposal if and only if y_j prefers x_i to the agent she is currently matched with, which we call $x_{i'}$. If y_j accepts the proposal, then x_i, y_j are now matched and $x_{i'}$ is rejected. Otherwise, y_j rejects x_i. When all agents are matched, the algorithm finishes.

Theorem 3.6. *The X-oriented Gale-Shapley algorithm produces a stable matching.*

Proof. Suppose, for a contradiction, that there is a blocking pair (x_i, y_j) as defined in Definition 2.3. Since x_i prefers y_j to $y_{j'}$, x_i would have proposed to y_j before $y_{j'}$ in the X-oriented Gale Shapley algorithm. However, y_j is matched to $x_{i'}$. This means that y_j prefers $x_{i'}$ to x_i because otherwise x_i would be accepted when x_i proposed to $y_{j'}$. This is a contradiction because (x_i, y_j) would not be a blocking pair. Therefore, there are no blocking pairs in the matching returned by X-optimal Gale-Shapley. □

Example 3.7. *The stable matching problem from Example 2.4 is solved by X-oriented Gale-Shapley as follows.*

- *Round 1: x_1 proposes to y_3 and is accepted. x_2 and x_3 propose to y_1. x_2 is accepted and x_3 is rejected because $x_2 \succ_{Y,1} x_3$.*

- *Round 2: The current match is* $(x_1, y_3), (x_2, y_1)$. x_3 *proposes to* y_3. y_3 *prefers* x_3 *to* x_1. *Hence,* y_3 *accepts* x_3 *and rejects the currently matched agent* x_1.
- *Round 3: The current match is* $(x_2, y_1), (x_3, y_3)$. x_1 *proposes to* y_2 *and is accepted.*
- *The algorithm terminates as every agent is matched. The stable matching is* $(x_1, y_2), (x_2, y_1), (x_3, y_3)$.

Our implementation of the Gale-Shapley algorithm supports an extension of the stable matching problem defined in Section 2.1, where the sizes of X and Y can be different, and each $y_j \in Y$ can be matched with multiple $x_i \in X$. However, we refer the reader to Klaus et al. [2016] and omit any explanation here.

A concept that will become useful in Section 3.7.2 is the shortlist.

Definition 3.8. *Let* x_i *be an agent in* X. x_i*'s* ***shortlist*** *or* ***reduced preference list*** s_X *is an (ordered) list of agents in* Y *that*

1. *satisfies the preference ranking* $\succ_{X,i}$.
2. *does not contain any agent in* Y *who rejected* x_i*'s proposal.*

Let y_j *be an agent in* Y. y_i*'s shortlist* s_Y *is an (ordered) list of agents in* X *that*

1. *satisfies the preference ranking* $\succ_{Y,j}$.
2. *does not contain any agent in* X *who are worse than the best agent from which* y_j *received a proposal.*

Theorem 3.9. *[Gusfield and Irving, 1989] The* X*-oriented Gale-Shapley algorithm produces an* X*-optimal matching such that*

1. *there is no stable matching in which any agent from* X *can be assigned to a better agent from* Y *than the one assigned by* X*-oriented Gale-Shapley.*
2. *in every stable matching, every agent from* Y *is assigned to an agent from* X *that is better than or equal to the one assigned by* X*-oriented Gale-Shapley.*

From Theorem 3.9 and Definition 3.8, y_j appears on x_i's shortlist if and only if x_i appears on y_j's shortlist, and y_j is first on x_i's shortlist if and only if x_i is last on y_j's shortlist.

Example 3.10. *At the end of Example 3.7, the shortlists are as follows.*

Figure 3.3: Shortlists for Agents in X

x_1	$[y_2]$
x_2	$[y_1]$
x_3	$[y_3, y_2]$

Figure 3.4: Shortlists for Agents in Y

y_1	$[x_2]$
y_2	$[x_3, x_1]$
y_3	$[x_3]$

3.7.2 Irving's Algorithm

As introduced at the beginning of this section, Irving's algorithm is the only efficient cardinal algorithm for stable matching that we are aware of. In Section 4.2, we will use Irving's algorithm to propose and implement an application of an elicitation-based

algorithm to stable matching. In this section, we introduce the algorithm and present our implementation which achieves $O(n^4)$ runtime complexity. To our knowledge, this work is the first implementation of Irving's algorithm. Throughout this section, we will use our own notation introduced in Section 2.1.

Irving's algorithm works by performing a sequence of actions related to *rotations* on a stable matching to obtain another stable matching. Such an action either increases or decreases the social welfare. The algorithm finds a sequence of rotations that maximizes increases in social welfare. We begin by defining rotation.

Definition 3.11. *A **rotation** ρ exposed to some shortlists of X, Y is a sequence of pairs $\rho = (x_0, y_0), \ldots, (x_{r-1}, y_{r-1})$ where each $x_k \in X, y_k \in Y$ such that for every $k \in \{0, \ldots, r-1\}$,*

1. *y_k is first in x_k's shortlist*
2. *$y_{(k+1) \mod r}$ is second in x_k's shortlist*

Irving et al. [1987] claims that at least one rotation is always exposed unless the given shortlist is Y-optimal.

Example 3.12. *The problem from Example 2.4 is too simple to demonstrate Irving's algorithm. From now on, consider the following stable matching problem. Note that we abbreviate the subscripts on $\succ$ for readability. We also express the cardinal profile as integers where each row sums to 10, for reasons explained later.*

Figure 3.5: Ordinal Profile for X

x_1	$y_4 \succ y_3 \succ y_2 \succ y_1 \succ y_5 \succ y_6$
x_2	$y_3 \succ y_2 \succ y_5 \succ y_6 \succ y_1 \succ y_4$
x_3	$y_2 \succ y_5 \succ y_6 \succ y_3 \succ y_4 \succ y_1$
x_4	$y_2 \succ y_6 \succ y_3 \succ y_1 \succ y_4 \succ y_5$
x_5	$y_5 \succ y_1 \succ y_4 \succ y_6 \succ y_2 \succ y_3$
x_6	$y_1 \succ y_6 \succ y_4 \succ y_5 \succ y_3 \succ y_2$

Figure 3.6: Ordinal Profile for Y

y_1	$x_3 \succ x_2 \succ x_4 \succ x_1 \succ x_5 \succ x_6$
y_2	$x_6 \succ x_5 \succ x_1 \succ x_2 \succ x_3 \succ x_4$
y_3	$x_5 \succ x_6 \succ x_3 \succ x_4 \succ x_1 \succ x_2$
y_4	$x_6 \succ x_5 \succ x_4 \succ x_3 \succ x_2 \succ x_1$
y_5	$x_4 \succ x_1 \succ x_6 \succ x_2 \succ x_3 \succ x_5$
y_6	$x_1 \succ x_2 \succ x_5 \succ x_3 \succ x_4 \succ x_6$

$$v_X = \begin{bmatrix} 1 & 1 & 3 & 5 & 0 & 0 \\ 1 & 2 & 2 & 1 & 2 & 2 \\ 1 & 3 & 1 & 1 & 2 & 2 \\ 1 & 3 & 2 & 1 & 1 & 2 \\ 3 & 1 & 1 & 1 & 3 & 1 \\ 4 & 0 & 0 & 2 & 1 & 3 \end{bmatrix}, v_Y = \begin{bmatrix} 0 & 5 & 5 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 3 & 7 \\ 2 & 0 & 0 & 7 & 0 & 1 \\ 9 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

After running X-optimal Gale-Shapley on this example, we will obtain the following stable matching

$$M_X = (1,4), (2,3), (3,2), (4,6), (5,5), (6,1)$$

and shortlists

Figure 3.7: Shortlists for Agents in X

x_1	$[y_4, y_3, y_2, y_1, y_5, y_6]$
x_2	$[y_3, y_2, y_5, y_6, y_1, y_4]$
x_3	$[y_2, y_5, y_6, y_3, y_4, y_1]$
x_4	$[y_6, y_3, y_1, y_4, y_5]$
x_5	$[y_5, y_1, y_4, y_6, y_2, y_3]$
x_6	$[y_1, y_4, y_5, y_3, y_2]$

Figure 3.8: Shortlists for Agents in Y

y_1	$[x_3, x_2, x_4, x_1, x_5, x_6]$
y_2	$[x_6, x_5, x_1, x_2, x_3]$
y_3	$[x_5, x_6, x_3, x_4, x_1, x_2]$
y_4	$[x_6, x_5, x_4, x_3, x_2, x_1]$
y_5	$[x_4, x_1, x_6, x_2, x_3, x_5]$
y_6	$[x_1, x_2, x_5, x_3, x_4]$

Here, only one rotation $\rho_1 = (1,4),(2,3),(3,2),(5,5),(6,1)$ is exposed. x_2 is matched to y_3, who is second on x_1's shortlist. x_3 is matched to y_2, who is second on x_2's shortlist, and so on.

We operate on these rotations by performing an action called elimination.

Definition 3.13. *A rotation ρ is **eliminated** when we remove the following pairs from each others' shortlists. These pairs are said to be eliminated by ρ.*

1. *members of the rotation, i.e. (x_k, y_k)*
2. *(x, y_k) for all x between x_{k-1} and x_k on y_k's shortlist.*

We then further reduce the shortlists of X to enforce property 2 in Definition 3.8. When ρ is eliminated, a new stable matching is induced where each x_k is matched with $y_{(k+1) \mod r}$, and all other agents not in ρ keep their match. It is not difficult to show that elimination preserves the stability property defined in Definition 2.3. It is also easy to see that eliminating a rotation will expose a new set of rotations under the further reduced shortlists. This makes eliminating some rotations a prerequisite for accessing other rotations.

Definition 3.14. *A rotation π is an **explicit predecessor** of rotation ρ if for some $(x_k, y_k) \in \rho$ and some $y \neq y_k$, the pair (x_k, y) is eliminated by π and x_k prefers y to $y_{(k+1) \mod r}$. Let the relation $\pi < \rho$ hold if and only if π is an explicit predecessor of ρ. The reflexive transitive closure of $<$, denoted as $\leq$, is a partial order [Irving and Leather, 1986]. Hence, the set of all rotations that can be exposed, equipped with $\leq$, forms the rotation poset $P, \leq$.*

Clearly, a rotation will only become exposed if and only if all its explicit predecessors have been eliminated. We now demonstrate elimination with our toy example.

Example 3.15. *In Example 3.12, when we eliminate ρ_1, we find a new stable matching*

$$M_{\rho_1} = (1,3),(2,2),(3,5),(4,6),(5,1),(6,4)$$

which exposes the rotation $\rho_2 = (1,3),(2,2),(3,5),(4,6)$. Indeed, in this problem, there are 5 rotations related by $<$ which we can express in the following rotation poset graph.

$$\rho_1 \longrightarrow \rho_2 \longrightarrow \rho_3 \longrightarrow \rho_4 \longrightarrow \rho_5$$

Note that here, we do not include the edges which hold with transitivity. The pairs in

each rotation are

$$\rho_1 = (1,4),(2,3),(3,2),(5,5),(6,1)$$
$$\rho_2 = (1,3),(2,2),(3,5),(4,6)$$
$$\rho_3 = (5,1),(3,6),(4,3)$$
$$\rho_4 = (1,2),(2,5),(5,6)$$
$$\rho_5 = (1,5),(2,6),(4,1)$$

So far, we have shown that traversing the rotation poset graph will allow us to encounter different stable matchings. However, the number of stable matchings is known to be exponential in the number of agents [Knuth, 1976] [Irving and Leather, 1986], and thus it is not possible for an efficient algorithm to search the entire space of stable matchings. Irving et al. [1987] establishes the following definition, which reduces the problem to a problem of finding the set of rotations that can be eliminated together while maximizing some metric.

Definition 3.16. *Let ρ be a rotation that is exposed in a stable matching M, and let M_ρ be the stable matching obtained by eliminating ρ. The **weight** $w(\rho)$ is the difference in the social welfare of M and M_ρ. We can compute $w(\rho)$ efficiently as follows because only $x_k, y_k \in \rho$ get a different pair before and after the elimination.*

$$w(\rho) = \sum_{k=0}^{r-1} \left(v_{X x_k, y_{(k+1) \mod r}} - v_{X x_k, y_k}\right) + \sum_{k=0}^{r-1} \left(v_{Y y_k, x_{(k-1) \mod r}} - v_{Y y_k, x_k}\right)$$

From this definition, we can derive a corollary which says that the social welfare of the stable matching M_S obtained by eliminating rotations in some set S is

$$SW(M_S \mid v) = SW(M \mid v) + \sum_{\rho \in S} w(\rho) \tag{3.1}$$

Hence, we can obtain the optimal stable matching by maximizing the sum of the weights of rotations. Irving and Leather [1986] presented the following theorem which establishes the optimality and efficiency of this approach.

Theorem 3.17. *Let a closed subset C of rotation poset be a subset where for every $\rho \in C$, all explicit predecessors of ρ are in C. There is a one-to-one correspondence between the stable matchings $\mathbf{M}$ of a stable matching problem and the closed subsets of the rotation poset.*

Optimality is established because all stable matchings correspond to a closed subset. Hence, any stable matching can also be found using the approach. Efficiency is established because from Equation 3.1 and Theorem 3.17 we can reduce the problem to the maximum weight closed subset problem, which is a widely-known problem with polynomial time solutions.

In the maximum weight closed subset problem, the input is a general directed graph with weighted nodes, and we want to find a subset C of nodes which maximizes the weights such that for every node $v \in C$, all nodes with outgoing edges to v are also in C.

This problem has been explored in other contexts, such as open pit mining and project planning [Ahuja et al., 1993]. The most prominent solution is a network flow-based approach, which we adopt.

Now, we present our implementation of Irving's algorithm, guided by Irving et al. [1987]. For an overview, refer to the pseudocode below.

Algorithm 1 IRVING($v_1, v_2, \sigma_1, \sigma_2$)

1: $M_X \leftarrow$ GALE-SHAPLEY(σ_1, σ_2)
2: $s_X, s_Y \leftarrow$ FIND-SHORTLISTS(M_X, σ_X, σ_Y)
3: $\Pi \leftarrow$ FIND-ALL-ROTATIONS(s_X, s_Y) ▷ In this step, also create a map of pairs (x_i, y_j) to a rotation that eliminated it.
4: Construct P', a sparse graph representing the rotation poset P
5: $w \leftarrow$ WEIGHTS(Π, v_1, v_2)
6: $C^* \leftarrow$ MAXIMUM-WEIGHT-CLOSED-SUBSET(P', w)
7: $M^* \leftarrow$ ELIMINATE-ROTATIONS(M_X, Π, C^*)
8: **return** M^*

We have previously presented line 1 in Section 3.7.1, line 2 in Definition 3.8, line 5 in Definition 3.16, and line 7 in the comment after Definition 3.13. We will show the rest of the steps in the order of lines 3, 4, 6.

The first key step in implementing Irving's algorithm is to find the set of all rotations Π which can be exposed at any point. This is an important step because we require all rotations to be known to search for the most optimal one. We implemented the method described in Irving et al. [1987] which runs in $O(n^3)$ time, although Gusfield [1987] has an improve version that runs in $O(n^2)$ time. The method works by iteratively performing rounds in which it finds and eliminates all exposed rotations. Rotations that are found in the same iteration are put into the same *layer*, where the initially exposed rotations are in layer 0. More formally, we present the following pseudocode.

Algorithm 2 FIND-ALL-ROTATIONS(s_X, s_Y)

1: $\Pi \leftarrow []$
2: $D \leftarrow \{\}$ ▷ the map which keeps track of which rotation eliminated each pair
3: **while** $(\rho^{(1)}, \rho^{(2)}, \ldots, \rho^{(k)}) =$ FIND-ROTATIONS(s_X, s_Y) is not empty **do**
4: $\quad \Pi \leftarrow \Pi + (\rho^{(1)}, \rho^{(2)}, \ldots, \rho^{(k)})$
5: $\quad$ **for** each rotation $\rho^{(l)}$ in $(\rho^{(1)}, \rho^{(2)}, \ldots, \rho^{(k)})$ **do**
6: $\quad\quad$ Eliminate $\rho^{(l)}$ in s_Y
7: $\quad\quad$ Update D as pairs are eliminated
8: $\quad$ **end for**
9: $\quad$ Reduce s_X by examining reduced s_Y
10: **end while**
11: **return** Π, D

The central insight is that in line 9, it is only necessary to keep updated the first and second agents in each X shortlist, and the last agent in each Y shortlist. This is because the existence of a rotation is determined completely by those positions. The rest of the

shortlist is not reduced when it is supplied to FIND-ROTATIONS. With this workaround, we only have to traverse each shortlist once in total throughout the whole algorithm. Since each shortlist is $O(n)$ in size and there are $O(n)$ shortlists, the traversal takes $O(n^2)$ time. Another cost arises from looping through rotations. As proven in Irving et al. [1987], the total number of rotations is $O(n^2)$ since each pair can appear in only one rotation. In line 5, we loop through the rotations once. While the original paper claims this can be done in $O(n^2)$ time total, each iteration of our loop takes $O(n)$ time since each rotation contains $O(n)$ pairs. In advance of discussing FIND-ROTATIONS, we claim that it runs in $O(n)$ time and that there are at most $O(n^2)$ levels. The latter can be seen if at each level only one rotation was exposed, as Example 3.15 was. Hence, the algorithm runs in $O(n^3)$ time.

Next, we briefly examine the algorithm that finds all exposed rotations, used as an internal routine in the algorithm above. In this algorithm, we construct the following directed graph $G(M)$ so that each cycle corresponds to an exposed rotation.

Definition 3.18. *Let M be a stable matching, and let s_X, s_Y be the reduced shortlists for M. Then, let the graph $G(M) = (V,E)$ be a directed graph defined as follows.*

1. *Define a node for each agent in X, i.e. $V = X$*
2. *Define an edge from x_i to $x_{i'}$ if and only if $x_{i'}$ is matched to $y_{j'} \in Y$ in M and $y_{j'}$ is second on x_i's shortlist in s_X.*

Finding every cycle in $G(M)$ can be achieved in $O(n)$ with a depth-first search.

We now discuss item 4 from the overview, which constructs a sparse representation of the rotation poset P. Trivially, we can think of a graph P, where nodes are the rotations and edges are the relation $<$. However, this can be very dense. Given that there are $O(n^2)$ nodes, P in the worst case can have $O(n^4)$ edges. Irving et al. [1987] introduces a graph P', a subgraph of P, which only has $O(n^2)$ edges and preserves the closed subsets.

Definition 3.19. *Let P be a rotation poset. Let $P' = (V,E)$ be a directed acyclic graph defined as follows.*

1. *Define a node for each rotation in P.*
2. *Define an edge between from rotation π to rotation ρ if either*
   - *Rule 1: $(x,y) \in \pi$ and y' is the first agent below y in x's shortlist such that (x,y') is in some rotation, and this rotation is ρ.*
   - *Rule 2: $(x,y) \in \rho$, (x,y') is eliminated by π,y is the first agent above y' in some rotation,* ***and y' is more preferred than the agent in Y that x will be matched with upon eliminating ρ.***

Remark 3.20. *We believe that there is a mistake in the presentation of Rule 2 in Irving et al. [1987], which originally read*

- *Rule 2: $(x,y) \in \rho$, (x,y') is eliminated by π, and y is the first agent above y' in some rotation.*

Proof. We prove that the bolded condition in Rule 2 is necessary. We show that there is an application of Rule 2 which creates an edge from a rotation π to rotation ρ even though the relation $\pi < \rho$ does not hold. Upon showing this, it no longer holds that P' is a subgraph of P.

Suppose that all of the conditions outlined in the original Rule 2 hold. For this proof, let $\succ$ denote the preference ordering of x on agents from Y. Any shortlist for x will satisfy this ordering. Let y'' be the agent that x will receive after eliminating ρ. If x was in the k-th pair of ρ, then y'' will be in the $(k+1) \mod r$th pair of ρ. The original Rule 2 admits the following partial ordering.

$$y \succ y'' \succ y'$$

Recall from Definition 3.14 that π is an explicit predecessor of ρ if x prefers y' to y''. However, this is not the case. Hence, the condition outlined in Rule 2 alone does not establish that π is an explicit predecessor of ρ. Therefore, it is possible that an edge is drawn from π to ρ in P' even if $\pi > \rho$ does not hold. □

Even with the added condition, we can construct P' efficiently as done in Algorithm 3. In Algorithm 3, the input D is the map created by Algorithm 2 which maps pairs to their eliminating rotation (if any). Note that the three loops account for $O(n^2)$ iterations because any increase in j' will also increase j in one step. In our implementation, an additional complexity of $O(n)$ is introduced by finding y'' and checking whether $y'' \succ y'$ in $O(n)$ time. However, the runtime complexity of $O(n^3)$ is not the bottleneck of this algorithm. Even with the modification, the following theorem for correctness still holds.

Theorem 3.21. *P is the transitive closure of P'. Hence, any closed subset in P is also a closed subset in P'.*

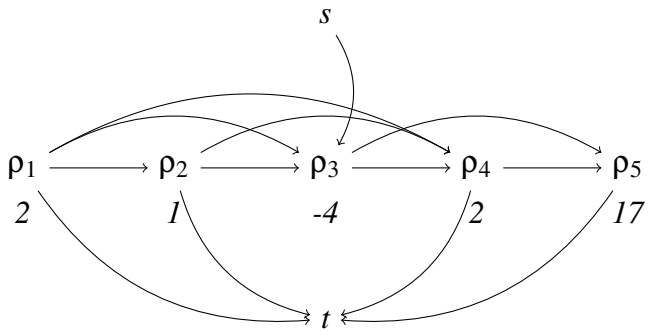
Proof. In the proof to this theorem, Irving et al. [1987] claimed without proof that P' is a subgraph of P. The rest of their proof focuses on showing that if for some rotations π, ρ, π explicitly precedes ρ, then there is a directed path from π to ρ in P'. From Remark 3.20, it should be clear that the only edge that will be eliminated by adding the bolded condition in Definition 3.19 is one where π is not an explicit predecessor of ρ. Hence, the correctness of the theorem follows. □

The final challenge of implementing Irving's algorithm is to find the maximum weight closed subset of P' efficiently. There are multiple known approaches to this problem, and we choose the network flow-based approach because it is the simplest and sufficiently fast. The network flow-based method works by creating the flow network $P'(s,t)$ as follows. First, add the source and sink nodes to P'. Then, add an edge from the source to each rotation with a strictly negative weight with capacity $-w(\rho)$. Next, add an edge from each rotation with a strictly positive weight to the sink with capacity $w(\rho)$. Finally, add any edges from P' with infinite capacity. To get a maximum weight closed subset, we obtain a minimum cut in $P'(s,t)$ using a general max-flow algorithm. Then, all the nodes with positive weights in the t-side of the cut and their predecessors form a maximum weight closed subset. Since this is a well-known method, we will omit the proof and refer the reader to Ahuja et al. [1993].

Algorithm 3 CONSTRUCT-SPARSE-ROTATION-POSET-GRAPH(Π, s_X, D)

$P' \leftarrow \{\pi : \{\,\} \text{ for } \pi \in \Pi\,\}$
Construct D', a map from each pair to a rotation it appears in by iterating through the pairs in every rotation. Pairs without such rotation will be excluded.
for each x in X **do**
    $j \leftarrow 0$
    while $j <$ length of $s_X[x]$ **do**
        $y \leftarrow j$th element in $s_X[x]$
        if $(x,y) \notin D'$ **then** ▷ In both rules (x,y) must be in some rotation
            $j \leftarrow j+1$, **continue**
        end if
        $j' \leftarrow j+1$
        while $j' <$ length of $s_X[x]$ **do**
            $y' \leftarrow j'$th element in $s_X[x]$ ▷ In both rules, y is above y'
            if $(x,y') \in D'$ **then** ▷ Rule 1 satisfied
                $\pi \leftarrow$ rotation of (x,y) in D'
                $\rho \leftarrow$ rotation of (x,y') in D'
                Add ρ to $P'[\pi]$, **break**
            end if
            if $(x,y') \in D$ **then**
                Find y'' by examining $\rho \in \Pi$
                if $y'' \succ y'$ **then** ▷ Rule 2 satisfied
                    $\pi \leftarrow$ eliminating rotation of (x,y') from D'
                    $\rho \leftarrow$ rotation of (x,y) in D'
                    Add ρ to $P'[\pi]$
                end if
            end if
            $j' \leftarrow j'+1$
        end while
        $j \leftarrow j'$
    end while
end for
return P'

Example 3.22. *The flow network $P'(s,t)$ constructed from Example 3.12 is as follows.*



For readability, we do not label edges with their capacities. Instead, we label the rotation nodes with their weights.

The maximum weight closed subset contains all nodes in P, and the total weight of the rotations is 18*. Quite clearly, the minimum cut in* $P'(s,t)$ *is* $\{s\},\{\rho_1,\rho_2,\rho_3,\rho_4,\rho_5,t\}$ *where the capacity across the cut is* 4*. This also induces the maximum weight closed subset above. The stable matching chosen by Irving has a social welfare of* 37*, while the stable matching chosen by X-optimal Gale-Shapley has a social welfare of* 19*.*

Since the graph has $O(n^2)$ nodes and edges, most flow algorithms will run in $O(n^6)$ time. Upon implementation, we compiled a list of algorithms available in Python and their runtime complexity on our problem in Table 3.3. Irving et al. [1987] showed that the Ford-Fulkerson algorithm runs in $O(n^4)$ time by proving the following lemma.

Table 3.3: Runtime Complexity of Popular Flow Algorithms for Irving's Algorithm

Algorithm	**Library**	**Complexity**	**Comments**
Ford-Fulkerson	socialchoicekit	$O(n^4)$	Naive implementation used in the Birkhoff-von Neumann algorithm from Section 3.6. Requires integral capacities.
Edmonds-Karp	Graph Tool, SciPy	$O(n^6)$	Claims to support non-integer capacities.
Dinic (1970)	SciPy	$O(n^6)$	
Goldberg's push-relabel	Graph Tool	$O(n^6)$	
Boykov-Kolmogorov	Graph Tool	$O(n^7)$	Claims to typically perform much better than the upper bound.

Lemma 3.23. *The capacity of the minimum cut in* $P'(s,t)$ *is bounded by* $O(n^2)$*.*

Irving et al. [1987] also refers the reader to Sleator and Tarjan [1983], which solves the problem in $O(n^4 \log n)$ time. However, we were not able to find a reliable working implementation in Python for this algorithm.

One major shortcoming of the flow-based approaches is that they require integral capacities, meaning that the values of the cardinal utility functions must be integral. In the normalized social choice approach, we employ an equal utility system where every agent's total utility across all of the choices sums to 1. However, such a normalization scheme would produce non-integer utilities that cannot be handled by Ford-Fulkerson. A workaround is to give each agent a fixed integer utility allowance and require integral utilities, as we did with Example 3.12. We further discuss the implications of this constraint in Section 6.1.

Chapter 4

Improving Distortion with Elicitation

As we briefly covered in the introduction, elicitation is a recent area of study that explores how distortion can be reduced by having access to limited cardinal information. Our goal in this chapter is to present our implementation of the major elicitation-based algorithms and propose a new elicitation-based algorithm for stable matching. The former can be found in Section 4.1 and the latter can be found in Section 4.2.

Elicitation, also called querying, was first considered by the pioneering work of Amanatidis et al. [2021]. Under their model, an elicitation-based algorithm would be able to obtain partial cardinal information by *querying* agents about their cardinal profile. In this work, we work with the following type of query.

Definition 4.1. *A **value query** asks an agent to answer the cardinal utility for a specific choice. More formally, it is a function $\mathcal{V} : N \times A \to \mathbb{R}_{\geq 0}$ which, given an agent i and a choice j, returns agent i's cardinal utility for choice j, i.e. v_{ij}. For stable matching, the function is $\mathcal{V} : X \times Y \to \mathbb{R}_{\geq 0}$ for queries to agents in X and $\mathcal{V} : Y \times X \to \mathbb{R}_{\geq 0}$ for queries to agents in Y.*

The purpose of querying is to obtain useful information about the agents' underlying cardinal profile in cases when supplying the full cardinal profile is too demanding for the agents.

Definition 4.2. *Any elicitation-based algorithm is a **mechanism** $\mathcal{M} = (Q, f)$ with access to a value or comparison oracle which is equipped with*

1. *an algorithm Q which makes queries to the oracle. Q takes as input an ordinal preference profile σ, decides which agent-choice combination(s) to query, and returns the answer(s) to these queries.*
2. *an ordinal algorithm, also known as a social choice function, f, as defined in Section 2.1, with the exception that f also takes as input the answers from Q, which we name $Q(\sigma)$.*

In stable matching, a mechanism $\mathcal{M} = (Q_X, Q_Y, f)$ consists of two query algorithms, one for each set of agents. The social choice function f will also take as input two sets $Q_X(\sigma_X), Q_Y(\sigma_Y)$.

Table 4.1: Composition of Elicitation-based Algorithms and Settings

	Voting	**One-sided Matching**	**Stable Matching**
Binary Search-based	k-ARV: Amanatidis et al. [2021]	λ-TSF: Amanatidis et al. [2022]	Double λ-TSF: Section 4.2
Two Query-based	SC-TwoQueries: Amanatidis et al. [2024], under limitations	Match-TwoQueries: Amanatidis et al. [2024]	None

The distortion of a mechanism is the distortion of the social choice function f.

Previous works have shown that elicitation-based methods are effective in reducing distortion while only requiring a small portion of the cardinal profile. For deterministic voting rules, Amanatidis et al. [2021] showed that elicitation of one query per individual can lower the distortion to $O(m)$, where m is the number of alternatives. Furthermore, they proposed a mechanism called k-ARV, which achieves a distortion of $O(\sqrt{m})$ with $O(\log m)$ queries per agent and constant distortion with $O(\log^2 m)$ queries per agent. More recently, Amanatidis et al. [2024] proposed a mechanism known as SC-TwoQueries, which makes only two queries but achieves $O(\sqrt{m})$ distortion. However, this mechanism works under certain conditions explained in Section 4.1.2. Amanatidis et al. [2024] also established a general lower bound of $\Omega(m^{\frac{1}{\lambda}})$ distortion for any elicitation-based mechanism with $\lambda = \Theta(1)$ queries per agent in the voting setting.

Elicitation has also been studied in the one-sided matching setting. Amanatidis et al. [2022] showed that with n as the number of items, $O(n^{\frac{1}{\lambda}})$ can be achieved with $O(\lambda \log n)$ queries per agent. This result is equivalent to the result of k-ARV and is achieved using a similar mechanism named λ-TSF. The same paper established a lower bound of $\Omega(n^{\frac{1}{\lambda}})$ for the distortion of any mechanisms making $\lambda = \Theta(1)$ queries. Amanatidis et al. [2024] also came up with the Match-TwoQueries mechanism that achieves $O(\sqrt{n})$ distortion with only two queries in one-sided matching.

The previously mentioned works [Amanatidis et al., 2022], [Amanatidis et al., 2024] also studied the application of their algorithm on more general combinatorial problems, including two-sided matching. However, there has been no previous exploration of elicitation mechanisms in the stable matching setting. Table 4.1 is a concise summary of the algorithms we cover in this chapter.

4.1 Implementation of Elicitation-based Algorithms

In this section, we will provide a high-level overview of the state-of-the-art mechanisms proposed by Amanatidis et al. [2021], Amanatidis et al. [2022], and Amanatidis et al. [2024], and we then will discuss their implementation. These mechanisms can be split into two types, binary search-based and two query-based, which we show in Table 4.1.

Since the authors of all three papers provided pseudocode for their mechanisms, the

main implementational challenge was to design the oracle which would provide answers to the query algorithm Q. This is an oracle superclass called `Elicitor` which has an `elicit` method taking an agent i and a choice j as parameters and returns a float. Users are supposed to instantiate a subclass of `Elicitor`, which implements a superclass method `_elicit_impl`. This private method is called from the `elicit` method. One subclass of `Elicitor` we provide is `ValuationProfileElicitor`, which takes as input a cardinal profile v upon initialization and automatically provides answers from v when `_elicit_impl` is called. While this oracle requires the entire cardinal profile, which we assume is not available in a practical elicitation environment, this oracle is useful for automating empirical computational simulations. Another subclass of `Elicitor` we provide is `SynchronousStdInElicitor`, which prints a basic English query to standard output and reads a number from standard input. While this oracle would be more applicable in scenarios where we are querying real agents, we expect that more complicated oracles will need to be developed by the user. We therefore also provide a subclass `LambdaElicitor`, to which users can provide custom `_elicit_impl` functions to use. To make the elicitation process easier for agents, we also provide a `memoize` option to every oracle, enabled by default. When this is enabled, the oracle will remember answers to previous `_elicit_impl` calls and will not call `_elicit_impl` with the same parameters. We expect that this will also help reduce the number of queries in practice for binary search-based mechanisms.

4.1.1 Binary Search-based Mechanisms

The binary search-based method is the first elicitation-based method proposed by Amanatidis et al. [2021]. The main idea of this approach is to construct a *simulated valuation function*, $\tilde{v}$, which approximates the true cardinal profile v. In this section, we limit our definition to be applicable to only the voting and one-sided matching settings. We will expand these definitions to the stable matching setting in Section 4.2.

Definition 4.3. *Let v be the cardinal profile we want to simulate, and let it be an (n,m) matrix. Let λ be a natural number $\lambda \leq m$, which we parameterize the mechanism with. Define $\lambda+1$ thresholds $\alpha_0, \alpha_1, \ldots, \alpha_\ell, \ldots, \alpha_\lambda$ where*

$$\alpha_\ell = m^{\frac{l}{\lambda+1}}$$

*Notice that $\alpha_0 = 1$. Let v_i^* be the cardinal utility of agent i's favorite choice. An α_ℓ-**acceptable set** $Q_{i,\ell}$ is a set of choices such that*

1. *$Q_{i,0}$ contains the favorite choice of agent i*
2. *$Q_{i,\ell}$ for $\ell \in \{1,\ldots,\lambda\}$ contains all elements a_j such that $v_i(a_j) \geq \frac{v_i^*}{\alpha_\ell}$ and $a_j \notin Q_{i,\ell-1}, \ldots, a_j \notin Q_{i,0}$.*

*Let $Q_i = \bigcup_{\ell \in \{0,\ldots,\lambda\}} Q_{i,\ell}$. A **simulated valuation function** $\tilde{v}_i$ is a function with the same domain and codomain as v such that*

$$\tilde{v}_i(a_j) = \begin{cases} \frac{v_i^*}{\alpha_\ell}, & a_j \in Q_{i,\ell} \text{ for some } \ell \in \{0,\ldots,\lambda\} \\ 0, & a_j \notin Q_{i,\ell} \text{ for all } \ell \in \{0,\ldots,\lambda\} \end{cases}$$

From the simulated valuation functions, we can construct a *simulated cardinal profile* $\tilde{v}$, a matrix representation of $(\tilde{v}_i)_{i \in N}$.

A binary search-based mechanism, an overview of which is presented below, would calculate $\tilde{v}$ by first constructing each α_ℓ-acceptable set. This can be efficiently done by performing a binary search to find the least preferred choice a_j such that $\frac{v_i^*}{\alpha_\ell} \leq \tilde{v}_i(a_j)$ holds. Then, any choice $a_{j'}$ which is less preferred than the least preferred choice in $Q_{i,\ell-1}$ but more preferred than a_j is in $Q_{i,\ell}$. The algorithm Q knows the most preferred choice and can perform a binary search because Q is provided an ordinal profile σ.

Binary Search-based Mechanism Overview

Compute $\alpha_0, \alpha_1, \ldots, \alpha_\lambda$.
For each agent i, perform the following routine to construct $\tilde{v}_i$ independently.

1. Query agent i for her most preferred choice. Let this be v_i^*.
2. For each $\ell = \{1, \ldots, \lambda\}$, going from lowest to highest, compute $Q_{i,\ell}$.
3. Set $\tilde{v}_{ij} = \frac{v_i^*}{\alpha_\ell}$ for all choices in $Q_{i,\ell}$.

For choices that are not in any setting, set their simulated value to 0. Note that this can be done by initializing $\tilde{v}_{ij} = 0$.
Return an outcome that maximizes the social welfare.

As seen in Table 4.1, k-Acceptable Range Voting, or k-ARV, is the original mechanism that works with the voting setting. In k-ARV, the authors use the symbol k instead of the symbol λ in Definition 4.3. Confusingly, λ_ℓ is used to indicate our thresholds α_ℓ. Moreover, $S_{i,\ell}$ is used instead of $Q_{i,\ell}$ for acceptable sets, and $S_{i,\ell}$ contains $S_{i,\ell-1}$. λ-Threshold Step Function, or λ-TSF, was introduced in a followup work, which adapts k-ARV to the one-sided matching setting. In λ-TSF, α_ℓ is defined to be multiplicative factors on v_i^*, i.e. the paper's α_ℓ is equal to $\frac{1}{\alpha_\ell}$ in our definition. Since the authors of k-ARV and λ-TSF have included the pseudocode for their mechanisms in their respective papers, we will only briefly describe their high-level design in this section. We have attached their pseudocode in Appendix B.1 and Appendix B.2, with some syntactic modifications. In Section 4.2, we propose the Double λ-Threshold Step Function, or Double λ-TSF, in which we apply λ-TSF to the stable matching setting.

In general, we can ensure that the outcome returned by the binary search-based mechanism is only optimal with respect to the simulated cardinal profile, not to the real cardinal profile. Hence, we need theorems to prove that this approach achieves low distortion. As mentioned at the beginning of this chapter, both k-ARV and λ-TSF achieve $O(m^{\frac{1}{\lambda+1}})$ distortion with $O(\lambda \log m)$ queries per agent, where m is the number of choices. This implies $O(\sqrt{m})$ distortion with $O(\log m)$ queries per agent and $O(1)$ distortion with $O(\log^2 m)$ queries per agent. We choose to demonstrate this for λ-TSF with our own notation, as the proof will become relevant when we discuss extending λ-TSF to stable-matching.

Theorem 4.4. *[Amanatidis et al., 2022] Let n be the number of items in a one-sided matching problem. λ-TSF makes $O(\lambda \log n)$ value queries per agent and achieves a distortion of $2n^{\frac{1}{\lambda+1}}$.*

Proof. Since λ-TSF uses binary search on n choices to find $\lambda+1$ sets, it is clear that the mechanism makes $O(\lambda \log n)$ queries per agent.

The rest of the proof focuses on proving the distortion bound. Consider an arbitrary cardinal profile v. Let M be an optimal matching, and let $\tilde{M}$ be the matching returned by λ-TSF. We denote the item that agent i received in M as a_i.

To begin, we split the set of agents N into those whose received item is in some $Q_{i,\ell}$, say S, and those whose item is not in any $Q_{i,\ell}$, say $\bar{S}$. Then, $S, \bar{S}$ is a partition of N. The social welfare for optimal matching M can be rewritten as

$$SW(M \mid v) = \sum_{i \in N} v_i(a_i) = \sum_{i \in \bar{S}} v_i(a_i) + \sum_{i \in S} v_i(a_i)$$

We evaluate the bounds separately on the two terms. In the term with items in $\bar{S}$,

$$\sum_{i \in \bar{S}} v_i(a_i) < \frac{\sum_{i \in \bar{S}} v_i^*}{\alpha_\lambda} \leq \frac{n \cdot \max_{i \in N} v_i^*}{\alpha_\lambda} \leq \frac{n \cdot SW(\tilde{M} \mid \tilde{v})}{\alpha_\lambda} \leq \frac{n \cdot SW(\tilde{M} \mid v)}{\alpha_\lambda} \tag{4.1}$$

Note that $\alpha_\lambda \neq 0$. The first inequality is established because the item is from $\bar{S}$. The second inequality follows because $\bar{S} \subseteq N$, therefore $|\bar{S}| \leq |N| = n$. The third inequality holds because $\tilde{M}$ is the optimal matching with respect to the simulated cardinal profile $\tilde{v}$. If there was an agent whose utility would exceed $SW(\tilde{M} \mid \tilde{v})$ from receiving their favorite item, then we can achieve better social welfare by giving that item to the agent, and $\tilde{M}$ will no longer be optimal. The fourth inequality holds because the simulated cardinal values are lower bounds on the actual cardinal values.

For the term with items in S, define S_ℓ for each $\ell \in \{0, \ldots, \lambda\}$ as the set of agents who received an item in $Q_{i,\ell}$. It follows that

$$\sum_{i \in S} v_i(a_i) = \sum_{l=0}^{\lambda} \sum_{i \in S_\ell} v_i(a_i) \tag{4.2}$$

When $\lambda = 0$,

$$\sum_{i \in S} v_i(a_i) = \sum_{i \in S} \tilde{v}_i(a_i) \leq SW(\tilde{M} \mid v)$$

In the original paper, the first equality was $\leq$. We believe that this holds as an equality because the single element in S holds the value v_i^* for agent i in both v and $\tilde{v}$. The second inequality holds due to a similar argument made for the third inequality of Equation 4.1.

When $\lambda > 0$, we know by definition for any $\ell \in \{1, \ldots, \lambda\}$ and for any $a_j \in Q_{i,\ell}$,

$$v_i(a_j) \leq \frac{v_i^*}{\alpha_{\ell-1}} = \frac{\alpha_\ell}{\alpha_{\ell-1}} \cdot \frac{1}{\alpha_\ell} \cdot v_i^* = \alpha_1 \cdot \tilde{v}_i(a_j) \tag{4.3}$$

Let $Q_{i,0} = \{a_i^*\}$. Then, also by definition,

$$v_i(a_i^*) = \tilde{v}_i(a_i^*) \leq \alpha_1 \cdot \tilde{v}_i(a_i^*) \tag{4.4}$$

Hence,

$$\sum_{i \in S} v_i(a_i) \leq \alpha_1 \sum_{l=0}^{\lambda} \sum_{i \in S_\ell} \tilde{v}_i(a_i) \leq \alpha_1 \cdot \sum_{i \in N} \tilde{v}_i(a_i) \leq \alpha_1 \cdot SW(\tilde{M} \mid \tilde{v}) \leq \alpha_1 \cdot SW(\tilde{M} \mid v) \quad (4.5)$$

The first inequality follows naturally from Equations 4.2, 4.3, and 4.4. The second inequality holds because all agents are now considered. The third inequality holds because $\tilde{M}$ is an optimal solution and M isn't. The last inequality holds because the $\tilde{v}_{ij}$ is a lower bound on v_{ij}. From Equations 4.1 and 4.5, we have

$$SW(M \mid v) \leq \left(\alpha_1 + \frac{n}{\alpha_\lambda}\right) \cdot SW(\tilde{M} \mid v) = 2n^{\frac{1}{\lambda+1}} \cdot SW(\tilde{M} \mid v) \quad (4.6)$$

The last equality holds because $\alpha_1 = n^{\frac{l}{\lambda+1}}$ and $\frac{n}{\alpha_\lambda} = n^{1-\frac{\lambda}{\lambda+1}}$. □

A similar theorem exists for k-ARV, which is proven in Amanatidis et al. [2021].

4.1.2 Two Query-based Mechanisms

The two query-based method is the second group of elicitation mechanisms. As previously mentioned, Match-TwoQueries uses this method for one-sided matching and SC-TwoQueries is the application of Match-TwoQueries for voting.

As could be understood from the name, this approach only makes two queries to achieve a distortion of $O(\sqrt{n})$. Since Amanatidis et al. [2022] proved a lower bound of $\Omega(n^{\frac{1}{\lambda}})$, this mechanism closes the gap between the distortion bounds for $\lambda = 2$. We implemented Match-TwoQueries but did not implement SC-TwoQueries, as the latter requires certain conditions to be true about the voting problem. In this section, we will introduce Match-TwoQueries first and consider an extension to voting.

Match-TwoQueries uses the first query to ask each agent for their favorite choice. It uses the second query to create the following sufficiently representative assignment.

Definition 4.5. *A **sufficiently representative assignment** M_0 is a many-to-one assignment (where multiple agents can be assigned to the same item, but only one item is assigned to an agent) such that*

1. *for every item $a_j \in A$, there are at most $O(\sqrt{n})$ agents assigned to a_j*
2. *for any matching M, there are at most $\sqrt{n}$ agents who prefer the item they are matched to in M to the item they are assigned to in M_0.*

The main idea of the two query-based mechanisms is to compute a sufficiently representative data structure, such as the sufficiently representative assignment above for one-sided matching. Then, it creates a simulated cardinal profile, albeit with different characteristics from the one in Definition 4.3. To do this, the mechanism queries each agent about a choice in this sufficient representative data structure and sets the simulated cardinal value to the utility of the queried choice for any choices more preferred over the queried choice (including the queried choice). All other choices will get a simulated

cardinal value of 0. At this point, the mechanism has a simulated cardinal profile which it feeds to a cardinal algorithm. Amanatidis et al. [2024] provides pseudocode for Match-TwoQueries, which we attach in Appendix B.3. As seen in the pseudocode, our implementation of Match-TwoQueries uses the maximum weight matching algorithm we defined in Section 3.6 as the cardinal algorithm.

Theorem 4.6. *[Amanatidis et al., 2024] Match-TwoQueries has distortion* $O(\sqrt{n})$.

Proof. We only provide an informal proof sketch as the full proof is provided in the original paper. Let $\tilde{M}$ be the matching produced by Match-TwoQueries. We can split the pairs in $\tilde{M}$ to those where the social welfare is revealed, or known because the agent received an item they were queried about, and to those where the social welfare is concealed. We let the welfare from the revealed pairs be denoted as $SW(\tilde{M})_R$ and the welfare from the concealed pairs be denoted as $SW(\tilde{M})_C$. Then, we show

$$SW(M) \leq (1+2\sqrt{n}) \cdot SW_R(\tilde{M})$$

where M is an optimal matching. The 1 in $1+2\sqrt{n}$ comes from

$$SW_R(M) \leq SW_R(\tilde{M})$$

which is deduced from the fact that $\tilde{M}$ is a matching that maximizes the social welfare based only on the revealed utilities. Since

$$SW(M) = SW_R(M) + SW_C(M)$$

we want to show that

$$SW_C(M) \leq 2\sqrt{n} \cdot SW_R(\tilde{M})$$

The proof for this bound can be found in Theorem 1 of Amanatidis et al. [2024]. □

In general, nothing in the two query-based mechanisms requires a complicated implementation, except computing the sufficiently representative data structure. For one-sided matching, computing the sufficiently representative matching can easily be done by Algorithm 10 in Appendix B.3, which Amanatidis et al. [2024] introduced and proved. In socialchoicekit, we have also implemented this algorithm. However, computing a sufficiently representative data structure becomes a roadblock for the voting setting. The mechanism SC-TwoQueries also presented by Amanatidis et al. [2024] considers a *sufficiently representative set*, as follows.

Definition 4.7. *A* ***sufficiently representative set*** *is a set of alternatives* $B \subseteq A$ *such that for any constant* $c \leq 1$, $|B| = c \cdot \sqrt{m}$ *and for every alternative* $a_j \in A$, *at most* $\sqrt{m}$ *agents prefer* a_j *over their favorite alternative in* B.

With a sufficiently representative set B, SC-TwoQueries can ask each agent about their favorite alternative in B in the second query. This would allow SC-TwoQueries to create a simulated cardinal profile similar to the one Match-TwoQueries made with one-sided matching. However, this is known to only exist when $m = \Omega(n)$, as deduced by Amanatidis et al. [2024] from the following theorem of Jiang et al. [2020].

Theorem 4.8. *For any $\xi \in \{1,\dots,n\}$, there exists a set S of alternatives with $|S| \le 16 \cdot \frac{n}{\xi}$ such that for every $a_j \in A$, there are at most ξ agents that prefer a_j over their favorite alternative in S.*

When $\xi = \sqrt{n}$ and $n = \Theta(m)$, Definition 4.7 is satisfied with $c = 16$. Note that this is a restrictive condition. We also remark that the set S mentioned in Theorem 4.8, called the approximately stable committee, is difficult to compute.

4.2 Double λ-TSF for Stable Matching

In this section, we extend the binary search-based method for one-sided matching by Amanatidis et al. [2022], λ-Threshold Step Function, to stable matching. We name the new mechanism Double λ-Threshold Step Function, or Double λ-TSF, as it involves having two λ-TSFs, one for each set of agents. We begin by adapting Definition 4.3 to the stable matching setting.

Definition 4.9. *Let v_X, v_Y be cardinal profiles we want to simulate, corresponding to utilities for X and Y, respectively. Then, both v_X and v_Y will be (n,n) matrices, as defined in Section 2.1. Use the same definition for v_i^*, λ and α_ℓ for each $\ell \in \{0,\dots,\lambda\}$ as Definition 4.3 substituting n for m. For each agent $x_i \in X$ and $\ell \in \{0,\dots,\lambda\}$, let the α_ℓ-acceptable set $Q_{i,\ell}$ be a set of agents in Y such that*

1. *$Q_{i,0}$ contains the favorite choice of agent x_i*
2. *$Q_{i,\ell}$ for $\ell \in \{1,\dots,\lambda\}$ contains all agents y_j such that $(v_X)_i(y_j) \ge \frac{v_i^*}{\alpha_\ell}$ and $y_j \notin Q_{i,\ell-1},\dots,y_j \notin Q_{i,0}$*

Let $Q_i = \bigcup_{\ell \in \{0,\dots,\lambda\}} Q_{i,\ell}$. Also define $\tilde{v}_X$ similarly as follows.

$$(\tilde{v}_X)_i(y_j) = \begin{cases} \frac{v_i^*}{\alpha_\ell}, & y_j \in Q_{i,\ell} \text{ for some } \ell \in \{0,\dots,\lambda\} \\ 0, & y_j \notin Q_{i,\ell} \text{ for all } \ell \in \{0,\dots,\lambda\} \end{cases}$$

Similarly, define α_ℓ-acceptable sets $R_{j,\ell}$, the union R_ℓ, and $\tilde{v}_Y$ by swapping X and Y.

The basic idea of λ-TSF is unchanged in Double λ-TSF, which is shown in Algorithm 4. We create two simulated cardinal profiles, $\tilde{v}_X, \tilde{v}_Y$, which simulate v_X, v_Y, respectively. We do so by performing the routine in Algorithm 5 separately for X and Y.

Algorithm 4 DOUBLE-λ-THRESHOLD-STEP-FUNCTION$(\lambda, \sigma_X, \sigma_Y)$

1: $\tilde{v}_X \leftarrow$ CONSTRUCT-SIMULATED-CARDINAL-PROFILE$(Q(\sigma_X), \sigma_X, \lambda)$
2: $\tilde{v}_Y \leftarrow$ CONSTRUCT-SIMULATED-CARDINAL-PROFILE$(Q(\sigma_Y), \sigma_Y, \lambda)$
3: $M \leftarrow$ IRVING$(\tilde{v}_X, \tilde{v}_Y, \sigma_X, \sigma_Y)$

The pseudocode in Algorithm 5 we show is for X, but it is easily replaceable with Y.

Algorithm 5 roughly follows the overview box in Section 4.1.1. The only major difference is that in Double λ-TSF,

Algorithm 5 CONSTRUCT-SIMULATED-CARDINAL-PROFILE($Q(\sigma_X), \sigma_X, \lambda$)

1: $\tilde{v}_X \leftarrow \mathbb{0}$ ▷ where $\mathbb{0}$ is the zero matrix
2: **for** $x_i \in X$ **do** ▷ from x_i, we also have the index i
3: $v_i^* \leftarrow Q(\sigma_X)$ ▷ where Q queries agent x_i for their favorite agent
4: $(\tilde{v}_X)_i(y_i^*) \leftarrow v_i^*$
5: $(p_i^*)' = v_i^*$ ▷ Cache for previous p_i^*
6: **end for**
7: **for** $l \in \{0, \ldots, \lambda\}$ **do**
8: $\alpha_\ell \leftarrow n^{\frac{l}{\lambda+1}}$
9: **for** $x_i \in X$ **do** ▷ from x_i, we also have the index i
10: $p_i^* \leftarrow \text{BSEARCH}(0, n, \alpha_\ell, v_i^*)$ ▷ Least preferred agent in $Q_{i,\ell}$
11: **for** $y_j \in Y$ such that $(p_i^*)' \succ_{X,i} y_j \succ_{X,i} p_i^*$ **do**
12: $(\tilde{v}_X)_i(y_j) \leftarrow (v_X)_i(p_i^*)$ ▷ $(v_X)_i(p_i^*)$ should be memoized by oracle
13: $(p_i^*)' \leftarrow p_i^*$
14: **end for**
15: **end for**
16: **end for**
17: **return** $\tilde{v}_X$

1. we require the cardinal profiles v_X, v_Y to only contain integral utilities
2. we use the cardinal utility of the least preferred agent p_i^* in $Q_{i,\ell}$ (or $R_{j,\ell}$), as the simulated utility for all agents in $Q_{i,\ell}$ (or $R_{j,\ell}$) instead of the non-integer value $\frac{v_i^*}{\alpha_\lambda}$

The BSEARCH routine is a common routine used in k-ARV and λ-TSF. An example pseudocode by Amanatidis et al. [2021] can be found in Appendix B.1.

To prove the theoretical correctness of Algorithm 4, we need to prove Theorem 4.10. This is because a normal cardinal algorithm for stable matching would only ensure stability in terms of the cardinal profiles provided, that is $\tilde{v}_X, \tilde{v}_Y$. Here, we assume that a blocking pair over the cardinal profile will employ the strict inequality ordering $>$.

Theorem 4.10. *The matching M produced by the mechanism Double λ-TSF satisfies the stability property defined in Definition 2.3 with respect to v_X, v_Y.*

Proof. We need to show that M admits no blocking pairs. We will rely on the following lemma about our implementation of Irving's algorithm.

Lemma 4.11. *A matching produced by the implementation of Irving's algorithm in Section 3.7.2 has no blocking pairs with respect to σ_X, σ_Y supplied to the algorithm.*

The proof for this lemma is that when we first construct the shortlists s_X, s_Y in line 2 of Algorithm 1, we use the ordinal profiles σ_X, σ_Y, instead of the cardinal profiles v_X, v_Y. Therefore, any ordering assumed by Irving's algorithm is with respect to σ_X, σ_Y. Since any two agents $(v_X)_i(y_j) > (v_X)_i(y_{j'})$ implies $y_j \succ_i y_{j'}$ and similarly for $>$ over v_Y and $\succ_j \in \sigma_Y$, a blocking pair in M with respect to v_X, v_Y exists only if there is a blocking pair in M with respect to σ_X, σ_Y. However, Theorem 3.17 states that the latter is false. □

We also attempt to show that Double λ-TSF achieves low distortion. Since Double λ-TSF is structured similarly to λ-TSF, one might consider using a similar proof structure to that of Theorem 4.4. However, we were not able to derive a distortion bound with this approach. We show an informal sketch. For example, let

$$SW(M \mid v_X, v_Y) = \sum_{(x_i,y_j)\in M} (v_X)_i(y_j) + (v_Y)_j(x_i) \leq n \cdot \max_{(x_i,y_j)\in M} (v_X)_i(y_j) + (v_Y)_j(x_i)$$

Define $S, \bar{S}$ and $T, \bar{T}$ similarly as we did in Theorem 4.4 for X, Y, respectively. Since we require the stability property, it might be the case that the maximum welfare-producing $(x_i, y_j) \in M$ is such that both $x_i \in \bar{S}$ and $y_j \in \bar{T}$. In this case, $SW(\tilde{M} \mid \tilde{v_X}, \tilde{v_Y}) = 0$. Therefore, we cannot place an upper bound on $\frac{SW(M|v_X,v_Y)}{SW(\tilde{M}|\tilde{v_X},\tilde{v_Y})}$. It is unknown whether there is a good way to bound $\frac{SW(M|v_X,v_Y)}{SW(\tilde{M}|v_X,v_Y)}$, which is needed for bounding distortion.

Theorem 4.12. *Double λ-TSF makes $O(\lambda \log n)$ queries per agent.*

Even though we do not have a formal proof to establish distortion bounds for Double λ-TSF, we finish with an example where Double λ-TSF works well.

Example 4.13. *When we apply Double λ-TSF with $\lambda = 1$ on the instance of the stable matching problem from Example 3.12, we obtain the following:*

$$\tilde{v}_X = \begin{bmatrix} 0 & 0 & 3 & 5 & 0 & 0 \\ 1 & 1 & 2 & 1 & 1 & 1 \\ 0 & 3 & 0 & 0 & 2 & 2 \\ 0 & 3 & 2 & 0 & 0 & 2 \\ 3 & 0 & 0 & 0 & 3 & 0 \\ 4 & 0 & 0 & 2 & 0 & 2 \end{bmatrix}, \quad \tilde{v}_Y = \begin{bmatrix} 0 & 5 & 5 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 3 & 7 \\ 0 & 0 & 0 & 7 & 0 & 1 \\ 9 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Since $\lambda = 1$, Double λ-TSF creates two acceptable sets $Q_{i,0}$ and $Q_{i,1}$ for each agent i. Each $Q_{i,0}$ contains just the agent's favorite choice, which has utility v_i^*. Each $Q_{i,1}$ contains choices whose utility is above $\frac{v_i^*}{\alpha_1} = \frac{v_i^*}{\sqrt{6}}$, and choices in $Q_{i,1}$ share the simulated cardinal value of the least preferred choice. For example, x_2 places a utility of 2 for multiple choices, but all except one choice is downgraded to $Q_{2,1}$, the elements in which are given a simulated cardinal value of 1. Even with $\lambda = 1$, the structure of the cardinal profiles is mostly preserved.

Using $\tilde{v}_X, \tilde{v}_Y$ as inputs to Irving, along with σ_X, σ_Y, will output the same stable matching referred to in Example 3.22, and the social welfare will be 37. With the simulated cardinal values, the graph P' will remain unchanged, but its weights will change. Under $\tilde{v}_X, \tilde{v}_Y$,

$$w(\rho_1) = 1, \quad w(\rho_2) = 0, \quad w(\rho_3) = -7, \quad w(\rho_4) = 0, \quad w(\rho_5) = 21$$

and it is also clear here that the maximum weight closed subset is $\{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$.

Chapter 5

Data Generation for Empirical Evaluation of Distortion

In this small chapter, we discuss the generation of cardinal profiles from ordinal profiles to maximize distortion. We consider the voting and one-sided matching settings. It is inherently difficult to generate distortion-optimal cardinal profiles for stable matching, as we must account for the stability structure. We also limit our focus to unit-sum cardinal profiles, which is a commonly used type of cardinal profile that treats the preferences of each agent equally by requiring the total utility of each agent to be 1. An ordinal profile σ has many consistent unit-sum cardinal profiles, which form a set denoted as $C(\sigma)$. As remarked in Section 3.3, randomly picking a consistent cardinal profile in $C(\sigma)$ will not result in a high distortion. We use the foundations provided by Ebadian et al. [2024] to solve this problem, although the original paper aims to achieve a different goal. We examine the use of a type of valuation function called the dichotomous utility function to achieve this.

Definition 5.1. *Let σ be an ordinal profile with n agents and m choices. For each agent i, define a ranking function $\sigma_i : [1,m] \to A$, where if a_j is the kth most preferred choice, $\sigma_i(k) = a_j$. For any $r \in [1,m]$, the **dichotomous utility function** with respect to σ_i is defined as*

$$\mathbb{1}_{\sigma_i,r}(a_j) = \begin{cases} \frac{1}{r}, & \sigma_i^{-1}(a_j) \leq r \\ 0, & \textit{otherwise} \end{cases}$$

The dichotomous utility function is a valuation function and creates a cardinal profile where each agent prefers some top r choices equally and has zero utility for the bottom $m-r$ choices. We call such a profile a *dichotomous utility profile*. Next, recall that randomized voting rules output a discrete probability distribution $p \in \Delta(A)$. As we assumed with randomized voting rules, we can compute the expected social welfare and expected distortion of p with respect to some cardinal profile v. In this chapter, we make explicit the cardinal profile used to calculate distortion, i.e. $dist(p,v)$.

For the voting setting, Ebadian et al. [2024] showed in the following theorem that among the distortion-maximizing unit-sum cardinal profiles consistent with an ordinal profile, there is one that is dichotomous.

Theorem 5.2. *[Ebadian et al., 2024] For any ordinal profile σ and distribution $p \in \Delta(A)$, there exists a dichotomous utility profile $u^* \in C(\sigma)$ such that $dist(p,u^*)$ is the highest among all unit-sum cardinal profiles consistent with σ.*

Proof. Let $d = \max_{u \in C(\sigma)} dist(p,u)$. Then, this equation follows.

$$\max_{u \in C(\sigma)} \frac{\max_{a^* \in A} SW(a^* \mid u)}{SW(p \mid u)} = d$$

$$\max_{u \in C(\sigma)} \left(\max_{a^* \in A} SW(a^* \mid u) - d \cdot SW(p \mid u) \right) = 0$$

$$\max_{a^* \in A} \max_{u \in C(\sigma)} \sum_{i \in N} (u_i(a^*) - d \cdot u_i(p)) = 0$$

Since the summation is linear over agents, we can move the maximum inside the summation to construct the optimal cardinal profile. As a small syntactic detail, we denote the set of unit-sum valuation functions consistent with some ranking function σ_i as $C(\sigma_i)$.

$$\max_{a^* \in A} \sum_{i \in N} \max_{u_i \in C(\sigma_i)} (u_i(a^*) - d \cdot u_i(p)) = 0 \tag{5.1}$$

Now, we show that for some alternative a^* and agent i, $u_i(a^*) - d \cdot u_i(p)$ is maximized at some dichotomous utility function. Let $\Delta^m = \{\alpha \in [0,1]^m \mid \sum_{r \in [1,m]} \alpha_r = 1\}$ be the $m-1$ simplex. Then,

$$\max_{u_i \in C(\sigma_i)} u_i(a^*) - d \cdot u_i(p) = \max_{\alpha \in \Delta^m} \left(\sum_{r \in [1,m]} \alpha_r \cdot \mathbb{1}_{\sigma_i,r}(a^*) - d \cdot \sum_{r \in [1,m]} \alpha_r \cdot \mathbb{1}_{\sigma_i,r}(p) \right) \tag{5.2}$$

$$= \max_{\alpha \in \Delta^m} \sum_{r \in [1,m]} \alpha_r \cdot (\mathbb{1}_{\sigma_i,r}(a^*) - d \cdot \mathbb{1}_{\sigma_i,r}(p)) \tag{5.3}$$

To show that the first equality holds, Ebadian et al. [2024] claims that $C(\sigma)$ is an m-dimensional subspace spanned by the dichotomous utility functions $(\mathbb{1}_{\sigma_i,r})_{r \in [1,m]}$. Note that the linear span of $(\mathbb{1}_{\sigma_i,r})_{r \in [1,m]}$ is not equal to $C(\sigma_i)$. As a simple counterexample, consider the linear combination $2 \cdot \mathbb{1}_{\sigma_i,1}$. This is not a unit-sum valuation function.

We clarify that $C(\sigma_i)$ is an m-dimensional vector subspace which can be represented as a linear combination of the dichotomous utility functions $(\mathbb{1}_{\sigma_i,r})_{r \in [1,m]}$ where every coefficient is between 0 and 1 and the coefficients sum to 1.

We now show that our claim is true. The elements in $(\mathbb{1}_{\sigma_i,r})_{r \in [1,m]}$ are clearly linearly independent. Now, given an arbitrary unit-sum valuation function v_i, we construct a linear combination of $(\mathbb{1}_{\sigma_i,r})_{r \in [1,m]}$. This can be done canonically as follows. Let $t_1, \ldots, t_m$ be the coefficients of the linear combination, corresponding to $\mathbb{1}_{\sigma_i,1}, \ldots, \mathbb{1}_{\sigma_i,m}$. We start with the coefficient t_m. Since $\mathbb{1}_{\sigma_i,m}$ is the only element in $(\mathbb{1}_{\sigma_i,r})_{r \in [1,m]}$ that assigns a nonzero value for $\sigma_i(m)$, we must set

$$t_m = m \cdot v_i(\sigma_i(m))$$

In any linear combination where the above holds, the utility of $\sigma_i(m)$ correctly becomes $v_i(\sigma_i(m))$ because

$$\sum_{r=1}^{m} t_r \cdot \mathbb{1}_{\sigma_i,r}(\sigma_i(m)) = t_m \cdot \mathbb{1}_{\sigma_i,m}(\sigma_i(m)) = m \cdot v_i(\sigma_i(m)) \cdot \frac{1}{m} = v_i(\sigma_i(m))$$

We use this fact as the base case of a mathematical induction. For the inductive hypothesis, assume that we have set $t_{k+1},\dots,t_m$, such that any linear combination of $(\mathbb{1}_{\sigma_i,r})_{r\in[1,m]}$ using $t_{k+1},\dots,t_m$ recreates the utility of $\sigma_i(k+1),\dots,\sigma_i(m)$ as $v_i(\sigma_i(k+1)),\dots,v_i(\sigma_i(m))$. Then, we want to show that we can set t_k such that any linear combination of $(\mathbb{1}_{\sigma_i,r})_{r\in[1,m]}$ using $t_k,\dots,t_m$ recreates the utility of $\sigma_i(k),\dots,\sigma_i(m)$ correctly. To do this, we set

$$t_k = k\cdot(v_i(\sigma_i(k)-v_i(\sigma_i(k+1)))$$

From the assumption, we know that

$$\sum_{r=k+1}^{m} t_r\cdot\mathbb{1}_{\sigma_i,r}(\sigma_i(k)) = \sum_{r=k+1}^{m} t_r\cdot\mathbb{1}_{\sigma_i,r}(\sigma_i(k+1)) = v_i(\sigma_i(k+1))$$

Since $\mathbb{1}_{\sigma_i,k},\dots,\mathbb{1}_{\sigma_i,m}$ are the only elements in $(\mathbb{1}_{\sigma_i,r})_{r\in[1,m]}$ that assign a nonzero value to $\sigma_i(k)$,

$$\begin{aligned}\sum_{r=k}^{m} t_r\cdot\mathbb{1}_{\sigma_i,r}(\sigma_i(k)) &= t_k\cdot\mathbb{1}_{\sigma_i,k}(\sigma_i(k)) + \sum_{r=k+1}^{m} t_r\cdot\mathbb{1}_{\sigma_i,r}(\sigma_i(k))\\ &= k\cdot(v_i(\sigma_i(k)-v_i(\sigma_i(k+1)))\cdot\frac{1}{k} + v_i(\sigma_i(k+1)) = v_i(\sigma_i(k))\end{aligned}$$

Since the value of t_k does not affect the recreation of $\sigma_i(k+1),\dots,\sigma_i(m)$, we have proved the inductive hypothesis. Each t_k is in $[0,1]$ because

$$\frac{1}{k}\geq v_i(\sigma_i(k))\geq v_i(\sigma_i(k+1))\geq 0$$

As such, we can inductively represent any unit-sum valuation function as a linear combination of $(\mathbb{1}_{\sigma_i,r})_{r\in[1,m]}$ with coefficients in $[0,1]$. The final thing to show is that the coefficients sum to 1, which holds due to the unit-sum assumption.

$$\begin{aligned}&1\cdot(v_i(\sigma_i(1))-v_i(\sigma_i(2)))+2\cdot(v_i(\sigma_i(2))-v_i(\sigma_i(3)))+\dots\\ &+(m-1)\cdot(v_i(\sigma_i(m-1))-v_i(\sigma_i(m)))+m\cdot v_i(\sigma_i(m)) = \sum_{r\in[1,m]} v_i(\sigma_i(r)) = 1\end{aligned}$$

Note that in Equation 5.3, there is some $r\in[1,m]$ for which the expression $\alpha_r\cdot(\mathbb{1}_{\sigma_i,r}(a^*)-d\cdot\mathbb{1}_{\sigma_i,r}(p))$ is maximized. Since the last expression is linear in α, the maximum must be achieved at an α^* such that $\alpha^*_r=1$ for this r and 0 for all others. □

We now propose two types of distortion-maximizing cardinal profiles. Both types are generated from a fixed ordinal profile σ. The first is defined as follows.

Definition 5.3. *Let p be the discrete probability distribution that achieves the lowest distortion $d=\max_{u\in C(\sigma)} dist(p,u)$. Then, the **pseudo-distortion-maximizing cardinal profile** is a cardinal profile that achieves maximum distortion with respect to p,d among the unit-sum cardinal profiles consistent with σ.*

This profile does not maximize distortion against every voting rule, but only against the optimal randomized voting rule p. However, we conjecture that it will achieve a relatively high distortion for most other voting rules. We show later that there is no cardinal profile that will achieve a sufficiently high distortion for every voting rule. Whether the pseudo-distortion-maximizing cardinal profile achieves high distortion for most voting rules is an open problem. Such a profile will be useful for users who want to compare the performance of multiple voting rules on a single hard cardinal profile.

To compute the pseudo-distortion-maximizing cardinal profile, we need to find p and d. Ebadian et al. [2024] uses Theorem 5.2 to construct a linear program that gives these in polynomial time. We attach the linear program in Appendix B.4. Given p and d, we know that there is an underlying distortion-maximizing dichotomous cardinal profile. We do not know this cardinal profile yet, but we need to know the alternative a^* that maximizes social welfare under this distortion-maximizing cardinal profile. We can use a brute-force method, looping over each $a_j \in A$ and compute the optimal $r \in [1,m]$ for each agent $i \in N$. The summation in Equation 5.1 will evaluate to 0 for the optimal alternative a^*. When calculating a^*, we also find r for each agent i.

We name the second type of distortion-maximizing cardinal profile a *rule-specific distortion-maximizing cardinal profile*. As can be understood from the name, this unit-sum cardinal profile attains the worst distortion given a probability distribution p representing some voting rule f, in addition to an ordinal profile σ. In this way, the user can generate the worst-case distortion for a specific algorithm they may want to test. The following theorem from Ebadian et al. [2024] shows that this can be done.

Theorem 5.4. *There is an $O(nm\log nm)$ algorithm to compute $\max_{u\in C(\sigma)} dist(p,u)$.*

We will not show the proof of this theorem, but Ebadian et al. [2024] provides pseudocode to achieve this with dichotomous utility profiles in Appendix B of their paper.

Unfortunately, there is no single cardinal profile that attains a high distortion on all voting rules. Suppose for a contradiction that there is a cardinal profile $\tilde{u}^*$, such that

$$dist(p,\tilde{u}^*) \geq \alpha \cdot dist(p,u^*) \quad \forall p \in \Delta(A)$$

where u^* is the rule-specific distortion-maximizing cardinal profile for the given p and α is any constant factor with value in the range $\frac{1}{dist(p,u^*)} < \alpha \leq 1$. We require the first inequality on α because we most conservatively interpret "high distortion" as a distortion larger than 1. If $\alpha = \frac{1}{dist(p,u^*)}$, then $dist(p,\tilde{u}^*) = 1$ and this becomes the lowest distortion possible. To contradict this, we let p select an alternative that maximizes the social welfare under $\tilde{u}^*$ with probability 1. Then, $dist(p,\tilde{u}^*)$ will be 1, and the condition on α will be violated. Hence, there is no cardinal profile which maximizes distortion for all voting rules up to some constant factor.

We now apply the idea of Theorem 5.2 to one-sided matching. Let S_n be the set of all (n,n) permutation matrices. A permutation matrix corresponds to a one-sided matching. Recall that a bistochastic matrix can be seen as a probability distribution over permutation matrices. Hence, it corresponds to a probability distribution over one-sided matchings, and we can calculate the expected social welfare and distortion on B.

Theorem 5.5. *Let σ be any ordinal profile with n agents and n items. Let B be any (n,n) bistochastic matrix B. Then, there exists a dichotomous cardinal profile $u^* \in C(\sigma)$ such that $dist(B,u^*)$ is the highest among all unit-sum cardinal profiles consistent with σ.*

Proof. We follow similar steps as done in the proof to Theorem 5.2. By definition,

$$\begin{aligned}
\max_{u \in C(\sigma)} \frac{\max_{M^* \in S_n} SW(M^* \mid u)}{SW(B \mid u)} &= d \\
\max_{u \in C(\sigma)} \left(\max_{M^* \in S_n} SW(M^* \mid u) - d \cdot SW(B \mid u) \right) &= 0 \\
\max_{M^* \in S_n} \max_{u \in C(\sigma)} \sum_{i \in N} (u_i(M_i^*) - d \cdot u_i(B_i)) &= 0 \\
\max_{M^* \in S_n} \sum_{i \in N} \max_{u_i \in C(\sigma_i)} (u_i(M_i^*) - d \cdot u_i(B_i)) &= 0
\end{aligned}$$

where M_i^* is the ith row of M^* and B_i is the ith row of B and $u_i(M_i^*), u_i(B_i)$ are expected utilities for agent i of M^*, B respectively. Note that the rows M_i^*, B_i are probability distributions, as discussed in Section 3.6. The last equation corresponds to Equation 5.1 in the proof of Theorem 5.2.

We now show that for a fixed matching M^* and agent i, $u_i(M_i^*) - d \cdot u_i(B_i)$ is maximized at some dichotomous utility function $u_i^* \in C(\sigma)$. We claim that $C(\sigma_i)$ is an n-dimensional vector subspace spanned by a linear combination of $(\mathbb{1}_{\sigma_i,r})_{r \in [1,n]}$ such that the coefficients are in $[0,1]$ and sum to 1. The proof of this claim is identical to the proof given in Theorem 5.2. Hence,

$$\begin{aligned}
\max_{u_i \in C(\sigma_i)} (u_i(M_i^*) - d \cdot u_i(B_i)) &= \max_{\alpha \in \Delta^n} \left(\sum_{r \in [1,n]} \alpha_r \cdot \mathbb{1}_{\sigma_i,r}(M_i^*) - d \cdot \sum_{r \in [1,n]} \alpha_r \cdot \mathbb{1}_{\sigma_i,r}(B_i) \right) \\
&= \max_{\alpha \in \Delta^n} \sum_{r \in [1,n]} \alpha_r \cdot (\mathbb{1}_{\sigma_i,r}(M_i^*) - d \cdot \mathbb{1}_{\sigma_i,r}(B_i))
\end{aligned}$$

where Δ^n is the $n-1$ simplex $\Delta^n = \{\alpha \in [0,1]^n \mid \sum_{r \in [1,n]} \alpha_r = 1\}$. Since the last equation is linear in α, we know that it is maximized when $\alpha_r = 1$ for some r and $\alpha_s = 0$ for all $s \neq r$. □

However, generating distortion-maximizing dichotomous cardinal profiles is not easy for one-sided matching. The linear programming approach for voting proposed by Ebadian et al. [2024] does not run in polynomial time in the one-sided matching because the number of matchings in S_n grows factorially with the number of agents and items. It is an open problem whether a different representation exists that allows us to create a distortion-maximizing profile or whether this can be shown to be impossible.

Chapter 6

Conclusion

Our work is a first attempt to create a comprehensive set of implementations for distortion-oriented social choice algorithms. In particular, we explored the application of elicitation-based algorithms and distortion-maximizing cardinal profile generation in the one-sided matching and stable matching settings. In some places, our results highlight the difficulty of computing distortion-efficient outcomes in new settings with structural differences. We have also shown that this is doable.

6.1 Limitations and Directions for Future Work

As briefly discussed at the end of Section 3.7.2, we currently do not have a way of efficiently computing an optimal stable matching from non-integral cardinal profiles. This is because the most efficient flow algorithm for our problem is Ford-Fulkerson, which runs in $O(n^4)$ time but requires capacities to be integers. As Irving et al. [1987] stated, whether there is a flow algorithm more practical than Ford-Fulkerson for computing weighted closed subsets is an open problem. Note that the maximum weight closed subset problem is a special case of the max-flow problem because all edges in the constructed network have infinite capacities except the edges from source s and the edges to sink t. Another area of difficulty was finding efficient flow implementations available in Python that worked with non-integral values. Even though there are max-flow algorithms with better asymptotic complexities, we suspect that they are too involved to outperform Ford-Fulkerson in practice, and the implementations do not exist. Even with the integral utility requirement, the unit-sum restriction can still be enforced by asking users to distribute a fixed integral number of points to each choice. However, this becomes more difficult for agents to do in elicitation, where agents are queried only about a small subset of the choices. Note that the simulated cardinal profile is not unit-sum, regardless of whether the true cardinal profile is integral.

Results in Chapter 5 have highlighted difficulties with generating distortion-maximizing cardinal profiles for empirical experiments. Namely, we showed that there is no cardinal profile that will attain a high distortion on all algorithms. The one-sided matching and stable matching problems inherently require us to search from a larger space of possible outcomes. We have shown that there is a dichotomous distortion-maximizing cardinal

profile for one-sided matching. We also found that the linear programming approach does not run in polynomial time for one-sided matching, and it is an open question whether there is a more efficient approach to computing one. It is also an open question whether we can quantify how ”good” the pseudo-distortion-maximizing cardinal profile is on some portion of the voting rules.

In Section 4.2, while we created an extension of λ-TSF to the stable matching and showed that the stability property holds, we were not able to show that the same distortion bound held. Moreover, it remains an open question whether there is a good way to prove distortion bounds on stable matchings in general, not just Double λ-TSF. Computing distortion for stable matchings is especially difficult because we must preserve the stability property when we are considering an optimal matching.

We remark that in practical settings with a small number of alternatives, the binary search-based elicitation mechanisms do not necessarily achieve a significant reduction in the agent's burden of providing cardinal information. This is because the number of queries quickly grows with λ. For example, in Example 4.13 with 6 choices, Double λ-TSF queries each agent two to four times with $\lambda = 2$. However, when we change λ from 2 to 3, the algorithm must now query each agent three to seven times. While socialchoicekit provides memoization to reduce the number of queries as much as possible, we believe that two query-based approaches are more practical for settings with a small number of choices. It remains to be answered whether there is a two query-based approach for the stable matching setting, i.e. whether there is a sufficiently representative data structure for stable matching.

In most of this work, we have assumed that we select one choice. In voting, we assumed that we only select one winning alternative, but there is a broad selection of literature Caragiannis et al. [2017] studying committee selection problems where the goal is to select multiple alternatives. Similarly, we assumed in one-sided matching that we only select one item for each agent and that the number of agents is equal to the number of items. In stable matching, we have largely considered instances where the number of agents in X and Y were equal, and each agent was matched with exactly one agent. While extensions on these constraints are likely to increase the complexity of the problem, it is interesting how strategies to improve distortion can be effectively applied to more general settings.

There are also limitations in the socialchoicekit library. Due to the breadth of the field of computational social choice and research surrounding distortion, the algorithms we implemented are only from a small part of the existing literature. The library can be extended to other settings we did not consider. In Chapter 3, we have also made assumptions about the size and type of the data. In particular, many of our algorithms could be extended to support incomplete profiles and profiles with ties. Adding support for this would allow users to work with more profiles since many ordinal profiles from PrefLib are not strict and complete. Finally, our work did not run case studies with potential users of socialchoicekit. A direction for follow-up work is to receive feedback from researchers and software engineers about the practicality of our implementation.

Bibliography

Matthew D. Adler. *Well-Being and Fair Distribution: Beyond Cost-Benefit Analysis*. Oxford University Press, Oxford, 2011.

Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. Prentice Hall Inc., Englewood Cliffs, NJ, 1993.

Georgios Amanatidis, Georgios Birmpas, Aris Filos-Ratsikas, and Alexandros A. Voudouris. Peeking behind the ordinal curtain: Improving distortion via cardinal queries. *Artificial Intelligence*, 296:103488, 2021.

Georgios Amanatidis, Georgios Birmpas, Aris Filos-Ratsikas, and Alexandros A. Voudouris. A few queries go a long way: Information-distortion tradeoffs in matching. *Journal of Artificial Intelligence Research*, 74:226–261, 2022.

Georgios Amanatidis, Georgios Birmpas, Aris Filos-Ratsikas, and Alexandros A Voudouris. Don't roll the dice, ask twice: The two-query distortion of matching problems and beyond. *SIAM Journal on Discrete Mathematics*, 38(1):1007–1029, 2024.

Elliot Anshelevich and John Postl. Randomized social choice functions under metric preferences. *Journal of Artificial Intelligence Research*, 58:797–827, 2017.

Elliot Anshelevich, Onkar Bhardwaj, Edith Elkind, John Postl, and Piotr Skowron. Approximating optimal social choice under metric preferences. *Artificial Intelligence*, 264:27–51, 2018.

Elliot Anshelevich, Aris Filos-Ratsikas, Nisarg Shah, and Alexandros A. Voudouris. Distortion in social choice problems: The first 15 years and beyond. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)*, pages 4294–4301, 2021.

Kenneth Arrow. *Social Choice and Individual Values*. Wiley, New York, 1951. Reprinted 1963.

Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán. Revista A.*, 5:147–151, 1946. [Three observations on linear algebra].

Craig Boutilier, Ioannis Caragiannis, Simi Haber, Tyler Lu, Ariel D. Procaccia, and Or Sheffet. Optimal social choice functions: A utilitarian view. *Artificial Intelligence*, 227:190–213, 2015.

Sylvain Bouveret and Marie Jeanne Natete. Whale4 (which alternative is elected), 2015. URL `https://whale.imag.fr/`.

Felix Brandt, Guillame Chabin, and Christian Geist. Pnyx: a powerful and user-friendly tool for preference aggregation. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, 2015.

Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors. *Handbook of Computational Social Choice*. Cambridge University Press, 2016.

Donald E Campbell and Jerry S Kelly. A strategy-proofness characterization of majority rule. *Economic Theory*, 22(3):557–568, 2003.

Ioannis Caragiannis and Ariel D. Procaccia. Voting almost maximizes social welfare despite limited communication. *Artificial Intelligence*, 175(9-10):1655–1671, 2011.

Ioannis Caragiannis, Swaprava Nath, Ariel D Procaccia, and Nisarg Shah. Subset selection via implicit utilitarian voting. *Journal of Artificial Intelligence Research*, 58:123–152, 2017.

Moses Charikar and Prasanna Ramakrishnan. Metric distortion bounds for randomized social choice. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2986–3004. SIAM, 2022.

Moses Charikar, Prasanna Ramakrishnan, Kangning Wang, and Hongxun Wu. Breaking the metric voting distortion barrier. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1621–1640. SIAM, 2024.

Soroush Ebadian, Anson Kahng, Dominik Peters, and Nisarg Shah. Optimized distortion and proportional fairness in voting. In *Proceedings of the 23rd ACM Conference on Economics and Computation (EC '22)*, page 38 pages, Boulder, CO, USA, 2022. ACM. July 11-15.

Soroush Ebadian, Aris Filos-Ratsikas, Mohamad Latifian, and Nisarg Shah. Computational aspects of distortion. In *The 23rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1–14, 2024.

Aris Filos-Ratsikas, Søren Kristoffer Stiil Frederiksen, and Jie Zhang. Social welfare in one-sided matchings: Random priority and beyond. In *International Symposium on Algorithmic Game Theory*, pages 1–12. Springer, 2014.

David Gale and Lloyd Stowell Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

Vasilis Gkatzelis, Daniel Halpern, and Nisarg Shah. Resolving the optimal metric distortion conjecture. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1427–1438, 2020.

Dan Gusfield. Three fast algorithms for four problems in stable marriage. *SIAM Journal on Computing*, 16:111–128, 1987.

Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA, 1989.

Thomas Hare. *Treatise on the Election of Representatives, Parliamentary and Municipal*. Longman, Green, Reader, and Dyer, London, 1859.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

Aanund Hylland and Richard Zeckhauser. The efficient allocation of individuals to positions. *Journal of Political Economy*, 87(2):293–314, 1979.

Robert. W. Irving and Paul Leather. The complexity of counting stable marriages. *SIAM Journal on Computing*, 15:655–667, 1986.

Robert W. Irving, Paul Leather, and Dan Gusfield. An efficient algorithm for the "optimal" stable marriage. *Journal of the Association for Computing Machinery,*, 34 (3):532–543, 1987.

Zhihao Jiang, Kamesh Munagala, and Kangning Wang. Approximately stable committee selection. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 463–472, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450369794. doi: 10.1145/3357713.3384238. URL `https://doi.org/10.1145/3357713.3384238`.

Bettina Klaus, David F Manlove, and Francesca Rossi. Matching under preferences. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors, *Handbook of Computational Social Choice*, pages 333–355. Cambridge University Press, Cambridge, 2016.

Donald E. Knuth. *Mariages Stables*. Les Presses de l'Université de Montréal, Montreal, Quebec, Canada, 1976.

Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. pytest x.y, 2004. URL `https://github.com/pytest-dev/pytest`.

Christian List. Social Choice Theory. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2022 edition, 2022.

Nicholas Mattei and Toby Walsh. Preflib: A library of preference data HTTP://PREFLIB.ORG. In *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT 2013)*, Lecture Notes in Artificial Intelligence. Springer, 2013.

Kamesh Munagala and Kangning Wang. Improved metric distortion for deterministic social choice rules. In *Proceedings of the 2019 ACM Conference on Economics and Computation (EC)*, pages 245–262, 2019.

John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1953.

Natsuhiko Ozawa. Getting started - socialchoicekit documentation, 2024. URL `https://socialchoicekit.natsuozawa.com/getting_started`.

Laurent Perron and Vincent Furnon. Or-tools, 2024. URL `https://developers.google.com/optimization/`.

Ariel D. Procaccia and Jeffrey S. Rosenschein. The distortion of cardinal preferences in voting. In *International Workshop on Cooperative Information Agents (CIA)*, pages 317–331, 2006.

Alvin Eliot Roth and Marilda Sotomayor. Two-sided matching. In R. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*, volume 1, pages 485–541. North Holland, Amsterdam, 1992.

Amartya Sen. Social choice theory. In K.J. Arrow and M.D. Intriligator, editors, *Handbook of Mathematical Economics*, volume III, pages 1073–1181. Elsevier Science Publishers B.V. (North-Holland), 1986.

Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26:362–391, 1983.

Christopher Small, Michael Bjorkegren, Timo Erkkilä, Lynette Shaw, and Colin Megill. Polis: Scaling deliberation by mapping high dimensional opinion spaces. *RECERCA. Revista De Pensament I Anàlisi*, 26(2), 2021. URL `https://doi.org/10.6035/recerca.5516`.

Dariusz Stolicki, Stanisław Szufa, and Nimrod Talmon. Pabulib: A participatory budgeting library, 2020.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

William S. Zwicker. Introduction to the theory of voting. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors, *Handbook of Computational Social Choice*, pages 23–74. Cambridge University Press, Cambridge, 2016.

Appendix A

Example Usage of socialchoicekit Library

This example from Ozawa [2024] demonstrates how the user can run social choice functions from socialchoicekit using out-of-the-box integration with PrefLib. This particular example demonstrates how we can test the empirical distortion of various voting algorithms on a course selection dataset from PrefLib. The usage and implementation of this library are described in Chapter 3.

```
import numpy as np

from preflibtools.instances import OrdinalInstance

from socialchoicekit.preflib_utils
    import preflib_soc_to_profile
from socialchoicekit.data_generation
    import UniformValuationProfileGenerator
from socialchoicekit.deterministic_scoring
    import Plurality, SocialWelfare
from socialchoicekit.elicitation_voting
    import KARV
from socialchoicekit.elicitation_utils
    import ValuationProfileElicitor, SynchronousStdInElicitor
from socialchoicekit.distortion import distortion

url = 'https://www.preflib.org/static/data/agh/00009-00000001.soc'

# 1.1) Import data
instance = OrdinalInstance()
instance.parse_url(url)
profile = preflib_soc_to_profile(instance)
print(profile)
```

```
# 1.2) Generate (hypothetical) cardinal profile
valuation_profile = UniformValuationProfileGenerator(
    high=1, low=0, seed=1
).generate(profile)
print(valuation_profile)

# 1.3) Compute optimal utility
# using cardinal information
social_welfare = SocialWelfare().score(valuation_profile)
print(social_welfare)
optimal_alternative = int(np.argmax(social_welfare))
print("Optimal alternative: ", optimal_alternative + 1)
optimal_welfare = np.amax(social_welfare)
print("Optimal welfare: ", optimal_welfare)

# 2) Test baseline: Plurality (pick favorite)
plurality = Plurality()
plurality_winner = plurality.scf(profile)
print(plurality.score(profile))
print("Plurality winner: ", plurality_winner)
print(
    "Distortion: ",
    distortion(plurality_winner, valuation_profile)
)

# 3) Elicitation (query)-based voting
karv = KARV(k=3)
valuation_profile_elicitor = ValuationProfileElicitor(
    valuation_profile=valuation_profile, memoize=True
)
karv_winner = karv.scf(profile, valuation_profile_elicitor)
print("KARV winner: ", karv_winner)
print(
    "Distortion: ",
    distortion(karv_winner, valuation_profile)
)
```

Appendix B

Omitted Pseudocode and Programs

B.1 Pseudocode for k-Acceptable Range Voting

This is the pseudocode for k-Acceptable Range Voting presented by Amanatidis et al. [2021]. This mechanism is described in Section 4.1.1. Note that the notation used is not modified from the original paper.

Algorithm 6 k-ACCEPTABLE-RANGE-VOTING(k, σ)

for $i \in N$ **do**
  $v_i^* \leftarrow v_i(\sigma_i(1))$, where $\sigma_i(p)$ is the alternative that agent i ranks at position p.
  $\tilde{v}_i(\sigma_i(1)) \leftarrow v_i^*$
  $S_{i,0} \leftarrow \{\sigma_i(1)\}$
  for $\ell \in \{1, 2, \ldots, k\}$ **do**
    $\lambda_\ell \leftarrow m^{\frac{\ell}{k+1}}$
    $p^* \leftarrow \text{BSEARCH}(1, m, \lambda_\ell, v_i^*)$
    $S_{i,\ell} \leftarrow \{a_j \in A : a_j \succ_i \sigma_i(p^*)\}$ ▷ define the λ_ℓ-acceptable set of agent i
    for $a_j \in S_{i,\ell} \setminus S_{i,\ell-1}$ **do** ▷ $S_{i,\ell}$ assumed to contain all $S_{i,\ell-1}, \ldots, S_{i,0}$
      $\tilde{v}_i(a_j) \leftarrow \frac{v_i^*}{\lambda_\ell}$ ▷ define the simulated cardinal profile
    end for
  end for
  for $a_j \in A \setminus S_{i,k}$ **do**
    $\tilde{v}_i(a_j) \leftarrow 0$
  end for
end for
for $a_j \in A$ **do**
  $SW_s(a_j|\tilde{v}) \leftarrow 0$ ▷ compute the simulated welfare of alternative j
  for $a_j \in A$ **do**
    $SW_s(a_j|\tilde{v}) \leftarrow SW_s(a_j|\tilde{v}) + \tilde{v}_i(a_j)$
  end for
end for
return $\arg\max_{a_j \in A} SW_s(a_j|\tilde{v})$

Algorithm 7 BSEARCH$(\alpha, \beta, \lambda, v)$

if $\alpha = \beta$ **then**
  return α
end if
Let $u \leftarrow v_i\left(\sigma_i\left(\frac{\alpha+\beta}{2}\right)\right)$ ▷ v_i is the valuation function for agent i, whose value is obtained through querying
if $u > \frac{v}{\lambda}$ **then**
  return BSEARCH$(\frac{\alpha+\beta}{2}, \beta, \lambda, v)$
else
  return BSEARCH$(\alpha, \frac{\alpha+\beta}{2}, \lambda, v)$
end if

B.2 Pseudocode for λ-Threshold Step Function

This is the pseudocode for the λ-threshold step function presented by Amanatidis et al. [2022]. This mechanism is described in Section 4.1.1. The mechanism is slightly modified to be consistent with our model.

Algorithm 8 λ-THRESHOLD-STEP-FUNCTION(λ, σ)

Let $\alpha_\ell = n^{\ell/(\lambda+1)}$ for $\ell \in \{0, \ldots, \lambda\}$.
for every agent $i \in N$ **do**
  Query i for her top-ranked item a_i^*; let v_i^* be this value.
  Let $Q_{i,0} = \{a_i^*\}$ and $\tilde{v}_i(a_i^*) = \frac{v_i^*}{\alpha_0} = v_i^*$.
  for $\ell \in \{1, \ldots, \lambda\}$, using binary search **do**
    Compute $Q_{i,\ell} = \{a_j \in A : \tilde{v}_i(a_j) \in [\frac{v_i^*}{\alpha_\ell}, \frac{v_i^*}{\alpha_{\ell-1}})\}$
    and let $\tilde{v}_i(a_j) = \frac{v_i^*}{\alpha_\ell}$ for every $j \in Q_{i,\ell}$.
  end for
  Let $Q_i = \bigcup_{\ell=0}^{\lambda} Q_{i,\ell}$ and set $\tilde{v}_i(a_j) = 0$ for $a_j \in A \setminus Q_i$.
end for
return a matching $M^* \in \arg\max_{M \in \mathbf{M}} SW(M|\tilde{v})$.

B.3 Pseudocode for Match-TwoQueries

Algorithm 9 MATCH-TWO-QUERIES(σ)

Query each $i \in N$ about her favorite item w.r.t. $\succeq_i$
Compute a *sufficiently representative* assignment M_0
Query each agent about the item she is assigned to in M_0
Set all non-revealed values to 0
return a maximum-weight perfect matching M

This is the pseudocode for the Match-TwoQueries mechanism presented by Amanatidis et al. [2024]. This mechanism is described in Section 4.1.2. The syntax is modified to be consistent with the notations used in the rest of the dissertation.

Algorithm 10 $\sqrt{n}$-SERIAL-DICTATORSHIP(σ)

Let B be a multiset containing $\sqrt{n}$ copies of each $a_j \in A$
for every agent $i \in N$ **do**
    Let α_i be the most preferred item of agent i in B
    Remove α_i from B
end for
return $(\alpha_i)_{i \in N}$

B.4 Linear Program for Computing the Optimal Distortion and Probability Distribution

This is the linear program by Ebadian et al. [2024] to determine the optimal distortion d and probability distribution p, given an ordinal profile σ for the voting setting. This linear program is referenced in Chapter 5. We now briefly show the derivation of the linear program and explain the terms used.

From Theorem 5.2, we begin by replacing the maximum over $u_i \in C(\sigma_i)$ with a maximum over the dichotomous utility functions.

$$dist(p,\sigma) = d \iff \max_{a^* \in A} \sum_{i \in N} \max_{r \in [1,m]} \frac{1}{r} \left(\mathbb{I}\left[\sigma_i^{-1}(a^*) \leq r\right] - d \cdot \sum_{\ell=1}^{r} p_{\sigma_i(\ell)} \right) = 0 \quad \text{(B.1)}$$

where $\mathbb{I}$ is the indicator function which returns 1 when the argument is true and 0 if otherwise. We have rewritten the rank function given in the original paper as σ^{-1}. Remember that p is a discrete probability distribution. As the original paper does, we write $p_{\sigma_i(\ell)}$ to indicate the probability that the alternative $\sigma_i(\ell)$ is chosen.

Ebadian et al. [2024] started with the following program.

$$\begin{aligned}
\min \quad & d \\
\text{s.t.} \quad & \delta_{i,a} \geq \frac{1}{r} \left(\mathbb{I}\left[\sigma_i^{-1}(a) \leq r\right] - d \cdot \sum_{\ell=1}^{r} p_{\sigma_i(\ell)} \right) && \forall i \in N, a \in A, r \in [1,m] \\
& \sum_{i \in N} \delta_{i,a} \leq 0 && \forall a \in A \\
& \sum_{a \in A} p_a = 1 \\
& p_a \geq 0 && \forall a \in A
\end{aligned}$$

With the above program, we will also find p that will achieve this minimal distortion. Note that $\delta_{i,a}$ is an upper bound on the inner maximum in Equation B.1, given a fixed $a = a^*$ and $i \in N$. Therefore, the second constraint makes the program equivalent to Equation B.1. The problem here is that the program is not linear because we multiply d, a decision variable, with some p_a, also a decision variable, in the first constraint. Ebadian et al. [2024] linearized this by introducing a new variable $\hat{p}_a = d \cdot p_a$ for

every $a \in A$. Since $d = \sum_{a \in A} \hat{p}_a$, we can obtain the following linear program where the decision variables are just $(\hat{p}_a)_{a \in A}$.

$$\begin{aligned}
\min \quad & \sum_{a \in A} \hat{p}_a \\
\text{s.t.} \quad & \delta_{i,a} \geq \frac{1}{r} \left(\mathbb{I}\left[\sigma_i^{-1}(a) \leq r\right] - \sum_{\ell=1}^{r} \hat{p}_{\sigma_i(\ell)} \right) && \forall i \in N, a \in A, r \in [1,m] \\
& \sum_{i \in N} \delta_{i,a} \leq 0 && \forall a \in A \\
& \hat{p}_a \geq 0 && \forall a \in A
\end{aligned}$$

From the solution $\hat{p}$ of this linear program, we can compute the optimal distortion d and the probability distribution p. This program currently has $O(nm)$ variables and $O(nm^2)$ constraints. The final step taken by Ebadian et al. [2024] is to reduce the number of constraints down to $O(mn)$. For $i \in N$, $r \in [1,m]$, let $s_{i,r} = \sum_{\ell=1}^{r} \hat{p}_{\sigma_i(\ell)}$. Then, we can rewrite the first constraint as

$$\delta_{i,\sigma_i(r)} \geq \max \left\{ \max_{\ell \in [1,r-1]} -\frac{1}{\ell} \cdot s_{i,\ell}, \max_{\ell \in [r,m]} \frac{1}{\ell} \cdot (1 - s_{i,\ell}) \right\}$$

Let $\alpha_{i,r} = \max_{\ell \in [1,r]} -\frac{1}{\ell} \cdot s_{i,\ell}$ and let $\beta_{i,r} = \max_{\ell \in [r,m]} \frac{1}{\ell} \cdot s_{i,\ell}$. Then, $\alpha_{i,r}$ and $\beta_{i,r}$ can be constrained with only $O(mn)$ constraints, and each $\delta_{i,r}$ can be bound with only two constraints.

$$\delta_{i,\sigma_i(r)} \geq \alpha_{i,r-1}, \quad \delta_{i,\sigma_i(r)} \geq \beta_{i,r}$$

This produces the following linear program with $O(mn)$ variables and constraints.

Linear Program $\mathcal{P}$

$$
\begin{aligned}
\min \quad & \sum_{a\in[1,m]} \hat{p}_a & \\
\text{s.t.} \quad & \delta_{i,\sigma_i(r)} \geq \alpha_{i,r-1} & \forall i \in N, r \in [2,m-1] \\
& \delta_{i,\sigma_i(r)} \geq \beta_{i,r} & \forall i \in N, r \in [1,m] \\
& \sum_{i=1}^{n} \delta_{i,a} \leq 0 & \forall a \in [1,m]
\end{aligned}
$$

Partial sums:

$$
\begin{aligned}
& s_{i,1} = \hat{p}_{\sigma_i(1)} & \forall i \in N \\
& s_{i,r} = s_{i,r-1} + \hat{p}_{\sigma_i(r)} & \forall i \in N, r \in [2,m]
\end{aligned}
$$

Top partial maximums:

$$
\begin{aligned}
& \alpha_{i,r} \geq \alpha_{i,r-1} & \forall i \in N, r \in [2,m-1] \\
& \alpha_{i,r} \geq \frac{1}{r}(-s_{i,r}) & \forall i \in N, r \in [1,m-1]
\end{aligned}
$$

Bottom partial maximums:

$$
\begin{aligned}
& \beta_{i,r} \geq \beta_{i,r+1} & \forall i \in N, r \in [1,m-1] \\
& \beta_{i,r} \geq \frac{1}{r}(1-s_{i,r}) & \forall i \in N, r \in [1,m]
\end{aligned}
$$

Variable ranges:

$$
\begin{aligned}
& \hat{p}_a \geq 0 & \forall a \in [1,m] \\
& \delta_{i,\sigma_i(r)}, \alpha_{i,r}, \beta_{i,r} \in \mathbb{R} & \forall i \in N, r \in [1,m]
\end{aligned}
$$

This linear program can be solved programmatically. For example, Google's OR-Tools [Perron and Furnon, 2024] offer functionality to express this linear program.