AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

FACULTY OF COMPUTER SCIENCE, ELECTRONICS AND TELECOMMUNICATIONS

DEPARTMENT OF COMPUTER SCIENCE

Master of Science Thesis

Implementacja algorytmów dla systemów Monroe i Chamberlina-Couranta z nieliniową funkcją satysfakcji
Implementation of algorithms for Monroe and Chamberlin-Courant systems under nonlinear satisfaction function

Author:	*Piotr Szmigielski*
Degree programme:	*Computer Science*
Supervisor:	*Piotr Faliszewski, PhD*

Kraków, 2015

Oświadczamy, świadomi odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonaliśmy osobiście i samodzielnie (w zakresie wyszczególnionym we wstępie) i że nie korzystaliśmy ze źródeł innych niż wymienione w pracy.

Contents

1. Introduction

We study the effectiveness of algorithms for approximate winner determination under the Monroe and Chamberlin-Courant multiwinner voting rules using nonlinear satisfaction function. Both of the rules aim to select a group of candidates that best represent the voters. Having good voting rules and algorithms for them is important, as multiwinner elections are used both in human societies (e.g. parliament elections) and software systems (e.g. recommendation systems). Rules studied in this paper are particularly interesting because they have both desired features of multiwinner rules: they provide accountability (there is connection between the elected candidates and the voters, so each voter has a representative assigned to her and each candidate knows who she represents) and proportional representation of the voters' views.

We assume that candidates participate in the election with multiple winners (a commitee with multiple members is selected) and they are elected by voters, each of whom ranks all the candidates (each voter provides a linear order over the set of candidates expressing their preferences). For each voter the Monroe and Chamberlin-Courant rules assign a single candidate as their representative (with some constraints, which are detailed in the further part of the thesis).

The candidates are selected and assigned to the voters optimally, either by maximizing the total satisfaction of all voters, or by minimizing the total dissatisfaction of all voters. The total satisfaction is calculated as a sum of individual satisfactions of the voters. We assume that there is a satisfaction function that measures how well a voter is represented by the candidate. The function is the same for each voter. It is a decreasing function, so a voter is more satisfied if the candidate assigned to her is ranked higher. The dissatisfaction is calculated in the similar way, except that the function is an increasing one. In this paper I study cases in which the satisfaction function is a nonlinear one.

The main drawback of the aforementioned rules is that election winner determination is NP-hard under each of them [1] which makes them hard to use in practice, as it would force the use of algorithms that do not provide an optimal result for every data set. Therefore, using these systems for real-life elections may rise some difficulties. However, they can be used for the recommendation systems conveniently, as a good but not optimal recommendation is still useful. Skowron et al. [2] provided approximation algorithms for both the rules that return near-optimal results for various test cases (including real-life data and synthetic data), but for linear satisfaction function only.

In this paper we focus on providing several algorithms for the Monroe and Chamberlin-Courant rules using nonlinear satisfaction function and evaluating them empirically against various data sets. For smaller data, results can be easily assessed by comparing them to the optimal result (calculated with the brute-force algorithm). For bigger data, the upper bound of the optimal result must be used for comparison. We implement and evaluate various heuristic algorithms, as well as the existing approximation algorithms for the linear satisfaction function, but applying them to the nonlinear cases.

2. Motivation and Goals

Monroe and Chamberlin-Courant systems may be potentially very useful, because they are one of the few multiwinner election systems that provide both accountability of candidates to the voters (each voter has one particular representative in the elected commitee) and proportional results. Most of the currently used voting rules lack at least one of these properties. For example, D'Hondt method used to elect members of Polish lower house of parliament lacks accountability (voters are accountable to political parties rather than specific parliament members), while single-member constituency plurality system used for United Kingdom parliament elections lacks proportionality.

As finding optimal solution for both of the aforementioned systems is NP-hard [1], there is a need to provide good algorithms which can compute result which is suboptimal but still as close to optimal as possible. Using such result in real-life elections (e.g. for parliament) is disputable. However, for some software applications, e.g. recommendation systems, it can be used seamlessly.

Skowron et. al [2] have already provided approximation algorithms for these systems, but only under linear satisfaction function. In this thesis we focus on providing algorithms for non-linear satisfaction functions, as they can better reflect real preferences of the voters.

3. Preliminaries

In the first part of the chapter we explain basic notions regarding multiwinner election problem. Then we present formal definitions of winner determination problem under Monroe and Chamberlin-Courant voting rules. Finally, we present some notions regarding computational complexity.

3.1. Basic notions

Preferences. For each $n \in \mathbb{N}$, by $[n]$ we mean $\{1, \ldots, n\}$. We assume that there is a set $N = [n]$ of *agents* and a set $A = \{a_1, \ldots a_m\}$ of *alternatives*. Each agent i has a *preference order* $\succ_i$ over A, i.e., a strict linear order of the form $a_{\pi(1)} \succ_i a_{\pi(2)} \succ_i \ldots \succ_i a_{\pi(m)}$ for some permutation π of $[m]$. For an alternative a, by $pos_i(a)$ we mean the position of a in the i'th agent's preference order. For example, if a is the most preferred alternative for i then $pos_i(a) = 1$, and if a is the least preferred one then $pos_i(a) = m$. A collection $V = (\succ_1, \ldots, \succ_n)$ of agents' preference orders is called a *preference profile*.

Subsets of the alternatives are often included in the descriptions of preference orders. If A is the set of all the alternatives and B is a nonempty strict subset of A, then by $B \succ A - B$ we mean that for the preference order $\succ$ all alternatives in B are preferred to those outside of B.

Positional scoring function. A *positional scoring function* (PSF) is a function $\alpha^m : [m] \to \mathbb{N}$. A PSF α^m is an *increasing positional scoring function* (IPSF) if for each $i, j \in [m]$, if $i < j$ then $\alpha^m(i) < \alpha^m(j)$. Analogously, a PSF α^m is a *decreasing positional scoring function* (DPFS) if for each $i, j \in [m]$, if $i < j$ then $\alpha^m(i) > \alpha^m(j)$.

If β^m is an IPSF then $\beta^m(i)$ can represent the *dissatisfaction* of an agent if she is assigned to an alternative ranked i'th in her preference order. We assume that for each IPSF $\beta^m(1) = 0$ (an agent is not dissatisfied with her best alternative). Analogously, a DPSF γ^m can represent an agent's satisfaction. We assume that for each DPSF $\gamma^m(m) = 0$ (an agent is completely not satisfied with her worst alternative). In some cases, we will write α instead of α^m to simplify notation.

An IPSF (DPSF) α^m is considered a *linear dissatisfaction (satisfaction) function* if for each $i \in [m-1]$ value $\alpha^m(i+1) - \alpha^m(i)$ is constant. Every IPSF (DPSF) not satisfying that constraint is considered a *nonlinear dissatisfaction (satisfaction) function*.

We define a family α of IPSFs (DPSFs) of the form $\alpha = (\alpha^m)_{m \in \mathbb{N}}$, where α^m is a PSF on $[m]$, such that:

1. For a family of IPSFs it holds that $\alpha^{m+1}(i) = \alpha^m(i)$ for all $m \in \mathbb{N}$ and $i \in [m]$.
2. For a family of DPSFs it holds that $\alpha^{m+1}(i+1) = \alpha^m(i)$ for all $m \in \mathbb{N}$ and $i \in [m]$.

We build families of IPSFs (DPSFs) by appending (prepending) values to functions with smaller domains. To simplify notation, we will refer to such families of IPSFs (DPSFs) as *normal* IPSFs (DPSFs).

Assignment functions. A *K-assignment function* is any function $\Phi : N \to A$, such that $\|\Phi(N)\| \leq K$ (it assigns agents to at most K alternatives). A *Monroe K-assignment function* is an assignment function that additionally satisfies the following constraint: For each alternative $a \in A$ we have that either $\left\lfloor \frac{\|N\|}{K} \right\rfloor \leq \|\Phi^{-1}(a)\| \leq \left\lceil \frac{\|N\|}{K} \right\rceil$ or $\|\Phi^{-1}(a)\| = 0$.

A *partial K-assignment function* is defined in the same way as a regular one, except that it may assign a null alternative, $\bot$, to some of the agents. It is convenient to think that for each agent i we have $pos_i(\bot) = m$. A *partial Monroe K-assignment* is a partial K-assignment that can be extended to a regular Monroe K-assignment. If he have an assignment function Φ, for each agent i we refer to alternative $\Phi(i)$ as the *representative* of i.

Having a normal IPSF (DPSF) α, we may consider the following three functions, each assigning a positive integer to a given assignment Φ:

$$l^{\alpha}_{sum}(\Phi) = \sum_{i=1}^{n} \alpha(pos_i(\Phi(i))), \tag{3.1}$$

$$l^{\alpha}_{max}(\Phi) = max_{i=1}^{n} \alpha(pos_i(\Phi(i))), \tag{3.2}$$

$$l^{\alpha}_{min}(\Phi) = min_{i=1}^{n} \alpha(pos_i(\Phi(i))). \tag{3.3}$$

These functions aggregate individual dissatisfaction (satisfaction) values of the agents to measure the quality of the assignment for the entire society. In the utilitarian framework (which is used in this thesis), we use the first function as the *total dissatisfaction function* in the IPSF case and as the *total satisfaction function* in the DPSF case. We use the second and the third functions, respectively, as the total dissatisfaction and satisfaction functions for IPSF and DPSF cases in the egalitarian framework.

3.2. Monroe and Chamberlin-Courant rules

We will now define the problems of winner determination under the Monroe and Chamberlin-Courant (abbreviated as CC) rules. In both cases the goal is to find an optimal assignment function, where the optimality is relative to on eof the total dissatisfaction or satisfaction functions introduced earlier. The former is to be minimized and the latter is to be maximized.

Definition 1. Let α be a normal IPSF. An instance of α-*CC-DisWinner* problem consists of a set of agents $N = [n]$, a set of alternatives $A = \{a_1, \dots, a_m\}$, a preference profile V of the agents, and positive integer K. We ask for a K-assignment function Φ such that $l^{\alpha}_{sum}(\Phi)$ is minimized. The problem α-*Monroe-DisWinner* is defined in the same way but we additionally require Φ to be a Monroe K-assignment function.

Definition 2. Let α be a normal DPSF. The problem α-*CC-SatWinner* is defined in the same way as α-*CC-DisWinner*, except that we seek a K-assignment Φ such that $l^{\alpha}_{sum}(\Phi)$ is maximized. The problem α-*Monroe-SatWinner* is defined in the same way but we additionally require Φ to be a Monroe K-assignment function.

In terms of solutions dissatisfaction-based problems are equivalent to the satisfaction-based ones. One can always transform an DPSF to an equivalent IPSF and vice versa.

The goal in our problems is to compute a particular (Monroe) K-assignment function. Such a function defines a set of K alternatives, who are viewed as the winners of the given multiwinner election.

For each subset of the alternatives $S \subseteq A$ such that $\|S\| \leq K$, we write Φ^S_α to denote the partial (Monroe) K-assignment that assigns agents only to the alternatives from S and such that Φ^S_α maximizes the utilitarian satisfaction $l^{\alpha}_{sum}(\Phi^S_\alpha)$.

3.3. Computational complexity

As for many normal IPSFs our problems are NP-hard [1], we seek approximate solutions.

Definition 3. Let r be a real number such that $r \geq 1 (0 < r \leq 1)$, let α be a normal IPSF (DPSF), and let R be either Monroe or CC. An algorithm is an r-approximation algorithm for α-R-DisWinner problem (α-R-SatWinner problem) if on each instance I it returns a feasible assignment Φ such that $l^{\alpha}_{sum}(\Phi) \leq r \cdot OPT$ (such that $l^{\alpha}_{sum}(\Phi) \geq r \cdot OPT$), where OPT is the optimal total dissatisfaction (satisfaction) $l^{\alpha}_{sum}(\Phi_{OPT})$.

4. State of the Art

5. Implemented Algorithms

In this chapter we present implemented algorithms for the utilitarian versions of Monroe and Chamberlin-Courant multiwinner voting rules in the satisfaction-based framework.

Proposition 1 (Implicit in the paper of Betzler et al. [3]). Let α be a normal DFSF, N be a set of agents, A be a set of alternatives, V be a preference profile of N over A, and S a K-element subset of A (where K divides $\|N\|$). Then there is a polynomial-time-algorithm that computes a (possibly partial) optimal K-assignment Φ_{α}^{S} (Monroe K-assignment Φ_{α}^{S}) of the agents to the alternatives from S.

5.1. Algorithm A

Algorithm A was first presented by Skowron et al. [2] and tries to solve α-Monroe-SatWinner. It builds a solution iteratively (greedily). In each step we pick some not-yet-assigned alternative a_i (using some criterion) and assign it to those $\frac{N}{K}$ agents that are not assigned to any other alternative yet and whose satisfaction of being matched with a_i is maximal. This algorithm runs in polynomial time. Pseudocode is presented in Algorithm 1.

5.2. Algorithm B

Algorithm B is an extension to Algorithm A and was presented in the same paper. The idea is to run Algorithm A first and then, using Proposition 1, optimally reassign the alternatives to the voters. It should noticeably improve the results of the algorithm and it still runs in polynomial time.

5.3. Algorithm C

Algorithm C is a further improvement over Algorithm B, also presented by Skowron et al. [2]. The idea is that instead of keeping only one partial function Φ that is iteratively extended up to the full assignment, we keep a list of up to d partial assignment functions, where d is a parameter of the algorithm. At each iteration, for each assignment function Φ among the d stored ones and for each alternative a that does not yet have agents assigned to by this Φ, we compute an optimal extension of

Algorithm 1 Algorithm A

```
1: procedure ComputeMonroeSatWinner
2:     $\Phi \leftarrow$ a map defining a partial assignment, iteratively built by the algorithm
3:     $\Phi^{\leftarrow} \leftarrow$ the set of agents for which the assignment is already defined
4:     $\Phi^{\rightarrow} \leftarrow$ the set of alternatives already used in the assignment
5:     if $K \leq 2$ then
6:         compute the optimal solution using an algorithm of Betzler et al. [2] and return
7:     $\Phi$ = $\{\}$
8:     for $i \leftarrow 1$ to $K$ do
9:         $score \leftarrow \{\}$
10:        $bests \leftarrow \{\}$
11:        for all $a_i \in A \setminus \Phi^{\rightarrow}$ do
12:            $agents \leftarrow$ sort $N \setminus \Phi^{\leftarrow}$ so that if agent $j$ preceeds agent $j'$ then $pos_j(a_i) \leq pos_{j'}(a_i)$
13:            $bests[a_i] \leftarrow$ choose first $\frac{N}{K}$ elements from $agents$
14:            $score[a_i] \leftarrow \sum_{j \in bests[a_i]}(m - pos_j(a_i))$
15:        $a_{best} \leftarrow argmax_{a \in A \setminus \Phi^{\rightarrow}} score[a]$
16:        for all $j \in bests[a_{best}]$ do
17:            $\Phi[j] \leftarrow a_{best}$
```

this Φ that assigns to a. As a result we obtain possibly more than d (partial) assignment functions. For the next iteration we keep those d of them that give highest satisfaction. If we take $d = 1$, we obtain Algorithm B. Pseudocode is presented in Algorithm 3.

Unlike previous algorithms, Algorithm C can be used for both Monroe and Chamberlin-Courant rules. To adapt it for the Chamberlin-Courant rule, we have to replace the contents of the first for all loop with the appropriate code, presented in Algorithm 2.

Algorithm 2 Algorithm C - for all code replacement

```
1: for all $a_i \in A \setminus \Phi^{\rightarrow}$ do
2:     $\Phi' \leftarrow \Phi$
3:     for all $j \in N$ do
4:         if agent $j$ prefers $a_i$ to $\Phi'(j)$ then
5:             $\Phi'(j) \leftarrow a_i$
6:     $newPar.push(\Phi')$
```

Algorithm 3 Algorithm C

```
procedure COMPUTEMONROESATWINNER
    $\Phi \leftarrow$ a map defining a partial assignment, iteratively built by the algorithm
    $\Phi^{\leftarrow} \leftarrow$ the set of agents for which the assignment is already defined
    $\Phi^{\rightarrow} \leftarrow$ the set of alternatives already used in the assignment
    $Par \leftarrow$ a list of partial representation functions
    $Par = []$
    $Par.push(//)$
    for $i \leftarrow 1$ to $K$ do
        $newPar = []$
        for $\Phi \in Par$ do
            $bests \leftarrow \{\}$
            for all $a_i \in A \setminus \Phi^{\rightarrow}$ do
                $agents \leftarrow$ sort $N \setminus \Phi^{\leftarrow}$ (agent $j$ preceeds agent $j'$ implies that $pos_j(a_i) \leq pos_{j'}(a_i)$
                $bests[a_i] \leftarrow$ choose first $\frac{N}{K}$ elements of $agents$
                $\Phi' \leftarrow \Phi$
                for all $j \in bests[a_i]$ do
                    $\Phi'[j] \leftarrow a_i$
                $newPar.push(\Phi')$
        sort $newPar$ according to descending order of the total satisfaction of the assigned agents
        $Par \leftarrow$ choose first $d$ elements of $newPar$
    for $\Phi \in Par$ do
        $\Phi \leftarrow$ compute the optimal representative function using an algorithm of Betzler et al. [3] for the set of winners $\Phi^{\rightarrow}$
    return the best representative function from $Par$
```

5.4. Algorithm R

As shown by Skowron et al. [2], algorithms A, B and C achieve very high approximations ratios under linear satisfaction function for the cases where K is small relative to m. For the remaining cases, we can use a sampling-based randomized algorithm (called Algorithm R). We expect that under nonlinear satisfaction function algorithms should behave analogously in relation to each other.

The idea of this algorithm is to randomly pick K alternatives and match them optimally to the agents, using Proposition 1. Such an algorithm may be very unlucky and pick K alternatives that all of the agents rank low. Yet, if K is comparable to m then it is likely that such a random sample would include a large chunk of some optimal solution. Algorithm can naturally be used for both Monroe and Chamberlin-Courant systems.

5.5. Algorithm AR

Algorithm family A-C and algorithm R are naturally suitable for different cases. Therefore, Skowron et al. [2] proposed to combine algorithms A and R. Pseudocode is presented in Algorithm 4.

TODO: requires further description

Algorithm 4 Algorithm AR

```
1: procedure ComputeMonroeSatWinner
2:     $\lambda \leftarrow$ required probability of achieving the approximation ratio equal $0.715 - e$
3:     if $\frac{H_K}{K} \geq \frac{e}{2}$ then
4:         compute the optimal solution using an algorithm of Betzler et al. [3] and return
5:     if $m \leq 1 + \frac{2}{e}$ then
6:         compute the optimal solution using a simple brute force algorithm and return
7:     $\Phi_1 \leftarrow$ solution returned by Algorithm A
8:     $\Phi_2 \leftarrow$ run the sampling-based algorithm - $\log(1 - \lambda) \cdot \frac{2+e}{e}$ times; select the assignment of the
   best quality
9:     return the better assignment among $\Phi_1$ and $\Phi_2$
```

5.6. Algorithm GM

Algorithm GM (greedy marginal improvement) was introduced by Lu and Boutilier [4] for the Chamberlin-Courant rule. It was generalized by Skowron et al. [2] to apply it to the Monroe rule as well, for which it can be viewed as an extension to Algorithm B.

We start with an empty set S. Then we execute K iterations. In each iteration we find an alternative a that is not assigned to agents yet, and that maximizes the value $\Phi_\alpha^{S\cup\{a\}}$. It requires a large number of computations of Φ_α^S, which is a notable disadvantage for the Monroe case, as a computation is a slow process based on min-cost/max-flow algorithm [3]. Pseudocode is presented in Algorithm 5.

Algorithm 5 Algorithm GM

```
1: procedure COMPUTESATWINNER
2:     $\Phi_\alpha^S$ - the partial assignment that assigns a single alternative to at most $\frac{n}{K}$ agents, that assigns to
   the agents only the alternatives from $S$, and that maximizes the utilitarian satisfaction $l_{sum}^\alpha(\Phi_\alpha^S)$
3:     $S \leftarrow \emptyset$
4:     for $i \leftarrow 1$ to $K$ do
5:         $a \leftarrow argmax_{a\in A\setminus S} l_{sum}^\alpha(\Phi_\alpha^{S\cup\{\alpha\}})$
6:         $S \leftarrow S \cup \{a\}$
7:     return $\Phi_\alpha^S$
```

5.7. Algorithm P

This algorithm, introduced by Skowron et al. [2], computes a certain value x and greedily computes an assignment that (approximately) maximizes the number of agents assigned to one of their top-x alternatives. If after this process some agent has no alternative assigned, we assign her to the most preferred alternative from those already picked.

$w(x)$ used in the algorithm is a Lambert's W-function, defined to be the solution of the equation $x = w(x)e^{w(x)}$. Pseudocode of Algorithm P is presented in Algorithm 6. It applies to Chamberlin-Courant rule.

5.8. Genetic Algorithm

This algorithm starts with an initial set of creatures (each of them presenting a possible solution under Chamberlin-Courant rule) which are later mutated and crossed over with each other. The best creatures (ones with the highest total satisfaction) are preferred for further mutation and crossover in order to better investigate neighbourhood of local maxima, but on the other hand algorithm also produces new creatures by crossing over random existing ones to better explore entire solution space, not limiting to local extrema.

In each iteration of the algorithm creatures are evaluated (satisfaction is computed). Best creature in terms of total satisfaction is compared with currently best found creature and takes its place if it is better. Half of the evaluated creatures (the best ones) are chosen for further propagation. Each of them is then

Algorithm 6 Algorithm P

```
1: procedure ComputeCCSatWinner
2:     $\Phi \leftarrow$ a map defining a partial assignment, iteratively built by the algorithm
3:     $\Phi^{\leftarrow} \leftarrow$ the set of agents for which the assignment is already defined
4:     $\Phi^{\rightarrow} \leftarrow$ the set of alternatives already used in the assignment
5:     $num\_pos_x(a) \leftarrow \|\{i \in [n] \setminus \Phi^{\leftarrow} : pos_i(a) \leq x\}\|$ - the number of not-yet assigned agents that
   rank alternative $a$ in one of their first $x$ positions
6:     $w(\cdot)$ - Lambert's W-function
7:     $\Phi = \{\}$
8:     $x = \left\lceil \frac{mw(K)}{K} \right\rceil$
9:     for $i \leftarrow 1$ to K do
10:        $a_i \leftarrow argmax_{a \in A \setminus \Phi^{\rightarrow}} num\_pos_x(a)$
11:        for all $j \in [n] \setminus \Phi^{\leftarrow}$ do
12:            if $pos_j(a_i) < x$ then
13:                $\Phi[j] \leftarrow a_i$
14:    for all $j \in A \setminus \Phi^{\leftarrow}$ do
15:        $a \leftarrow$ such server from $\Phi^{\rightarrow}$ that $\forall_{a' \in \Phi^{\rightarrow}} pos_j(a) \leq pos_j(a')$
16:        $\Phi[j] \leftarrow a$
```

mutated randomly. Remaining creatures are created by crossing over random creatures from the "better" half with each other. Resulting set of creatures is used for the next iteration. Pseudocode of the algorithm is presented in Algorithm 7.

5.9. Simulated Annealing

TODO

Algorithm 7 Genetic Algorithm

```
1: procedure ComputeCCSatWinner
2:     $I$ - number of iterations
3:     $c$ - number of creatures
4:     $\Phi_{best}$ - best creature (preference profile)
5:     $creatures \leftarrow$ generate initial random set of $c$ creatures
6:     for $i \leftarrow 1$ to I do
7:         $creaturesSorted \leftarrow$ sort $creatures$ by total satisfaction
8:         if $satisfaction(creaturesSorted[1]) > satisfaction(\Phi_{best})$ then
9:             $\Phi_{best} = creaturesSorted[1]$
10:        $bestCreatures \leftarrow$ choose first $c/2$ elements from $creaturesSorted$
11:        $mutated \leftarrow$ mutate all creatures from $bestCreatures$ randomly
12:        $crossed \leftarrow$ crossover random creatures from $bestCreatures$ to produce $c/2$-element set
13:        $newCreatures \leftarrow mutated \cup crossed$
14:        $creatures \leftarrow newCreatures$
```

6. Evaluation Results

7. Summary

References

[1] Ariel D. Procaccia, Jeffrey S. Rosenschein, and Aviv Zohar. On the complexity of achieving proportional representation. *Social Choice and Welfare*, 30(3):353–362, apr 2008.

[2] P. Skowron, P. Faliszewski, and A. Slinko. Achieving fully proportional representation, approximability results. *Artificial Intelligence*, 222:67–103, may 2015.

[3] N. Betzler, A. Slinko, and J. Uhlmann. On the computation of fully proportional representation. *Journal on Artificial Intelligence Research*, 47:475–519, 2013.

[4] Tyler Lu and Craig Boutilier. Budgeted social choice: From consensus to personalized decision making. *IJCAI*, 11, 2011.

List of Tables