Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Ranking Alternatives Online

Bachelor thesis

Philipp Rimle
primle@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Georg Bachmeier
Prof. Dr. Roger Wattenhofer

August 29, 2016

Acknowledgements

I thank Professor Roger Wattenhofer for the opportunity to write my thesis in his research group. I would also like to especially thank my advisor Georg Bachmeier for contributing his ideas and the valuable feedback and advice he provided.

Abstract

The problem of finding an optimal social choice for pairwise elections between alternatives regarding the Slater, Kemeny and FLAP welfare functions is NP-hard. We present an approach to calculate global rankings using online heuristics that can efficiently handle new incoming votes. A single vote can be updated in $\mathcal{O}(n)$ and a set of complete votes can be processed in $\mathcal{O}(r \cdot n^3)$, where n is the number of alternatives and r is the number of complete votes. Our online heuristics have on average only 2–3 percent higher objective costs than the optimum offline solutions.

Contents

Acknowledgements i

Abstract ii

1 Introduction 1

1.1 Related Work . . . 2

1.2 Outline . . . 2

2 Model 3

3 Test Data 6

3.1 Generated Data . . . 6

3.2 Real World Data . . . 6

3.3 Test Environment . . . 7

4 Optimal Offline Solution 8

4.1 First Version . . . 8

4.2 Second Version . . . 9

5 Online Algorithms 12

5.1 Slater . . . 12

5.1.1 Online Heuristic . . . 12

5.1.2 Move to Edge Variant . . . 13

5.1.3 Objective Cost and Runtime Analysis . . . 15

5.2 Kemeny . . . 19

5.2.1 Online Heuristic . . . 19

5.2.2 Runtime and Cost Analysis . . . 20

5.3 FLAP . . . 24

5.3.1 Online Heuristic . . . 24

5.3.2 Runtime and Cost Analysis . . . 25

5.4 Comparison between Social Welfare Functions . . . 30

5.4.1 Simulation . . . 30

5.4.2 Real World Data . . . 32

5.5 Constant Number of Alternatives 32
5.6 Bucket Updates 37

6 Conclusion **39**

Bibliography **40**

Chapter 1

Introduction

In a lot of circumstances one is interested in the result of group preferences of individual opinions, interests or preferences. What is the social choice? How well does it represent each individuals opinion? One might be interested in political elections, e.g. who wins the election having ballots with an order of the candidates? What is the final ranking of the candidates? Perhaps one is interested in ranking products against each other. A group might want to decide in which restaurant they go or what movie they watch. One might want to compare different websites, food products, cities or a multitude of other possibilities.

There are a lot of methods how one can evaluate votes and find a social choice. For example we can have a plurality voting, where each individual voter gives exactly one vote from a set of options and the winner is simply determined by the number of votes.

If the voters are required to rank the alternatives in a relative preference we call it a *preferential voting*. In preferential voting, the voters give an order of all alternatives according to their preferences. A simple method to obtain the final ranking would be the Borda Count. For m candidates each voter would simply assign m points to the first choice, $m-1$ to the second choice and so on. The final ranking can be obtained by ordering the alternatives descending regarding their total received points. However, Borda might not give the social choice one is interested in due to Borda's Voting Paradox described in [1]. E.g. Borda could rank an alternative that wins most of the time against the other alternatives but in a few situation looses against all in the middle even though the alternative would win each pairwise election. A lot of research is conducted in the area of pairwise election to obtain a "best" social choice.

Social welfare functions classify a social choice as more or less desirable regarding the individual preferences. We will look at the two well known social welfare functions Slater [2] and Kemeny [3] and a third social welfare function called FLAP. All three social welfare functions rely on pairwise elections between the alternatives.

However to find the best global ranking for Slater, Kemeny and FLAP is NP-Hard [4]. There are several approaches for offline heuristics to derive such a global ranking assuming the knowledge of the full data set, but as soon as new votes are added, one has to recalculate everything, which is time intensive.

In this thesis, we present approaches to calculate global rankings using online heuristics regarding Slater, Kemeny and FLAP under the assumption that we already have a current global ranking and we receive new comparisons between the alternatives. In contrast to the offline solutions, our online approaches can efficiently handle new incoming votes and update the global ranking regarding the social welfare functions without complete recalculations.

1.1 Related Work

Social choice theory is widely studied [5, 6, 7] and there are several approaches for calculating offline solutions. Vincent Conitzer published an approach to calculate a Slater ranking in linear time if the pairwise election graph is hierarchically structured [8]. In [9] Alnur Ali and Marina Meilă give an overview over several methods for rank aggregation regarding the Kemeny welfare function. Clair Mathieu and Adrian Vladu described in [10] the problem of producing a global ranking when new alternatives arrive. However the online case where we receive a new comparison between known alternatives is not well studied yet.

1.2 Outline

The rest of the thesis is structured as follows:

- In the **second chapter** we will describe our model and give some definition that we will use throughout this thesis.
- The **third chaper** includes how the simulated data for the different test runs are generated and which real world data we use.
- In **chapter four** we describe how optimal offline solutions can be implemented using ILPs.
- In the **fifth chapter** we present our approaches for online heuristics regarding the Slater, Kemeny and FLAP welfare functions. For each welfare function we describe how our online heuristic works and evaluate it with different datasets. Furthermore we compare the different version against each other and present some additional features.
- Finally, **chapter six** contains the conclusion, where we summarise our approaches and describe possible future research that can be conducted in this area.

Chapter 2

Model

In the following chapter we describe our model and give some definitions for frequently used terms in the thesis. In the following let $\mathcal{A} := \{a_1, a_2, \ldots, a_n\}$ be a set of alternatives. A *single vote* between two alternatives is given as $v_{a_1 a_2} := (a_1 \succ a_2)$, where $a_1 \succ a_2$ denotes that a_1 is ranked higher than a_2.

Definition 2.1 (Complete Vote)**.** A *complete vote* $v := (a_1 \succ a_2 \succ \cdots \succ a_n)$ is a strict order over all alternatives in $\mathcal{A}$. Complete votes correspond to permutations of the n elements of $\mathcal{A}$.

Definition 2.2 (Voting Set)**.** A *voting set* $\mathcal{V}_\mathcal{A}$ is a finite set of complete votes v over the alternatives in $\mathcal{A}$.

In a voting set $\mathcal{V}_\mathcal{A}$ we have pairwise elections between alternatives. Let $c_{a \succ b}$ be the number of times a is ranked before b in $\mathcal{V}_\mathcal{A}$ for some alternatives $a, b \in \mathcal{A}$ and $c_{b \succ a}$ the number of times b is ranked before a respectively. If $(c_{a \succ b}) > (c_{b \succ a})$ we call a the winner of the pairwise election between a and b with respect to the voting set $\mathcal{V}_\mathcal{A}$. Note that we can also have pairwise elections in a set of single votes over some $\mathcal{A}$, i.e. we do not have complete votes but nevertheless can count how many times an alternative is ranked higher than another by simply looking at all single votes with both alternatives.

Definition 2.3 (Pairwise Election Graph)**.** For a voting set $\mathcal{V}_\mathcal{A}$, the *pairwise election graph* $PEG := (V, E)$ consists of a set of vertices and a set of directed edges between them. The set of vertices is simply the set of alternatives, $V := \mathcal{A}$ and there is a directed edge from a vertex a to vertex b if a is the winner of the pairwise election in $\mathcal{V}_\mathcal{A}$, i.e. $(c_{a \succ b}) > (c_{b \succ a})$, with weight $w_{ab} := (c_{a \succ b}) - (c_{b \succ a})$.

For a given voting set we try to find a global ranking which "represents" the voting set the best, i.e. a permutation on $\mathcal{A}$ representing the votes in $\mathcal{V}_\mathcal{A}$. For that we need to define what is meant by "represent best". One way is to define a social welfare function, which classifies possible global rankings as more desirable or less desirable. More formally a social welfare function in our case is a function that assigns a cost c to a voting set $\mathcal{V}_\mathcal{A}$ and a global ranking r. A global ranking r that minimizes c is an optimal solution. For our model, we consider three social welfare functions: Slater, Kemeny and FLAP, defined as follows:

Definition 2.4 (Slater Rule)**.** Given a voting set $\mathcal{V}_\mathcal{A}$, a global ranking r and two alternatives $a, b \in \mathcal{A}$, let $\delta_{ab}(r) = 1$ if r ranks the winner of the pairwise election between a and b lower than the loser, and 0 otherwise. The Slater rule tries to minimize:

$$\sum\nolimits_{a,b \in \mathcal{A}} \delta_{ab}(r) \tag{2.1}$$

Definition 2.5 (Kemeny Rule)**.** Given a global ranking r and a single vote v_{ab} for two alternatives a,b. Let $\delta_{ab}(r, v) = 1$ if r and v disagree on the relative ranking of a and b, and 0 otherwise. The Kemeny rule tries to minimize:

$$\sum_{a,b \in \mathcal{A}} \sum_{v \in \mathcal{V}_\mathcal{A}} \delta_{ab}(r, v) \tag{2.2}$$

The Kemeny rule tries to minimize the number of single votes disagreeing with the global ranking and the Slater rule tries to minimize the number of pairwise election winners, disagreeing with the global ranking.

If we apply a *linear order* over the alternatives of a pairwise election graph, we get another view of the problem of finding a global ranking over a set of votes. A global ranking is simply a permutation of a set of alternatives $\mathcal{A}$ and hence represents a linear order over these alternatives. We call an edge $e_{ab} \in E$ "backward edge" for some alternative a and b, if for a given global ranking over a pairwise election graph PEG, a is ranked lower than b. Similarly we call an edge e_{ab} to be "forward" if alternative a is ranked higher than b. Note that the weight $w_e = w_{ab}$ of the edge is the difference of the pairwise elections where a wins against b and the pairwise elections where b wins against a. A forward edge e_{ab} in the pairwise election graph over a linear order therefore indicates that a global ranking agrees with the pairwise election of a and b. Similarly a backward edge e_{ab} indicates that a global ranking disagrees with the pairwise election of a and b in a voting set $\mathcal{V}_\mathcal{A}$. Figure 2.1 shows an example of applying a linear order over a pairwise election graph and the resulting forward and backward edges. Throughout this thesis we read a pairwise election graph with a linear order from left to right, i.e. the highest ranked alternative is on the left and the lowest ranked alternative on the right.

The Slater rule can therefore be reinterpreted as finding a global ranking where the number of backward edges in the pairwise election graph is minimal. Note that this is closely related to the problem of finding a minimum feedback arc set. Similarly the Kemeny rule can be seen as a minimization over the sum of weights of all backward edges in the pairwise election graph over a linear order.

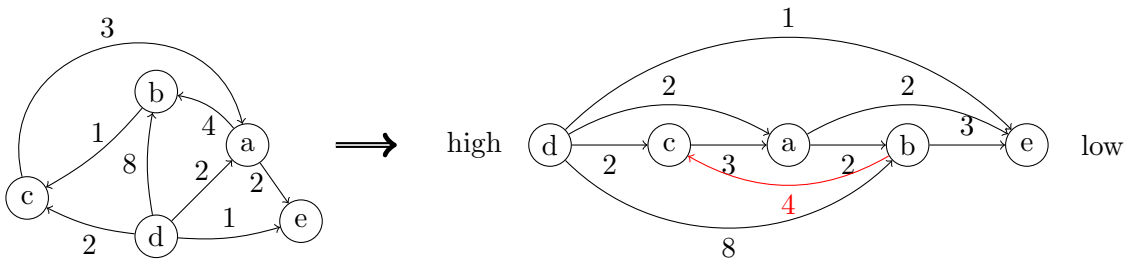


Figure 2.1: Example of a linear order applying to a pairwise election graph: On the left hand side we have a pairwise election graph built from a voting set $\mathcal{V}_\mathcal{A}$, where $\mathcal{A} := \{a, b, c, d, e\}$. On the right hand side we see the pairwise election graph ordered according to a linear order (here $d \succ c \succ a \succ b \succ e$). Note that in this example, where d is the highest and e the lowest ranked alternative in the global ranking, the global ranking is an optimal solution for Slater with cost of one, but not for Kemeny, as we could place b in front of c and obtain a total cost of two instead of four.

Definition 2.6 (The Feedback Linear Arrangement Problem – FLAP)**.** Let $\pi : \mathcal{A} \to \mathcal{A}$ be a permutation over the set of alternatives and let S_π be the set of all permutations over the set of

alternatives $\mathcal{A}$. Given a pairwise election graph $PEG = (V, E)$ over a voting set $\mathcal{V}_{\mathcal{A}}$, we define the FLAP rule as:

$$\min_{\pi \in S_n} \sum_{\substack{e=(i,j) \in E \\ \pi(i) < \pi(j)}} w(e) \cdot (\pi(j) - \pi(i)) \tag{2.3}$$

In other words the FLAP rule not only tries to minimize the weight of backward edges in our pairwise election graph as the Kemeny rule does, but also minimizes over the weights times the length of backward edges, where the length is given as the distance between the alternatives in the global ranking. The idea behind FLAP is that if the global ranking has to disagree with some pairwise elections between two alternative a and b, we prefer a global ranking where the alternatives a and b are closer together.

Chapter 3

Test Data

3.1 Generated Data

Our simulated data are based on a consensus probability between voters for each single vote presented in [11]. A voting set $\mathcal{V}_{\mathcal{A}}$ is constructed based on an underlying probability model. We first generate a total order v_{cons} of the alternatives over $\mathcal{A}$ representing a consensus order and then generate a complete vote for each voter. To construct a complete vote we build an acyclic graph over the set of alternatives regarding a consensus probability $p \in [0, 1]$ as follows:

- Take two alternatives $a, b \in \mathcal{A}$ without an edge in between and v_{cons} ranks a higher than b.
- We build a directed edge from a to b with probability p, otherwise an edge from b to a.
- For a new edge e_{ab} we add an edge from all alternatives that can reach a to all alternatives b can reach. Similarly for a new edge e_{ba} we add an edge from all alternatives that can reach b to all alternatives a can reach. (Note that in this way we can prevent cycles)
- Repeat the above steps until we have a directed edge between all alternatives.

The code to generate a complete vote was implemented by Raphael Anderegg. To generate a voting set we fix the consensus probability p and generate the desired number of complete votes. For our test runs we have built datasets for five up to 250 alternatives and consensus probabilities $p \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. Each dataset consists of 1000 complete votes. Furthermore there are twenty different datasets per number of alternatives per consensus probability, used to run several test runs on similar problems. So a dataset is described with (n_a, p, n_v) for $n_a \in \{5, \ldots, 250\}$, $p \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ and $n_v \in \{1, \ldots, 1000\}$. Note that for a given dataset with 1000 rankings we also can randomly take a number of complete votes less than 1000.

3.2 Real World Data

There are a few real world data sets with both the number of votes and alternatives being large enough to be interested here. For our tests with real world data we use a dataset collected by http://www.preflib.org.

The dataset contains the preferences about various kinds of sushi of 5000 individuals, each ranked 10 sushi types from a set of 100 sushi types in total. The dataset is a result of a series of surveys conducted by Toshihiro Kamishima.

3.3 Test Environment

All tests run on the EULER cluster provided by ETH Zürich. For our test runs we used four cores if the algorithms for the offline solution were involved and two cores otherwise to speed up the parallelisable offline algorithms. The cores are part of 12-core Intel Xeon E5-2697v2 processors with a nominal clock rate of 2.7 GHz. We assigned each core 4096 MB DDR3 memory clocked at 1866 MHz.

Chapter 4

Optimal Offline Solution

Finding an optimal global ranking for Slater, Kemeny and FLAP regarding pairwise election is NP-hard [4]. One way of solving the ranking problem is using Integer Linear Programming (ILP). In the following chapter we present two approaches for finding an optimal solution using ILPs. The first is a straightforward approach with a slow runtime and thus can only be used up to around twenty alternatives within four days if we test on twenty datasets per number of alternatives. The second approach models the problem from a different perspective and can be used up to around eighty alternatives. The faster runtime is due to the smaller space of feasible solutions. However, the second approach can only be used for Slater and Kemeny. We describe how the ILPs are constructed in both approaches, given a pairwise election graph built from a voting set over alternatives in $\mathcal{A}$.

4.1 First Version

The first version of the ILP consists of three types of decision variables, all integer constrained, an objective function and constraints according the social welfare functions. Let us start with the decision variables:

Permutation We have a two dimensional permutation matrix of decision variables $p_{ij} \in \{0, 1\}$ for $i, j \in \{1, \ldots, |\mathcal{A}|\}$ which are used to model a permutation of the alternatives for the global ranking, i.e. $p_{ij} = 1$ if alternative i has rank j.

Final Rank We have an array consisting of decision variables r_i for the final rank of alternative $i \in \mathcal{A}$. E.g $r_i = k$ denotes alternative i has rank k in the final global ranking for some $k \in \{1, \ldots, |\mathcal{A}|\}$.

Backward Edge We have a decision variable $e_{back_{ij}}$ for $i, j \in \{1, \ldots, |\mathcal{A}|\}$ if there is an edge from i to j in the pairwise election graph. For Slater and Kemeny the variable $e_{back_{ij}}$ indicates if an edge from alternative i to j is backward in the final global ranking and for FLAP $e_{back_{ij}}$ is the length of the respective backward edge.

The objective function is given from the according welfare function. For Slater we try to minimize the sum of the backward edges (Equation 4.1) and for Kemeny we minimize the sum of weights over all backward edges (Equation 4.2). FLAP minimizes over the weight of the backward edges times their length, but since we model $e_{back_{ij}}$ as the length of a backward edge for FLAP, we can use the same objective as for Kemeny (Equation 4.2).

$$\min \sum e_{back_{ij}} \tag{4.1}$$

$$\min \sum (w_{ij} \cdot e_{back_{ij}}) \tag{4.2}$$

We have several constraints holding for each model (Slater, Kemeny and FLAP). To model a permutation with the $|\mathcal{A}|^2$ many decision variables $p_{ij} \in \{0, 1\}$ (Equations 4.3, 4.4), we assign to each position in the global ranking exactly one alternative (Equation 4.5) and to each alternative exactly one position in the global ranking (Equation 4.6). The final rank r_i (position in the global ranking) of an alternative i is given through the column position where $p_{ij} = 1$ in the permutation matrix (Equation 4.7). For Slater and Kemeny we are only interested in whether an edge is backward (in the global ranking) or not. An edge from node i to j is backward if the position of alternative i is behind j, i.e. $r_i - r_j > 0$. Our decision variables $e_{back_{ij}}$ take the value 1 if we have a backward edge from i to j and 0 otherwise. Equations 4.8, 4.9 and 4.10 achieve this goal. Our ILP for FLAP just replaces the constraint 4.9 and 4.10 by 4.11, as $e_{back_{ij}}$ is 0 if the edge from i to j is a forward edge and otherwise the length of the edge.

$$p_{ij} \geq 0 \quad \text{for} \quad i, j \in \{1, \ldots, |\mathcal{A}|\} \tag{4.3}$$
$$p_{ij} \leq 1 \quad \text{for} \quad i, j \in \{1, \ldots, |\mathcal{A}|\} \tag{4.4}$$

$$\sum_{i=1}^{|\mathcal{A}|} (p_{ij}) = 1 \quad \text{for} \quad j \in \{1, \ldots, |\mathcal{A}|\} \tag{4.5}$$
$$\sum_{j=1}^{|\mathcal{A}|} (p_{ij}) = 1 \quad \text{for} \quad i \in \{1, \ldots, |\mathcal{A}|\} \tag{4.6}$$
$$\sum_{j=1}^{|\mathcal{A}|} (j \cdot p_{ij}) = r_i \quad \text{for} \quad i \in \{1, \ldots, |\mathcal{A}|\} \tag{4.7}$$

$$(All) \qquad e_{back_{ij}} \geq 0 \qquad \text{for all} \quad e_{back_{ij}} \tag{4.8}$$
$$(Slater, Kemeny) \qquad e_{back_{ij}} \leq 1 \qquad \text{for all} \quad e_{back_{ij}} \tag{4.9}$$
$$(Slater, Kemeny) \qquad e_{back_{ij}} \geq \frac{r_i - r_j}{|\mathcal{A}|} \qquad \text{for all} \quad e_{back_{ij}} \tag{4.10}$$
$$(FLAP) \qquad e_{back_{ij}} \geq r_i - r_j \qquad \text{for all} \quad e_{back_{ij}} \tag{4.11}$$

The optimal global ranking is given through the values of the final rank decision variables r_i, indicating the position of alternative i in the global ranking.

4.2 Second Version

In a second approach we try to improve the runtime of finding an optimal offline solution by modifying the linear program according to a model described in [12]. The approach is slightly different and closely related to the minimum feedback arc set problem, which tries to find the smallest set of edges in a directed graph such that removing this set results in an acyclic graph. The idea is that we can find a minimum feedback arc set by removing edges from disjoint cycles in the global ranking. Note that if we break cycles we get an acyclic graph. The Slater rule is equivalent to finding such a minimum feedback arc set. For Kemeny we try to break cycles

by removing the edges with the smallest weight in the disjoint cycles, i.e. the problem can be reformulated as:

$$\text{For any set of edge-disjoint cycles C,} \quad \sum\nolimits_{c \in C} \min\nolimits_{e \in c} w_e \quad \text{is an optimal solution for Kemeny} \tag{4.12}$$

Whereas it is easy to see the relation between the second approach and Kemeny and Slater, no optimality preserving reduction from FLAP to the minimum feedback arc set problem is apparent to us and therefore we only use the first ILP for FLAP. The Integer Linear Program for Slater and Kemeny is built as follows:

Our ILP has integer constrained binary decision variables x_{ij} for $i, j \in \{1, \ldots, |\mathcal{A}|\}$ representing an edge between any pair of nodes. Note that we start with a fully connected graph with edges in both directions. Let E be the set of actual edges in our pairwise election graph and $e \in E$ an actual edge. In the following we write $x_e := x_{ij}$ if there is an edge from alternative i to alternative j and similarly $w_e := w_{ij}$ for the weight of an edge. We set $w_{ij} := 0$ if there is no edge in the pairwise election graph from i to j. The objectives are given through 4.13 (Slater) and 4.14 (Kemeny).

$$\min \sum\nolimits_{e \in E} x_e \tag{4.13}$$

$$\min \sum\nolimits_{e \in E} (w_e \cdot x_e) \tag{4.14}$$

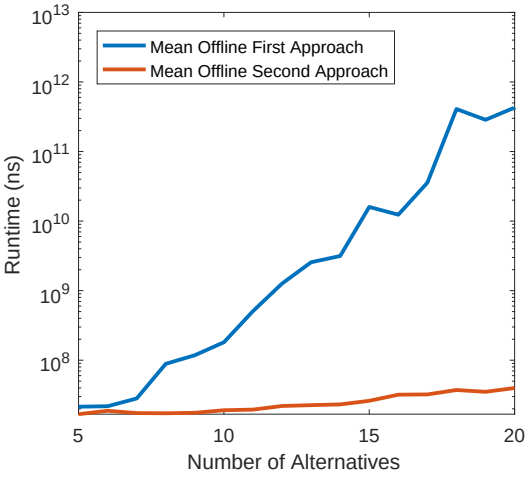
Our decision variables x_{ij} indicating an edge from i to j are set to one if i is ranked behind j in the global ranking and 0 otherwise. Constraint 4.17 ensures that exactly one of the two alternatives is ranked behind the other (in the end we must have a graph with directed edges) and Constraint 4.18 enforces transitivity between the alternatives.

$$x_{ij} \geq 0 \quad \text{for} \quad i, j \in \{1, \ldots, |\mathcal{A}|\} \tag{4.15}$$
$$x_{ij} \leq 1 \quad \text{for} \quad i, j \in \{1, \ldots, |\mathcal{A}|\} \tag{4.16}$$

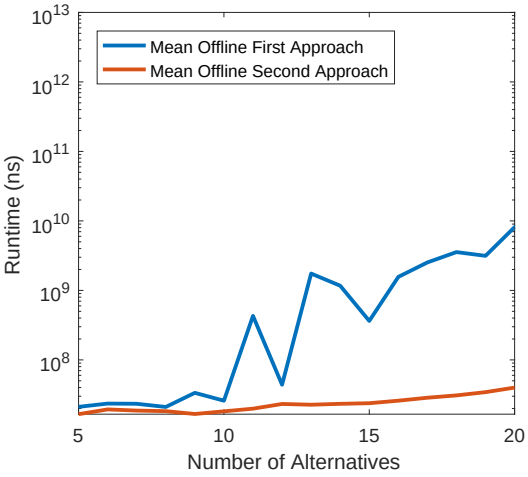
$$x_{ij} + x_{ji} = 1 \quad \text{for all distinct} \quad i, j \in \{1, \ldots, |\mathcal{A}|\} \tag{4.17}$$
$$x_{ij} + x_{jk} + x_{ki} \geq 1 \quad \text{for all distinct} \quad i, j, k \in \{1, \ldots, |\mathcal{A}|\} \tag{4.18}$$

The optimal solution of the ILP is given as an acyclic graph and to get the global ranking we simply do a topological sorting.

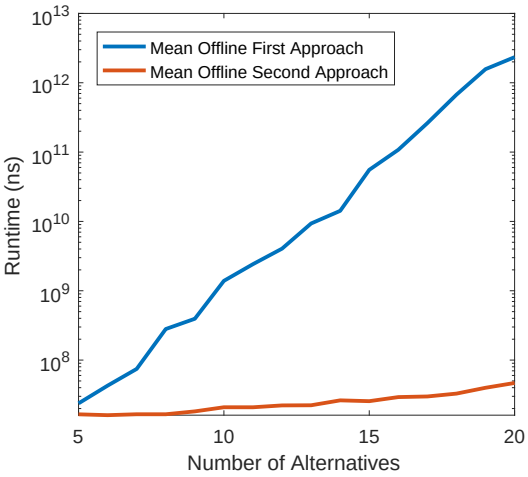
Figure 4.1 shows the logarithmic runtime in nanoseconds of the first and the second approach. The second approach is a lot faster for both consensus probabilities 0.6 and 0.8, but the difference is greater for lower consensus probabilities. In the remainder of the thesis we will only use the second approach for our offline test runs regarding the Slater and Kemeny welfare function.



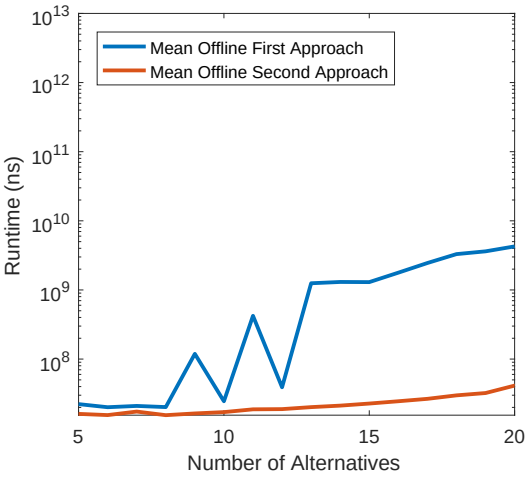
(a) Slater, consensus probability of 0.6



(b) Slater, consensus probability of 0.8



(c) Kemeny, consensus probability of 0.6



(d) Kemeny, consensus probability of 0.8

Figure 4.1: Runtime in nanoseconds of the optimum offline solution regarding Slater and Kemeny, implemented with the first and the second offline approach. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings. The Figures on the left side show the runtime using datasets with consensus probability of 0.6 and the Figures on the right side show the runtime using datasets with consensus probability of 0.8.

Chapter 5

Online Algorithms

In the following chapter we will present online heuristics for the three welfare functions Slater, Kemeny and FLAP. The state is independent of the welfare function, it consists of a current global ranking and a pairwise election graph with the comparisons seen so far. The initial state is given with a random global ranking and a pairwise election graph with no edges. Over time, new single votes between alternatives are revealed, and the global ranking is changed with an update method according to the new information.

5.1 Slater

The Slater welfare function (Def. 2.4) applied to our problem tries to minimize the number of backward edges in the pairwise election graph with regard to a global ranking. It does not optimize the weights of the edges nor the length of the edges.

5.1.1 Online Heuristic

The idea is based on the intuition that for a new single vote between two alternatives a and b, possible changes in the global ranking only happen in the region between those alternatives. Our approach is therefore to find the position of a and b in the current global ranking and to determine if the cost of the objective function decreases when changing the position. A first observation is that it only makes sense to change the global ranking if the new single vote disagrees with the current ranking. Therefore we will leave the ranking as it is if the incoming single vote agrees with it. If the new single vote v_{ab} disagrees with the current ranking, we have the following four cases:

Case 1.1 We increment the weight of an already existing backward edge. In this case we will not change the global ranking because the slater welfare function tries to minimize the number of backward edges and changing the global ranking would not make any improvements nor degradations.

Case 1.2 A new backward edge appears (see Figure 5.1), i.e. there was a tie between the two alternatives before. We simply go through S_2, the set of alternatives between a and b in the global ranking, to see if we can decrease the objective cost by moving a directly in front of b or b directly behind a or both. Let $\xi_{a_{back}}$ and $\xi_{b_{back}}$ be the number of forward edges that would become a backward edge by moving a in front of b, or moving b behind a respectively. Let $\xi_{a_{for}}$ and $\xi_{b_{for}}$ be the number of backward edges that would become forward edges again changing the position of a in the global ranking or b respectively. It

only makes sense to change a if $\xi_{a_{back}} - \xi_{a_{for}} \leq 1$ as we could reduce the objective cost by one (the new backward edge will then be a forward edge). The same holds for b, but we only move both alternatives if $\xi_{a_{back}} - \xi_{a_{for}} \leq 0$ and $\xi_{b_{back}} - \xi_{b_{for}} \leq 0$ because otherwise we could increase the objective cost. For $(\xi_{a_{back}} - \xi_{a_{for}}) = (\xi_{b_{back}} - \xi_{b_{for}}) = 1$ we choose randomly a or b.

Case 2.1 We decrement the weight of a forward edge. Similar to case 1.1 we leave the ranking as it is, since we cannot make any improvements as Slater only considers the number of edges (instead of their weights) for the objective cost function. The old ranking will therefore not be worse with the new single vote.

Case 2.2 An existing forward edge disappears (see Figure 5.2), i.e. with the new single vote there is a tie between the two alternatives. Moving a directly in front of b or b directly behind a will not worsen our objective cost regarding the comparison between those alternatives only. Similar to case 2.1 we can go through S_2 and look if we can improve the objective cost by changing a or b. This is the case if $\xi_{a_{back}} - \xi_{a_{for}} \leq 0$ or $\xi_{b_{back}} - \xi_{b_{for}} \leq 0$ similar to Case 1.2 with zero instead of one on the right hand side of the equation. The idea is that we might were not able to move the alternatives a or b with the existing forward edge, as this edge would have increased our objective cost by one. Now we do not have the handicap anymore and reducing the objective cost might be possible.

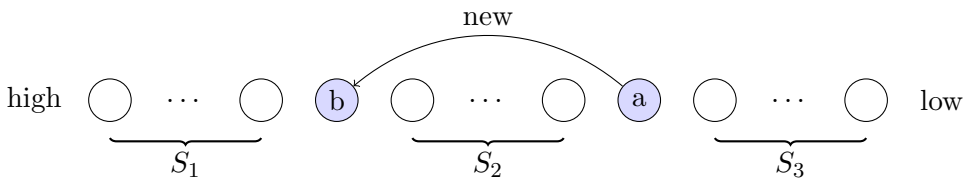


Figure 5.1: Backward Edge Appears

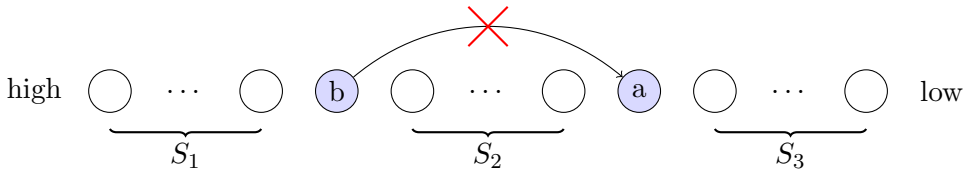


Figure 5.2: Forward Edge Disappears

5.1.2 Move to Edge Variant

A second approach tries to handle poor local minima, which Algorithm 1 might fall in. The idea is that even though it makes sense to test in the set between two alternatives from an incoming vote disagreeing with the ranking, it might be possible to balance the ranking in a better way. We are now not only interested in moving one of the alternatives directly behind the other but additionally try to move the alternative as far as possible to the edge of the ranking, i.e. find the furthermost position for an alternative without increasing the objective function. In other words we try to . The basic principle from Algorithm 1 stays the same. We additionally go through S_1 and S_3 as shown in Figure 5.1 and 5.2 and find the position where the objective cost is minimal.

Algorithm 1: SLATER ONLINE: SINGLE UPDATE

Input : *globalRanking* (1D array, storing the currently global ranking)
pairwiseElectionGraph (2D matrix, storing the weight of the edges between alternatives according to previous updates)
(a, b), a new single vote $v_{ab} = (a \succ b)$

Output: updated *globalRanking*

$pairwiseElectionGraph[a][b] + +$;
$pairwiseElectionGraph[b][a] - -$;
$globalIndexA := index(a)$ `// index of a in the global ranking`
$globalIndexB := index(b)$ `// Index of b in the global ranking`
$edgeWeightAB := pairwiseElectionGraph[a][b]$;
$disagree := globalIndexA > globalIndexB$; `// vote` v_{ab} `disagrees with the global ranking`
if *disagree and* $(edgeWeightAB = 0$ *or* $edgeWeightAB = 1)$ **then**
    `// global ranking: Set1 | b | Set2 | a | Set3`
    `// edgeWeightAB = 0: forward edge b -> a disappeared`
    `// edgeWeightAB = 1: backward edge a -> b appeared`
    $counterA := 0$;
    $counterB := 0$;
    $Set2 := globalRanking[index(b) : index(a)]$; `// nodes between` b `and` a `in global ranking`
    foreach $n \in Set2$ **do**
        `// forward edge would become backward edge -> increment counter`
        `// backward edge would become forward edge -> decrement counter`
        if $pairwiseElectionGraph[n][a] > 0$ **then**
            $counterA + +$;
        else if $pairwiseElectionGraph[n][a] < 0$ **then**
            $counterA - -$;
        if $pairwiseElectionGraph[b][n] > 0$ **then**
            $counterB + +$;
        else if $pairwiseElectionGraph[b][n] < 0$ **then**
            $counterB - -$;
        if $counterA > edgeWeightAB$ *and* $counterB > edgeWeightAB$ **then**
            return *globalRanking*; `// no improvement in changing a or b`
    $indexNewA := globalIndexB - 1$;
    $indexNewB := globalIndexA + 1$;
    $minA := counterA$;
    $minB := counterB$;
    `// Execute Algorithm 2 here for move to edge variant`
    if $minA \leq edgeWeightAB$ *and* $minB \geq edgeWeightAB$ **then**
        move a to $indexNewA$ in *globalRanking*;
    else if $minA \geq edgeWeightAB$ *and* $minB \leq edgeWeightAB$ **then**
        move b to $indexNewB$ in *globalRanking*;
    else
        swap a and b int *globalRanking*;
return *globalRanking*;

Algorithm 2: SLATER MOVETOEDGE

```
1  Set1 := globalRanking[0 : index(b) − 1];
2  Set3 := globalRanking[index(a) + 1 : globalRanking.length];
   // find final place for a
3  foreach n ∈ Set1 do
4      if pairwiseElectionGraph[n][a] > 0 then
5          counterA + +;
6      else if pairwiseElectionGraph[n][a] < 0 then
7          counterA − −;
8      if counterA ≤ minA then
9          indexNewA := index(n)              // index of n in the global ranking
10         minA := counterA;
   // find final place for b
11 foreach n ∈ Set3 do
12     if pairwiseElectionGraph[b][n] > 0 then
13         counterB + +;
14     else if pairwiseElectionGraph[b][n] < 0 then
15         counterB − −;
16     if counterB ≤ minB then
17         indexNewB := index(n)              // index of n in the global ranking
18         minB := counterB;
```

5.1.3 Objective Cost and Runtime Analysis

Claim 5.1. A single update runs in $\mathcal{O}(|\mathcal{A}|)$

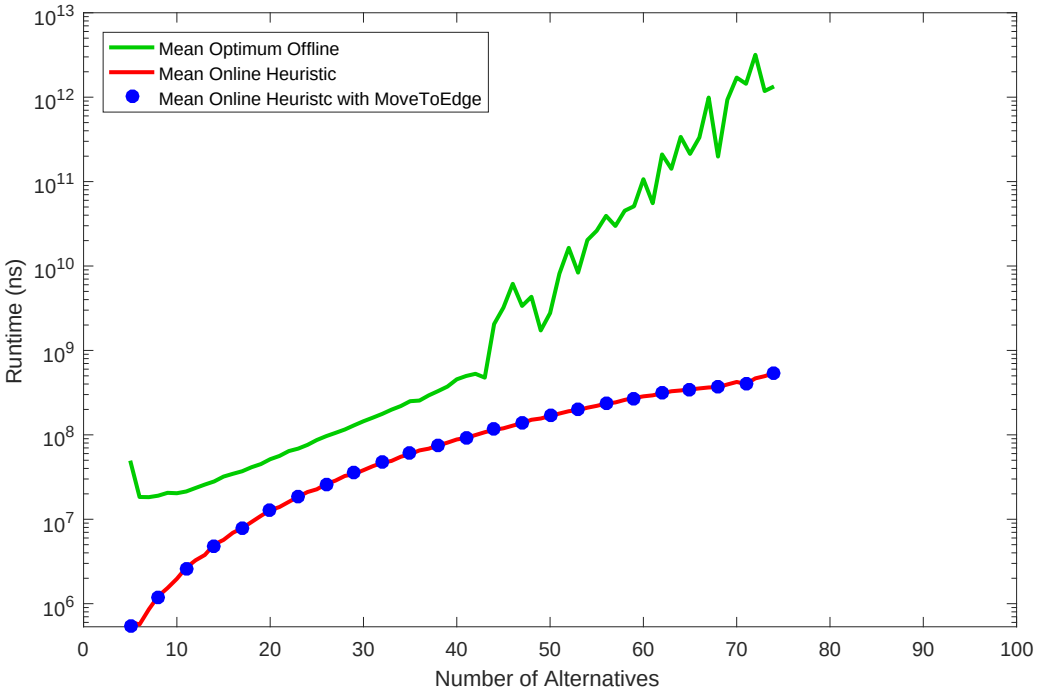
Proof. For a single update, i.e. an incoming single vote v_{ab} for two alternatives a and b, we need $\mathcal{O}(|\mathcal{A}|)$ to find the position of a in the global ranking and $\mathcal{O}(|\mathcal{A}|)$ to find the position of b. The iteration through S_2 in algorithm 1 and the iteration through the whole global ranking (S_1, S_2, S_3) in Algorithm 1 including Algorithm 2 takes $\mathcal{O}(|\mathcal{A}|)$. The possible position change of a,b or both has also a linear runtime regarding to the number of alternatives. Therefore we have a total runtime of $\mathcal{O}(|\mathcal{A}|)$. □

Claim 5.2. The update of a voting set $\mathcal{V}_\mathcal{A}$ runs in $\mathcal{O}(|\mathcal{V}_\mathcal{A}| \cdot |\mathcal{A}|^3)$

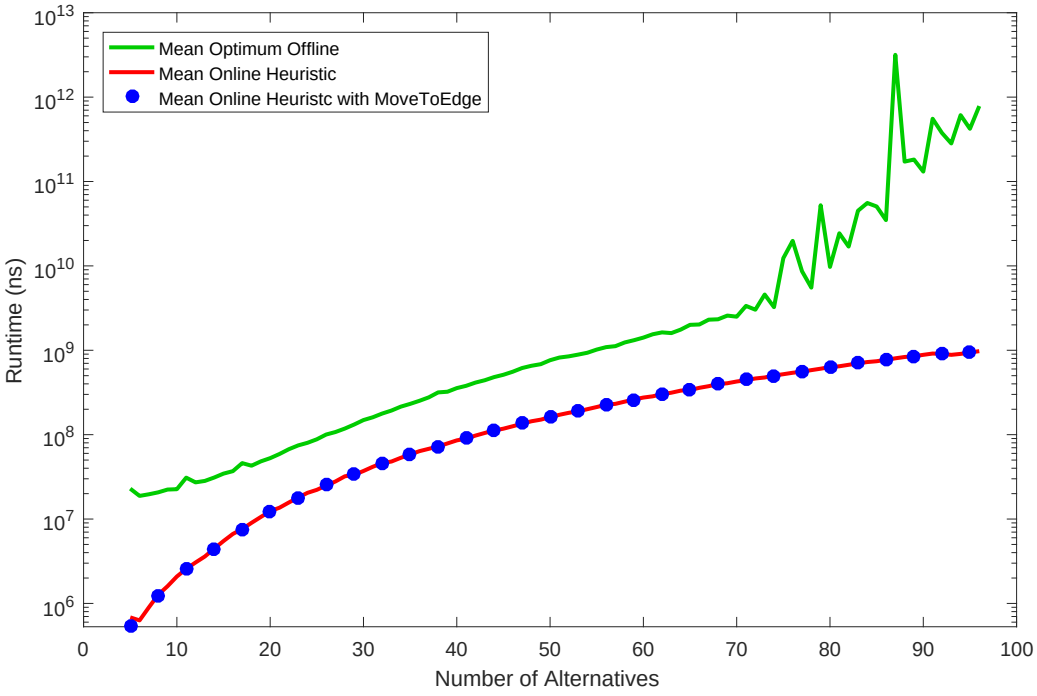
Proof. A complete vote $v \in \mathcal{V}_\mathcal{A}$ consists of $\mathcal{O}(|\mathcal{A}|^2)$ comparisons between the alternatives. Thus we have a total runtime of $\mathcal{O}(|\mathcal{A}|^2) \cdot \mathcal{O}(|\mathcal{A}|) = \mathcal{O}(|\mathcal{A}|^3)$ for updating v. Since there are $|\mathcal{V}_\mathcal{A}|$ complete votes, we get a runtime of $\mathcal{O}(|\mathcal{V}_\mathcal{A}| \cdot |\mathcal{A}|^3)$. □

Figure 5.3 reflects our asymptotic runtime analysis. It shows the runtime of the optimal offline algorithm and the two online approaches. One can see that the optimal offline algorithm increases rapidly after a certain number of candidates and it depends on the consensus probability how high this number is. In Figure 5.3a and 5.3b we can see this behaviour. Note that the test runs for the lower consensus probability includes fewer alternatives as we reach the point earlier where the runtime increases rapidly.

The test runs have shown how well the first and the second approach competes with the optimal offline solution regarding the objective cost. Figure 5.4 shows the objective cost of the two approaches and the optimal offline cost. For each number of alternatives we tested twenty different models, each with 1000 rankings. Figure 5.4a shows it for a consensus probability of 0.6 and Figure 5.4 for 0.8. We can see that the cost for higher consensus probability is lower, as the votes are similar and we therefore can find a better global ranking representing all the votes. On average our heuristics are just 2–3 percent worse than the optimal regarding the objective cost.

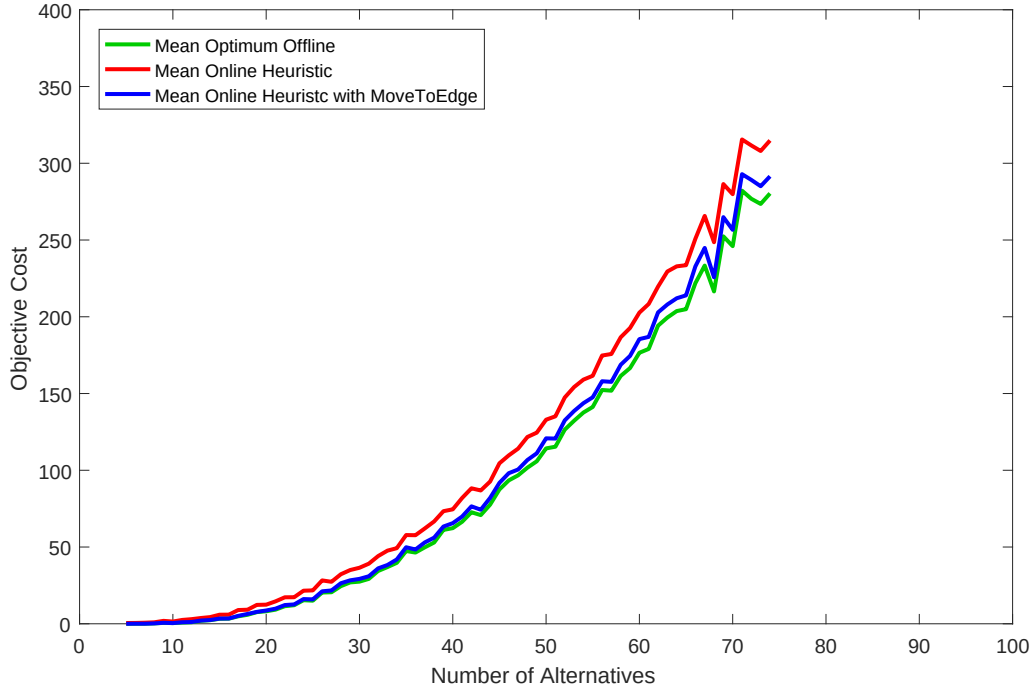


(a) Consensus probability 0.6

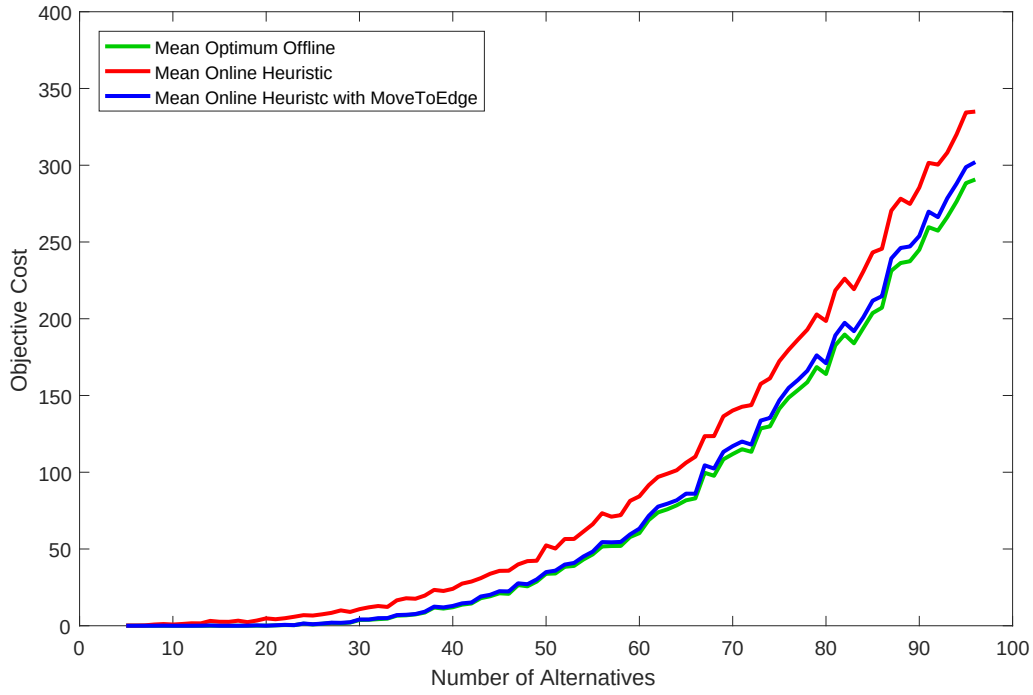


(b) Consensus probability 0.8

Figure 5.3: Runtime in nanoseconds of the optimum offline solution against our two online heuristics regarding the Slater welfare function. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings. The consensus probability for (a) is 0.6 and for (b) 0.8.



(a) Consensus probability 0.6



(b) Consensus probability 0.8

Figure 5.4: Objective cost of the optimum offline solution against our two online heuristics regarding the Slater welfare function. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings. The consensus probability for (a) is 0.6 and for (b) 0.8.

5.2 Kemeny

The Kemeny welfare function (Def. 2.5) not only tries to minimize the amount of backward edges in the pairwise election graph over a given ranking as Slater does, it also minimizes the sum of the weights of all backward edges. That means the objective cost changes for every new incoming single vote. Our heuristic is therefore similar to the one for Slater, but differs regarding when the update method is invoked and how we increment the counters.

5.2.1 Online Heuristic

For a new incoming single vote v_{ab} we only try to move an alternative a or b if the single vote disagrees with the current global ranking similarly to Algorithm 1 for the Slater welfare function. For agreeing single votes we either increment forward edges, which does not affect the objective cost or we decrement backward edges, which would result in a lower objective cost and hence improve our result. Since the Kemeny rule minimizes over weights, we are not only interested in whether a new backward edge appears or an existing forward edge disappears, but also if the weight of a backward edge increases or the weight of a forward edge decreases. This is the case for every new considered single vote disagreeing with the ranking for which we therefore invoke the update method in Algorithm 3. The following cases arise:

Case 1 We increment the weight of a backward edge (see Figure 5.5). It does not matter if the backward edge newly appears or if an already existing backward edge increases, as in both cases the weight is incremented by one. Let S_2 be the set of alternatives between a and b as shown in Figure 5.5 and let $\Sigma_{a_{back}}$ be the sum of the weights of all edges that would become backward edges by moving a directly in front of b and $\Sigma_{a_{for}}$ the sum of the weights of all edges that would become a forward edge by moving a directly in front of b. Similarly, we have $\Sigma_{b_{back}}$ and $\Sigma_{b_{for}}$ for b. Now a and b are moved according to exactly one of the following subcases: (If none of the following cases occurs we will leave the global ranking as it is)

- **Case 1.1** If $(\Sigma_{a_{back}} - \Sigma_{a_{for}}) + (\Sigma_{b_{back}} - \Sigma_{b_{for}}) \leq w_{ab}$ we will move a and b. Here w_{ab} is simply the weight of the edge from a to b.
- **Case 1.2** If $(\Sigma_{a_{back}} - \Sigma_{a_{for}}) \leq (\Sigma_{b_{back}} - \Sigma_{b_{for}})$ and $(\Sigma_{a_{back}} - \Sigma_{a_{for}}) \leq w_{ab}$ we will only move a before b.
- **Case 1.3** If $(\Sigma_{b_{back}} - \Sigma_{b_{for}}) \leq (\Sigma_{a_{back}} - \Sigma_{a_{for}})$ and $(\Sigma_{b_{back}} - \Sigma_{b_{for}}) \leq w_{ab}$ we will only move b behind a.

Case 2 A forward edge is decremented (see Figure 5.6). Again there is no difference if the weight of the forward edge is decremented or if the edge disappears. As in Case 1 we go through S_2 to calculate $\Sigma_{a_{for}}$, $\Sigma_{a_{back}}$, $\Sigma_{b_{for}}$ and $\Sigma_{b_{back}}$. The difference is now that we have an edge from b to a with weight w_{ba} which would become a backward edge. So the sum of weights of edges changing from a backward edge to a forward edge must be greater or equal than the sum of weights of edges changing from forward to backward plus the weight w_{ba}. Since we have directed edges and $w_{ba} = -w_{ab}$, the moving behaviour is equally to Case 1.1, Case 1.2 and Case 1.3.

Algorithm 3 which optimizes according to the Kemeny rule also works with the "moveToEdge" extension (explained in Section 5.1.2 for the Slater heuristic). We again go through

S_1 and S_3 to find the position of a and b where the objective cost is minimal and move the alternatives similar to the above cases (Case 1.1 - Case 2.3).

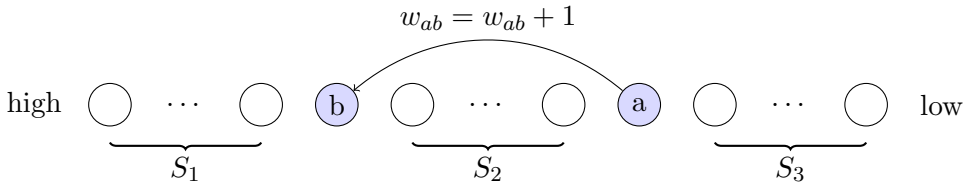


Figure 5.5: Incrementing Backward Edge

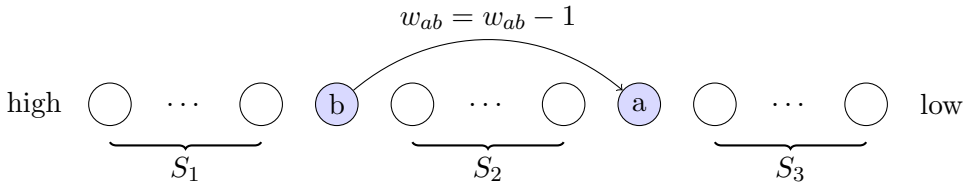


Figure 5.6: Decrementing Forward Edge

5.2.2 Runtime and Cost Analysis

The asymptotic runtime of Algorithm 3 is $\mathcal{O}(|\mathcal{V}_\mathcal{A}| \cdot |\mathcal{A}|^3)$. This can easily be shown as the only difference to Algorithm 1 is that the counter variables increment the weights instead of one, zero and minus one. As we perform an update for each single vote disagreeing with the current global ranking, the performance in practice is slightly worse. Compared to the optimal offline algorithm, our online heuristic is again clearly faster. Figure 5.7 shows the runtime of the optimal offline algorithm (Solver for linear program) and the two online heuristics presented above. Similarly to Slater we can see that the runtime of the offline algorithm starts to increase rapidly after a certain number of alternatives and that it depends on the consensus probability.

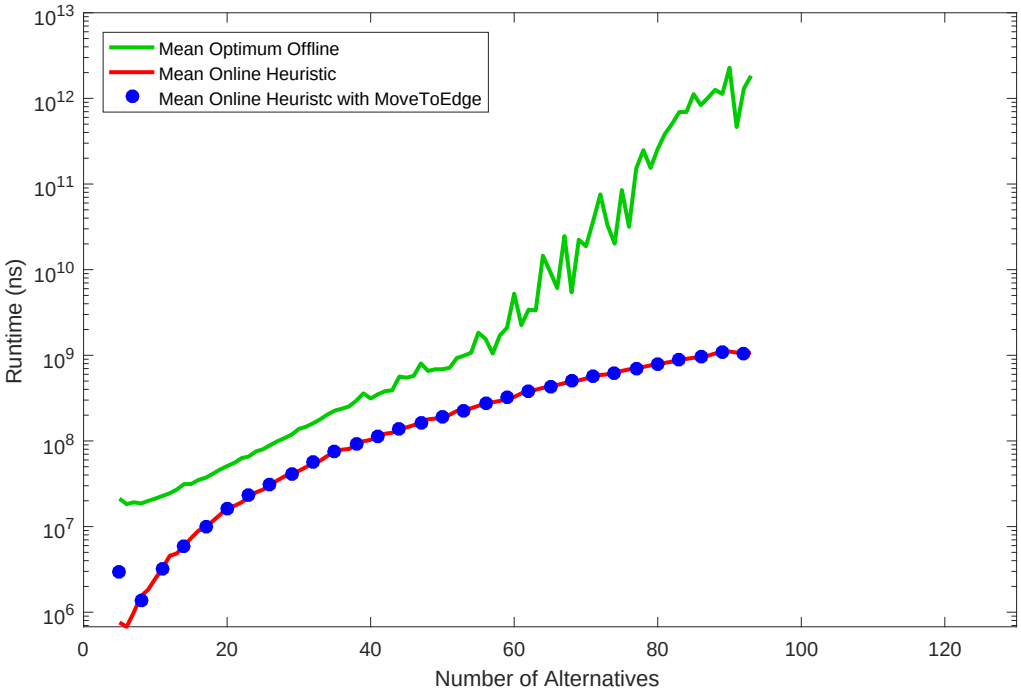
Figure 5.8 shows the objective cost of the presented approaches and the optimum offline case regarding the Kemeny function. In contrast to Algorithm 1 for the Slater function, the "moveToEdge" extension does not improve our cost in the Kemeny approach in a significant manner. One of the reasons for this behaviour is that Algorithm 3 better identifies trends due to the additional information given through the weights and the increased number of times where the update method is called. Our tests have shown that Algorithm 3 does not tend to fall in poor local minima and therefore does not need the "moveToEdge" extension to balance it out.

Algorithm 3: KEMENY ONLINE: SINGLE UPDATE

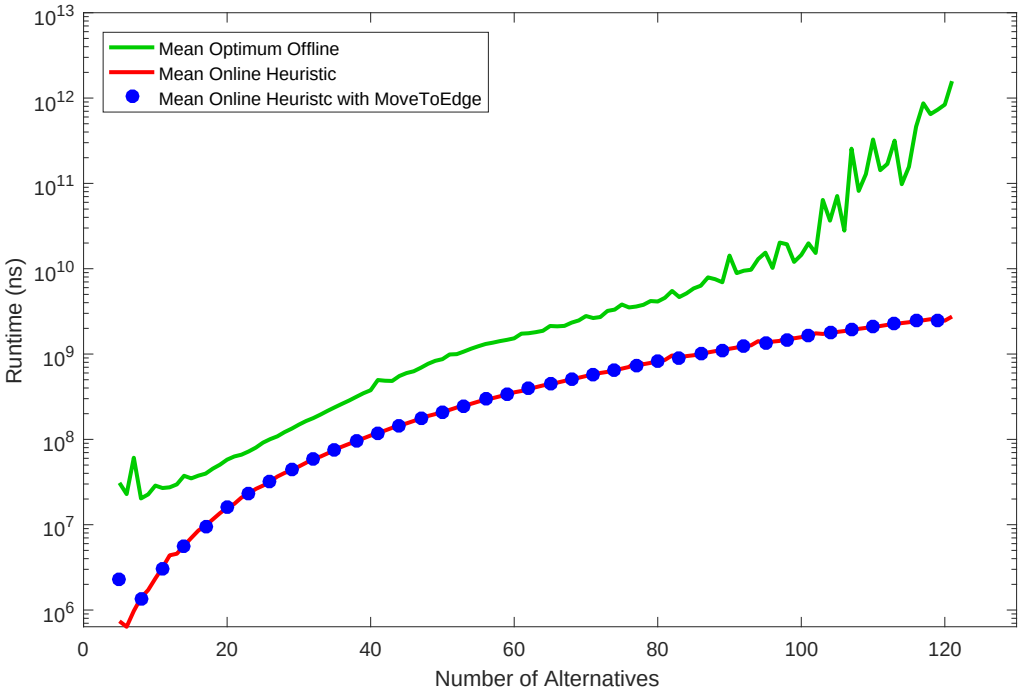
Input : *globalRanking* as 1D array, storing the currently global ranking
pairwiseElectionGraph as 2D matrix, storing the weight of the edges between alternatives according to previous updates
(a, b), a new single vote $v_{ab} = (a \succ b)$

Output: updated *globalRanking*

```
1  globalIndexA := index(a)                    // index of a in the global ranking
2  globalIndexB := index(b)                    // Index of b in the global ranking
3  pairwiseElectionGraph[a][b] + +;
4  pairwiseElectionGraph[b][a] − −;
5  if globalIndexA > globalIndexB then
6      weightAB := pairwiseElectionGraph[a][b];
7      counterA := 0;
8      counterB := 0;
9      Set2 := globalRanking[index(b) : index(a)];      // Set1 | b | Set2 | a | Set3
10     foreach n ∈ Set2 do
           // forward edge would become backward edge -> increment counter
           // backward edge would become forward edge -> decrement counter
           // pairwiseElectionGraph[i][j] > 0:  edge from i to j
           // pairwiseElectionGraph[i][j] < 0:  edge from j to i
11         counterA := counterA + pairwiseElectionGraph[n][a];
12         counterB := counterB + pairwiseElectionGraph[b][n];
13     if counterA > weightAB and counterB > weightAB then
14         return globalRanking;                    // no improvement in changing a or b
15     indexNewA := globalIndexB − 1;
16     indexNewB := globalIndexA + 1;
17     minA := counterA;
18     minB := counterB;
19     if moveToEdge then
20         Set1 := globalRanking[0 : index(b) − 1];  // nodes from index 0 to index of
               b - 1
21         Set3 := globalRanking[index(a) + 1 : globalRanking.length − 1];
           // find final place for a
22         foreach n ∈ Set1 do
23             counterA := counterA + pairwiseElectionGraph[n][a];
24             if counterA ≤ minA then
25                 indexNewA := index(n);           // index of n in the global ranking
26                 minA := counterA;
           // find final place for b
27         foreach n ∈ Set3 do
28             counterB := counterB + pairwiseElectionGraph[b][n];
29             if counterB ≤ minB then
30                 indexNewB := index(n);           // index of n in the global ranking
31                 minB := counterB;
32     if minA + minB ≤ weightAB then
33         swap a and b in globalRanking;
34     else if minA ≤ minB then
35         move a to indexNewA in globalRanking;
36     else
37         move b to indexNewB in globalRanking;
38 return globalRanking;
```

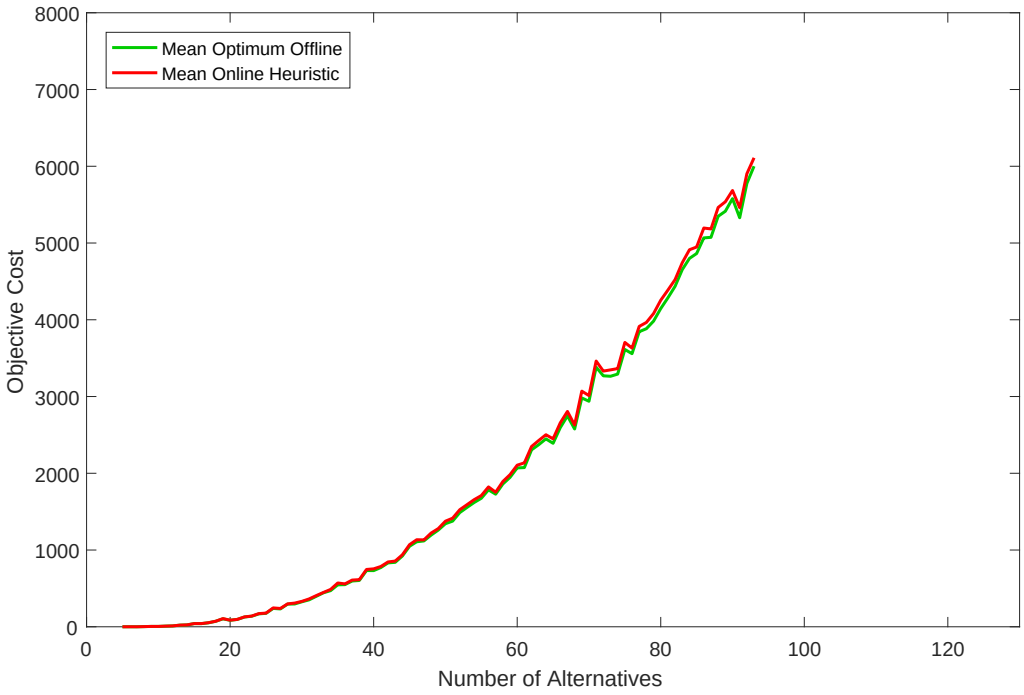


(a) Consensus probability 0.6

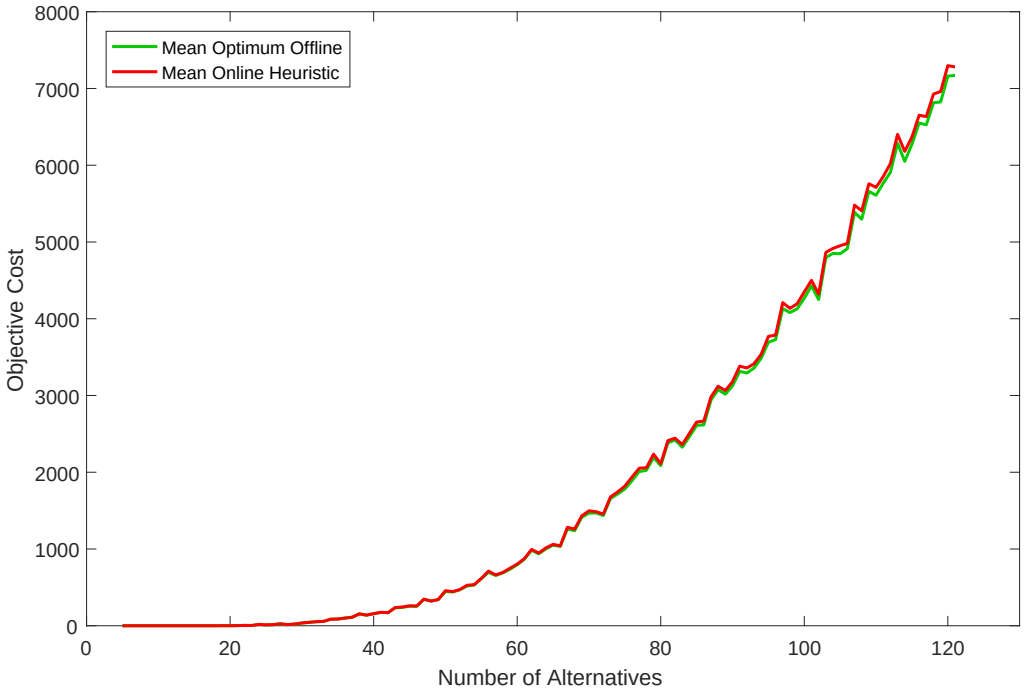


(b) Consensus probability 0.8

Figure 5.7: Runtime in nanoseconds of the optimum offline solution against our two online heuristics regarding the Kemeny welfare function. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings. The consensus probability for (a) is 0.6 and for (b) 0.8.



(a) Consensus probability 0.6



(b) Consensus probability 0.8

Figure 5.8: Objective cost of the optimum offline solution against our online heuristic regarding the Kemeny welfare function. Note that we omitted the online heuristic with the "moveToEdge" extension in the plot, since the line is almost identically to the normal online heuristic. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings. The consensus probability for (a) is 0.6 and for (b) 0.8.

5.3 FLAP

The FLAP welfare function (Def. 2.6)minimizes the length times the weight of backward edges. Similar to our Kemeny approach, we invoke the update method each time a single vote arises. In contrast to our Slater and Kemeny approach we cannot simply count edges or their weights respectively, since the objective cost is also determined by the length of the edges and we have to carry the possible new position of the alternatives. A first observation is that the "moveToEdge" extension does not fit well in our problem as we do not know the final position of a and b in this approach. It is therefore unhandy to keep track of each possible new position of a and b. Furthermore we have seen that if we have more information such as the weight in the Kemeny approach the "moveToEdge" extension does not yield significantly better results.

5.3.1 Online Heuristic

We again have a new single vote v_{ab} and the set S_2 of alternatives between a and b in the current global ranking. Let a' be the node a placed in front of b in the possible new global ranking, i.e. the the index of a' is the index of b - 1 with all the edges from and to a, and let b' be the node b placed behind a in the possible new global ranking respectively (index of b' is therefore $a + 1$). Figure 5.9 shows an example for moving a. Lets define the cost of an edge between two arbitrary nodes a_1 and a_2 as follow:

$$\tau_{a_1a_2} := \begin{cases} w_{a_1a_2} \cdot |i_{a_1} - i_{a_2}| & \text{if there is a directed edge from } a_1 \text{ to } a_2 \\ 0 \quad \text{otherwise} \end{cases} \tag{5.1}$$

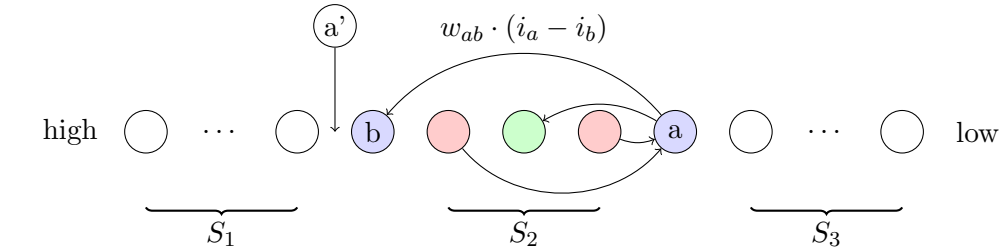
where $w_{a_1a_2}$ is the weight of the edge and i_{a_1},i_{a_2} are the indices of a_1 and a_2 in the global ranking. Then the cost for moving a directly in front of b is given as $\mathcal{C}_a := (\sum_{n \in S_2} \tau_{na'}) - (\sum_{n \in S_2} \tau_{an})$ and the cost for moving b directly behind a is $\mathcal{C}_b := (\sum_{n \in S_2} \tau_{b'n}) - (\sum_{n \in S_2} \tau_{nb})$. Note that if we have for example an edge from some $n_i \in S_2$ to a in the current global ranking, this forward edge would become a backward edge where its length is given through the new position of a, i.e the length is $|i_{a'} - i_{n_i}|$. On the other hand if a we have a backward edge in the global ranking from a to some $n_i \in S_2$, the edge would become a forward edge and we can decrement the objective cost by the weight times the old length of the edge. We only consider forward and backward edges from S_1 to a or b and forward and backward edges from a or b to S_1, as we only consider the cases of moving a directly in front of b or moving b directly behind a.

As in our approaches for Slater and Kemeny we leave the global ranking as it is if the new vote v_{ab} agrees with it and only increment the weight from a to b in our pairwise election graph. If v_{ab} disagrees with the current global ranking we have the following two cases:

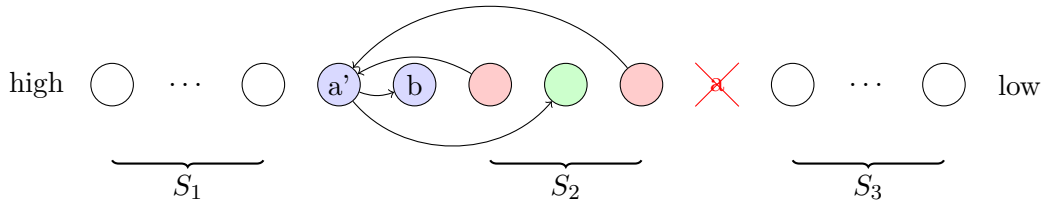
Case 1 We increment the weight of a backward edge. In this case we will move a in front of b if $\mathcal{C}_a \leq \tau_{ab}$ and $\mathcal{C}_a \leq \mathcal{C}_b$ or we move b behind a if $\mathcal{C}_b \leq \tau_{ab}$ and $\mathcal{C}_b < \mathcal{C}_a$. If none of the above holds we leave the ranking as it is.

Case 2 We decrement the weight of a forward edge and therefore move a in front of b if $\mathcal{C}_a + w_{ba} \leq 0$ or move b behind a if $\mathcal{C}_b + w_{ba} \leq 0$. Again if none of these holds we leave the ranking as it is

Note the difference between Case 1 and Case 2. In the first case we can decrease the objective function with the value τ_{ab} if we move one of the alternatives a or b, since we change



(a) Current global ranking



(b) Possible new global ranking by moving a in front of b

Figure 5.9: A new global ranking (b), where alternative a is moved in front of b, is preferred if the objective cost decreases. Forward edges e_{ia} for alternatives $i \in S_2$ in the current gloabl ranking (a) become backward edges with new length of $index(i) - index(a')$ and backward edges e_{ai} become forward edges with length $index(i) - index(a')$ respectively. The objective cost is therefore reduced by the cost of the "old" backward edges and increased by the "new" backward edges.

the backward edge in a forward edge. In the second case we decrease the cost of a forward edge (no changes in the objective cost) and therefore need costs $\mathcal{C}_a$, $\mathcal{C}_b$ which are negative and smaller than w_{ab} (absolute value is greater than w_{ba}). Notice that a negative cost will decrease our objective cost.

5.3.2 Runtime and Cost Analysis

Our online heuristic for FLAP runs as the previous heuristics in $\mathcal{O}(|\mathcal{V}_\mathcal{A}| \cdot |\mathcal{A}|^3)$. The additional calculation regarding the length for each node in S_2 takes constant time. The rest is similar to the proof for Claim 5.2.

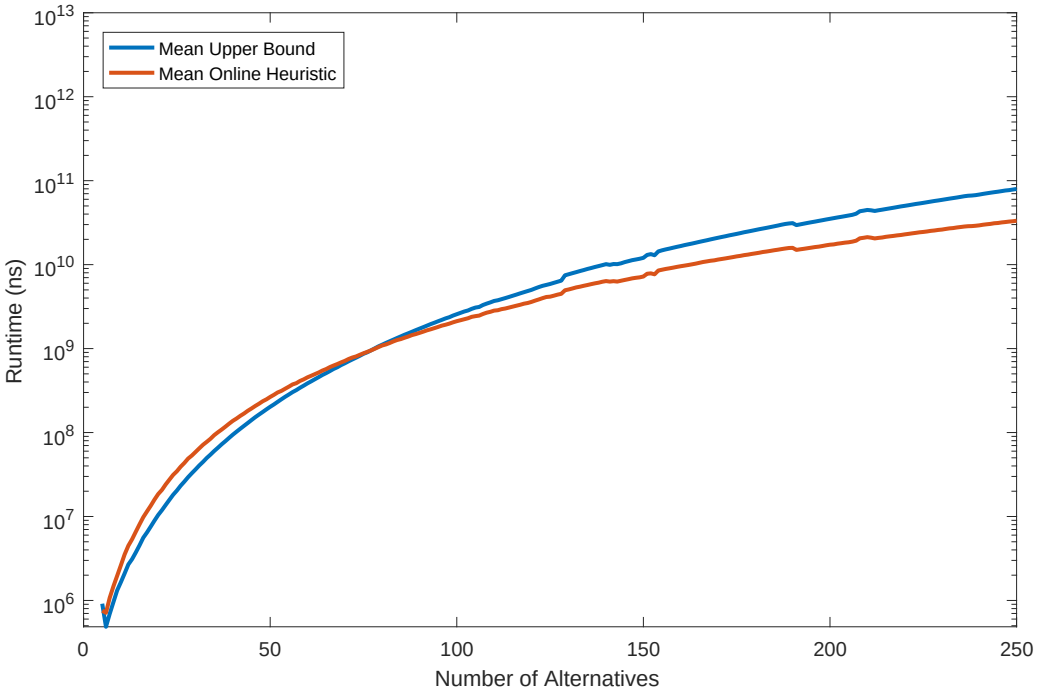
As we only can use the first version for the optimal offline solution (See Section 4.2), we compare our approach with a upper bound which calculates $(100 \cdot |\mathcal{A}|)$ random permutations over $\mathcal{A}$ and takes the permutation with the minimum objective cost for a given set of complete votes $\mathcal{V}_\mathcal{A}$ regarding the FLAP rule. The upper bound runs faster than our online heuristic but has a similar runtime for a number of alternatives up to 200 (Figure 5.10). In Figure 5.11 we can see that our presented online heuristic clearly stays below the upper bound. Figure 5.12 shows the objective cost and the runtime of our online heuristic against the optimum offline solution calculated with the first version of the ILP described in Section 4.1 up to 22 alternatives.

Algorithm 4: FLAP ONLINE: SINGLE UPDATE

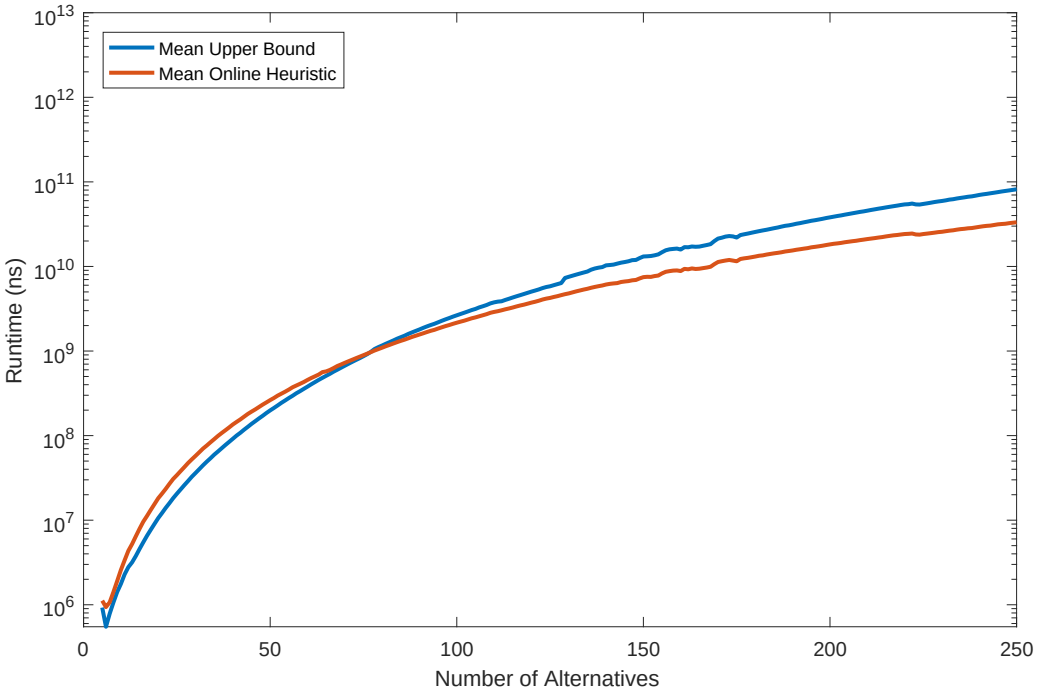
Input : *globalRanking* as 1D array, storing the currently global ranking
pairwiseElectionGraph as 2D matrix, storing the weight of the edges between alternatives according to previous updates
(a, b), a new single vote $v_{ab} = (a \succ b)$

Output: updated *globalRanking*

```
globalIndexA := index(a)                    // index of a in the global ranking
globalIndexB := index(b)                    // Index of b in the global ranking
pairwiseElectionGraph[a][b] + +;
pairwiseElectionGraph[b][a] − −;
disagree := globalIndexA > globalIndexB;    // True if vote v_ab disagrees with
                                               the global ranking
backwardEdgeInRanking := pairwiseElectionGraph[a][b] ≥ 0;
if disagree then
    if backwardEdgeInRanking then
        weightAB := pairwiseElectionGraph[a][b] · (globalIndexA − globalIndexB)
    else
        weightAB := pairwiseElectionGraph[a][b];        // weightAB is negative
    counterA := 0;
    counterB := 0;
    Set2 := globalRanking[index(b) : index(a)];     // Set1 | b | Set2 | a | Set3
    foreach n ∈ Set2 do
        if pairwiseElectionGraph[n][a] ≥ 0 then
            // plus new backward edge
            counterA := counterA
              +pairwiseElectionGraph[n][a] · ((index(n) + 1) − globalIndexB);
        else
            // minus old backward edge
            counterA := counterA
              +pairwiseElectionGraph[n][a] · (globalIndexA − index(n));
        if pairwiseElectionGraph[b][n] > 0 then
            // plus new backward edge
            counterB := counterB
              +pairwiseElectionGraph[b][n] · (globalIndexA − (index(n) − 1));
        else
            // minus old backward edge
            counterB := counterB
              +pairwiseElectionGraph[b][n] · (index(n) − globalIndexB);
    if counterA > weightAB and counterB > weightAB then
        return globalRanking;                   // no improvement in changing a or b
    indexNewA := globalIndexB − 1;
    indexNewB := globalIndexA + 1;
    if counterA ≤ counterB then
        move a to indexNewA in globalRanking;
    else
        move b to indexNewB in globalRanking;
return globalRanking;
```

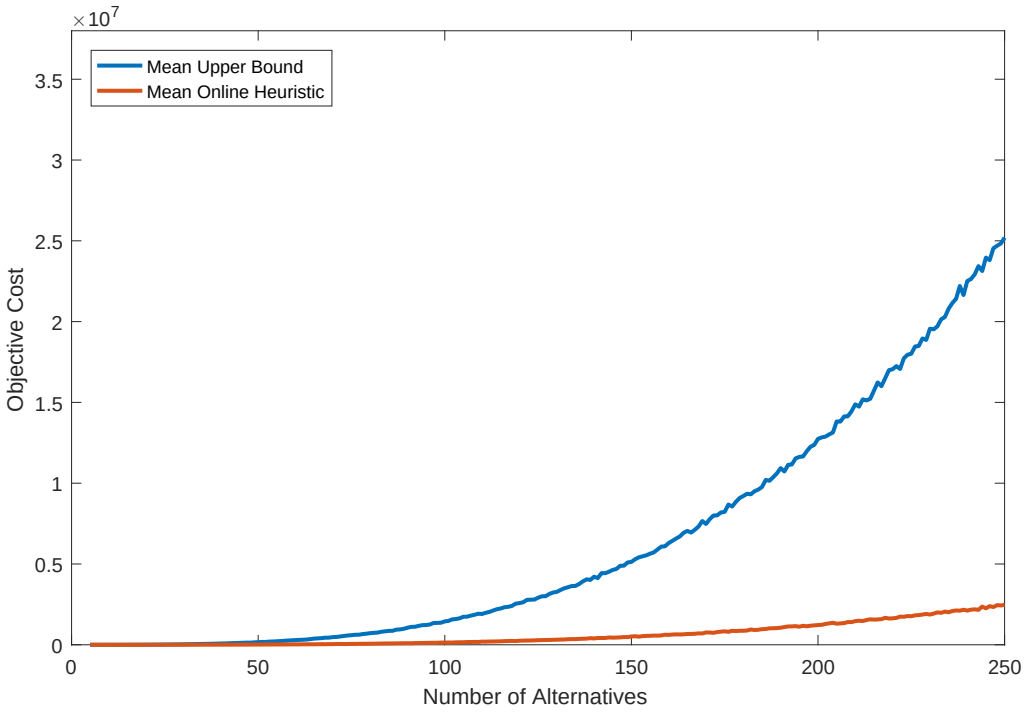


(a) Consensus probability 0.6

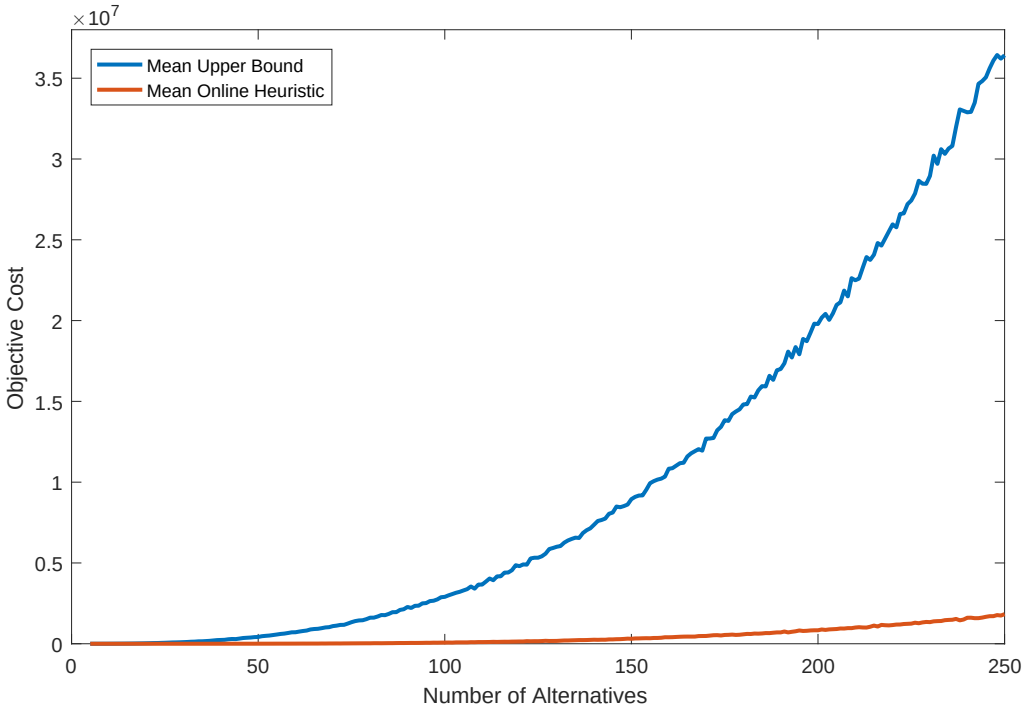


(b) Consensus probability 0.8

Figure 5.10: Runtime in nanoseconds of the upper bound against our online heuristic regarding the FLAP welfare function. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings. The consensus probability for (a) is 0.6 and for (b) 0.8.

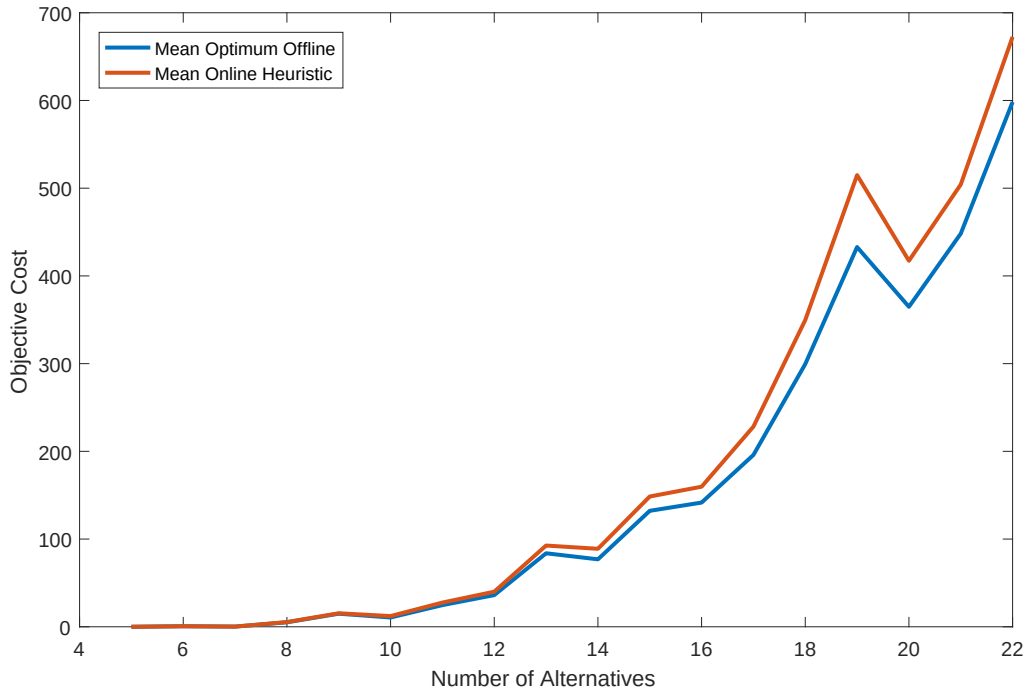


(a) Consensus probability 0.6

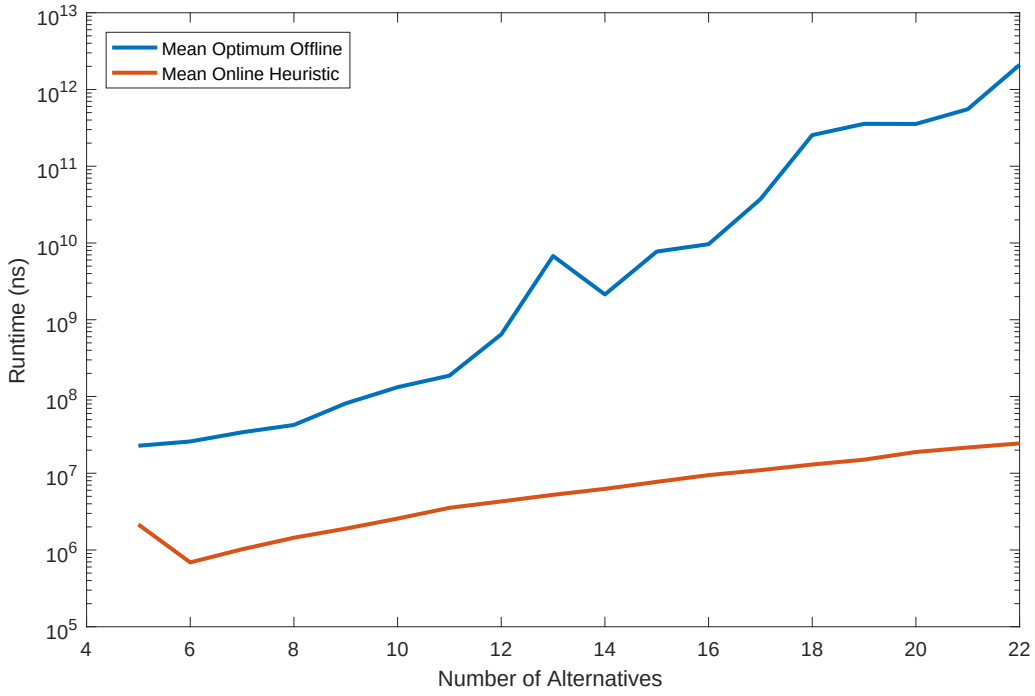


(b) Consensus probability 0.8

Figure 5.11: Objective cost of the upper bound against our online heuristic regarding the FLAP welfare function. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings. The consensus probability for (a) is 0.6 and for (b) 0.8.



(a) Objective Cost



(b) Runtime in nanoseconds

Figure 5.12: Optimum offline solution against our online heuristic regarding the FLAP welfare function. There are 20 different datasets per number of alternatives, each with a consensus probability of 0.6 and consisting of 1000 rankings.

5.4 Comparison between Social Welfare Functions

We have seen the performance for each of our online heuristics regarding the optimal offline solution. In the following section we describe how our approaches perform for the other welfare functions respectively. For a set of complete votes $\mathcal{V}_{\mathcal{A}}$ we calculate the global ranking for each approach separately and compare the cost they achieve for the other two welfare functions. In the following we refer to Algorithm 1 with the "moveToEdge" extension (Algorithm 2) as the Slater heuristic, Algorithm 3 without the "moveToEdge" extension as Kemeny heuristic and Algorithm 4 as FLAP heuristic.

5.4.1 Simulation

Figure 5.13 shows the objective cost for each approach regarding the Slater welfare function. We can see that all approaches perform similarly but our online heuristic for Slater has the lowest objective cost. In Figure 5.14 and 5.15 we see the objective cost regarding the Kemeny and the FLAP welfare function. One can see that the online heuristics for Kemeny and FLAP perform similarly whereas the online heuristic for Slater performs slightly worse. This follows from the fact that our Slater approach only optimizes over backward edges and not their respective weights.

Although the asymptotic runtime of all online heuristics is the same, there is a runtime difference in practise. Our online heuristic for Slater has the fastest runtime (See Figure 5.16) since it only updates the global ranking if a new backward edge appears or an existing forward edge disappears.

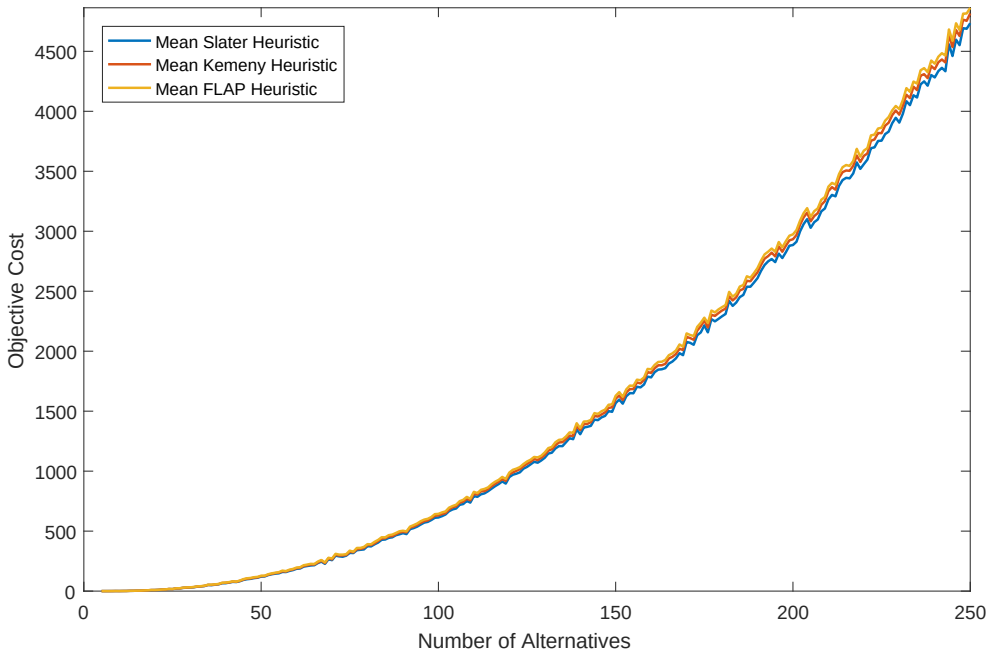


Figure 5.13: Objective cost of the three online heuristics developed for Slater, Kemeny and FLAP, with respect to the Slater welfare function. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings and a consensus probability of 0.6

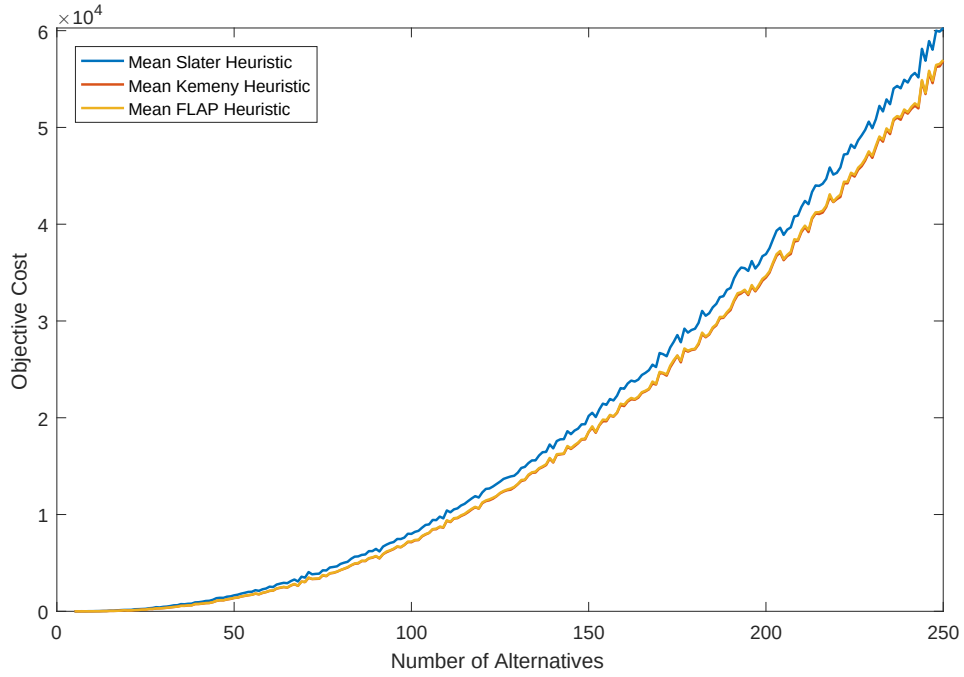


Figure 5.14: Objective cost of the three online heuristics developed for Slater, Kemeny and FLAP, with respect to the Kemeny welfare function. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings and a consensus probability of 0.6

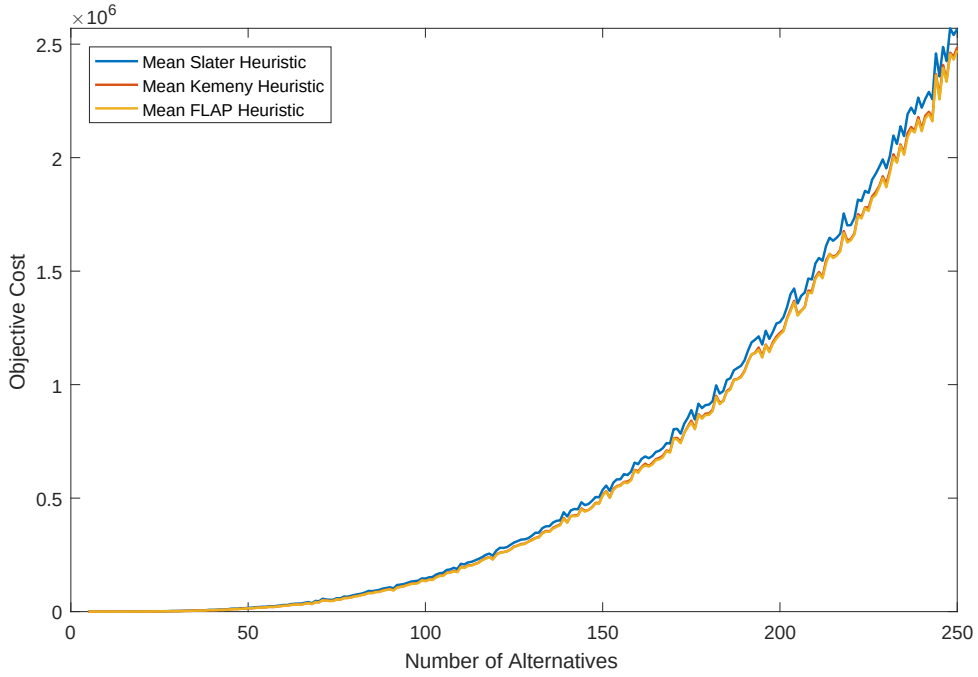


Figure 5.15: Objective cost of the three online heuristics developed for Slater, Kemeny and FLAP, with respect to the FLAP welfare function. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings and a consensus probability of 0.6

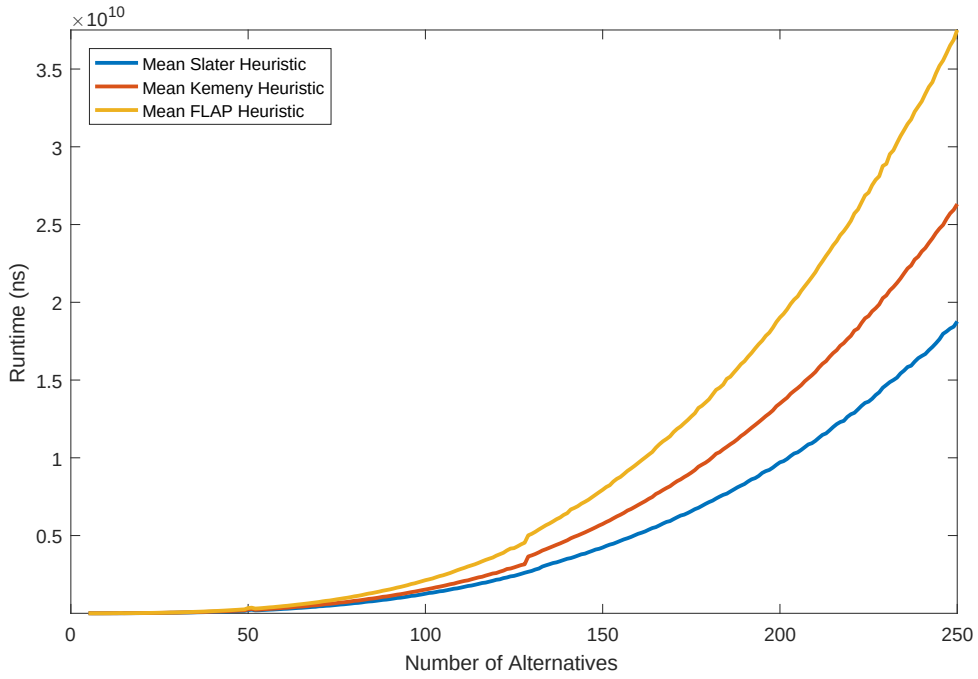


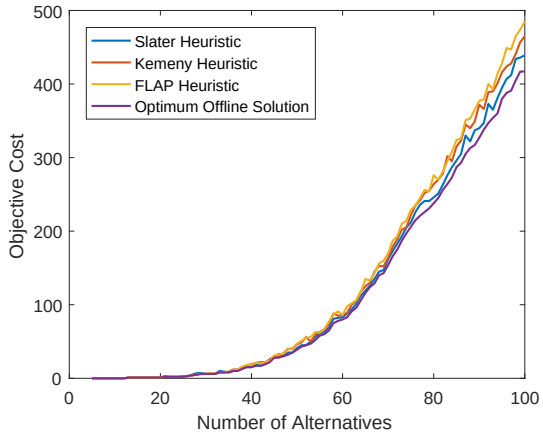
Figure 5.16: Runtime in nanoseconds of the three online heuristics for Slater, Kemeny and FLAP. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings and a consensus probability of 0.6

5.4.2 Real World Data

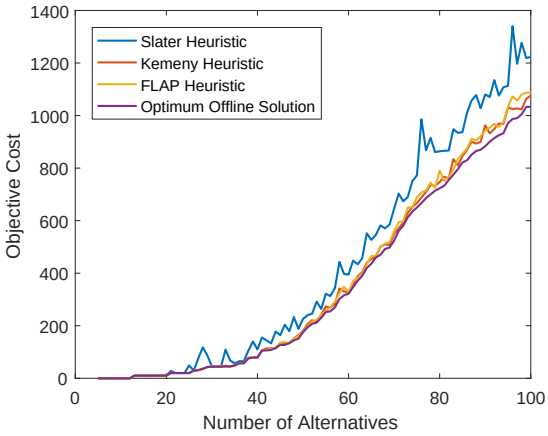
To test our online heuristic on real world data we use the "sushi" dataset described in Section 3.2. Figure 5.17 shows the objective cost and runtime of our online approaches and the optimum offline solution regarding Slater, Kemeny and FLAP. We can see that our online heuristics perform similarly on simulated data and real world data.

5.5 Constant Number of Alternatives

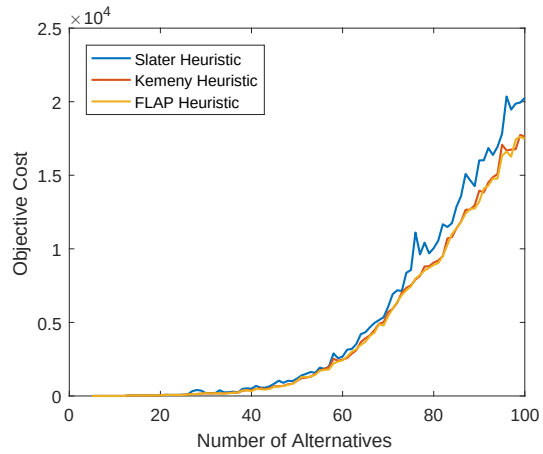
We have seen the performance of our online approaches and the optimum offline solutions according to different number of alternatives. One can ask the question how the global ranking behaves for incoming votes after we have already seen a number of votes. Our hypothesis is that after a number of votes the objective cost function of the global ranking becomes increasingly flat. In other words if we have seen enough votes representing the consensus of the voters, the objective cost will even out. Our test runs for Kemeny and FLAP (Figure 5.19 and 5.20) support our hypothesis and we can see that after we have seen enough complete votes, the cost function flattens out and even decreases a little bit for high consensus probabilities. In contrast to Kemeny and FLAP, the objective cost for Slater (Figure 5.18) looks differently but behaves as expected. Note that Slater minimizes over the number of backward edges and the cost therefore grows rapidly at the beginning regarding the maximum possible cost, if the first amount of complete votes disagree. This also happens for Kemeny and FLAP, however as the cost is not only calculated from the number of backward edges but also according the weight of the edges — which is very small for only a few votes — the objective cost stays small regarding



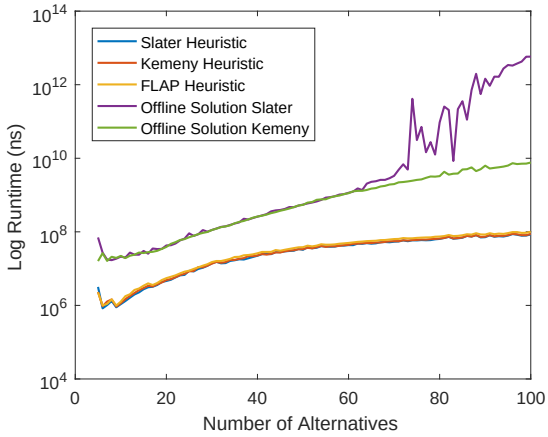
(a) Objective Cost regarding Slater



(b) Objective Cost regarding Kemeny



(c) Objective Cost regarding FLAP



(d) Runtime of all approaches

Figure 5.17: Result for the sushi dataset (Section 3.2) for a different number of alternatives: Figure (a) and (b) show the objective cost of our online heuristics and the optimal offline solution regarding the Slater and Kemeny welfare functions. Figure (c) shows the objective cost of our online heuristics with respect to the FLAP welfare function. Note that the upper bound for FLAP is omitted in figure (c) and (d) because of its worse order of magnitude. In Figure (d) we can see the runtime in nanosecond of our online heuristics and the optimum offline solutions for Slater and Kemeny.

the maximum objective cost of 1000 complete votes over a given set of alternatives. A second observation is the spreading between the different datasets, which is higher for a few complete votes and lower for a lot of complete votes. The reason is that the number of possible cases is much higher for only a few complete votes than it is for a lot of complete votes constructed under the same consensus probability.

Similar behaviour can be seen in Figure 5.21 showing the objective cost regarding the Slater and Kemeny welfare function on the "sushi" dataset (Section 3.2). We can see that our online heuristics produce nearly the same objective cost as the optimum offline solution.

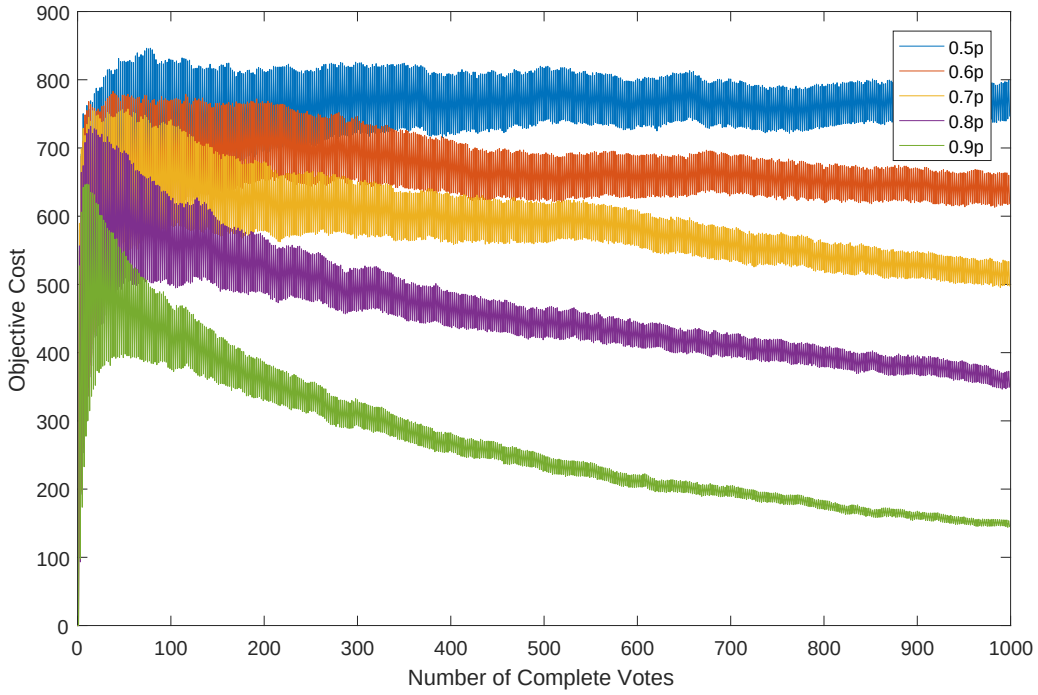


Figure 5.18: Objective cost of the online heuristic regarding the Slater welfare function for different consensus probabilities and a fixed amount of alternatives over 20 different datasets.

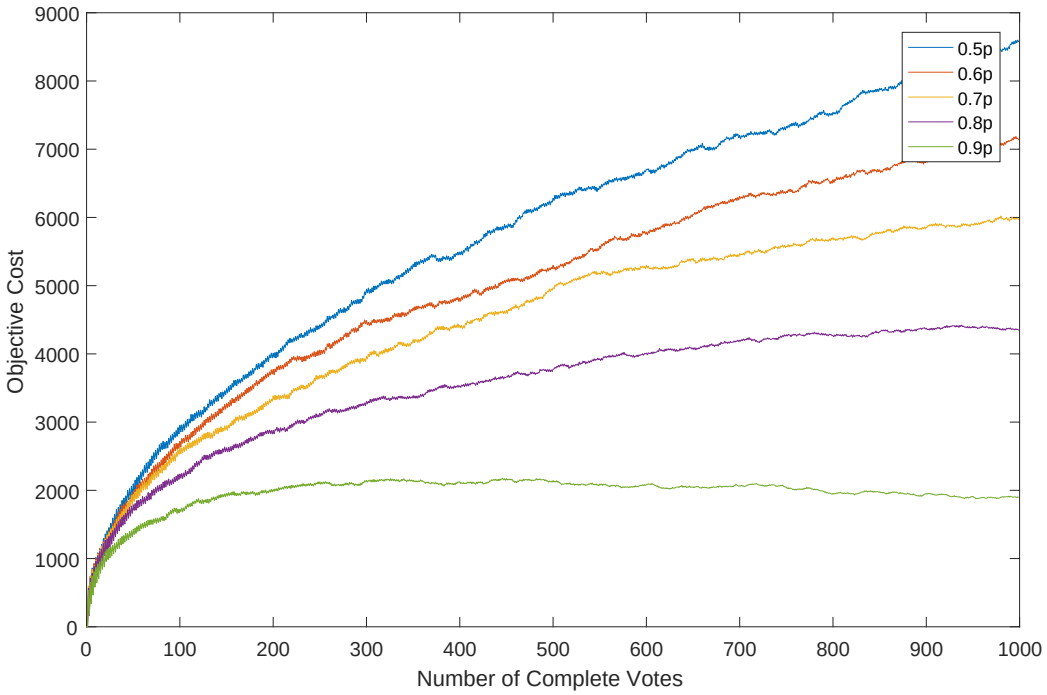


Figure 5.19: Objective cost of the online heuristic regarding the Kemeny welfare function for different consensus probabilities and a fixed amount of alternatives over 20 different datasets.

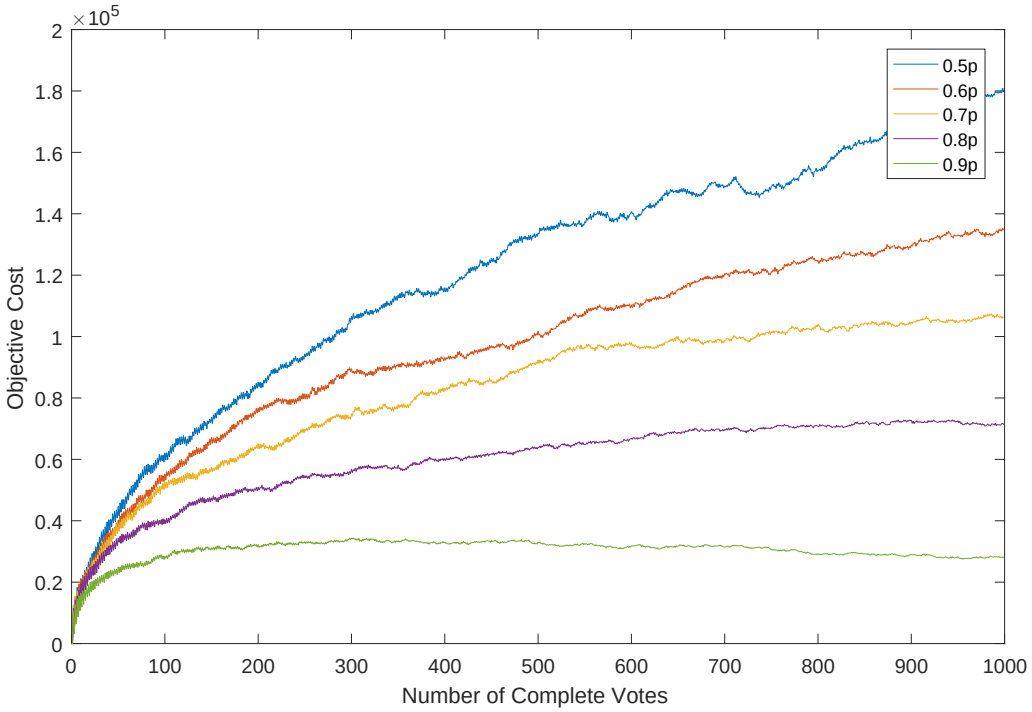
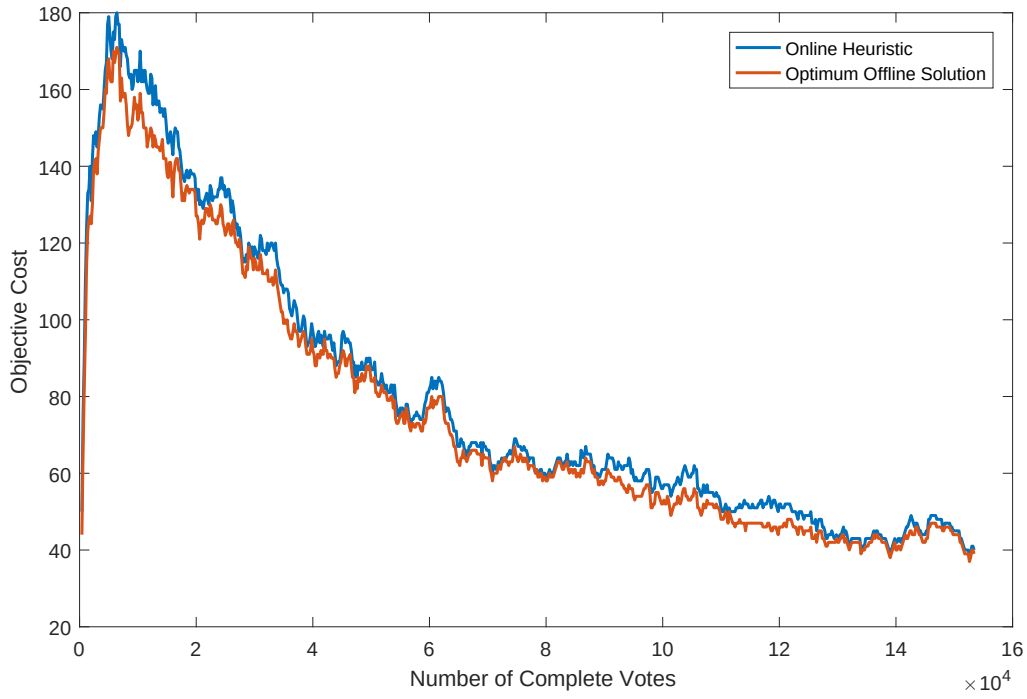
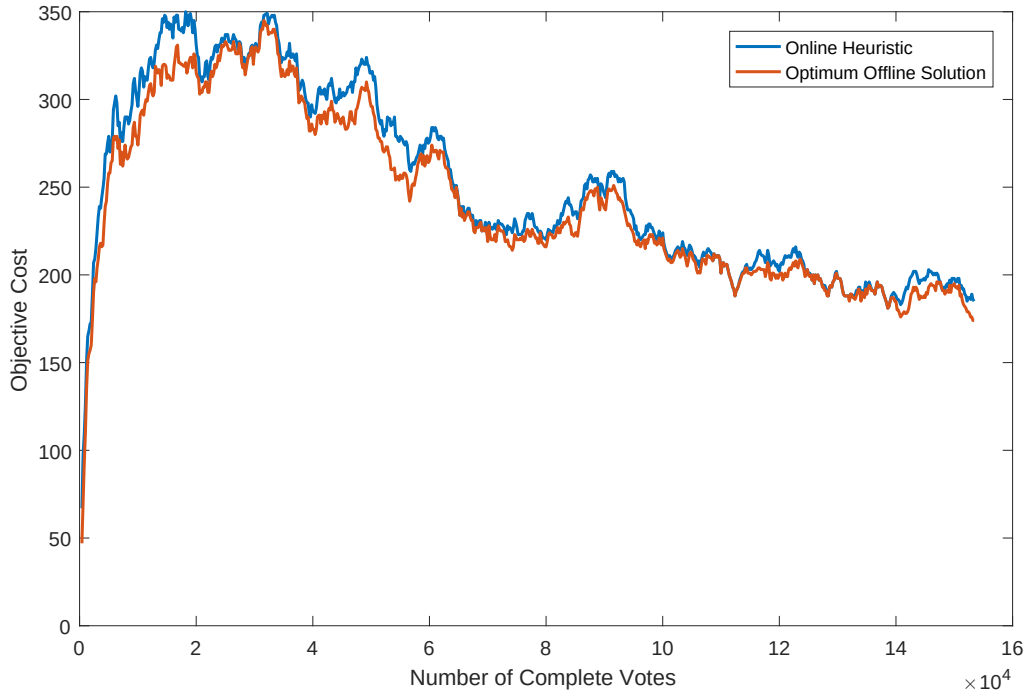


Figure 5.20: Objective cost of the online heuristic regarding the FLAP welfare function for different consensus probabilities and a fixed amount of alternatives over 20 different datasets.



(a) Slater welfare function

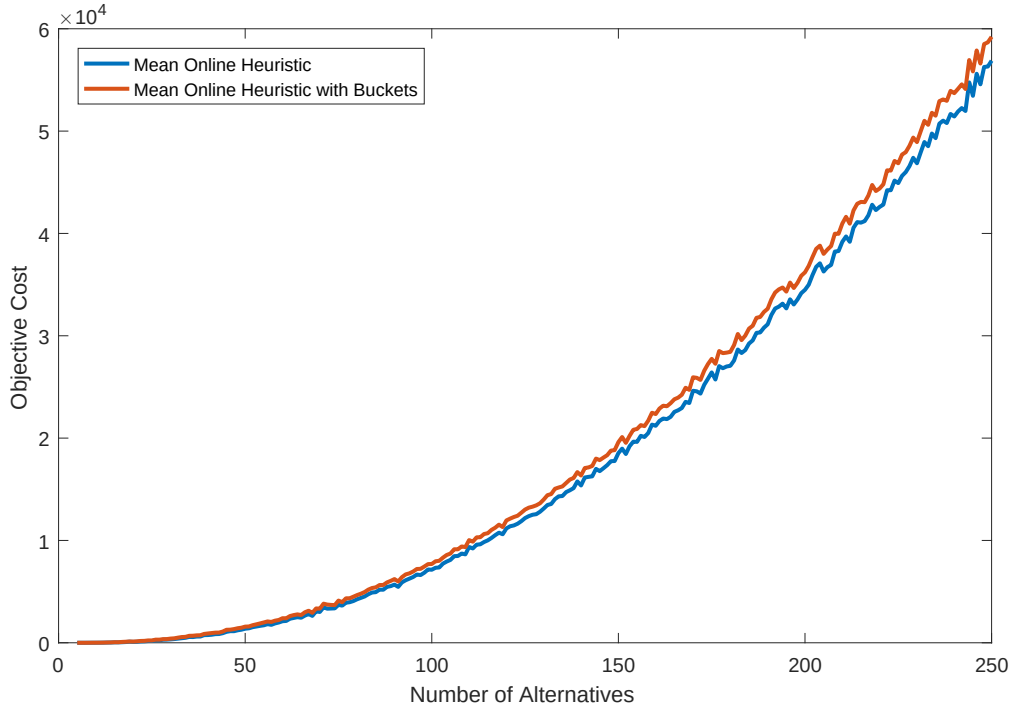


(b) Kemeny welfare function

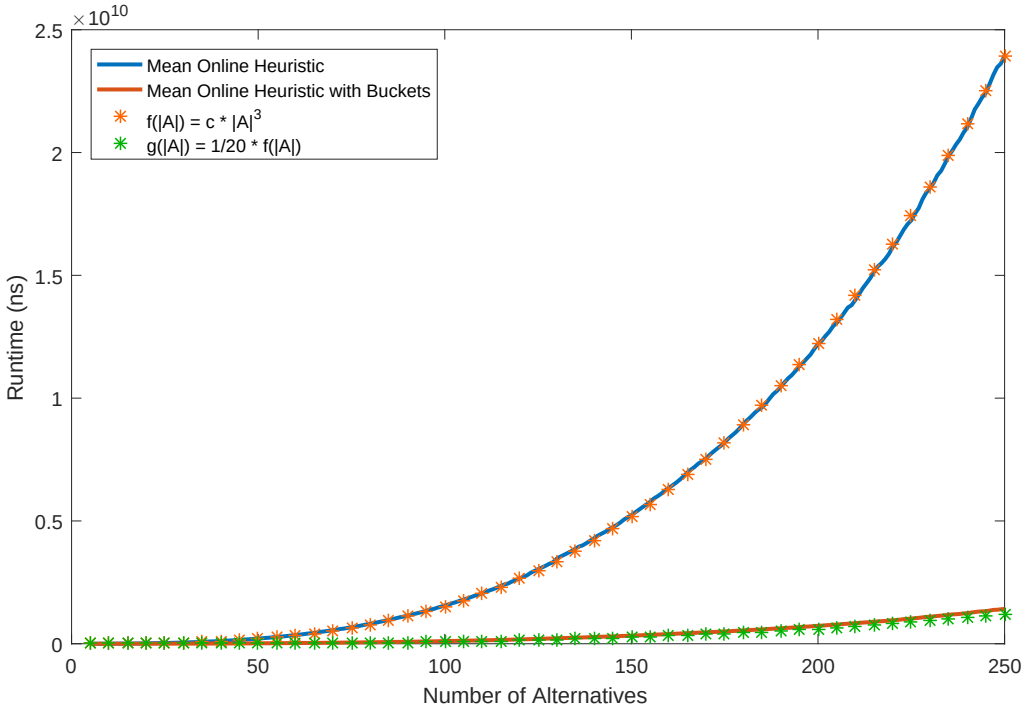
Figure 5.21: Objective cost of the online heuristic and the optimum offline solution regarding the Slater and Kemeny welfare function for the sushi dataset (Section 3.2) with a fixed number of alternatives $|\mathcal{A}| = 50$.

5.6 Bucket Updates

In practise when we have many single votes, one can imagine that a lot of similarities exists. We are therefore interested in how we can speed up our online algorithms by some constant factor. The idea is that we do not invoke the update method each time a single vote appears, instead we collect an amount of similar votes in a bucket and update it together. We have a bucket b_{ab} with size s for each possible single vote v_{ab} over $\mathcal{A}$. Test runs have shown that our so called "bucket"–approach produces similar results and can be used for all previous approaches. There is a tradeoff between runtime and the objective cost though, as if the bucket size is too large the cost function is getting worse. Figure 5.22 shows the objective cost and the runtime of our bucket approach with bucket size $s = 20$ against the normal online heuristics regarding the Kemeny social welfare function. As can be expected, the "bucket" approach is around twenty times faster than the normal online heuristic. In general, the bucket versions with size $s := |b_{ab}|$ are around s times faster than the approaches without buckets. This follows from the fact that we invoke the update method only every s-th time.



(a) Objective cost



(b) Runtime in nanoseconds

Figure 5.22: Objective cost and runtime of our online heuristic with and without the "bucket"–approach. There are 20 different datasets per number of alternatives, each consisting of 1000 rankings and a consensus probability of 0.6. In Figure (b) we have a reference function $f(|\mathcal{A}|) = c \cdot |\mathcal{A}|$ for some machine dependent constant c and a function $g(|\mathcal{A}|) = \frac{1}{s} \cdot f(|\mathcal{A}|)$, where $s = 20$ is the bucket size.

Chapter 6

Conclusion

In this thesis, we presented three main approaches for online heuristics regarding the Slater, Kemeny and FLAP welfare functions. All approaches update the global ranking according to new incoming votes efficiently. For the Slater and Kemeny welfare function we presented an additional approach which improved the online heuristic regarding Slater and performed equally well as the online heuristic for Kemeny. We ran several tests on different datasets, either generated by our model or constructed from real world data. Our online heuristics compare favourably to the optimum offline solutions and on average have only 2–3 percent higher objective costs.

The difference of the objective cost between the optimum solution and our online heuristics has the potential to be further reduced by future research. One could for example analyse if we obtain better results for the FLAP heuristic by moving a in the direction of b or vice versa, to obtain a position which minimizes the objective cost . Another potential direction would be a dynamic "backoff" mechanism, as we have seen that the objective cost of our global ranking will even out after seeing enough votes, representing the consensus of the voters. We could therefore update the global ranking only after a dynamic number of votes, which increases with the number of votes we have already seen.

Bibliography

[1] Colman, A.M., Pountney, I.: Borda's voting paradox: Theoretical likelihood and electoral occurrences. Behavioral Science **23**(1) (1978) 15–20

[2] Hudry, O.: On the complexity of Slater's problems. European Journal of Operational Research **203**(1) (May 2010) 216–221

[3] Kemeny, J.G.: Mathematics without numbers. Daedalus **88**(4) (1959) 577–591

[4] Bartholdi, J., Tovey, C.A., Trick, M.A.: Voting schemes for which it can be difficult to tell who won the election. Social Choice and Welfare **6**(2) (1989) 157–165

[5] Arrow, K.: Social Choice and Individual Values. Cowles Foundation Monographs Series. Yale University Press (1963)

[6] Pacuit, E.: Voting methods. In Zalta, E.N., ed.: The Stanford Encyclopedia of Philosophy. Winter 2012 edn. (2012)

[7] Satterthwaite, M.A.: Strategy-proofness and arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. Journal of Economic Theory **10**(2) (1975) 187–217

[8] Conitzer, V.: Computing slater rankings using similarities among candidates. In: Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA. (2006) 613–619

[9] Ali, A., Meilă, M.: Experiments with kemeny ranking: What works when? Mathematical Social Sciences **64**(1) (2012) 28 – 40 Computational Foundations of Social Choice.

[10] Mathieu, C., Vladu, A. In: Online Ranking for Tournament Graphs. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 201–212

[11] Davenport, A., Kalagnanam, J.: A computational study of the kemeny rule for preference aggregation. In: Proceedings of the 19th National Conference on Artifical Intelligence. AAAI'04, AAAI Press (2004) 697–702

[12] Conitzer, V., Davenport, A., Kalagnanam, J.: Improved bounds for computing kemeny rankings. In: Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1. AAAI'06, AAAI Press (2006) 620–626