Software implementations and analysis of three voting systems

Adrian Salamon

Kungsholmens gymnasium

Senior thesis

Supervised by:
Maja Kankaanranta

May 14, 2017

Abstract

Various voting methods are in use across the world. They range from simple processes that everyone can understand to complex mathematical systems. This essay examines differences and similarities of the behavior of three voting methods: the Single Transferable Vote (STV), first-past-the-post (FPTP), and Schulze Method. This is done by implementing each voting method in Typescript and then running election data into them. This paper provides descriptions, pseudocode and implementations of each method. A program to generate voting results was developed in order to test the voting methods. The methods are evaluated in both singe-seat and multi-seat elections. No revolutionary results are presented, but the behavior of each method is evaluated experimentally. Several conclusions are drawn about the behavior of the methods, including to what degree the different methods provided proportionate results.

Referat

Mjukvaruimplementationer och analys av tre valsystem
Flera valsystem används i världen. De kan vara allt ifrån enkla processer som alla kan förstå till komplexa matematiska system. Denna uppsats undersöker skillnader och likheter mellan tre valsystem: Enkel Överförbar Röst, Majoritetsval och Schulze-metoden. Detta genom att implementera alla valsystem i Typescript och testa de med olika valdata. Denna uppsats innehåller beskrivningar, pseudokod samt implementationer för alla valsystem. Ett program för att generera valdata utvecklades för att testa de olika systemen. Valsystemen testades i både enmans- och flermansval. Inga revolutionerande resultat presenteras, men varje systems beteende undersökts experimentellt. Flera slutsatser dras från de olika systemens beteende, inklusive hur proportionella resultat systemen producerade.

Contents

1 Introduction **5**
- 1.1 Background . . . 5
- 1.2 Purpose . . . 5

2 Theory **5**
- 2.1 Definitions . . . 6

3 Methodology **6**
- 3.1 Modeling test data . . . 7
- 3.2 Constructing test scenarios . . . 8
  - 3.2.1 Scenario 1 . . . 9
  - 3.2.2 Scenario 2 . . . 10
  - 3.2.3 Scenario 3 . . . 11
- 3.3 Metrics for evaluating results . . . 12
  - 3.3.1 Standard deviation . . . 13
  - 3.3.2 Misrepresentation . . . 13
  - 3.3.3 Most likely winners . . . 13

4 Specification of voting methods **14**
- 4.1 First-past-the-post . . . 14
  - 4.1.1 Description . . . 14
  - 4.1.2 Justification . . . 14
  - 4.1.3 Pseudocode . . . 14
  - 4.1.4 Implementation . . . 14
- 4.2 Single transferable vote . . . 15
  - 4.2.1 Description . . . 15
  - 4.2.2 Justification . . . 16
  - 4.2.3 Pseudocode . . . 16
  - 4.2.4 Implementation . . . 17
- 4.3 Schulze . . . 19
  - 4.3.1 Description . . . 19
  - 4.3.2 Justification . . . 20
  - 4.3.3 Pseudocode . . . 21
  - 4.3.4 Implementation . . . 21

5 Evaluation of results **23**
5.1 Scenario 1 23
5.2 Scenario 2 24
5.3 Scenario 3 25

6 Conclusion **26**

References **29**

Appendix A Code implementation **31**

1 Introduction

1.1 Background

Taking a collective decision as a population is difficult. To solve this issue, various voting systems with defined rules are used. They are used to show common preferences within a population, for example what politician a population wants to see elected. Several types of systems have been designed for this purpose and there are a myriad of variations of those systems (FairVote, 2017). They range from simple methods such as "most votes win" to complex processes that can only be practically carried out by a computer. However, practically all voting systems are algorithmic in nature, which makes them interesting to study for both Computer Scientists and Mathematicians. When deciding on what systems that shall be used, it is useful to know what differences and similarities they have.

1.2 Purpose

The purpose of this essay is to evaluate differences and similarities in terms of election results in three different voting systems: Single Transferable Vote (STV), First-Past-The-Post (FPTP) and the Schulze Method. This paper will only be examining and discussing differences and similarities in the results of the methods, not touching on how practically feasible the methods might be in actual elections.

2 Theory

One would think that by testing all different voting systems and by setting up good criteria for how to evaluate them, it would be possible to come to a definitive conclusion of what system should be used. However, this is not the case. By setting up different mathematical criteria for how a voting system shall behave, it is impossible to satisfy all criteria simultaneously. This is what several theorems, most notably *Arrow's Impossibility Theorem* show (Arrow, 1950). This means that any decision regarding choice of electoral system is a balance of their strengths and weaknesses. Simply put: there is no perfect voting system and never will be. This does, however, not mean that all voting systems are equally efficient, elegant or accurate.

There are several ways of evaluating voting methods (Green-Armytage, 2017). One is the mathematical approach of evaluating what criteria are important to meet and finding methods that satisfy them (Woodall, 1994). Another is by testing and finding probabilities that some criteria are broken — A method that fails to meet a certain criterion only in very few cases is perhaps not a big problem. A third way of evaluating methods is not by mathematical criteria, but other metrics such as voter satisfaction or how likely someone is to vote tactically.

2.1 Definitions

- A *candidate* is someone a voter can vote for to elect. It can also be considered as being a party that a voter can vote for.
- An *election* is an event in which voters cast ballots to vote for candidates
- A *seat* is something a candidate can be elected to. A candidate may for example be elected to hold a seat of parliament.
- An *ideology* is a group that both candidates and voters can identify with. They split the whole voting population into several blocks. Several candidates can belong to one ideology.

3 Methodology

All voting algorithms have been implemented in Typescript – mostly due to the author having previous experience with JavaScript. Typescript is a superset of JavaScript with multiple extra features. Most importantly, Typescript has optional static type checking. Sample code written in Typescript can be seen in code example 1. The full source code with instructions for how to run the code can be found in Appendix A.

Code example 1: Basic Typescript syntax

```
let helloWorld: string = 'Hello World' // This is a comment
console.log(helloWorld)
```

3.1 Modeling test data

In order to test the methods and their implementations, test data is needed. There are cases where extensive election data is published, such as in Maltese elections, but full ballot data, which is needed as input for the implementations used in this paper has not been found. Instead, the algorithms will be tested with data generated from a simple ballot-generating computer program. The program used for this purpose in this paper is highly primitive, as modeling preferences within a population is far beyond the scope of this paper. The full ballot generation program can be found in the `/generator` folder. The goal of the program is to produce ballots that rank every candidate in order of preference (see figure 4 on page 15). The program incorporates the idea of an *ideology* where each candidate is assigned one. Each ideology has has a list of weighted preferences indicating how a voter for this ideology is likely to vote. The weighed preferences are generated via an abstraction. A single configuration object is used to create all ideologies.

Code example 2: Configuration object for creating ideologies

```
const ideologyConfig: Ideology[] = [{
    // The size of the ideology
    size: 0.2,
    // Spread of votes between candidates within ideology
    // Large -> steep spread, Small -> flat spread
    candidatePower: 0.2,
    // How inclined voters of this ideology are to vote for a
    ↪  candidate of this ideology
    // Higher -> more inclined
    ideologyPower: 1.2
}, {
//  ...
},
// ...
]
```

The program then works via 4 major steps:

1. It assigns each candidate to an ideology based on the size of the ideology. A larger ideology has more candidates.

2. It creates a base "strength" to each candidate. Within each ideology, every candidate is assigned its relative strength. The relative strength is generated by a negative exponential function with a different constant (based on the `candidatePower`). The strength of each candidate is then multiplied by the size of their ideology and then normalized. This gives a base strength of all candidates. This simulates the probability for vote from a non-ideological voter for a particular candidate.

3. Based on the base strength of each candidate, a probability distribution of votes for each candidate is created for each ideology. This process takes the `ideologyPower` constant into account. An ideology's probability distribution represents how a voter aligned with the ideology is likely to vote. A strong ideology may thus attract votes from voters aligned with another ideology.

4. It creates a certain number of individual ballots. Each ballot is given an ideology and based on the probability distribution of that particular ideology, it creates its own list of preferences for candidates. This is done via weighed random number generation.

The output of the program is a set of ballots, where each ballot is an ordered set of preferences between candidates. The implementation is highly arbitrary and as are all constants. However, expanding on the generator program would be beyond the scope of this project. The vote generation algorithm is an attempt to provide more realistic voting data than what is achieved by a purely random allocation of votes. The program also allows for the possibility to control the data by changing the input variables. This makes it possible to test different election scenarios.

3.2 Constructing test scenarios

Constructing test scenarios, ie. election conditions, is difficult task. Due to there being so many variables both in real elections and the simulated elections in the generator program, it makes it practically impossible to control for all variables. Instead, 3 arbitrary test scenarios will be used. An election will be run 1000 times for each scenario and the election data will then processed by each voting algorithm and the results will be compared. The three simulation scenarios are arbitrary but should allow for comparison of

the different voting methods. All simulations will have three ideologies and 500 ballots.

3.2.1 Scenario 1

The scenario is built around electing one candidate out of eight. The spread of votes within ideologies is "flat" and all ideologies are so called "weak". This means that the general spread of votes between candidates will be even. An example of the distribution of first-preference votes generated in this scenario can be seen in figure 1. The full configuration object can be seen in code example 3.

Code example 3: Configuration object for scenario 1

```
const candidates = 8
const seats = 1
const ideologies: Ideology[] = [{
    size: 0.3,
    candidatePower: 0.3,
    ideologyPower: 0.8
}, {
    size: 0.3,
    candidatePower: 0.3,
    ideologyPower: 0.7
}, {
    size: 0.4,
    candidatePower: 0.5,
    ideologyPower: 0.8,
}]
```

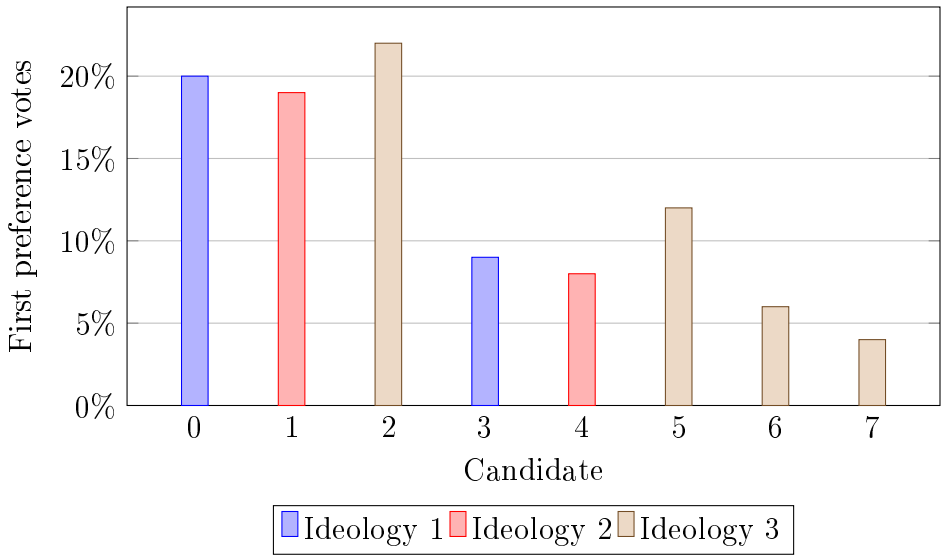

Figure 1: Example distribution of first-preference votes in scenario 1. Avreages of 10 simulations.

3.2.2 Scenario 2

The second scenario elects 3 candidates out of 9 total. The parameters are set to produce a "normal" spread of votes between candidates. An example distribution of first-preference votes generated in this scenario can be seen in figure 2. The full configuration can be seen in code example 4.

Code example 4: Scenario 2

```
const candidates = 9
const seats = 3
const ideologies: Ideology[] = [{
    size: 0.2,
    candidatePower: 0.5,
    ideologyPower: 1.2
}, {
    size: 0.3,
    candidatePower: 0.2,
    ideologyPower: 1
}, {
```

```
    size: 0.5,
    candidatePower: 0.3,
    ideologyPower: 0.8
}]
```

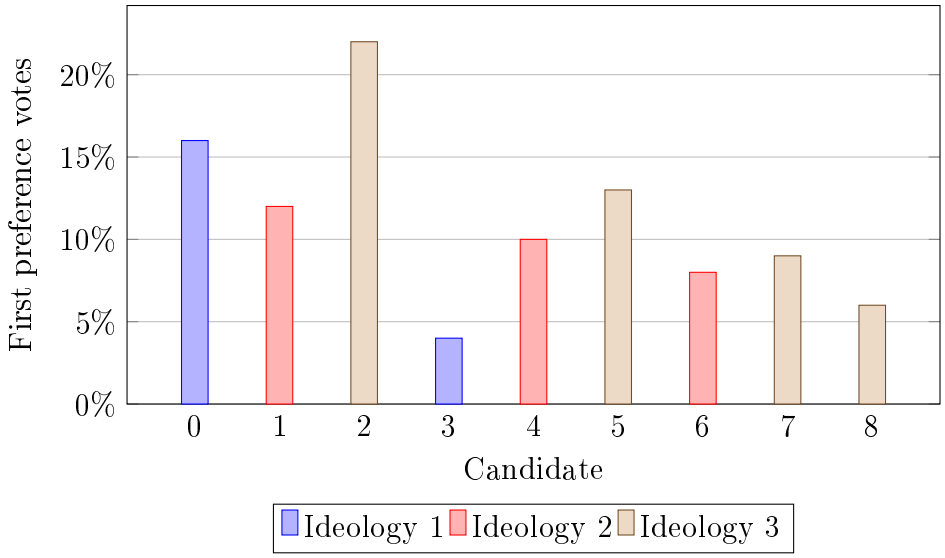

Figure 2: Example distribution of first-preference votes in scenario 2. Avreages of 10 simulations.

3.2.3 Scenario 3

The second scenario elects 3 candidates out of 9 total. It features a steep spread of votes between candidates. An example distribution of first-preference votes generated in this scenario can be seen in figure 3. The full configuration can be seen in code example 5.

Code example 5: Scenario 3

```
const candidates = 9
const seats = 3
const ideologies: Ideology[] = [{
    size: 0.2,
```

```
    candidatePower: 1.8,
    ideologyPower: 0.6
}, {
    size: 0.3,
    candidatePower: 1.6,
    ideologyPower: 0.6
}, {
    size: 0.5,
    candidatePower: 2,
    ideologyPower: 0.6
}]
```

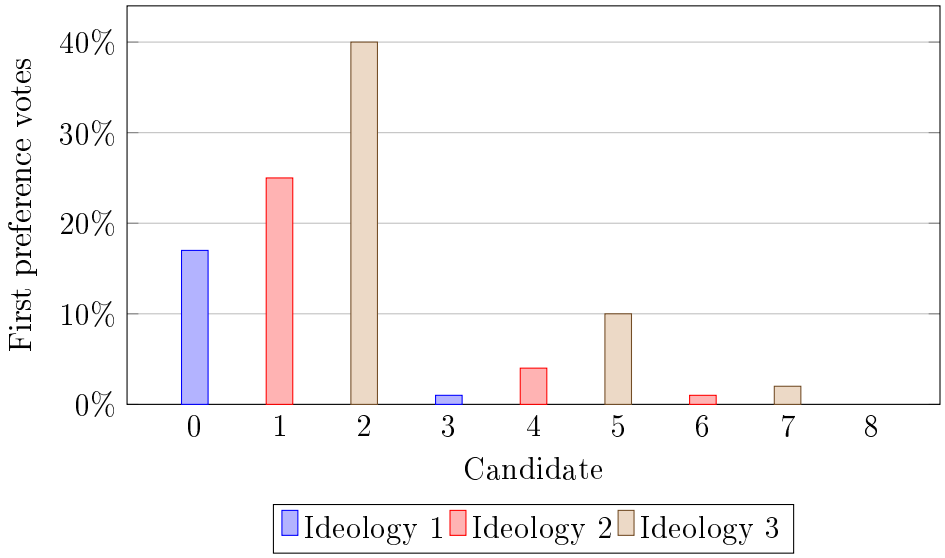

Figure 3: Example distribution of first-preference votes in scenario 3. Avreages of 10 simulations.

3.3 Metrics for evaluating results

Three metrics will be used for comparing the results obtained in the different scenarios. The results obtained with each voting method will be compared against the other methods. Comparisons to the configuration files outlined in Sections 3.2.1 - 3.2.3 will also be made.

3.3.1 Standard deviation

The standard deviation of the outcomes represents the variance in outcome due to the pseudorandom nature of the generator program outlined in Section 3.1. The standard deviation is calculated according to equation 1.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}, \quad \text{where} \quad \mu = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{1}$$

3.3.2 Misrepresentation

The misrepresentation metric gives an idea of how *proportional* the outcomes of a certain voting method in any scenario is. The level of proportionality of a result is a measure of how closely the preferences of a population in terms of ideology are matched with the result (King, 2000). Higher proportionality means that the election outcome more closely resembles the ideological preferences of the voters. The method of calculating the misrepresentation used in this paper is defined as follows:

1. Calculate what theoretical election outcome would be the most proportionate in terms of ideology based on the configuration file used to generate the ballots.

2. Compare the actual outcomes of a certain method with the theoretical result obtained in Step 1.

   (a) Calculate the difference between the measured and theoretical percentage for each ideology.

   (b) Sum all the differences.

3.3.3 Most likely winners

Shows what candidates, given a certain scenario, are most likely to win in a particular method.

4 Specification of voting methods

4.1 First-past-the-post

4.1.1 Description

The first-past-the-post voting method is a method designed for electing one or several (N) candidates out of a set of candidates. Each voter has one vote that may be allocated to any candidate. In a FPTP election the N candidates with most votes get elected (Wilkinson, 2017).

4.1.2 Justification

The method is simple and can easily be understood. The first-past-the-post method is widely used in for example the United Kingdom and United States (ACE Electoral Knowledge Network, 2017). Comparisons with this method can also easily be made.

4.1.3 Pseudocode

let V be the set of votes with V_i being the number of votes for candidate i
let N be number of candidates to be elected
sort V based on V_i
slice V between 0 and N

4.1.4 Implementation

The first-past-the-post program is implemented in a single file (`/fptp/index.ts`). The inputs of the program can be seen in code example 6.

Code example 6: Inputs for FPTP program

```
1 ballots: number[][] // Ex. [[0,1,2],[1,0,2],...]
2 seats: number // Ex. 2
```

The program sums all the first-preference votes for each candidate and elects the ones with most votes, in accordance with specified amount of seats.

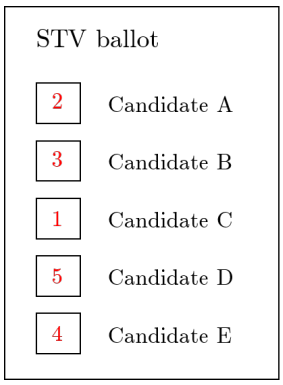

Figure 4: Sample STV ballot

4.2 Single transferable vote

4.2.1 Description

The Single Transferable Vote (STV) is a more complicated voting system than the first-past-the-post method (Electorial Reform Society, 2015). STV is a multi-seat election method, although it can be used for single-seat elections too. STV becomes equivalent to the Instant-runoff voting method (IRV) in single-seat elections. In an STV election, voters are given ballots in which they are supposed to rank candidates in order. The specific rules for wether one needs to rank all candidates or just a few can vary. For the sake of simplicity, all candidates must be ranked at unique and linear values in this implementation. An example of an STV ballot can be seen in figure 4. By having voters convey more information in a ballot, there is more data available that the method can take advantage of.

Using this data, votes are transferred between candidates within a ballot, hence the name Single Transferable Vote. The method relies on the concept of a quota, which is the number of votes one needs to be elected. The Droop-Quota, defined as $(\frac{\text{total valid poll}}{\text{seats}+1}) + 1$, is often used and is the one implemented in this paper. If a candidate receives equal or more votes than the quota, he/she is elected. If the number of votes for a candidate exceed the quota, a fraction of his/her votes are transferred to the next choice on the ballots that voted for that particular candidate. This process is repeated until there are no candidates with more votes than the quota. If there still are seats yet to be filled, the candidate with fewest votes is eliminated and his/her votes are transferred to the next choice on the ballots which supported the candidate.

This process is repeated until all seats are filled (King, 2000). The process can become quite complex with many ballots and transferring of fractional votes. A computer is therefore essential to resolve large scale elections.

4.2.2 Justification

STV is an established voting method in use in several countries. It is used for parliamentary elections in Ireland, Malta and Australia as well as being used in local and regional elections across the world (Tideman, 1995).

4.2.3 Pseudocode

let $quota$ be the quota
let $seats$ be the number of seats
let $winners$ be the list of elected candidates
let $votes$ be the set of votes with $votes_i$ signifying votes for candidate i
while $|winners| < seats$
    if any candidate $i \notin winners$ and $votes_i \geq quota$
        add i to $winners$
        continue
    end if
    if any candidate i where $votes_i > quota$
        multiply $votes_i$ with $(1 - \frac{\text{surplus votes}_i}{votes_i})$
        multiply next preferences for $votes_i$ with $\frac{\text{surplus votes}_i}{votes_i}$ and
        distribute into $votes$
        continue
    end if
    if $|votes| = seats$
        add all $votes \notin winners$ to $winners$
        continue
    end if
    $k :=$ index of smallest $votes_i$
    eliminate $votes_k$ and distribute all next-preference votes for candidate
    k into $votes$
end while

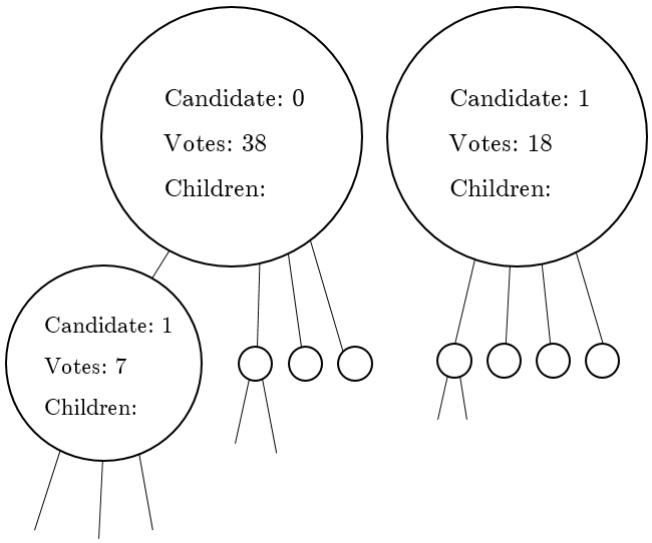

Figure 5: Simplified and shortened version of the tree structure. Each candidate is a top-level node with children, signifying the preferences of the voters for that candidate.

4.2.4 Implementation

The STV algorithm uses a tree structure to represent the state of the election. Every candidate is a top-level node with an attribute showing the number of current votes for the candidate. Each node has child nodes representing the next preferences of the voters for that particular candidate, see figure 5. It is easy to manipulate branches via recursive functions. Merging and multiplying branches is used when transferring votes from one candidate node to another. A candidate node is a Typescript class. It can be found in the `stv/candidate.ts` file and has the following properties and functions:

Code example 7: The CandidateNode class

```
class CandidateNode {
  public cand: number
  public count: number
  public children: CandidateNode[]
  // Add this and another branch together. Returns a new
   ↪ branch.
```

```
  add(other: CandidateNode): CandidateNode {
    //...
  }
  // Multiplies the count of this node and all its children
   ↪  and returns the new branch
  multiply(factor: number): CandidateNode {
    //...
  }
}
```

With the `add()` and `multiply()` functions it is possible to recursively modify branches. There are two different manipulations that need to be preformed on the tree: transferring surplus votes and eliminating candidates. Both functions rely on the `distribute()` function which is used to distribute a set of nodes onto the tree.

Code example 8: The distribute function

```
// Distributes nodes onto the tree. Returns new tree.
function distribute(nodeList: CandidateNode[], tree:
 ↪  CandidateNode[], winners: number[]): CandidateNode[] {
    // Loops through each candidate node in the list
    for (let i = 0; i < nodeList.length; i++) {
        let cand = nodeList[i];
        // Find if candidate is in the tree and/or is a winner
        let itExistsInTree = tree.some(obj => obj.cand ===
         ↪  cand.cand)
        let isWinner = winners.includes(cand.cand)

        // If it doesn't exist in tree (eliminated or no
         ↪  votes) or is a winner, distribute its children
         ↪  instead (recursive).
        if (!itExistsInTree || isWinner) {
            tree = distribute(cand.children, tree, winners)
        } else {
            // Find the index of candidate in tree
            let treeIndex = tree.findIndex(item => item.cand
             ↪  === cand.cand)
```

```
16              // Add the trees/branches together
17              tree[treeIndex] = tree[treeIndex].add(nodeList[i])
18          }
19      }
20      return tree
21  }
```

In order to eliminate and transfer surplus votes from any candidate node A, the children of A are passed into the distribute function. In the case of transferring surplus votes, the votes for each child node is first multiplied by $\frac{\text{surplus votes}}{\text{total votes}}$ as outlined in the pseudocode in Section 4.2.3. The full source code for functions that manipulate the tree structure can be found in `/stv/tree.ts`. The main process of the program is found in the `stv/election.ts` file and runs the process defined in the pseudocode in Section 4.2.3 on page 16.

4.3 Schulze

4.3.1 Description

The Schulze Method is a method used mainly to determine results in single-winner elections but can also be used to provide rankings and elect multiple candidates in a single election (Schulze, 2011). Ballots are identical to those of the Single Transferable Vote (see figure 4 on page 15). The method is compliant with the Condorcet Criterion, which means that if there is a candidate who the majority prefers in a pairwise comparison with every other candidate, that candidate (Johnson, 2005). As mentioned, this method relies on comparing the preferences of every candidate to one another as shown in table 1. If there is a Condorcet winner the process is simple: declare the Condorcet winner a winner and run another iteration without that candidate. Repeat this process until all seats are filled. However, there can be Condorcet ties, where there is no Condorcet winner. There are multiple methods used for resolving Condorcet ties, with one of them being the Schulze Method.

The Schulze Method resolves ties by investigating possible paths between candidates. For example, if out of a total of 50 ballots 30 ballots prefer $B > A$, 28 ballots prefer $A > C$ and 27 ballots prefer $C > B$, a path from A to B can be created by going $A \rightarrow C \rightarrow B$. The path is said to have the *strength* of the weakest link in the path. In this example, the path strength

Table 1: Pairwise comparison matrix used in the Schulze method. For example, 17 ballots prefer $A > B$ and 33 ballots prefer $B > A$.

	A	B	C	D	E
A		17	14	35	30
B	33		24	47	36
C	36	26		40	42
D	15	3	10		18
E	20	14	8	32	

between A and B is 27 due to the link between C and B having a strength of 27. There may be several paths between two candidates and the goal of the process is to find the strongest path between any candidate A and B, written as $p[A, B]$. By finding the strongest path between all nodes a result can be obtained where $p[X, Y] \geq p[Y, X]$ which means that candidate X wins. The Schulze Method also provides a linear ranking between all candidates, for example that $E > B > A > C > D$. By selecting the N top candidates one can use the method for a multi-seat election. As the process can become very complex as the number of candidates grow, a computer is needed to resolve large elections.

The difficult problem in this method is finding the strongest path between candidates. The problem is known as the widest path problem in graph theory. An efficient and relatively simple way to compute this problem is via the Floyd-Warshall algorithm which is used in the implementation (Wikipedia, 2017a). See Code Example 10 for the full algorithm.

4.3.2 Justification

The Schulze method is used by multiple organizations around the world (Wikipedia, 2017b). It has been used by various software organizations such as The Wikimedia Foundation, The Debian Project and Ubuntu. It is also used by political parties such as the Pirate Party in Sweden and various other countries. The process also differs greatly in both method and implementation from the two other voting methods used in this paper.

4.3.3 Pseudocode

let $d[i, j]$ be the number of ballots that prefer candidate i to candidate j
let $p[i, j]$ be the strength of the strongest path from candidate i to j
for i from 1 to C
    for j from 1 to C
        if $(i \neq j)$
            if $(d[i, j] > d[j, i])$
                $p[i, j] \coloneqq d[i, j]$
            else
                $p[i, j] \coloneqq 0$
            end if
        end if
    end for
end for
for i from 1 to C
    for j from 1 to C
        if $(i \neq j)$
            for k from 1 to C
                if $(i \neq k$ and $j \neq k)$
                    $p[j, k] \coloneqq \max\ (p[j, k],\ \min\ (p[j, k], p[i, k]))$
                end if
            end for
        end if
    end for
end for

4.3.4 Implementation

The Schulze Method is implemented within the file `schulze/index.ts`. It takes two inputs: a list of ballots and the number of seats to be elected. A ballot is an ordered list of candidates. The output of the program is a list of winners. Data representation can be seen in Code Example 9.

Code example 9: Data representation

```
// d[i][j] gives the number of ballots preferring i > j
let d: PairMap = getPairs(input)
```

```
// o[i][j] stores the strongest path between i and j
let p: PairMap = {}
// Number of candidates
let c = input[0].length
```

The program then runs the algorithm outlined in the pseudocode in Section 4.3.3.

Code example 10: Main algorithm

```
// Computing strongest path strength. Variant of the
 ↪  Floyd-Warshall algorithm
for (var i = 0; i < c; i++) {
    for (var j = 0; j < c; j++) {
        if (i !== j) {
            if (d[i][j] > d[j][i]) {
                p[i][j] = d[i][j]
            } else {
                p[i][j] = 0
            }
        }
    }
}

for (var i = 0; i < c; i++) {
    for (var j = 0; j < c; j++) {
        if (i !== j) {
            for (var k = 0; k < c; k++) {
                if (i !== k && j !== k) {
                    p[j][k] = Math.max(p[j][k],
                     ↪  Math.min(p[j][i], p[i][k]))
                }
            }
        }
    }
}
```

5 Evaluation of results

5.1 Scenario 1

Scenario 1 featured eight candidates where one was to be elected. Vote spread was flat, meaning that votes tended to be diverse in their preferences. This means that there is not an obvious winner present. This can be seen from the first-preference votes in the scenario seen in figure 1 on Page 10. The full result in this scenario can be seen in figure 6. The outlier in this scenario seems to be the FPTP method, electing candidate 0 more often than both other methods. Key metrics for the results in this scenario can be found in table 2. The misrepresentation metric show that FPTP is not entirely representative. The STV and Schulze method seem to produce very similar results.

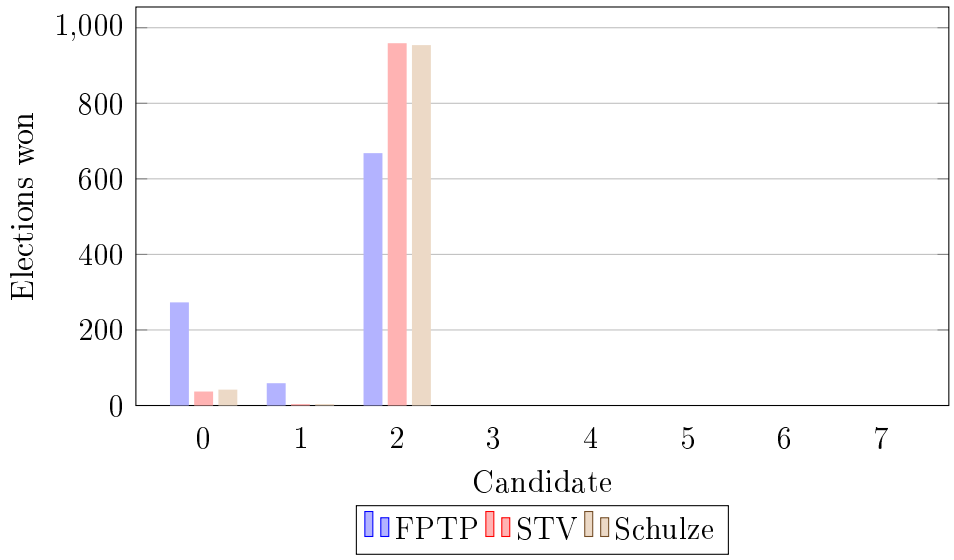

Figure 6: Results in scenario 1

Table 2: Key metrics for result in scenario 1

Method	Standard deviation	Misrepresentation	Most likely winners
FPTP	223	66	2
STV	315	8	2
Schulze	313	9	2

5.2 Scenario 2

Scenario 2 featured nine candidates of which three were to be elected. The distribution of votes was "normal", and as seen in figure 2 on page 2, the election is very close with many candidates having a similar percentage of first-preference votes. The results obtained in scenario 2 are very interesting. FPTP tended to underrepresent candidate 1, and elected a candidate from ideology 2 only in 28% of elections, despite the ideology theoretically having 30% of popular support. The Schulze method also tended to underrepresent ideology 2. It instead overrepresented ideology 3, electing 3 candidates of ideology 3 in 50% of elections. This indicates that the Schulze method is not a representative method in a multi-seat election. The STV result showed less misrepresentation than the FPTP result, which was to be expected since the method has access to more data of any voter's preferences. The results in scenario 2 can be observed in figure 7 and the key metrics can be seen in table 3.

Table 3: Key metrics for result in scenario 2

Method	Standard deviation	Misrepresentation	Most likely winners
FPTP	418	50	0, 2 & 5
STV	388	47	1, 2 & 5
Schulze	393	100	2, 5 & 7

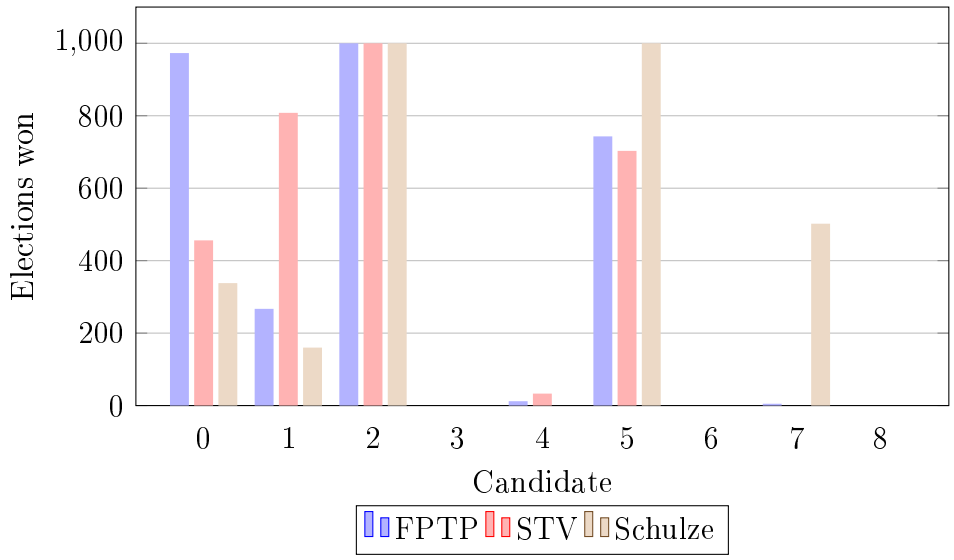

Figure 7: Results in scenario 2

5.3 Scenario 3

Scenario 3 featured nine candidates of which three were to be elected. The distribution of votes was "steep". This can be seen by the first-preference votes in figure 3 on Page 12. The results obtained with each method in this scenario are practically identical. From this, it can be inferred that the results of the methods converge as the preferences in a population become more clear. The results obtained in this scenario can be seen in figure 8. The key metrics can be found in table 4.

Table 4: Key metrics for result in scenario 3

Method	Standard deviation	Misrepresentation	Most likely winners
FPTP	470	0	0, 1 & 2
STV	471	0	0, 1 & 2
Schulze	471	0	0, 1 & 2

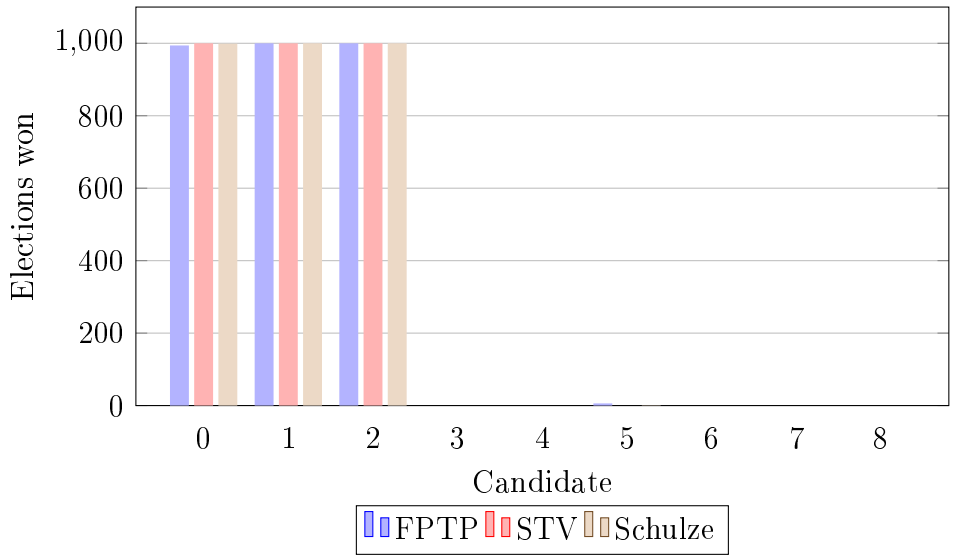

Figure 8: Results in scenario 3

6 Conclusion

The results obtained in the three different scenarios provide some insight into the varying properties of the different voting systems. Each of the methods showed different behavior in the different scenarios. For instance, the methods provided practically the same result in scenario 3 but wildly varying results in scenario 2. While the testing method used in this paper may not have been the most robust way of comparing the voting methods, several conclusions can be drawn from the data. They are outlined in 4 points below.

1. Schulze did not seem to be a proportional voting method in multi-seat elections.

   As seen from the results obtained in scenario 2, which can be seen in Section 5.2, the Schulze Method heavily overrepresented candidates from ideology 1 (candidates 3, 5 & 7). While ideology 1 in fact was the most popular ideology, a more proportionate way of representing the electorate in the scenario would have been to grant one seat to each ideology. The Schulze tended to elect candidates from the strongest ideology only, instead of electing candidates from all ideologies in a

more proportionate manner. This is not a problem in a single-seat election, as can be seen in the results of scenario 1 in Section 5.1, which was a single-seat elections, where the Schulze Method simply elected the most popular candidate.

2. In single-seat elections, Schulze and STV (and IRV in extension) seemed to provide similar results.

   As seen in the results of scenario 1 outlined in Section 5.1, both Schulze and STV produced nearly identical results, while the results from FPTP deviated slightly. While the methods work very differently, the results in these methods seem to behave similarly in single-seat elections.

3. FPTP did not seem to be a very proportional method in close elections.

   FPTP preformed considerably worse at being representative than the other methods. This was true for both close single- and multi-seat elections, excluding The Schulze Method in scenario 2 (see Point 1). At the core, both the STV and Schulze Method have more data to use for determining winners than what FPTP does. The FPTP method only has access to first-preference votes which does not convey as much information as a ballot listing all candidates in order of preference does.

4. STV seems to be the most proportional method out of the 3 tested, but is not without its flaws.

   STV provided the most proportionate results overall of all three methods tested. However, as it goes with voting system, it has several flaws. It is for example possible to increase the chance of a candidate winning by lowering him/her on a ballot, and vice versa. STV is the only multi-seat proportional voting system tested in this paper, and the results obtained reinforce its position as just that.

While there is no perfect voting system, they all preform differently. From the conclusions outlined above, a decision can be taken about what system analyzed in this paper to be used. Generally, if a proportional representation is desired (which in a democracy it usually is), STV will likely be the most appropriate method to use. In single-seat elections, the Schulze Method should also be considered. FPTP is the simplest method to both use and understand, but cannot be considered a proportionate voting method.

While the findings of this paper are hardly revolutionary, the nature of the three voting systems explored in this paper and their behavior is experimentally evaluated. The paper also provides implementations of all voting methods in both Typescript and JavaScript.

The key area for furthering this work would be in data collection or creation. A more sophisticated procedure for generating test data could be developed and run with more extensive test scenarios. Data from the real world could also be collected in actual elections and run through the different algorithms to test them. One could also write software that generates lots of scenarios with different parameters and analyzes how they differ automatically. Since the source code for this paper is open sourced under the MIT License, anyone can freely copy, modify and build on this work. This also makes for an open and transparent voting method where every voter can audit the system in use.

References

ACE Electoral Knowledge Network (2017). *Comparative data - Electoral System*. URL: http://aceproject.org/epic-en/CDTable?question=ES005 (visited on 04/27/2017).

Arrow, Kenneth J (1950). "A difficulty in the concept of social welfare". In: *Journal of political economy* 58.4, pp. 328–346.

Electorial Reform Society (2015). *What is STV?* URL: http://web.archive.org/web/20170227191342/http://electoral-reform.org.uk/sites/default/files/What-is-STV.pdf (visited on 04/02/2017).

FairVote (2017). *Electoral Systems*. URL: http://www.fairvote.org/electoral_systems (visited on 04/17/2017).

Green-Armytage, James (2017). *A Survey of Basic Voting Methods*. URL: http://jamesgreenarmytage.com/personal/voting/survey.htm#single (visited on 04/17/2017).

Johnson, Paul E (2005). *Voting Systems*. URL: http://pj.freefaculty.org/Ukraine/PJ3_VotingSystemsEssay.pdf (visited on 04/02/2017).

King, Charles (2000). *ELectoral Systems*. URL: http://web.archive.org/web/20170323033814/http://faculty.georgetown.edu/kingch/Electoral_Systems.htm (visited on 04/27/2017).

Schulze, Markus (2011). "A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method". In: *Social Choice and Welfare* 36.2, pp. 267–303. ISSN: 1432-217X. DOI: 10.1007/s00355-010-0475-4. URL: http://dx.doi.org/10.1007/s00355-010-0475-4.

Tideman, Nicolaus (1995). "The single transferable vote". In: *The Journal of Economic Perspectives* 9.1, pp. 27–38.

Wikipedia (2017a). *Floyd–Warshall algorithm*. URL: https://en.wikipedia.org/w/index.php?title=Floyd%E2%80%93Warshall_algorithm&oldid=770306762 (visited on 04/11/2017).

— (2017b). *Schulze Method*. URL: https://en.wikipedia.org/w/index.php?title=Schulze_method&oldid=774976249 (visited on 04/11/2017).

Wilkinson, Michael (2017). "What is the 'First Past The Post' voting system?" In: *The Telegraph*.

Woodall, Douglas R (1994). "Properties of Preferential Election Rules". In: *Voting matters* (4).

Appendix A Code implementation

```
/
├── fptp/
│   └── index.ts
├── generator/
│   ├── ballots.ts
│   ├── index.ts
│   ├── candidates.ts
│   └── interfaces.ts
├── schulze/
│   └── index.ts
├── stv/
│   ├── candidate.ts
│   ├── tree.ts
│   ├── election.ts
│   └── index.ts
├── .gitignore
├── LICENSE.txt
├── package.json
├── README.md
├── tsconfig.json
└── tslint.json
```

Figure 9: Project structure

The full source code for this project can be found at `https://github.com/adriansalamon/gymnasiearbete`. It is open sourced under the MIT License. In order to run any of the implementations, the Typescript files (`.ts`) need to be compiled to JavaScript files (`.js`) using the Typescript Compiler. Then the code can be executed using Node.js. Code Example 11 shows an example of how to run an election.

Code example 11: Example of how to run a Schulze Election with the code.

```
1 import schulze from '/path/to/schulze'
2 const ballots = [[0,1,2],[2,1,0],[0,2,1]]
```

```
3 const seats = 1
4 const electionWinners = schulze(ballots, seats)
```

All source code is commented and includes extensive use type annotations to help with orientation. The full project structure can be seen in Figure 9.