Analysis of Voting Algorithms: a comparative study of the Single Transferable Vote

GABRIEL ABENSOUR SELLSTRÖM
and MEIDI RUNEFELT TÕNISSON

KTH Computer Science
and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2012

Analysis of Voting Algorithms: a comparative study of the Single Transferable Vote

GABRIEL ABENSOUR SELLSTRÖM
and MEIDI RUNEFELT TÕNISSON

DD143X, Bachelor's Thesis in Computer Science (15 ECTS credits)
Degree Progr. in Computer Science and Engineering 300 credits
Royal Institute of Technology year 2012
Supervisor at CSC was Henrik Eriksson
Examiner was Mårten Björkman

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2012/abensour_sellstrom_gabriel_OCH_runefelt_tonisson_meidi_K12001.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Abstract

A voting system is defined as a procedure through which political power is distributed among candidates - from the ballot box to the parliament. This essay specifically seeks to contrast the Single Transferable Vote system with two other voting algorithms (Modified Sainte-Laguë and First-Past-The-Post), by constructing Java implementations of the algorithms and running example data through them. Thus, the suitability of a possible real-life implementation of the Single Transferable Vote method in a Swedish parliament context is evaluated. Furthermore, an alternative version of the original STV method which has been modified to fit these conditions is suggested. The effects of such an implementation on election outcomes are not entirely conclusive, and the conclusion is that more research is needed before a definite evaluation can be made.

Referat

Analys av röstningsalgoritmer:
en jämförande studie av ”enkel överförbar röst”

Ett valsystem är det tillvägagångssätt med vilket den politiska makten fördelas bland kandidater efter ett val. Den här uppsatsen syftar till att jämföra valsystemet ”enkel överförbar röst” med två andra valsystem, genom att konstruera Java-implementationer av algoritmerna för att köra exempeldata i. På så sätt utvärderar vi möjligheten att använda systemet med ”enkel överförbar röst” i svenska riksdagsval. Utöver det föreslås en alternativ version av det ursprungliga system ”enkel överförbar röst”, där modifikationer gjorts för att passa de förutsättningar som råder i Sverige. Effekterna av ett sådant byte av valsystem är inte helt klarlagda och slutsatsen är att mer forskning krävs innan det finns grund för ett definitivt omdöme.

Statement of collaboration

The authors would like to thank the following people for their contributions to the project:

Henrik Eriksson, for providing an excellent starting point for the project.

Svante Linusson and **Svante Janson**, for contributing with superb source material.

Marita Strand from the information department of Feministiskt Initiativ, for supplying highly useful election poll data.

This candidate project has been acheived as a collaboration between two authors. An outline of the main responsibilities of each author follows below.

Meidi has been responsible for designing and implementing the algorithms in Java as well as writing the corresponding sections in the final essay (including illustrations). She has constructed the augmented version of the 2010 Swedish election results fitted for the Single Transferable Vote algorithm. She has also been highly involved in the structuring and writing of the essay itself.

Gabriel has been responsible for constructing the hypothetical sub-threshold example for use with the MSL and STV systems. Furthermore he has been responsible for retrieving and processing data from the Maltese parliament elections, which have been used with the STV and FPTP methods. He has also created the graphs which appear in the essay. He has been in charge of typesetting the essay in LaTeX and handling the references, in addition to writing parts of the essay.

Contents

1 Introduction **7**
- 1.1 Background . . . 7
- 1.2 Purpose . . . 7
- 1.3 Definitions . . . 7

2 Problem Statement **8**
- 2.1 Evaluating voting systems . . . 8
  - 2.1.1 Defining desirability of outcomes . . . 8
  - 2.1.2 Evaluating voting system implementations . . . 8
- 2.2 Adapting the algorithms and real-life examples to suit our needs . . . 8
- 2.3 Methodology . . . 9

3 Specification of voting algorithms **9**
- 3.1 Modified Sainte-Laguë method . . . 9
  - 3.1.1 Description . . . 9
  - 3.1.2 Justification . . . 10
  - 3.1.3 Theoretical evaluation . . . 10
  - 3.1.4 Pseudo code . . . 10
  - 3.1.5 Implementation . . . 11
- 3.2 Single transferable vote . . . 12
  - 3.2.1 Description . . . 12
  - 3.2.2 Justification . . . 13
  - 3.2.3 Theoretical evaluation . . . 13
  - 3.2.4 Pseudo code . . . 13
  - 3.2.5 Implementation . . . 15
- 3.3 First-Past-The-Post . . . 17
  - 3.3.1 Description . . . 17
  - 3.3.2 Justification . . . 18
  - 3.3.3 Theoretical evaluation . . . 18
  - 3.3.4 Pseudo code . . . 18
  - 3.3.5 Implementation . . . 18

4 Constructing outcome examples **19**
- 4.1 The 2010 Swedish election results . . . 20
  - 4.1.1 Proportional allotment . . . 20
  - 4.1.2 Modification of the example data . . . 20
  - 4.1.3 Hypothetical differences in rankings . . . 22
- 4.2 The 2008 Malta election results, 4th district . . . 22
- 4.3 Sub-threshold example . . . 23
  - 4.3.1 Proportional allotment . . . 24

5 Evaluation of algorithms based on examples **24**
5.1 Modified Sainte-Laguë method 24
5.1.1 Example 1: 2010 Swedish election results 24
5.1.2 Example 2: Sub-threshold example 25
5.2 Single transferable vote 25
5.2.1 Example 1: 2010 Swedish election results 25
5.2.2 Example 2: Maltese election results 2008, 4th District 25
5.2.3 Example 3: Sub-threshold example 26
5.3 First-past-the-post 27
5.3.1 Example 1: Maltese election results 2008, 4th District 27

6 Visual representations **27**

7 Evaluations of methodology **30**

8 Conclusions **30**
8.1 Modified Sainte-Laguë 30
8.1.1 Implementation 30
8.1.2 Outcome 30
8.2 Single Transferable Vote 30
8.2.1 Implementation 30
8.2.2 Outcome 31
8.3 First-Past-The-Post 32
8.3.1 Implementation 32
8.3.2 Outcome 32

9 Further research **32**

10 Appendix **35**

1 Introduction

1.1 Background

In a democratic system, a set of rules is needed in order to fairly distribute the political power amongst the candidates. Several voting systems have been designed and tested throughout human history; some of them simple (such as the popular probabilistic method of drawing straws), some of them more complex (such as the Electoral College method currently employed in the United States[1]). What most voting systems have in common, though, is that they are decidedly algorithmic in nature, which makes this field of study interesting for Computer Scientists and Political Scientists alike.

1.2 Purpose

The purpose of this essay is to evaluate the possible use of a different version of the Single Transferable Vote (STV) method in a Swedish parliament context through comparisons with two other established voting algorithms: Modified Sainte-Laguë (MSL) and First-Past-The-Post (FPTP). As an auxiliary to this principal purpose, we aim to implement and evaluate these other voting algorithms in order to be able to draw apt comparisons between them and the Single Transferable Vote method - both in their designs and their achieved outcomes.

1.3 Definitions

- A *party* or *candidate* is someone for whom a vote can be cast in an election.
- An *election* is an event during which votes are cast by voters for parties or candidates.
- A *seat*, or *seat of office*, is a unit of political power distributed between the parties after the election has been carried out. The number of seats in a parliament might, for example, determine the number of votes a party can cast in parliament polls.
- A *vote threshold*, commonly employed in many democratic systems, is defined as a minimum percentage of votes that must be attained in order to be considered elected. In some systems, this percentage is a fixed number (as in Sweden, where a 4% vote threshold is used); in others, it might be the result of a more complex mathematical calculation. In the most common version of the Single Transferable Vote system, the equivalent to the vote threshold is the *Droop quota*. The Droop quota is described with the following formula: $\lfloor \frac{VOTES}{SEATS+1} + 1 \rfloor$.

2 Problem Statement

2.1 Evaluating voting systems

When evaluating a voting system, several factors contribute to the perceived appropriateness of a given system. For example, one might prefer systems that only demand of the voters to have a clear opinion of their highest-ranking candidate (and not that they rank all of the candidates), or that the votes can be tallied in a way that minimizes errors. For the purposes of this essay, we will define the desirability of a voting system as the combination of two factors: the desirability of the outcomes it typically produces, and how easy the system is to implement.

2.1.1 Defining desirability of outcomes

The desirability of an election outcome can be defined in several ways. One might for example come to the conclusion that seats should be awarded in a way that accurately reflects the votes that have been cast, resulting in the most seats being awarded to the party that has received the largest amount of votes. It might also be seen as desirable to award seats in such a way that deadlocks (situations where decisions cannot be reached due to an equal amount of members of parliament on opposing sides) are prevented.

For the purposes of this paper, we will primarily define the desirability of an election outcome in terms of how close to complete proportionality the seats are awarded. This means that a party that has received 30% of the vote should receive 30% of the seats in parliament. In the vast majority of cases, complete proportionality cannot be attained, as the total number of seats must be an integer, and there usually cannot be as many seats as there are voters. Different voting algorithms have different solutions to this problem.

2.1.2 Evaluating voting system implementations

The complexity of implementing the algorithm will also be a factor in determining its quality. Of course, to programmatically implement an algorithm differs significantly from the process involved in actually tallying the votes from a real election, but the similarities between the processes are nevertheless sufficiently large so as to provide some insight into the real-life difficulties of instituting a certain voting algorithm.

2.2 Adapting the algorithms and real-life examples to suit our needs

An obvious problem in empirically evaluating different voting algorithms using the same real-life example data is that some of the election data does not fit the construction of the voting algorithm. As an example: consider running the result of the Swedish 2010 election through the Single Transferable Vote algorithm. Since the Swedish electoral system does not in any way register the second preference choices of the voters, there is no way for us to extract this data in any exact manner. In order to produce an apt

example drawing from this initial data, we have been forced to draw some conclusions of our own, based on voter demographics. This is further detailed in section 4.1.2.

Unfortunately, this can also be the case when running real-life examples that do fit the proposed voting algorithm. The published voting figures from the Maltese public elections[2] (where the Single Transferable Vote method is in use) simply report which candidates have been eliminated in what round of the voting count and how the excess/wasted votes have been distributed, and not how the original votes were ranked. Seeing as how it's not possible to find out exactly how the original votes were ranked, we were forced to draw some conclusions of our own from the numbers that have been published. This is further detailed in section 4.2.

2.3 Methodology

Along with the Single Transferable Vote method, the Modified Sainte-Laguë method and the First-Past-The-Post method have been implemented. These voting algorithms have been contrasted and compared to each other (where applicable), and their defining features have been evaluated. The voting algorithms have been implemented in Java - mostly due to the fact that it is an established programming language that the authors have previous experience with. Several test cases (based on real-life scenarios, some with slight modifications as explained in section 4 below) have been provided as input to the different algorithms. The results have been noted, and the data has been collected in a structured way so as to be able to study the outcome differences as closely as possible.

As mentioned in Section 2.1.2, we have decided to study the implementational difficulties of the voting algorithms as well as their achieved outcomes. This has been done both through personally noting the perceived facility of programmatic implementation, and through reading of source material.

3 Specification of voting algorithms

We have decided to evaluate the following voting algorithms in this paper.

3.1 Modified Sainte-Laguë method

3.1.1 Description

The Sainte-Laguë method, named after the French mathematician André Sainte-Laguë, is a voting system employed in many multiparty parliaments. This method seeks to accomplish approximate proportionality of seat apportionment between parties, through successive calculations of party quotients. The vote tally for each party is divided by a series of odd divisors, and the party with the largest such quotient is awarded the current available seat. The Modified Sainte-Laguë method, which we will study during the course of our work, only differs from the original Sainte-Laguë method slightly; as it uses a different divisor in the first quotient calculation (see below). The modified version of the method is the voting system currently in use in the Swedish parliament elections.[3]

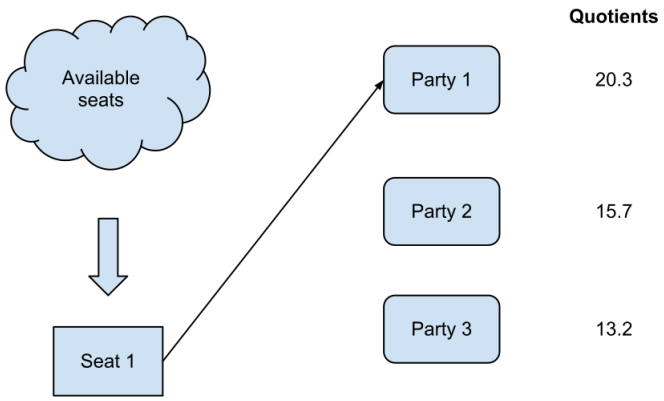

Figure 1: A seat allotment in the Modified Sainte-Laguë method.

3.1.2 Justification

Due to its aforementioned use in Swedish elections, and due to the fact that the authors are Swedish (and the readers most likely will be as well), the Modified Sainte-Laguë is an apt algorithm to implement and evaluate in this context. Election data from elections where the Modified Sainte-Laguë method has been used is readily available to us, which facilitates the usage and possible modification of example data.

3.1.3 Theoretical evaluation

The Modified Sainte-Laguë method and the original Sainte-Laguë method differ only in the first divisor of the quotient calculation; using 1.4 instead of 1 as the first divisor. This is in order to not dole out the first seat "too cheaply" to small parties.[3] Studies have shown that the Modified Sainte-Laguë method generally manages to attain a medium amount of proportionality, although less so than its unmodified predecessor.[4] This is due to the previously stated fact that the Modified Sainte-Laguë method slightly favors larger parties over smaller parties.

3.1.4 Pseudo code

The Modified Sainte-Laguë voting system can be described in pseudocode as follows:

let V be the set of votes, with V_i denoting the number of votes for party i
let $p_0, ..., p_n$ be the set of parties
let $SEATS$ be the total amount of available seats
let $seats_i$ be the number of seats currently awarded to p_i
let $quotients_i$ be the current quotient for party i
while $SEATS > 0$
    if $seats_i = 0$

$quotient_i := V_i/(1.4)$
else
$quotient_i := V_i/(2 * seats_i + 1)$
end if
$seats_{max(quotients)} := seats_{max(quotients)} + 1$
$SEATS := SEATS - 1$
end while

3.1.5 Implementation

The Modified Sainte-Laguë method has been implemented in a single Java class (see ModifiedSainteLague.java in Appendix). The class relies on the following global data structures and fields:

Code example 1: Global data structures and fields

```
int[] votes; // an integer array where the i:th element
    denotes the number of votes cast for party i
int[] seats; // an integer array where the i:th element
    denotes the number of seats currently awarded to
    party i
int SEATS; // the total number of seats in the example
    parliament
int PARTIES; // the total number of parties in the
    example
```

The example data is read from a file structured in the following way:

Code example 2: Example input file

```
3 // the total number of parties
// one line for each party follow, starting with the
    number of votes cast for each party, followed by the
    party name
15 SuperParty
10 MegaParty
12 UberParty
```

The data structures are then initialized with these values.

The seat apportionment is carried out by calculating successive quotients for each party. If a party has yet to receive a seat in the parliament, its quotient is set to $votes[i]/1.4$ - the number of votes cast for the party, divided by 1.4. In all subsequent calculations (when the party has been allotted a seat), the quotient is set to $votes[i]/(2 * seats[i] + 1)$ - the number of votes cast for the party, divided by two times the number of currently allocated seats for the party plus one.

These quotients are recalculated for each seat that is to be allocated. The seat is allocated to the party with the highest quotient.

3.2 Single transferable vote

3.2.1 Description

The Single Transferable Vote method is a voting system based on preferential voting. Instead of one vote being cast by each voter, a list of candidates is submitted in order of preference. If a high-ranking candidate gets eliminated, the vote transfers to the next candidate on the list. This method can be used in single-winner systems as well as multiple-winner systems.[5]

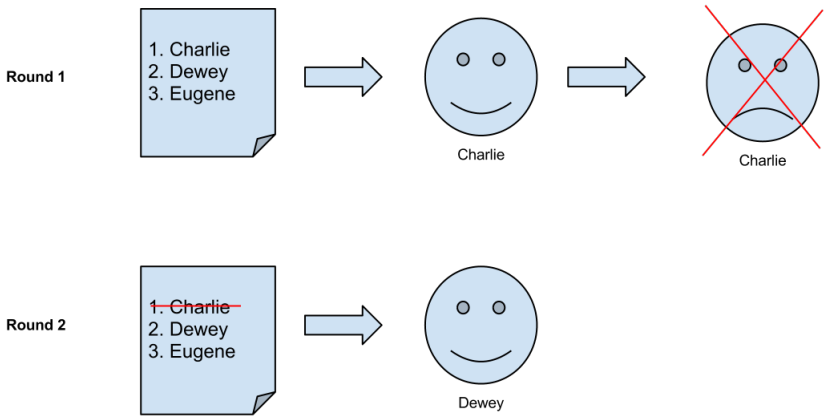

Figure 2: The transfer of a ranked vote in the STV system.

The original definition of the Single Transferable Vote algorithm includes transferring excess votes from already elected candidates to the voters second-preference choices, as well as transferring the votes cast for the parties that did not meet the quota. For single-winner systems, or systems where voters can vote for candidates from several different parties, this makes sense. However, in systems where elections are typically held on the basis that voters vote for *parties* and not candidates, such as the case currently is in Sweden, this would not be applicable. Since our work includes running the data from the Swedish elections through this algorithm, we have decided to modify the algorithm for such examples. The modification seeks to minimize the number of "wasted" votes for parties that do not reach the minimum vote threshold (for Swedish elections, this would be 4% of the total vote). This modification will work as follows:

Votes cast for parties that already fulfill the minimum vote threshold will continue to be added to the parties' vote tallies. Votes cast for parties that do not fulfill the minimum vote threshold when the tally of votes is completed will be transferred to the second preference candidate of the voter casting the vote, and so on. This version will be referred to as the *party version*.

The original definition of the Single Transferable Vote algorithm has also been implemented, to be used for test cases where voters cast votes for individual candidates insetad of parties. This version will be referred to as the *candidate version*.

3.2.2 Justification

The natural way of implementing a preferential voting system in Sweden would have been to use the classical STV method where voters rank individual candidates regardless of what party they belong to. This is a proven method which is already in use in parliamentary elections on Malta and in Ireland.[5]

This essay does not explore the possibilities of using STV in the same manner in Sweden. The reason is that there seems to be a different, less individual, culture among the parties in Sweden, resembling a more closed corporate-like culture. This is not surprising, since there is a tradition among party members of remaining loyal to their own party during the votes in the parliament. The idea is that a member is represented in the parliament in order to realize the programme of the party, rather than acting on their own desires.[6]

There have been changes in the voting process in Sweden aiming at promoting politicians with individual campaigns. The open list system in use in Sweden was modified in 1998. Voters now have the possibility to mark one single candidate from the party for which they vote, giving them an advantage over the other candidates on the party list.[7] However, this change has not had significant effects on the way candidates are elected. In the election of 2010, only 24.9% of the voters took advantage of this opportunity, which makes the number of candidates benefitting from this very limited.[8] Hence, there would not be much reason for employing the classical STV method in Swedish elections.

3.2.3 Theoretical evaluation

Critics of the Single Transferable Vote method usually stress the complexity of the method. It demands a certain level of knowledgeability on the part of the voters, and takes for granted that a voter has a clear view of how they would like to rank the candidates.[9]

Another consequence of the Single Transferable vote method is that it should, theoretically, reduce the level of partisanship in parliament. Since parties and/or candidates can come to power simply from the second-preference votes from other parties, one has less to gain from strict party lines and negative campaigning.[9]

The Single Transferable Vote method is not immune to tactical voting. There are several ways of tactically casting a vote within the STV system; among them, casting a first-preference vote for a candidate that is highly unlikely to be elected. The second-preference votes will then be transferred to other candidates at a later stage in the count, thus carrying more weight.[9]

3.2.4 Pseudo code

The party version of the Single Transferable Vote system can be described in pseudocode as follows:

let *votes* be the set of ranked votes
let *currentvotes* be the set of votes currently awarded to each party, with $currentvotes_i$ denoting

the number of current votes for party i
let *limit* be the least amount of votes needed in order to be considered elected
for each v in *votes*
    $i :=$ first preference of v
    $currentvotes_i := currentvotes_i + 1$
end for
while there are un-eliminated parties still below the vote limit
    $i :=$ index of the minimum value of *currentvotes*
    if $currentvotes_i < limit$
        divide the votes among the second-preferences of the voters that voted for i
        eliminate party i from the election
    end if
end while

The candidate version of the Single Transferable Vote method is similar:

let *votes* be the set of ranked votes
let *currentvotes* be the set of votes currently awarded to each candidate, with $currentvotes_i$ denoting the number of current votes for candidate i
let *limit* be the least amount of votes needed in order to be considered elected
let *elected* be the number of candidates that are to be elected
for each v in *votes*
    $i :=$ first preference of v
    $currentvotes_i := currentvotes_i + 1$
end for
while the number of elected candidates are less than *elected*
    while there are candidates still above the vote limit
        $i :=$ index of the maximum value of *currentvotes*
        if $currentvotes_i > limit$
            divide the votes among the second-preferences of the voters that voted for i
            elect candidate i
        end if
        if $currentvotes_i = limit$
            elect candidate i
        end if
    end while
    while there are un-eliminated candidates still below the vote limit
        $i :=$ index of the minimum value of *currentvotes*
        if $currentvotes_i < limit$
            divide the votes among the second-preferences of the voters that voted for i
            eliminate candidate i from the election
        end if
    end while

end while

3.2.5 Implementation

The Single Transferable Vote system has been implemented in a Java class that processes the number of first preference votes, as well as a set of probabilities of an "i-party voter" voting for the "j-party" as a second preference (resulting in a limit of the number of preferences to two, although votes can be redistributed several times). This is simply due to practical reasons. We do not have access to a large set of individual votes for any election (as previously stated in section 2.2), and thus we cannot process the original data in the same way as the election holders for the original elections did. We have decided not to implement a proof-of-concept version of the voting system that processes each vote individually, since the data extracted from this implementation would not be of any use for outcome comparisons.

It is important to stress that our implementation of the Single Transferable Vote method is not analogous to a real-life voting method implementation. The complexity of tallying votes combined with simple probabilities is far smaller than the complexity of tallying and re-tallying actual ranked votes. Thus, in evaluating the complexity of this implementation, one must remember that the outcome of this evaluation cannot simply be applied to a "pure" version of the Single Transferable Vote method.

The Java implementation of the Single Transferable Vote method has been achieved in a single class (see SingleTransferableVote.java in Appendix) - but in two versions, depending on whether a party-proportional parliament is desired or not. The first version only redistributes votes from parties that have been eliminated; the second version also redistributes excess votes from parties/candidates that have been elected.

The class relies on the following global data structures and fields:

Code example 3: Global data structures and fields

```
double[][] votes; // an integer matrix, with element (i,
    j) denoting the probability that an i-voter will vote
     for the j-party. the diagonal elements denote the
    number of votes currently given to the party: -1 if
    they have been eliminated.
int SEATS; // the total number of seats in the example
    parliament
int VOTES; // the total number of votes cast in the
    example election
int PARTIES; // the total number of parties for which
    there is sufficient data
boolean DROOP; // a boolean denoting whether the Droop
    quota should be used in the vote threshold
    calculation. If not, the vote quota will default to
    4%.
```

```
boolean party; // whether the party version or the
    candidate version should be run
```

The example data is read from a file structured in the following way:

Code example 4: Example input file

```
3 // the total number of parties
// one line for each party follow, starting with the
    number of votes cast for each party, followed by the
    party name, followed by the vote transition
    probabilities (0 for the party itself)
5 SuperParty 0 50 50
10 MegaParty 25 0 75
12 UberParty 100 0 0
```

The data structures are then initialized with these values. After this point, the two versions of the algorithm differ as described below.

Party version The first-preference votes are tallied and the parties whose votes do not suffice according to the vote threshold quota have their votes set to -1 before their votes are divided among the other parties, in order of smallest non-threshold-fulfilling party to largest. The probability of the second-preference vote going to the k:th party is multiplied with the number of votes that are to be distributed. Votes are not transferred to the party itself or to parties that have already been eliminated.

Code example 5: Vote redistribution

```
for(int k = 0;k<PARTIES;k++) {
  if(k==tmp||currvotes[k]==-1) {
      continue;
  }
  double probability = (double)votes[tmp][k]/100.0;
  double surplusVotes = (double)surplus;
  currvotes[k] += Math.floor(probability*surplusVotes);
}
```

This recounting procedure continues until there are no more votes to be redistributed.

Candidate version The first-preference votes are tallied, and the candidates whose vote count exceed (or equal) the vote threshold quota have their votes set to the vote threshold before the surplus of their votes are divided among the other candidates, in order of largest threshold-fulfilling candidate to smallest. These candidates are deemed elected, and a parameter keeping count of the number of elected candidates is increased.

Code example 6: Vote redistribution

```
while(elected!=SEATS) {
```

```
    int tmp = max(currvotes);
    if(currvotes[tmp]>limit) {
      electedParties.put(tmp, true);
      elected = electedParties.size();
      ...
    }
    ...
  }
```

When all of the excess votes have been redistributed, the algorithm moves on to redistribute the votes from the eliminated candidates, just as in the previous example. If after these two sets of calculations, the number of elected candidates is not equal to the number of seats that are to be apportioned, the calculations recommence (with first redistributing excess votes and then redistributing votes from eliminated candidates).

3.3 First-Past-The-Post

3.3.1 Description

The First-Past-The-Post voting method is an election method designed for electing one or several candidates out of a set of many candidates. It consists of simply electing the N candidates that have received the highest amount of votes. As such, it is a suitable algorithm for cases where several candidates are to be elected with each of the candidates holding the same amount of political power - as well as for cases where only one candidate is to be elected.[10]

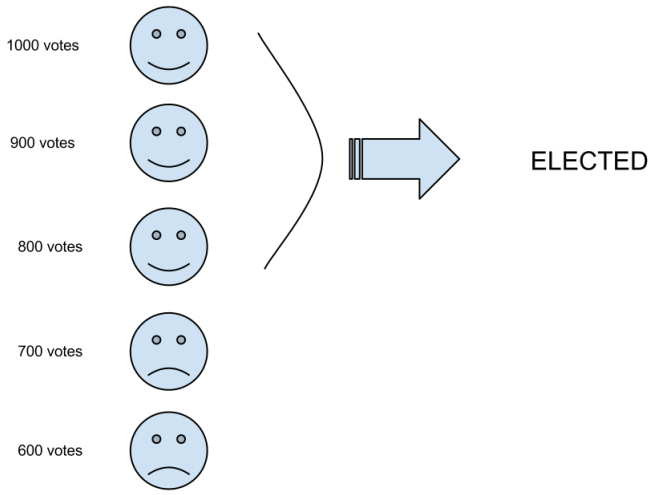

Figure 3: First-Past-The-Post election with three seats and five candidates

3.3.2 Justification

Being a relatively elementary voting system, the reasons for including the First-Past-The-Post method are mainly reasons of practicality. It is quite simply a method with which comparisons easily can be made - especially comparisons with the candidate version of the Single Transferable Vote method. As such, the results of the First-Past-The-Post method might look obvious, but they are nevertheless needed in order to provide a sufficient amount of data for evaluation.

3.3.3 Theoretical evaluation

The First-Past-The-Post voting method has been subject to much criticism; mainly that the system encourages tactical voting.[10] A voter that prefers an unpopular candidate might reason that since their favorite candidate is unlikely to be elected, their vote would go to waste if they went with their first preference - and as such, they might cast their vote for a more popular although less personally favorable candidate. This leads to voters having to somehow calculate the possible winners before the election takes place, which leads to a significant risk of having their final choice influenced by media outlets and pre-election polls.

3.3.4 Pseudo code

let V be the set of votes, with V_i denoting the number of votes for party i
let N be the number of parties that are to be elected
while less than N parties have been elected
    for all V_i in V
        if $V_i = max(V)$
            elect party i
        end if
        $V_i := -\infty$
    end for
end while

3.3.5 Implementation

The First-Past-The-Post voting method has been implemented in a single Java class (see FirstPastThePost.java in the Appendix). The class relies on the following global data structures and fields:

Code example 7: Global data structures and fields

```
int[] votes; // an integer array where the i:th element
    denotes the number of votes cast for party i
int PARTIES; // the total number of parties in the
    election data
```

```
int SEATS; // the total number of parties that are to be
    elected
```

The example data is read from a file structured in the following way:

Code example 8: Example input data

```
3 // the number of parties in the example data
// one line for each party follow, starting with the
   number of votes cast for each party, followed by the
   party name
5 SuperParty
10 MegaParty
12 UberParty
```

The data structures are then initialized with these values.

After the initialization is complete, the N parties with the highest amount of votes are declared to be elected:

Code example 9: Winner calculation

```
int count = 0;
while(count<SEATS) {
  int elected = max(votes);
  System.out.println("Party " + elected) + " has been
     elected");
  votes[elected] = -1; // do not include in further
     calculations
  count++;
}
```

4 Constructing outcome examples

In constructing our example data, we have chosen to focus primarily on real-life examples, as well as slightly modified real-life examples. This has entailed looking up election data online, mainly from countries that already employ one of the algorithms that we have implemented. The reason for doing so, instead of constructing fictional examples completely from scratch, is that the real-life examples we have found (as outlined below) are sufficiently diverse and interesting enough in order to demonstrate the differences between the voting systems.

As outlined in section 2.2 above, we have had to modify some of the data in the example set for the algorithm experiments to make sense. These modifications are explained in further detail in the sections below.

4.1 The 2010 Swedish election results

As the authors of this paper are of Swedish origin, as will most of the readers be, it seems reasonable to use numbers from a Swedish election as an example. This way the reader will have a relation to the numbers and to the results, therefore it will be easier to understand and motivate different algorithms. In terms of voting algorithms, the results of the 2010 election themselves are not more interesting than are the results of any of the earlier elections. One thing that is unusual about the 2010 election is that a new party entered the parliament, namely Sverigedemokraterna, something that had not happened in almost 20 years.[11]

It should be noted that adjustment seats have not been taken into consideration in our implementation of the voting algorithms. Therefore the results from the Swedish 2010 election, when run through our implementation, will differ slightly from the real-life 2010 seat division.

4.1.1 Proportional allotment

If the 349 seats would be allotted in a completely proportional way - giving a party with 10% of the vote 10% of the seats - the seat distribution for this example would consist of fractional seats being allotted to the parties. The following numbers[12] are thus included purely for comparative reasons.

Party	V	S	MP	C	FP	KD	M	SD	PP	FI	SPI
Seats	19.54	107.00	25.62	22.89	24.64	19.54	104.91	19.89	2.27	1.40	0.66

It must be noted that the figures above do not take the Swedish vote threshold of 4% into account.

4.1.2 Modification of the example data

In order to make use of the data from the 2010 Swedish election while analysing the Single Transferable Vote algorithm, we were forced to augment this data set with some creative guessing regarding how a Swedish voter would rank their parties if they had the opportunity to do so. Since we have already applied a modification of the Single Transferable Vote algorithm that leads to excess votes not being redistributed among other parties (as described in section 3.2.5), we only really need to justify the hypothetical candidate ranking of the voters whose parties did not meet the Swedish vote threshold of 4% of the total vote.

The three largest parties that are just below the 4% threshold are Piratpartiet (with 0.65% of the vote), Feministiskt Initiativ (with 0.40% of the vote), and SPI - Sveriges Pensionärers Intresseparti (with 0.19% of the vote).[12] We have decided to only include these three sub-threshold parties in our ranking estimates, as relevant data regarding voters for even smaller parties is hard to find.

There are many papers on the subject of Swedish voter trends, but unfortunately they almost always only describe the flow of votes between parties that are already in

parliament. As such, we have had to rely on numbers from the SVT exit poll VALU[13] that was carried out on the day of the election, in order to analyze what the probabilities are of a person from a certain demographic voting for another party. Specifically, we have decided to use the results of the question "which parties would you like to see in government?" Given the fact that voters were free to choose more than one party as an answer to this question, this does not reflect their actual second-choice preferences.

This obviously results in little more than a loosely founded guess, but even a guess such as this should yield results that are interesting enough to warrant the inclusion of such a guess. We have decided to only account for hypothetical second rankings of sub-threshold party voters, though, as the construction of third and fourth-hand guesses would be much too hypothetical to offer any real insight.

Piratpartiet

Of the voters for Piratpartiet on election day in 2010, this is the percentage breakdown of the government party preferences:[13]

Party	V	S	MP	C	FP	KD	M	SD
Preference	11	19	21	19	21	9	26	9

Drawing from these numbers, we can construct a guess of what a Piratpartiet voter would have chosen as a second-choice party in the 2010 election. Normalizing these numbers, we get the following percentages:

Party	V	S	MP	C	FP	KD	M	SD
Percentage	8.15	14.07	15.56	14.07	15.56	6.67	19.26	6.67

This gives us a sufficient indication of the party sympathies of the Piratpartiet voters. The data from the 2010 election can then be augmented in the obvious fashion - giving 8.15% of the Piratpartiet voters Vänsterpartiet as a second choice, 14.07% Socialdemokraterna, and so on.

Feministiskt initiativ

Of the voters for Feministiskt Initiativ on election day in 2010, this is the percentage breakdown of the government party preferences:[13]

Party	V	S	MP	C	FP	KD	M	SD
Preference	49	37	64	6	12	6	15	2

Normalizing these numbers, we get the following percentages:

Party	V	S	MP	C	FP	KD	M	SD
Percentage	25.65	19.37	33.51	3.14	6.28	3.14	7.85	1.05

The data from the 2010 election can then be augmented in the obvious fashion, as above.

SPI - Sveriges Pensionärers Intresseparti

The SPI voters have not been included in the SVT VALU exit poll, making it necessary to use other means of constructing a hypothetical second choice for these voters. Since the founding principle of the SPI party is to provide a government alternative that caters to senior citizens,[14] we have decided to generalize the demographic of the SPI party to people of the ages 61 and above.

The percentage breakdown of voters of the ages 61 and above is as follows:[15]

Party	V	S	MP	C	FP	KD	M	SD	Others
Percentage	3.83	37.17	4.44	8	8.61	6.61	27.51	3.22	0.61

Disregarding the 0.61% voting for other parties (since there is no detailed statistical breakdown of this figure), data from the 2010 election can then be augmented in the obvious fashion, as above.

4.1.3 Hypothetical differences in rankings

One of the key differences of voting in a Single Transferable Vote system as compared to a Sainte-Laguë system (currently in place in Sweden), is that voters are not discouraged from voting for their favorite candidate simply from fear of the candidate not getting into parliament. Being able to rank the candidates in this way would possibly eliminate the fear of one's vote being "wasted" on a party that did not manage to meet the parliament threshold.[5]

Taking this into account, it is easy to come to the conclusion that the voting numbers for the 2010 Swedish election might have looked extremely different with the Single Transferable Vote system in place. For example, according to a phone survey carried out by Sifo in 2006, 11% of voters would consider voting for Feministiskt Initiativ if they were confident that the party would make the 4% threshold.[16] This makes the augmented 2010 Swedish election results example somewhat lacking, and probably not very representative of what the numbers would have looked like if the Single Transferable Vote algorithm were in place by the 2010 election.

4.2 The 2008 Malta election results, 4th district

In our search for example data to draw from when evaluating the Single Transferable Vote method, we immediately started looking for countries that already have implemented a variant of this method in their elections. Malta emerged from several sources,[5,17] and we managed to get a hold of data from elections held in the different Maltese districts in 2008. The numbers show how the successive eliminations and vote redistributions were carried out, but the details of the individual votes have not been reported.[2]

Based on this data, we have constructed a Maltese example with the same structure as the previous Single Transferable Vote examples. For each vote redistribution, a probability of vote transfer was calculated based on the actual number of votes that were transferred from one party to another. E.g: since 4 out of 1230 votes from Saviour

Parnis we redistributed to Jesmond Mugliett, the probability of vote transfer from Parnis to Mugliett is $4/1230 \approx 0.3\%$.

The main problem with this approach is that data becomes more and more sparse as the election progresses. For the later eliminations, there is only data describing which of the non-eliminated, non-elected candidates the redistributed votes have been allocated to. This means that a voter whose first-preference candidate was eliminated very late in the election may very well have had a previously eliminated candidate as their second-preference - but this has not been reported. However, since there is no way of deducing the actual rankings of the entire set of voters, we have had to settle for what information could be extracted from the data set.

4.3 Sub-threshold example

The purpose of this example is to investigate what were to happen if one of the parties which are represented in the Swedish parliament today were to receive less than 4% of the votes (the Swedish vote threshold), and therefore lose their representation in the parliament.

One of the parties which usually lies close to the 4% vote threshold is Kristdemokraterna (KD), which received 5.60% of the votes in the Swedish election of 2010.[12] Now, suppose that they had instead received 3.9% of the votes; just under the vote threshold. This would mean that, with the current voting system in use in Sweden, they would not have received a single seat in the parliament. Since KD is part of the right wing coalition which currently have the majority of seats in the parliament,[18] this would also mean that the coalition as a whole would lose 3.9% of their votes, and thereby lose some of their seats as a result.

Of the voters for KD on election day 2010, this is the percentage breakdown of the government party preferences.[13]

Party	V	S	MP	C	FP	M	SD
Preference	1	6	11	64	70	75	2

Normalizing these numbers, we get the following percentages:

Party	V	S	MP	C	FP	M	SD
Percentage	0.44	2.62	4.8	27.95	30.57	32.75	0.87

If KD had received 232 456 votes, they would have had 3.9% of the total 5 960 408 of votes. In reality they received 333 696 votes, which is 5.60% of the total amount of votes. So for the sake of this example, we will distribute the remaining 101 240 votes among the seven other parties of the government, using the normalized party preferences above.

Party	V	S	MP	C	FP	KD	M	SD
Votes	334 498	1 830 149	442 295	419 101	451 473	232 456	1 824 922	340 491
Percentage	5.612%	30.71%	7.421%	7.031%	7.575%	3.9%	30.62%	5.713%

4.3.1 Proportional allotment

If the 349 seats would be allotted in a completely proportional way, the seat distribution for this example would consist of fractional seats being allotted to the parties. The following numbers are thus included purely for comparative reasons.

Party	V	S	MP	C	FP	KD	M	SD	PP	FI	SPI
Seats	19.59	107.18	25.90	24.54	26.44	13.61	106.86	19.94	2.27	1.40	0.66

It must be noted that the figures above do not take the Swedish vote threshold of 4% into account.

5 Evaluation of algorithms based on examples

The examples, as described in section 4 above, have been run as input data through the three voting algorithms. Specific parameters have been set in order to make the algorithms closely fit the aim of the original election: for example, the Modified Sainte-Laguë method was given a total seat number of 349 when calculating the seat distribution for the 2010 Swedish parliament election.

The vote threshold quotient for each example and algorithm has also been set in an individual fashion, with the intent to fit the aim of the original election from which the data was retrieved. A vote threshold quotient of 4% was used for the 2010 Swedish election results example, and the Droop quota was used as a vote threshold for the 2008 Maltese election results.

5.1 Modified Sainte-Laguë method

5.1.1 Example 1: 2010 Swedish election results

The vote tally of the 2010 Swedish election results was run through the Java implementation of the Modified Sainte-Laguë method with a parameter setting of 349 seats (the total number of seats in the Swedish parliament) and 8 parties (the parties that fulfilled the 4% vote quota), yielding the following results:

Party	V	S	MP	C	FP	KD	M	SD
Votes	334 053	1 827 497	437 435	390 804	420 524	333 696	1 791 766	339 610
Seats	20	109	26	23	25	20	106	20

The difference between the results of the implemented Modified Sainte-Laguë method and the actual apportionment of seats in parliament is due to the usage of adjustment seats in Swedish parliament.

5.1.2 Example 2: Sub-threshold example

The vote tally of the fictitious election where KD did not attain the Swedish vote threshold of 4% was run through the Java implementation of the Modified Sainte-Laguë method with a parameter setting of 349 seats and 7 parties, yielding the following results:

Party	V	S	MP	C	FP	M	SD
Votes	334 498	1 830 149	442 295	419 101	451 473	1 824 922	340 491
Seats	21	113	27	26	28	113	21

5.2 Single transferable vote

5.2.1 Example 1: 2010 Swedish election results

The vote tally of the 2010 Swedish election results were run through the Java implementation of the Single Transferable Vote method, with the eleven parties with the largest amount of first-hand votes in the input file. The version that was run was the party version, which does *not* redistribute excess votes from already elected parties. The three parties that did not meet the Swedish vote threshold quota of 4% votes had their example data augmented with probabilities of the voters ranking a certain other party as their second preference choice (as detailed in section 4.1.2). The eight parties that *did* make the 4% vote threshold did not have their example data augmented, due to the aforementioned modification to the Single Transferable Vote algorithm that results in excess votes not being redistributed to other parties, but instead contributing to the seat apportionment.

The redistributed votes were then run as input data to the Modified Sainte-Laguë method, in order to calculate the new seat distribution in the Swedish parliament. This yielded the following results (with significant changes from the results of only running the example through the Modified Sainte-Laguë method denoted with a plus or minus sign depending on whether the party lost or gained seats):

Party	V	S	MP	C	FP	KD	M	SD
Votes	343 805	1 841 704	452 003	397 862	428 981	337 752	1 804 120	342 786
Seats	20	108-	27+	23	25	20	106	20

As follows from these results, it can be noted that the only difference in seat distribution is that one seat that was previously awarded to Socialdemokraterna is now awarded to Miljöpartiet.

5.2.2 Example 2: Maltese election results 2008, 4th District

The vote tally of the 2008 Maltese election in the 4th district, augmented with the vote transfer probabilities as previously described in section 4.2, was run through the Java implementation of the Single Transferable Vote method with a vote threshold (calculated according to the Droop quota) of 3660 votes, and the 16 candidates running for office

in the input file. The version that was run was the candidate version, which *does* redistribute excess votes from already elected candidates. This yielded the following results:

Candidate	Votes	Status
Seychell	86	ELIMINATED
Mizzi	182	ELIMINATED
Bonnici	544	ELIMINATED
Brincat	164	ELIMINATED
Cauchi	560	ELIMINATED
Chircop	4378	ELECTED
Mangion	2411	ELECTED
Muscat	49	ELIMINATED
Parnis	4890	ELECTED
Sammut	511	ELIMINATED
Azzopardi	3321	ELECTED
Bonavia	253	ELIMINATED
Galea	411	ELIMINATED
Mugliett	2484	ELECTED
Scerri	1561	ELIMINATED
Schembri	153	ELIMINATED

Table 1: Table showing the results with the number of total votes (first preference and transferred), and whether or not the candidate was elected

5.2.3 Example 3: Sub-threshold example

The vote tally of the fictitious election where KD did not attain the Swedish vote threshold of 4% was run through the Java implementation of the Single Transferable Vote method with a parameter setting of 349 seats, and the eleven parties with the largest amount of first-hand votes in the input file. The version that was run was the party version, which does *not* redistribute excess votes from already elected parties.

The redistributed votes were then run as input data to the Modified Sainte-Laguë method, in order to calculate the new seat distribution in the Swedish parliament. This yielded the following results (with significant changes from the results of running the example through the Modified Sainte-Laguë method denoted with a plus or minus sign depending on whether the party lost or gained seats):

Party	V	S	MP	C	FP	M	SD
Votes	345 290	1 850 552	468 215	492 264	532 231	1 914 733	345 724
Seats	20-	109-	28+	29+	31+	112-	20-

5.3 First-past-the-post

5.3.1 Example 1: Maltese election results 2008, 4th District

The vote tally of the 2008 Maltese election in the 4th district was run through the Java implementation of the First-Past-The-Post algorithm with a parameter setting of 5 seats and the 16 candidates running for office in the input file, yielding the following results:

Candidate	Votes	Status
Seychell	86	ELIMINATED
Mizzi	182	ELIMINATED
Bonnici	544	ELIMINATED
Brincat	164	ELIMINATED
Cauchi	560	ELIMINATED
Chircop	4378	ELECTED
Mangion	2411	ELECTED
Muscat	49	ELIMINATED
Parnis	4890	ELECTED
Sammut	511	ELIMINATED
Azzopardi	3321	ELECTED
Bonavia	253	ELIMINATED
Galea	411	ELIMINATED
Mugliett	2484	ELECTED
Scerri	1561	ELIMINATED
Schembri	153	ELIMINATED

Table 2: Table showing the results with the number of total votes (first preference and transferred), and whether or not the candidate was elected

6 Visual representations

The following graphs illustrate the differences between the outcomes of the previous experiments.

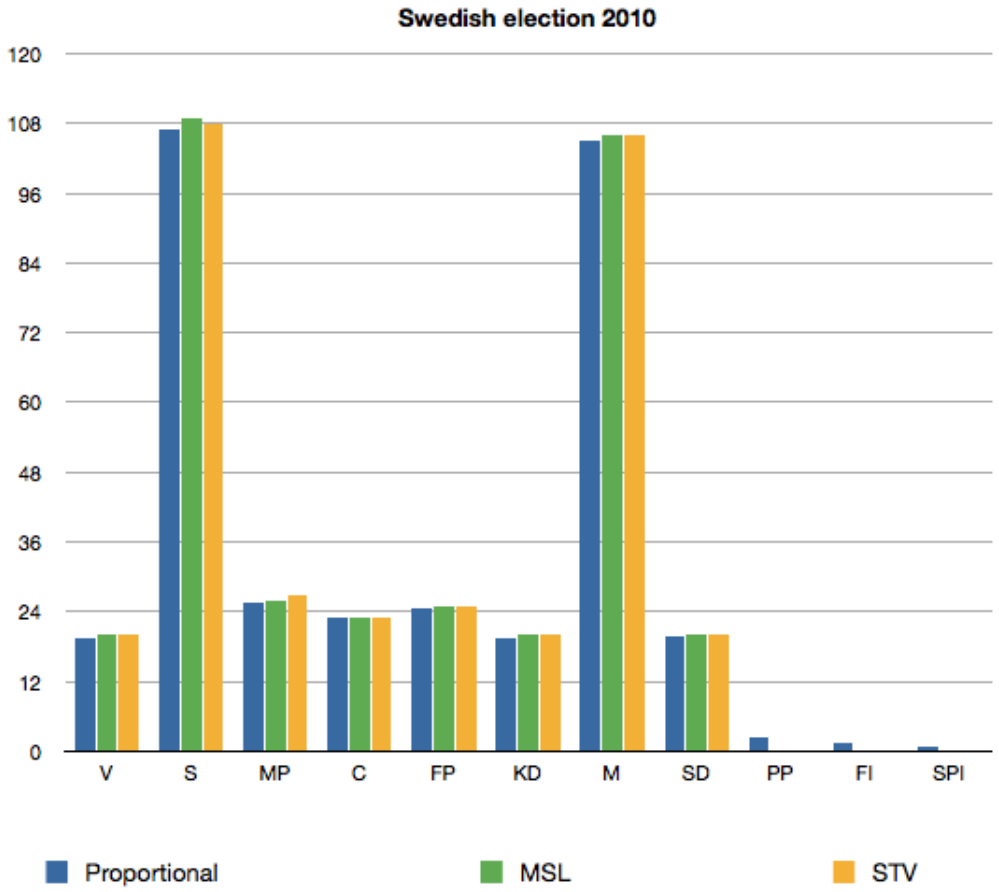

Figure 4: Differences in seat allotment for the Swedish 2010 election results, between completely proportional allotment, the Modified Sainte-Laguë method and the Single Transferable Vote

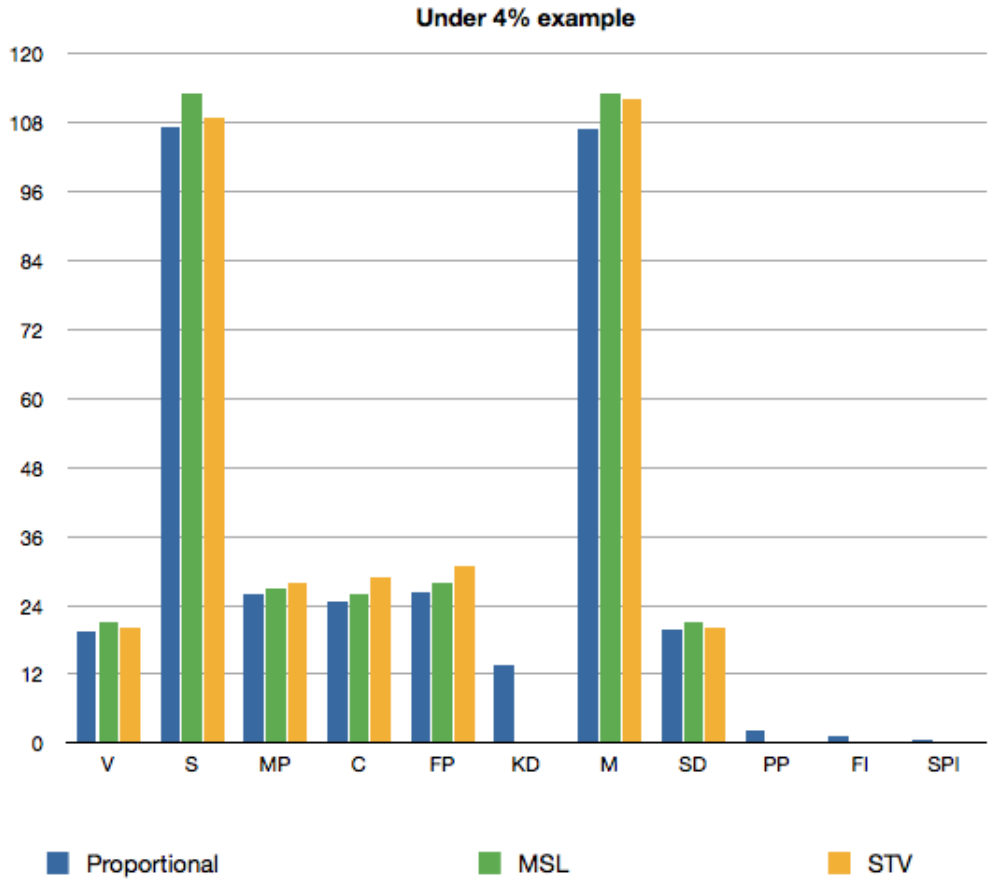

Figure 5: Differences in seat allotment for the sub-threshold example, between completely proportional allotment, the Modified Sainte-Laguë method and the Single Transferable Vote

7 Evaluations of methodology

Though efforts have been made to limit the scope of this essay as much as possible, it is evident that we might have benefitted from narrowing the scope down even further. Comparisons between voting algorithms as far apart as the three we have chosen to evaluate are hard to make, as example data from one election rarely fits the definition of another election system. This is especially evident in the case of trying to make data from Swedish elections fit the Single Transferable Vote method - some quite dramatic redesigns of both algorithm and input data had to be made. Nevertheless, we are content with our solution to these issues, and the result of the algorithmic redesign is a Single Transferable Vote method that possibly could be implemented in future elections in Sweden.

In conclusion: an essay with a significantly narrower scope would have been easier to write, and the implementations and issues that would have emerged from this would probably not have been as challenging as those of this essay. However, the challenges of this essay have proved constructive for answering our question formulation.

8 Conclusions

8.1 Modified Sainte-Laguë

8.1.1 Implementation

The Modified Sainte-Laguë method is relatively easy to implement in a standard election. Votes only need to be counted once, and can theoretically be discarded once the result has been tallied in a correct way. The seat distribution, as it depends on a well-defined mathematical formula, is unambiguous. The Java implementation within the scope of this essay correctly reflects this.

8.1.2 Outcome

Contrasting the outcome from the examples run through MSL with the outcome of a completely proportional seat allotment, one might notice a tendency on the part of MSL to favor larger parties over smaller parties. This is one of the key features of the Modified Sainte-Laguë method, as previously described in section 3.1.3, and our experiments correctly demonstrate this.

8.2 Single Transferable Vote

8.2.1 Implementation

Implementing a STV system in an actual election is not trivial. Votes must be saved until the election is complete, or their ranking lists have been exhausted (such as when the last person on a vote ranking list is eliminated). It is, again, important to note that the implementations that have been made in the scope of this essay do not correctly reflect the difficulties involved in utilizing such a system. This is because the focus of

our implementations is to estimate results based on probabilities and voting figures, and not actually counting separate votes.

Party version An election implementation of the proposed party version of the Single Transferable Vote method would not fundamentally differ from the unmodified version of the Single Transferable Vote method. Voters would submit a vote consisting of a ranked list - same principle as in all versions of STV - and the votes would be transferred from party to party in almost the same way; the only difference at this stage being that excess votes from already elected parties would not be transferred. After the tally is complete, the seats could then be distributed with a seat distribution method of choice.

Candidate version The candidate version of the STV, being the standard version of the algorithm, is implemented in many parliaments across the world today. As such, the difficulties involved in implementing this version reflect the difficulties of implementing the STV in general - votes must be recounted and correctly transferred from candidate to candidate, which demands a lot of work on part of the election personnel.

8.2.2 Outcome

The outcomes of elections held according to the STV system should be said to accurately reflect to collective will of the voters. Instead of settling for a non-favorite candidate for whom to cast their vote, voters are encouraged to cast their votes completely according to their own preferences.[5]

Party version The outcome of the examples run through the party version of the STV, as defined in this essay, do not differ significantly from the results of the same examples run through the Modified Sainte-Laguë method. One can note a slight tendency of redistributing seats from larger parties to smaller parties, but the shift is so slight that its significance is doubtful.

Again, one must remember that the construction of the numbers that constitute our examples are essentially based on little more than probabilistic guesses. As we previously noted in section 4.1.3, it is highly possible that the preferentially ranked votes of voters in these elections would have looked enormously different from the votes they cast in the non-STV elections.

Candidate version The outcome of the 2008 Maltese election for the fourth district, when run through the STV algorithm, corresponds to the outcome of the same example run through the First-Past-The-Post algorithm. In other words: the five candidates with the most amount of first-preference votes were the five candidates that ended up elected.

It would appear that the implementation of the Single Transferable Vote, in this case, was not actually significant for the outcome of the Maltese election, but it is important to note that the utilization of the STV did make a difference for other Maltese districts in the same election.[2] Due to the fact that the redistribution of votes for other districts

was ambiguous (sometimes eliminating more than one candidate per round, making it impossible to know whose votes were redistributed where), they were not applicable as examples for our algorithms. With more detailed data from these elections, we might have been able to demonstrate the effect of the STV in these cases.

8.3 First-Past-The-Post

8.3.1 Implementation

The First-Past-The-Post method is extremely simple to implement for real-life elections, given the resources to correctly count the votes and sort the results somehow. Our Java implementation also correctly reflects this.

8.3.2 Outcome

The outcome results of the First-Past-The-Post method are obvious, given the final vote figures. As previously stated in section 8.2.2, FPTP and the candidate version of STV managed to attain the same results for the example we have constructed for this essay, although this can hardly be said to be a typical outcome of such a comparison.

9 Further research

In this essay we have suggested an alternative version of the Single Transferable Vote system, which could be a potential replacement of the system currently in use in Sweden. Our work can only be regarded as a pilot study of the effects that the proposed STV alternative would have on the Swedish election process. To really evaluate the effects that the system would have in a Swedish election, one would have to make much more detailed evaluations.

One of the best ways to evaluate the system would be to perform an election exit poll, where respondents get both the questions of what party they chose to vote for and how they would have voted if they had the opportunity to rank candidates in the way that the alternative STV method proposes. Of course, this would require much work and planning since every respondent would have to be properly educated in how the system works. Furthermore, such a study would have to have a substantial amount of respondents from widely different demographic groups to really give an answer to the question of how the method could change the voting process.

References

[1] Balinski M, Young P. Fair Representation: Meeting the ideal of one man, one vote. Washington, D.C: Brookings Institution Press; 2001,

[2] Elections in Malta: The Single-Transferable-Vote System in Action, 1921 - 2009. District Results, Count by Count. [cited 2012 Apr 11]. Available from: http://www.maltadata.com/divs.htm

[3] Wikipedia. Sainte-Laguë method. [updated 2012 Apr 6; cited 2012 Apr 11]. Available from: http://en.wikipedia.org/wiki/Sainte-Laguë_method

[4] Benoit K. Which Electoral Formula Is the Most Proportional? A New Look with New Evidence. Political Analysis. 2000 [cited 2012 Apr 6]; 8(4):381-388. Available from: http://polmeth.wustl.edu/analysis/vol/8/PA84-381-388.pdf

[5] Tideman N. The Single Transferable Vote. Journal of Economic Perspectives. 1995 [cited 2012 Apr 11]; 9(1):27-38. Available from: http://www.jstor.org/stable/2138352

[6] Sveriges riksdag. Så arbetar ledamöterna. 2011 [cited 2012 Apr 11]. Available from: http://www.riksdagen.se/sv/Sa-funkar-riksdagen/Sa-arbetar-ledamoterna/

[7] Wikipedia. Personval. [updated 2012 Apr 6; cited 2012 Apr 11]. Available from: http://sv.wikipedia.org/wiki/Personval

[8] Statistiska centralbyrån. Flest personröster på manliga kandidater. 2011 [cited 2012 Apr 11]. Available from: http://www.scb.se/Pages/PressRelease____310892.aspx

[9] Wikipedia. Issues affecting the Single Transferable Vote. [updated 2011 Jul 5; cited 2012 Apr 11]. Available from: http://en.wikipedia.org/wiki/Issues_affecting_the_Single_Transferable_Vote

[10] Wikipedia. First-past-the-post voting. [updated 2012 Mar 25; cited 2012 Apr 11]. Available from: http://en.wikipedia.org/wiki/First-past-the-post_voting

[11] Wikipedia. Sveriges riksdag. [updated 2012 Mar 19; cited 2012 Apr 11]. Available from: http://sv.wikipedia.org/wiki/Sveriges_riksdag

[12] Valmyndigheten. Val till riksdagen - Röster. 2010 [cited 2012 Apr 11]. Available from: http://www.val.se/val/val2010/slutresultat/R/rike/index.html

[13] Sveriges Television. Riksdagsvalet 2010 Valu. 2010 [cited 2012 Apr 11]. Available from: http://svt.se/content/1/c8/02/15/63/14/ValuResultat2010_100921.pdf

[14] SPI – Sveriges Pensionärers Intresseparti. Partiprogram. 2010 [cited 2012 Apr 12]. Available from: http://spipartiet.org/partiprogram.html

[15] Statistiska centralbyrån. Allmänna val, valundersökningen - Andel röstande på partierna efter kön och ålder. 2011 [cited 2012 Apr 11]. Available from: http://www.scb.se/Pages/TableAndChart____272993.aspx

[16] Sifo Research International. Sifos Telefonbuss 2006. 2006 [cited 2012 Apr 11]. Available from: http://www.feministisktinitiativ.se/downloads/grasrot/sifo_060905.pdf

[17] Wikipedia. Table of voting systems by country. [updated 2012 Feb 1; cited 2012 Mar 28]. Available from: http://en.wikipedia.org/wiki/Table_of_voting_systems_by_nation

[18] Wikipedia. Regeringen Reinfeldt. [updated 2012 mar 29; cited 2012 Apr 11]. Available from: http://sv.wikipedia.org/wiki/Regeringen_Reinfeldt

10 Appendix

Code example 10: Java implementation of First-Past-The-Post

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;

/**
 * A class that takes election input data and calculates the winning
     candidates,
 * according to the First Past The Post voting algorithm.
 * @author Meidi Tõnisson
 *
 */
public class FirstPastThePost {
  private static int[] votes; // the set of votes for each party,
      where votes[i] is the number of votes for the i:th party
  private static int PARTIES; // the total number of parties
  private static int SEATS = 5; // the total number of seats to be
      filled
  private static HashMap<Integer,String> partynames; // a HashMap
      mapping party index to party name

  /**
   * This main method should be fed the filename of a file
     containing the following:
   * On the first line, an integer N denoting the number of parties
     in the election.
   * On the N following lines, N integers denoting the number of
     votes the party received
   * in the election, followed by the name of the party.
   *
   * @param args The input file name.
   * @throws IOException
   * @throws NumberFormatException
   */
  public static void main(String[] args) throws
      NumberFormatException, IOException {
    partynames = new HashMap<Integer,String>();
    BufferedReader in = new BufferedReader(new FileReader(args[0]));
    PARTIES = Integer.parseInt(in.readLine());
    votes = new int[PARTIES];
    int acc = 0; // will contain the total number of votes
    for(int i = 0; i<votes.length;i++) { // initialize votes
      String[] line = (in.readLine()).split(" ");
      votes[i] = Integer.parseInt(line[0]);
      partynames.put(i, line[1]);
      System.out.println("Votes for " + line[1] + "(party " + i + ")
          : " + votes[i]);
      acc+=votes[i];
    }
```

```
    int count = 0;
    while(count<SEATS) { // fill each seat
      int elected = max(votes);
      System.out.println(partynames.get(elected) + " has been
          elected, with " + votes[elected] + " votes, which is " +
          ((double)((double)votes[elected]/(double)acc))*100.0 + "%
          of the total vote.");
      votes[elected] = -1; // don't reuse this data
      count++;
    }
  }

  /**
   * Returns the index of the maximum element in the int array.
   * @param votes
   * @return maxIndex
   */
  private static int max(int[] votes) {
    double max = -1;
    int maxIndex = -1;
    for(int i = 0;i<votes.length;i++) {
      if(votes[i]>max) {
        max = votes[i];
        maxIndex = i;
      }
    }
    return maxIndex;
  }

}
```

Code example 11: Java implementation of Modified Sainte-Laguë

```
import java.io.*;
import java.util.HashMap;

/**
 * A class that takes election input data and calculates the seat
     distribution
 * between the parties, according to the Modified Sainte-Laguë
     voting algorithm.
 * @author Meidi Tõnisson
 *
 */
public class ModifiedSainteLague {

  private static int[] votes; // the set of votes for each party,
      where votes[i] is the number of votes for the i:th party
  private static int[] seats; // the set of seats for each party,
      where seats[i] is the number of seats for the i:th party
  private static final int SEATS = 349; // the total number of
      available seats
  private static int PARTIES; // the total number of parties
  private static HashMap<Integer,String> partynames; // a mapping of
```

```
    party index to party name

/**
 * This main method should be fed the filename of a file
     containing the following:
 * On the first line, an integer N denoting the number of parties
     in the election.
 * On the N following lines, N integers denoting the number of
     votes the party received
 * in the election, followed by the name of the party.
 *
 * @param args The input file name.
 * @throws IOException
 * @throws NumberFormatException
 */
public static void main(String[] args) throws
    NumberFormatException, IOException {
  BufferedReader in = new BufferedReader(new FileReader(args[0]));
  partynames = new HashMap<Integer,String>();
  PARTIES = Integer.parseInt(in.readLine());
  seats = new int[PARTIES];
  votes = new int[PARTIES];
  for(int i = 0; i<votes.length;i++) { // initialize votes
    String[] line = (in.readLine()).split(" ");
    votes[i] = Integer.parseInt(line[0]);
    partynames.put(i, line[1]);
    System.out.println("Votes for " + partynames.get(i) + "(party
        " + i + "): " + votes[i]);
  }

  double[] quotients = new double[PARTIES]; // a set of quotients
      for each party
  for(int j = 0; j<SEATS; j++) { // for every seat
    for(int i = 0;i<votes.length;i++) { // calculate quotients for
        each party
      if(seats[i] == 0) { // the first quotient is calculated with
          a different divisor
        quotients[i] = (double)((double)votes[i]/(1.4));
      } else {
        quotients[i] = (double)((double)votes[i]/(((double)seats[i
            ]*2.0)+1.0));
      }
    }
    seats[max(quotients)]++; // the party with the greatest
        current divisor gets the current seat
  }

  for(int i = 0;i<seats.length;i++) { // print the final result
    System.out.println("Seats for " + partynames.get(i) + ": " +
        seats[i]);
  }
}
```

```
  /**
   * Returns the index of the maximum element in the double array.
   * @param quotients
   * @return
   */
  private static int max(double[] quotients) {
    double max = -1;
    int maxIndex = -1;
    for(int i = 0;i<quotients.length;i++) {
      if(quotients[i]>max) {
        max = quotients[i];
        maxIndex = i;
      }
    }
    return maxIndex;
  }

}
```

Code example 12: Java implementation of Single Transferable Vote

```
import java.io.BufferedReader;
import java.io.*;
import java.io.IOException;
import java.util.*;

/**
 * A class that takes election input data and calculates the
    redistribution of votes
 * between the parties according to the Single Transferable Vote
    algorithm.
 *
 * @author Meidi Tõnisson
 *
 */
public class SingleTransferableVote {
  // an integer matrix, with element (i,j) denoting the probability
     that an i-voter
  // will vote for the j-party. the diagonal elements denote the
     number of votes currently
  // given to the party: -1 if they have been eliminated.
  private static double[][] votes;
  private static final int SEATS = 349; // the number of seats to be
     filled
  private static int VOTES; // the total number of votes cast in the
     election
  private static int PARTIES; // the total number of parties
  private static boolean party = true; // whether the party version
     or the candidate version should be run
  private static boolean DROOP = true; // whether the Droop vote
     quota should be used or not. If not, the vote quota will
     default to 4%
  private static HashMap<Integer,String> partynames; // a HashMap
     mapping index values to party names
```

```
/**
 * This main method should be fed the filename of a file
     containing the following:
 * On the first line, an integer N denoting the number of parties
     in the election.
 * On the N following lines, N integers denoting the number of
     votes the party received
 * in the election, followed by the name of the party, followed by
     the vote transition probabilities.
 *
 * @param args The input filename and the output filename.
 * @throws IOException
 */
public static void main(String[] args) throws IOException {
  BufferedReader in = new BufferedReader(new FileReader(args[0]));
  String outfile = args[1]; // the output will be written to a
      file that can be given as input to the Modified Sainte-Laguë
       class
  partynames = new HashMap<Integer,String>();
  PARTIES = Integer.parseInt(in.readLine());
  int[] firstvotes = new int[PARTIES]; // the set of first-
      preference votes for each party, where firstvotes[i] is the
      number of first-preference votes for the i:th party
  votes = new double[PARTIES][PARTIES]; // initialize vote matrix
  int acc=0; // will contain the total number of votes
  for(int i = 0; i<PARTIES;i++) {
    String[] line = (in.readLine()).split(" ");
    firstvotes[i] = Integer.parseInt(line[0]); // the first
        integer on every line denotes the number of firsthand
        votes cast for a certain party
    votes[i][i] = firstvotes[i]; // the diagonal elements denote
        the current number of votes for the party
    acc+=firstvotes[i];
    partynames.put(i, line[1]);
    System.out.println("First-preference votes for " + partynames.
        get(i) + "(party " + i + "): " + firstvotes[i]);
    if(line.length>2) { // if preferences have been supplied for
        the party
      for(int index = 2;index<line.length;index++) {
        if(index-2!=i) { // don't save preferences for the party
            itself
          votes[i][index-2] = Double.parseDouble(line[index]);
        }
      }
    }
  }
  VOTES = acc;
  int limit = voteLimit(SEATS,VOTES);
  System.out.println("Vote threshold: " + limit);
  int[] currvotes = new int[PARTIES];
  currvotes = firstvotes;
  int round = 1; // to keep track of recount rounds
```

```
HashMap<Integer,Boolean> electedParties = new HashMap<Integer,
    Boolean>();
if(party) {
  System.out.println("Running party version.");
  boolean finished = false; // in order to enter while loop
  while(!finished) {
    System.out.println("Recount round "+round);
    finished = true;
    int tmp = min(currvotes); // find party with least amount of
        votes
    if(currvotes[tmp]<limit&&tmp!=-1) { // do they fall below
       the threshold?
      finished = false; // make sure all redistributions have
          been carried out before finishing
      System.out.println(partynames.get(tmp)+": "+currvotes[tmp
          ]);
      int leftover = currvotes[tmp];
      currvotes[tmp] = -1; // eliminated
      System.out.println("The " + leftover + " votes from " +
          partynames.get(tmp) + " were divided:");
      for(int k = 0;k<PARTIES;k++) {
        if(k==tmp||currvotes[k]==-1) { // don't redistribute
            votes to the party itself or to eliminated parties
          continue;
        }
        double probability = (double)votes[tmp][k]/100.0; //
            vote transition probability
        double surplusVotes = (double)leftover; // the number of
             "leftover" votes
        currvotes[k] += Math.floor(probability*surplusVotes);
        System.out.println(Math.floor(probability*surplusVotes)
            + " (" + probability + "*" + surplusVotes + ") votes
             to " + partynames.get(k));
      }
      continue;
    }
    round++;
  }
} else {
  System.out.println("Running candidate version.");
  int elected = 0; // the number of elected candidates
  while(elected!=SEATS) {
    System.out.println("Recount round "+round);

    // redistribute leftover votes from elected parties

    int tmp = max(currvotes); // find party with most amount of
        votes
    if(currvotes[tmp]==limit) { // no redistribution, but they
        are elected
      electedParties.put(tmp, true);
      elected = electedParties.size();
      System.out.println(partynames.get(tmp)+": "+currvotes[tmp
```

```
        ]);
    }
    if(currvotes[tmp]>limit) { // set as elected and
        redistribute leftover votes
      electedParties.put(tmp, true);
      elected = electedParties.size();
      System.out.println(partynames.get(tmp)+": "+currvotes[tmp
          ]);
      int surplus = currvotes[tmp]-limit;
      currvotes[tmp] = limit;
      System.out.println("The surplus " + surplus + " votes from
           " + partynames.get(tmp) + " were divided:");
      for(int k = 0;k<PARTIES;k++) {
        if(k==tmp||currvotes[k]==-1||electedParties.get(k)!=null
            ) { // don't redistribute votes to the party itself,
              to eliminated parties or to elected parties
          continue;
        }
        double probability = (double)votes[tmp][k]/100.0;
        double surplusVotes = (double)surplus;
        currvotes[k] += Math.floor(probability*surplusVotes);
        System.out.println(Math.floor(probability*surplusVotes)
            + " (" + probability + "*" + surplusVotes + ") votes
             to " + partynames.get(k));
      }
      continue;
    }
    tmp = min(currvotes); // find party with least amount of
        votes
    if(currvotes[tmp]<limit&&currvotes[tmp]!=-1) { // are they
        below the threshold?
      System.out.println(partynames.get(tmp)+": "+currvotes[tmp
          ]);
      int leftover = currvotes[tmp];
      currvotes[tmp] = -1;
      System.out.println("The " + leftover + " votes from " +
          partynames.get(tmp) + " were divided:");
      for(int k = 0;k<PARTIES;k++) {
        if(k==tmp||currvotes[k]==-1||electedParties.get(k)!=null
            ) { // don't redistribute votes to the party itself,
              to eliminated parties or to elected parties
          continue;
        }
        double probability = (double)votes[tmp][k]/100.0;
        double surplusVotes = (double)leftover;
        currvotes[k] += Math.floor(probability*surplusVotes);
        System.out.println(Math.floor(probability*surplusVotes)
            + " (" + probability + "*" + surplusVotes + ") votes
             to " + partynames.get(k));
      }
      continue;
    }
    round++;
```

```
    }
  }

  System.out.println("Recount finished. Final result:");
  if(party) {
    for(int i = 0; i<PARTIES;i++) {
      if(currvotes[i] != -1)
        System.out.println(partynames.get(i) + ": " + currvotes[i]
            + " votes");
      else {
        System.out.println(partynames.get(i) + ": ELIMINATED");
      }
    }
  } else {
    for(int i = 0; i<PARTIES;i++) {
      if(electedParties.get(i) != null)
        System.out.println(partynames.get(i) + ": ELECTED");
      else {
        System.out.println(partynames.get(i) + ": ELIMINATED");
      }
    }
  }

  FileWriter fstream = new FileWriter(outfile);
  BufferedWriter out = new BufferedWriter(fstream);
  out.write(PARTIES + "\n");
  for(int i = 0;i<PARTIES;i++) {
    out.write(currvotes[i] + " " + partynames.get(i) + "\n");
  }
  out.close();

}

/**
 * This method returns the number of votes needed in order to be
    considered "elected". This
 * is either done by calculating the Droop quota, or by using the
    4% quota.
 * @param SEATS The total number of seats.
 * @param votes The total number of votes.
 * @return The number of votes needed in order to be considered
    elected.
 */
private static int voteLimit(int SEATS, int votes) {
  if(DROOP) {
    return (int)Math.floor((double)((double)votes/((double)SEATS
        +1)) + 1);
  } else { // default to 4% threshold
    return (int) Math.floor((double)((double)votes*0.04));
  }
}

/**
```

```
    * Returns the index of the smallest non-negative value in the
        array.
    * @param votes
    * @return minIndex
    */
  private static int min(int[] votes) {
    double min = Integer.MAX_VALUE;
    int minIndex = -1;
    for(int i = 0;i<votes.length;i++) {
      if(votes[i]<min&&votes[i]!=-1) {
        min = votes[i];
        minIndex = i;
      }
    }
    return minIndex;
  }

  /**
   * Returns the index of the greatest value in the array.
   * @param votes
   * @return maxIndex
   */
  private static int max(int[] votes) {
    double max = -1;
    int maxIndex = -1;
    for(int i = 0;i<votes.length;i++) {
      if(votes[i]>max) {
        max = votes[i];
        maxIndex = i;
      }
    }
    return maxIndex;
  }
}
```