Political optimization Strategies in Plurality Elections

Jackson Gyurisin

August 2024

1 Introduction

Perhaps a more informative but uninteresting title for this project would have been "Political optimization strategies in a 3-candidate system with mandatory voting and 2 fixed candidates". It would've been a subtitle if Overleaf wasn't such a confusing program.
sigh
In this paper I will explain different types of voting systems and how political candidates change their voting preferences in order to win elections (voting systems).

2 Context

This project is an expansion on some of Dr. Dragovic's previous work involving social choice in elections. Social choice is the idea that people have the power of choice. In a democracy this means that each voter has to ability to vote for candidate A, candidate B, candidate C... and henceforth (*CODEE Journal*) It is not just candidate A, like it would be in a monarchy. What Dr. Dragovic's research was primarily focused on was looking at what happens in a political population where there are two equal and opposite political views, and voters have the choice to not vote either candidate if they do not agree with either enough (American Mathematical Monthly) The question was: How does a candidate shift their political position in order to get the most votes and win the election?
For my project I be using the same idea that there are two equal and opposite political views, however, for the sake of simplicity I will assume that everyone votes in my elections. And for reasons that will become clear soon,

I will also be using three candidates in my research instead of two, which Dr. Dragovic and her colleagues used.

3 Election Systems

Something else to be mentioned before I get into my project.
My project is involved with different types of eletion systems. What is an election system? Well, I'm glad you asked.
An election system is how you count up the votes cast by the voters. The most simple method is known as the *Plurality method*. In this method, each voter gets one vote that they get to give to one candidate. Whichever candidate gets the most amount of votes wins the election. For example say that we have candidates Alice, Beatrice and Carl with 11, 10 and 12 votes respectively:

Alice	Beatrice	Carl
11	10	12

In our Plurality system it is obvious that Carl wins this election. However there is a system of voting out there that seems more fair, and it is known as *preference voting* (Hobart and William Smith Colleges) In this system each voter gets to vote who their first place choice would be, their second choice, third, and so on. It is more "fair" as even if a voter's first place candidate isn't selected there is a chance that their next favorite candidate will be selected. The type of election system decides how these secondary votes count. Let's take the Borda method as our first example:

3.1 Borda Method

Suppose we have our candidates Alice, Beatrice and Carl again. Let's say there are four groups of people who vote homogenously; that is, they vote the same way within their own group. Let's say the first group, now to be called a *bloc*, thinks that Alice should win the election, Beatrice would be a good follow up, and actively dislike Carl. On their ballots then they put the slip:

Alice	1
Beatrice	2
Carl	3

And... say there are eleven people who vote this way. Now say there are 6 people who want Beatrice to win, then Alice, then Carl. Their slips would look like:

Alice	2
Beatrice	1
Carl	3

The third bloc also votes that Beatrice should be first, but that Carl should be second, and Alice third (4 people). Their slip:

Alice	3
Beatrice	1
Carl	2

And finally the fourth bloc, with 12 people, who vote:

Alice	3
Beatrice	2
Carl	1

The final tally looks like:

	11	6	4	12
Alice	1	2	3	3
Beatrice	2	1	1	2
Carl	3	3	2	1

Now we get to adding up votes. Since we are using the Borda method, we will take each rank (A number in the same row as a candidate) a candidate received and then multiply it by the number of people who voted for them in that position. We then add up all those values, which is the candidate's final score. The goal, in this election, is to have the lowest score. So for this election:
Alice has: 11(1)+6(2)+4(3)+12(3) = 71 (points)
Beatrice has: 11(2)+6(1)+4(1)+12(2) = 56 (points)
Carl has: 11(3)+6(3)+4(2)+12(1) = 71 (points)
So Beatrice wins this election since she has the least amount of points.

3.2 Plurality with Runoff (Hare Method)

Now say we wanted to count votes a different way. Instead, we find out who has the least amount of first-place votes, eliminate them from the election, and give their votes to the next candidate the voters in those blocs vote for. Suppose we use the voting table from 3.1. Then we can eliminate Beatrice as she has the least amount of first-place votes (10 votes). We then give the votes to those blocs' second places; we do this by subtracting '1' from every rank in a column with Beatrice as the first rank. The table:

	11	6	4	12
Alice	1	~~2~~ 1	~~3~~ 2	3
Beatrice	2	~~1~~ 0	~~1~~ 0	2
Carl	3	~~3~~ 2	~~2~~ 1	1

Now, again, we add all of the first place votes together. This time Alice gets 11 + 6 = 17 votes, and Carl gets 4 + 12 = 16 votes. So Alice wins if we are using this type of election.

My research project will be using the Plurality with Runoff method. I demonstrated the Plurality and Borda method to show that even though candidates can receive the same number of votes in different systems it can change who wins. (Hobart and William Smith Colleges)

4 The Project

The purpose of this project is to determine, if there are three candidates, how they would move in order to win an election. We will assume that a Plurality with Runoff election is being used (It has been used in Minneapolis Mayoral elections and in Australia to give a few examples) (Hobart and William Smith Colleges; mggg/Votekit) and that all voters will vote. Suppose we have the bimodal graph:

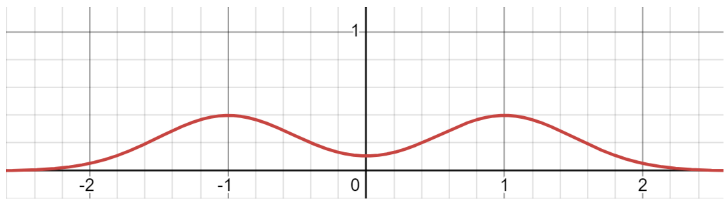

Figure 1: Bimodal Graph

The x-axis here represents the political spectrum, the more negative you are the more "left leaning" you are, and the more positive you are the more "right leaning" you are. Yes, it oversimplifies political views, but many social choice researchers are keeping it this way. And I'm only in my undergrad, so...

The y-axis stands for how many voters there are in that particular political stance. You will notice in this graph there are local maxima at -1 and 1, so most voters are moderately on either side of the spectrum. (The American Mathematical Monthly)

4.0.1 Example 1

Now suppose we have candidates whose political alignments are -1.5, 0.25, and 1. We will call these candidates A,B, and C, respectively. Then the graph will look like:

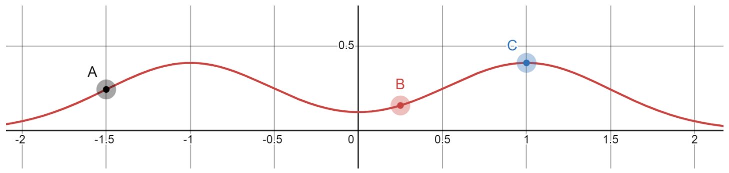

Figure 2: Political Positions

Now to find out how many votes each candidate will receive we will let the area under the curve be a percentage of the total number of votes. Which means we need to take an integral for each candidate. For this project I assumed that if you are closer to candidate X politically you would vote for them. So to find out who is closest to who I found the midpoints between A & B and B & C, which both, incidentally, happen to be at -0.625 and 0.625, respectively. The graph looks like:

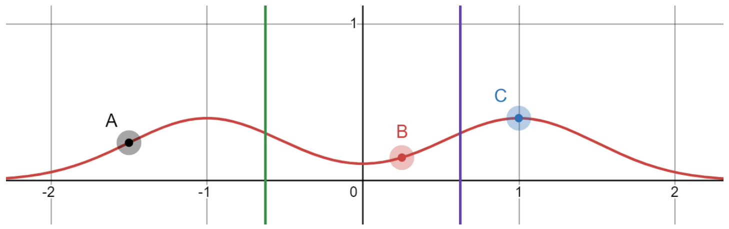

Figure 3: Positions + Midpoints

Thus, in this situation, the number of votes A gets is $\int_{-\infty}^{-0.625} f(x)dx$, B: $\int_{-0.625}^{0.625} f(x)dx$, and C: $\int_{0.625}^{\infty} f(x)dx$.

(with f(x) being the bimodal function.)

The integrals end up giving A 39% of the vote, B 22% of the vote, and C 39% of the vote. Using Plurality with Runoff we can see that B will be eliminated in this scenario. How then do we split up B's voting blocs between A and C? We do another midpoint.
If a voter is closer to A inside of B then that will be the new upper bound for A's integral. Otherwise the voter is closer to C, and that portion will go to C. Another graph:

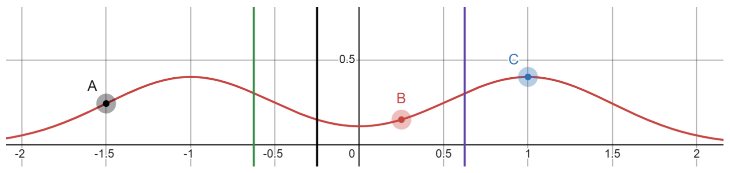

Figure 4: Midpoint AC added (black)

With the black line being the midpoint between A and C (@ -0.25). A and C both still have 39% of the vote, but the integrals that we have to add to them now, respectively [I love that word] are $\int_{-0.625}^{-0.25} = 8\%$ and $\int_{-0.25}^{0.625} = 14\%$.
And $39 + 8 < 39 + 14$, so C is the winner in this election.

4.0.2 Example 2

Now suppose that A isn't happy with their lot in life, they somehow know that they are going to lose the election. So they make some changes to their political opinion to secure more voters. They decide to move further right along the spectrum until they are more moderate at -0.9. Now the midpoint of A & B is at -0.325, and AC (how I will refer to midpoints from now on) stayed at 0.625. If we redo the integrals with new bounds we find that A starts with 46% of the vote, B with 15% of the vote, and C with 39% of the vote. B loses again, so we have to find the midpoint between A and C again, which this time is at 0.05.

$\int_{-0.325}^{0.05} = 5\% + 46\% = 51\%$

Since A received more than 50% of the vote we automatically know that they win this election (they cannot be last and will have more than anyone else). Take note as well that B lost and AC was greater than the median, 0.

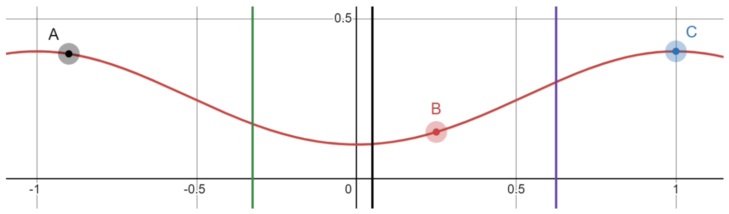

Figure 5: AC at 0.05

5 Winning Ranges

I stated at the beginning of this project that a more informative title would have included "2 fixed candidates". So we will have one candidate move along the function one at a time, namely A and B. C doesn't need to move since the graph is symmetric, so A = C. First a few facts about the bimodal function. The formula I used was

$$\frac{e^{-2(x-1)^2}+e^{-2(x+1)^2}}{\sqrt{2\pi}}$$

For this variation of a bimodal graph 50% of the area under the curve is between -1 and 1. This means that there is another 25% from -infinity to -1 and 1 to infinity. If A and C are at -1 and 1 then B is somewhere between them (B is defined as the middle candidate). If B is at 0 then the midpoints are at -0.25 and 0.25 and B will win $\approx$ 16% of the vote. A and C are already guaranteed 25% of the vote, so B loses. If B gets super close to A such that the distance between them is practically 0 then the midpoint between B and C will be almost at 0. And half of that area is 25%. But because B can never actually reach A B will never have 25%, so B is guaranteed to lose. Now that that is out of the way we can move on.

5.0.1 Candidate "A" moves

We will assume for this section that A is the opportunistic candidate and is moving along the x-axis. Where A is guaranteed to win is determined by where C and B are on the bimodal function.

If C ≤ 1 then A will win in the range from (-C, B). We know that B will lose the election as long as A doesn't go past -1. Then it is a competition between A and C. If the midpoint is less than zero then C wins, and if greater then A wins. We know the formula for a midpoint is $\frac{x+y}{2}$, so

$\Rightarrow \frac{A+C}{2} > 0 \Rightarrow A + C > 0 \Rightarrow A > -C$
The closer A moves to B the less votes B will get, and still helps move the midpoint above 0. But $A \neq B$ so that is why I keep open parentheses instead of square brackets. And if A = -C then there would be a tie, which we won't consider.

If C > 1 $\wedge$ BC > S Now suppose that C is greater than 1. Now B has a chance to move outside of -1 to 1 and win the election. But first lets establish what "S" is.

If each candidate were to receive an equal amount of 100 votes they would each get $33.\overline{3}$ percent of the votes. If a candidate gets more than that percent then they are guaranteed to survive to the next round, as they will have beaten some other candidate. "S" is the x-value for which $\int_S^\infty f(x)\,dx$ is equal to $33.\overline{3}$. Which, if you are wondering, is about 0.7849.
Back to voting. If BC is above S, then C is not guaranteed to survive that first round. If C does still beat B, then A needs the same range to win like the C $\leq$ 1 range. If B beats C however, then A needs to get the midpoint between it and B over 0. Hence the furthest left A can go is -B (but not equal -B). And since B is more moderate than C, by definition, -B > -C. So a guaranteed win range for A here is (-B, B).

If C > 1 $\wedge$ BC < S Otherwise suppose that C > 1 $\wedge$ BC < S. Again, for A, it is advantageous to get as close to B as possible, to win over as many left-wingers as possible. But if A wants to see how far left on the spectrum they can get without risking the win, they need to make sure that their midpoint with B is at least -S. So, $\frac{A+B}{2} > -S \Rightarrow A + B > -2S \Rightarrow A > -(B + 2S)$
And since BC < S, C is also guaranteed to not get out the first round. So B will be knocked out. *Now*, in order for A to win, the midpoint of A and C must be greater than 0. So A must also be greater than -C. Our final logical statement for the winning range is: $[A > -(B + 2S)] \wedge [A > -C] \wedge [A < B]$

5.0.2 Candidate "B" moves

Candidate B's behavior, when moving, is more dicey than when A moves. A has the entire left hand side of the graph to consider, while B has A and C to consider. Again, if A and C are at -1 and 1 respectively then B has no chance. However, let's say that C stays fixed and A moves further to the left. If B is greater than -C and manages to have more area than A, then B will win. With C at 1 this range happens to be -C < B < -(2+A). I got this by splitting the negative side of the graph in half, with half the area below -1, and the other between -1 and 0. If AB is below -1 then B will beat A for certain. The same also happens to apply for C<1, but it gets harder for B to stay in a range that exists unless A becomes an extreme leftist.

Now let's say that A and C are both beyond their respective "1"s. Then B has a better chance. However, they will have to be more careful about where they are positioned. Say we fix C to some value just beyond 1, say 1.2, and A is *very* extreme. A is so far out there that they will get virtually no points. Then B wins from (-C, C). But suppose A starts to creep closer to the median.
Things start to get tricky.
As A gets closer the AB midpoint moves up as well. If B stays close to A without violating -C then the midpoint will stay low. However, if B moves far enough right then the AB midpoint will be greater than -S. So either B or C will be eliminated in the first round, and as long as the AC midpoint is also below S, then B will not make it. So B has to get as close to C as possible to get that midpoint up. And strange enough, as $|A| < C$, B has to start following the opposite strategy, that is, B must be below -A instead and still keep those midpoints up.
Wow. That was a lot. Maybe a summary:

Say we have two arbitrary candidates, X and Y. Both are beyond the peaks of the bimodal graph on opposite sides of the median. Let's also say that $|X| > |Y|$. Then B needs to either get as close to X as possible without crossing -Y, *or* B needs to get close enough to Y such that BY is greater than S.

If $|X| < |Y|$ then $X \Rightarrow Y$ and $Y \Rightarrow X$

The closer X and Y get to their peaks the smaller the winning range B has.
The further they get from the peaks the bigger the winning range is for B.

5.0.3 A final note on winning ranges

In a two candidate system where everyone votes, the most ideal spot to be is as close to the other candidate as possible, hopefully closer to the median than them. In this election and voter spread it seemed that, in general, getting as close to another candidate was usually the best play as well. A moving closer to B was always an good option. For B, being really close to *one* of the candidates was helpful. Perhaps mandatory voting always encourages candidates to move closer?

6 Future Work

If I were to continue this project into the future the first thing I would like to work on is smoothing out the winning ranges for A and B. The code I wrote to show me the winning ranges didn't provide all the data I needed. I would have to revise it to give me more data such as: "Where were the midpoints for each pair of candidates?" or "Who lost the first round, and with what percent?". I had to go do those by hand, which instilled in me a mortal fear that there was a confounding variable that I missed when determining the winning ranges, because all I had was the winner of the election. Another thing I would tweak with the code would be to make it more continuous. The code would find who won when the moving candidate was at x position, and then incremented it by 0.05 repeatedly. I think the increment could be smaller next time. Lastly the code couldn't tell you if there was a tie. That should be fixed as even an increment off could skew the patterns I was looking for.

If I get past the code, some other things I would like to explore is what would happen if there were more than 3 candidates? How would that change everyone's positioning, as those all-important midpoints would increase tenfold (figuratively).
How would positioning change if I used a different election system? For example, the Borda system wouldn't eliminate candidates and reassign integral bounds, instead it would have to find every single voting bloc using midpoints, and then have to multiply those areas by their rankings to different candidates. How would that affect how a candidate moved?
I can think of other ideas (such as two candidates moving), but those are a little above my pay-grade. Maybe when I get a Masters.

References

[1] Christoph Börgers, Bruce Boghosian, Natasa Dragovic, and Anna Haensch. A blue sky bifurcation in the dynamics of political candidates. *The American Mathematical Monthly*, 0(0):114, 2023

[2] Christoph Börgers, Natasa Dragovic, Anna Haensch, Arkadz Kirshtein, and Lilla Orr. Odes and mandatory voting. *CODEE Journal*, 17, 2024

[3] Hobart and William Smith Colleges. Math 110: An introduction to the Mathematics of Voting. New York, Geneva.

[4] mggg/Votekit. Chris Donnay. Available from: https://github.com/mggg/VoteKit/blob/main/notebooks/4_election_systems.ipynb

7 Python Code; Ranges for 2 fixed candidates in a Bimodal Plurality with Runoff System:

```
#print("The weight is how many people are voting in this manner
                                    (called a 'bloc'). We will
                                   assume that the canidates have
                                   simplified names, starting with
                                    A, which is in position 1, B
                                   in position 2, and so forth.
                                   When inputting postions however
                                    write it as a column
                                   preference vote would be
                                   written, for example, if there
                                   are 3 canidates and a 'bloc'
                                   likes C the most, A second, and
                                    B last the order of numbers
                                   would look like: 2,3,1. That is
                                    how you input into the 'bloc's
                                   .\n")
from scipy.integrate import quad
import math
num_canidates = 3 # int(input("How many canidates are there?"))
num_rnds = num_canidates-1
num_blks = 4 # int(input("How many voting blocks are there?\n")
                                   )
blocs = {}
#positions = [0]*num_canidates
positions = [-1.5,-0.3,1.3] #For fixing positions in the tests.
fixed_variable_positions = [-2,-1.9,-1.8,-1.7,-1.6,-1.5,-1.4,-1
                                   .3,-1.2,-1.1,-1]
#[-1.4,-1.3,-1,-.9,-.8,-.7,-.6,-.5,-.4,-.3,-.2,-.1,0,.1,.2,.3,.
                                   4,.5,.6,.7,.8,.9,1,1.1,1.2,1.3,
                                   1.4]
candidate_moving = 1
fixed_variable_candidate = 0
increment = 0.05
```

```
if candidate_moving == 0:
    starting_point = -3
else:
    starting_point = positions[candidate_moving-1]+increment


end_point = positions[candidate_moving+1]
positions[candidate_moving] = starting_point




#
                                ######################################################

def zero_temp_voting_matrix():
    global temp_voting_matrix
    temp_voting_matrix = []
    for u in range(num_canidates):
        temp_voting_matrix.append(0)


def loser_value ():
    u = 999999999999
    for r in temp_voting_matrix:
        if r < u and r != 0:
            u = r
    return u


def find_ones_position (a_matrix):
    for r in a_matrix:
        if r == 1:
            return a_matrix.index(r)

def bimodal (x):
    return ((math.e)**(-2*(x+1)**2) + (math.e)**(-2*(x-1)**2))/
                                    ((2*(math.pi))**0.5)

def unimodal (x):
    return (math.exp(-(x**2)/2)/(2*(math.pi))**0.5)

#def Cauchy (x):
#    return (1/((math.pi)*2((x**2)+0.25)))

#
                                ######################################################
```

```
#range_to_move_over = [starting_point]
#for u in range (int(abs(starting_point-end_point)//increment))
                                  :
#     range_to_move_over.append(round(range_to_move_over[u]+
                                  increment, 2))

for fixed_position in fixed_variable_positions:

    positions[fixed_variable_candidate] = fixed_position

    if candidate_moving == 0:
        starting_point = -3
    else:
        starting_point = positions[candidate_moving-1]+
                                        increment
    print("The fixed candidate's(",chr(65+
                                    fixed_variable_candidate),"
                                    ) position is:",
                                    fixed_position)


    end_point = positions[candidate_moving+1]
    range_to_move_over = [starting_point]
    for u in range (int(abs(starting_point-end_point)//
                                    increment)):
        range_to_move_over.append(round(range_to_move_over[u]+
                                        increment, 2))
    winners_ranges = []



    for element in range_to_move_over:
        positions[candidate_moving] = element

        in_between_positions = [0]*(num_blks+1)
        in_between_positions[0] = float('-inf')#-10
        in_between_positions[1] = (positions[0]+positions[1])/2
        in_between_positions[2] = (positions[0]+positions[2])/2
        in_between_positions[3] = (positions[1]+positions[2])/2
        in_between_positions[4] = float('inf')#10

        integral_values = [0]*num_blks
        for u in range (num_blks):
            integral_values[u] = quad(bimodal,
                                            in_between_positions
                                            [u],
                                            in_between_positions
                                            [u+1])
```

```
for bloc_number in range(num_blks):
    canidate_preferences = [[1,2,3],[2,1,3],[3,1,2],[3,
                                   2,1]]
    blocs['Bloc '+ str(bloc_number)] =[integral_values[
                                   bloc_number][0],[]]
    #for u in range(num_canidates):
        #canidate_preferences.append(int(input("What is
                                           the number in
                                           the " + str(u+1
                                           ) +" position
                                           ?")))
    blocs['Bloc '+ str(bloc_number)][1] =
                                   canidate_preferences
                                    [bloc_number]



#
                                ##############################################

mat_all = []

for u in range(num_blks):
    mat_all.append(blocs['Bloc '+str(u)])

canidates_dict = {}
for u in range(num_canidates):
    canidates_dict[u] = chr(u+65)


elimination_matrix = []
for u in range(num_canidates):
        elimination_matrix.append(0)
#
                                ##############################################


for rounds in range(num_rnds):

    zero_temp_voting_matrix()

    for r in mat_all:
        for u in range(num_canidates):
            if r[1][u] == 1:
                temp_voting_matrix[u] += r[0]
        #print("For round", rounds+1, "the score is: ",
                                        temp_voting_matrix)

        losers_score = loser_value()

        for r in temp_voting_matrix:
            if r == losers_score:
                losers_position = temp_voting_matrix.index(
                                                r)
                elimination_matrix[losers_position] += 1


        for r in mat_all:
            if r[1][losers_position] == 1:
                for u in range(num_canidates):
                    r[1][u] = r[1][u] -1
                while elimination_matrix[find_ones_position
                                                (r[1])] > 0
                                                :
                    for u in range(num_canidates):
                        r[1][u] = r[1][u] -1
    #
                                    ##############################################


    for r in temp_voting_matrix:
            if r == max(temp_voting_matrix):
                winners_position = temp_voting_matrix.index
                                                (r)


    #print("Winner:", canidates_dict[winners_position],
                                    "!", canidates_dict[
                                    candidate_moving],"'s
                                    Position:", element)
    winners_ranges.append([winners_position,element])
#
                                ##################################################

filtered_ranges = [winners_ranges[0]]
range_starting_point = winners_ranges[0]
for u in range(len(winners_ranges)):
    if winners_ranges[u][0] != range_starting_point[0]:
        filtered_ranges.append(winners_ranges[u-1])
        filtered_ranges.append(winners_ranges[u])
        range_starting_point = winners_ranges[u]
filtered_ranges.append(winners_ranges[len(winners_ranges)-1
                                ])
    for u in range(len(filtered_ranges)-1):
        if filtered_ranges[u][0] == filtered_ranges[u+1][0]:
            if filtered_ranges[u][1] == filtered_ranges[u+1][1]
                                                             :
                print(canidates_dict[filtered_ranges[u][0]],"
                                                               won at",
                                                               filtered_ranges
                                                               [u][1])
            else:
                print(canidates_dict[filtered_ranges[u][0]],"
                                                               won in the
                                                               range of",
                                                               filtered_ranges
                                                               [u][1],"to",
                                                               filtered_ranges
                                                               [u+1][1])

    print("\n",canidates_dict[candidate_moving],"was moving.\n"
                                         ,"Switching to new position
                                          for fixed candidate \n\n\n
                                         ")
```