Multi-Objective Optimisation Applied to the Electoral Redistricting of Ireland

Eliza Somerville, Alexander Farren,
Ben McGloin and Mikey Whelan

31 August 2023

Contents

1
Introduction

In 2023, the Electoral Commission of Ireland was tasked with preparing a set of recommendations on possible changes to the constituency boundaries of Ireland. In doing so, they were required satisfy numerous criteria stipulated by the Irish Constitution, pertaining to features such as population per representative, contiguity of constituencies, and temporal continuity. The large number of objectives involved in the design of the constituency boundaries means that this is a prime use case for *multi-objective optimisation*.

Multi-objective optimisation is the process of finding a solution to a problem that best satisfies multiple objectives. While a wide variety of techniques are available for single-objective optimisation, there are comparatively fewer methods available for multi-objective optimisation. One example of such a method is an evolutionary algorithm, which involves generating solution candidates and then evolving them towards a more optimised solution [1].

In this project, an evolutionary algorithm was implemented in Python with the aim of generating a configuration of Irish constituencies that better satisfied the criteria laid out by the Constitution.

1.1 Background

Many representative democracies use electoral systems where a territory is split into a number of small regions called electoral districts. These can be considered the building blocks of *constituencies*, which are specified regions that may elect a certain number of representatives.

In these systems, the arrangement of constituencies can significantly impact election results. Indeed, one can sufficiently manipulate the map so as to favour a certain political party; this action is referred to as *gerrymandering*. The word was first used in Boston in reaction to the redrawing of constituencies through a bill signed by Elbridge Gerry, which favoured the Democratic-Republican party. The redistricting was said to resemble a mythological salamander, hence coining the term.

The problem of *redistricting* involves drawing a map of constituencies such that all members of the population are fairly represented in the electoral system. In this paper we focus on redistricting the constituencies of Ireland, using an evolutionary algorithm devised to optimise the configuration according to the objectives set out in the Constitution of Ireland.

1.2 Electoral Districting in Ireland

On 9 February 2023, a new state body called the Electoral Commission was established to oversee elections in Ireland [2]. One of the key roles of the Electoral Commission is reviewing

the the Dáil Éireann constituencies, and making a report and recommendations in relation to possible changes to constituency boundaries.

Basic Terminology

- The national parliament of Ireland is referred to as *Dáil Éireann*, or simply the Dáil.
- An elected representative sitting in the Dáil is referred to as a *Teachta Dála* (TD).
- Ireland is divided up into 3,440 regions known as *electoral divisions* (EDs).
- These EDs are currently split between 39 *constituencies*. The current configuration of constituencies in Ireland is shown in Figure 1.1. The constituency boundaries can be changed by transferring EDs from one constituency to another. The number of constituencies is not fixed and may be altered in a redrawing of boundaries.
- Ireland is also divided into 26 *counties*, which are administrative regions governed by bodies called county councils. The location of constituency boundaries is independent of the position of county boundaries, but it is considered desirable for constituencies to conform to county boundaries as far as is practicable. Larger counties are typically split into several constituencies, and some constituencies breach county boundaries to an appreciable extent, as illustrated in Figure 1.2.

In this project, we define the redistricting problem to mean redrawing the boundaries of constituencies such that criteria stipulated by the Irish Constitution and the Electoral Commission are satisfied. These requirements are explained below.

Constitutional Requirements

When deciding on any changes to the current constituency boundaries, the Electoral Commission is required to observe the following provisions of the Constitution [2, 3]:

- Article 16.2.2°of the Constitution provides that:
  The number of members shall from time to time be fixed by law, but the total number of members of Dáil Éireann shall not be fixed at less than one member for each thirty thousand of the population, or at more than one member for each twenty thousand of the population.
- Article 16.2.3°of the Constitution provides that:
  The ratio between the number of members to be elected at any time for each constituency and the population of each constituency, as ascertained at the last preceding census, shall, so far as it is practicable, be the same throughout the country.

Summary of Criteria

In determining the constituency boundaries of Ireland, the Electoral Commission to satisfy the requirements just mentioned, as well as taking several other criteria into account. The full set of criteria are the following [2]:

(i) Based on recent population figures, to satisfy Article 16.2.2°of the Constitution, the total number of members of the Dáil shall not be less than 171 and not more than 181 (compared to 160 currently).

(ii) Each constituency shall elect 3, 4 or 5 members.

(iii) The breaching of county boundaries shall be avoided as far as it is practicable. (The extent to which this is satisfied by the current configuration is illustrated in Figure 1.2.)

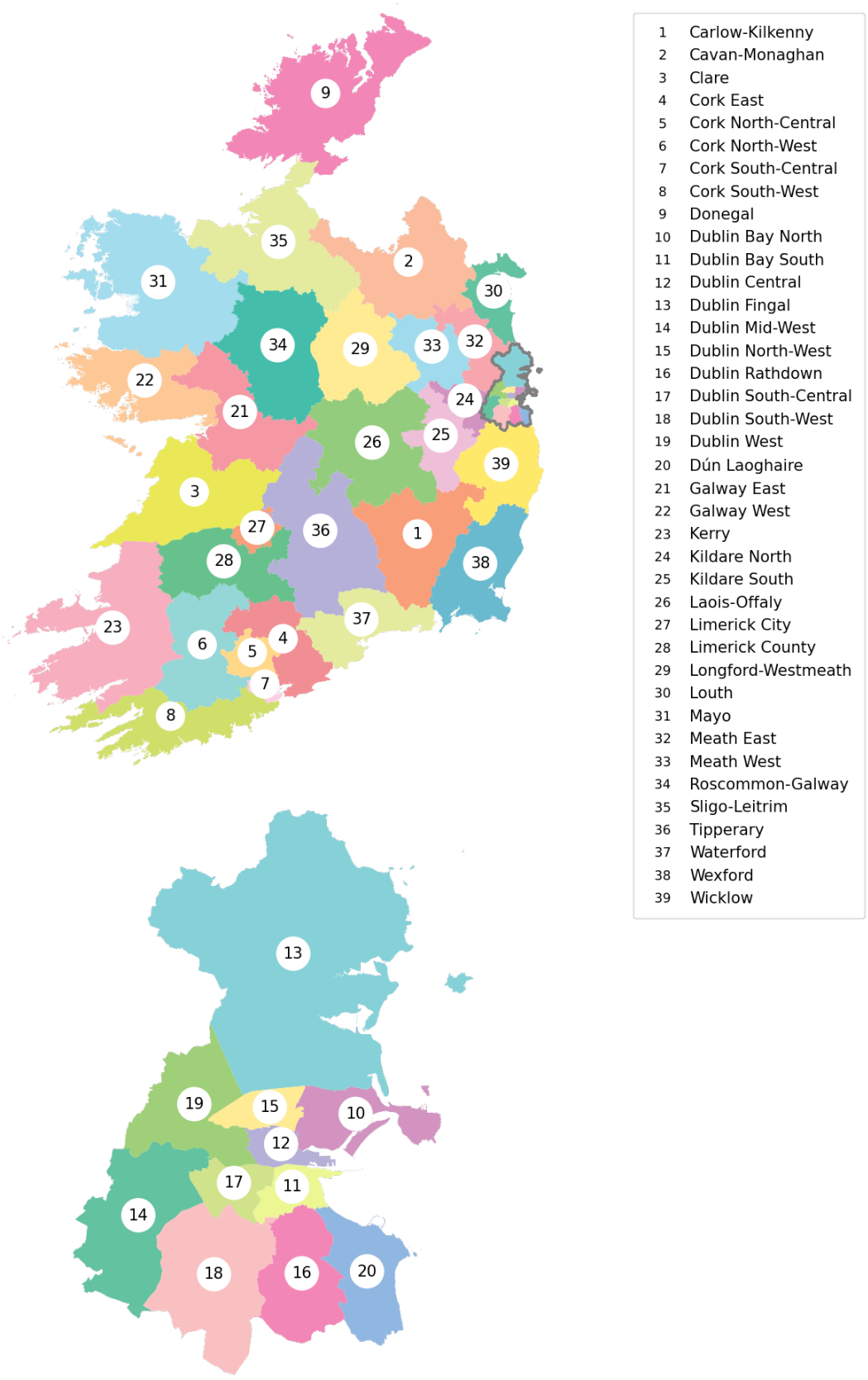


Figure 1.1: The current configuration of constituencies.

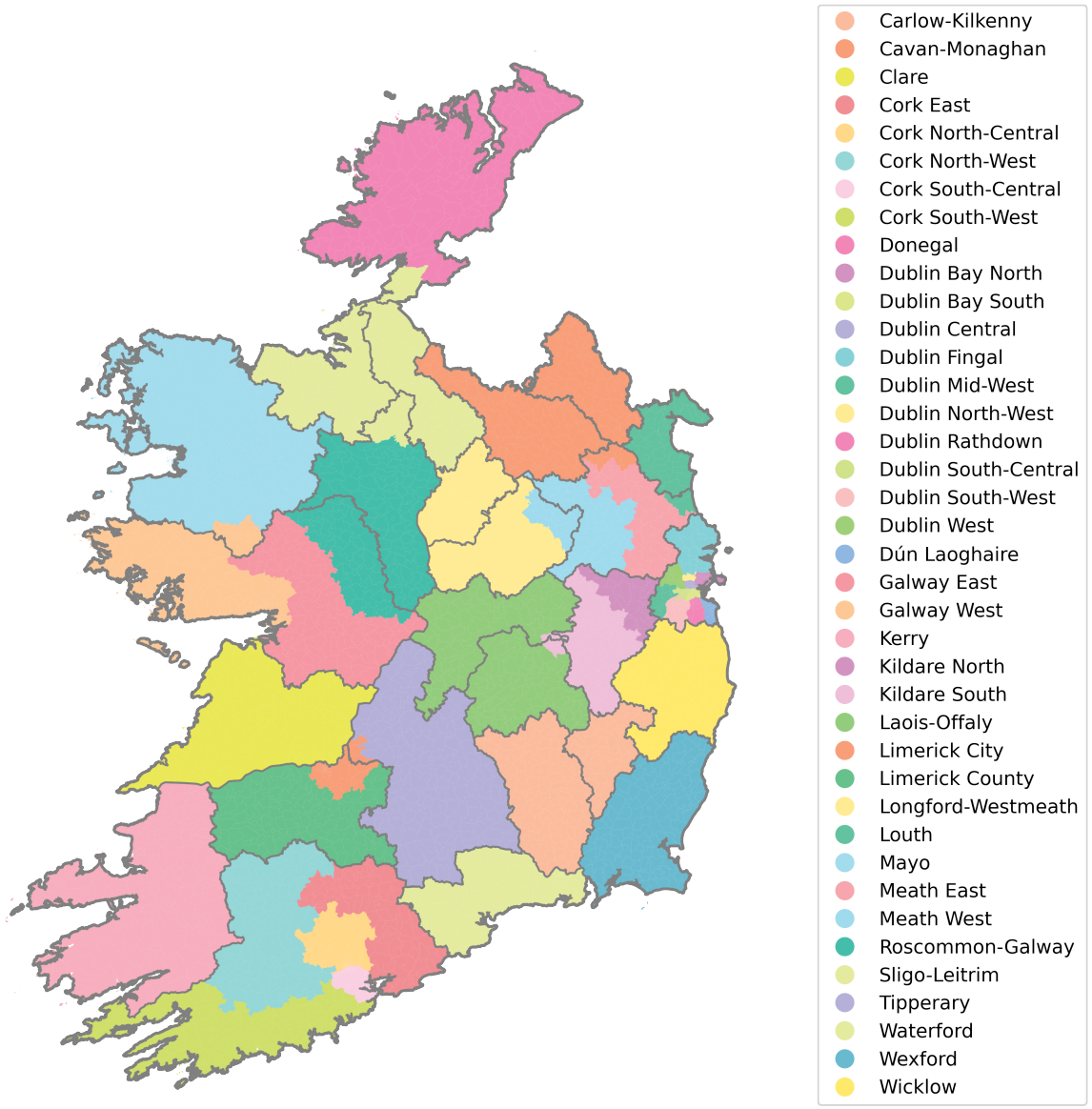


Figure 1.2: The current configuration of constituencies, overlaid with county boundaries shown as grey lines.

(iv) Each constituency shall be composed of contiguous areas.

(v) There shall be regard to geographic considerations including significant physical features and the extent of and the density of population in each constituency.

(vi) Subject to the above matters, the Commission shall endeavour to maintain continuity in relation to the arrangement of constituencies.

Note that adherence to (iv) is considered a priority, because it is a 'hard' requirement of the Constitution, and is designed to minimise frustration amongst voters.

The large number of constraints imposed by the Constitution mean that determining the optimal constituency boundaries is a challenging problem in multi-objective optimisation. In this project, numerical methods were implemented in order to find a solution which better satisfied the requirements and recommendations set out above.

1.3 Definitions and Nomenclature

In this section, the definitions of various terms and metrics used throughout the report are given. The metrics are based on those used in [4, 5].

National Ratio

The *National Ratio* of Ireland is defined as

$$\text{National Ratio} = \frac{\text{Recorded Census Population}}{\text{Number of Dáil Seats}}. \tag{1.1}$$

This metric represents the number of people each TD would represent in a scenario with perfectly equal representation.

Seat Equivalent Representation

The *Seat Equivalent Representation* (SER) of a constituency is defined as

$$\text{SER} = \frac{\text{Population}}{\text{National Ratio}}. \tag{1.2}$$

It quantifies the number of Dáil seats deserved by a constituency based on its population size. To satisfy the requirements of the Irish Constitution, constituency boundaries should be chosen such that the SER of each constituency is very close to an integer between 3 and 5, so that the SER is very close to the actual number of seats allocated to that constituency.

Variance from the National Average

In order to ensure equality of representation, the metric of *Variance from the National Average* (VNA) is defined:

$$\text{VNA} = \frac{\text{SER} - \text{Assigned Seats}}{\text{Assigned Seats}}. \tag{1.3}$$

Positive and negative values correspond, respectively, to SER values above and below the national average; ideally all constituencies would have a VNA of zero. Past Constituency Commission reports such as [5] have chosen a VNA of $\pm 5\%$ as an acceptable threshold value (the Constituency Commission was responsible for recommending changes to constituency boundaries before the establishment of the Electoral Commission in 2023). In this project, we follow this choice and aim to reduce the absolute value of the VNA of all constituencies to below 5%.

Table 1.1 shows the current SER and VNA values of all 39 Irish constituencies.

Table 1.1: The current number of seats allocated to the 39 constituencies, along with their SER and VNA values. VNA values that lie outside the acceptable threshold of ±0.05 are shown in bold.

Constituency	Seats	SER	VNA
Carlow-Kilkenny	5	5.575	**0.115**
Cavan-Monaghan	5	5.048	0.010
Clare	4	4.258	**0.064**
Cork East	4	4.075	0.019
Cork North-Central	4	4.200	0.050
Cork North-West	3	2.989	−0.004
Cork South-Central	4	4.090	0.023
Cork South-West	3	2.826	**−0.058**
Donegal	5	5.286	0.057
Dublin Bay North	5	4.962	−0.008
Dublin Bay South	4	4.066	0.017
Dublin Central	4	3.818	−0.046
Dublin Fingal	5	5.467	**0.093**
Dublin Mid-West	4	3.959	−0.01
Dublin North-West	3	2.423	**−0.192**
Dublin Rathdown	3	3.170	**0.057**
Dublin South-Central	4	4.103	0.026
Dublin South-West	5	4.917	−0.017
Dublin West	4	4.182	0.046
Dún Laoghaire	4	4.146	0.036
Galway East	3	2.993	−0.002
Galway West	5	4.866	−0.027
Kerry	5	5.234	0.047
Kildare North	4	4.523	**0.131**
Kildare South	4	4.212	**0.053**
Laois-Offaly	5	5.399	**0.080**
Limerick City	4	4.150	0.037
Limerick County	3	3.037	0.012
Longford-Westmeath	4	4.455	**0.114**
Louth	5	5.327	**0.065**
Mayo	4	4.418	**0.105**
Meath East	3	3.320	**0.107**
Meath West	3	3.330	**0.110**
Roscommon-Galway	3	3.040	0.013
Sligo-Leitrim	4	4.060	0.015
Tipperary	5	5.405	**0.081**
Waterford	4	4.162	0.040
Wexford	5	5.464	**0.093**
Wicklow	5	5.183	0.037

1.4 Data Aggregation and Preparation

This project was written in Python. The library `pandas` [6] was used for handling data, `geopandas` [7] was used to implement geographic methods such as unions and intersections, and to plot geographical data, and `matplotlib` [8] was used to render maps and plot graphs and charts.

1.4.1 Data Sources

The data used in the project came primarily from Tailte Éireann (formerly Ordnance Survey Ireland). In particular, we used 2019 data on electoral divisions [9], and 2017 data on constituency boundaries [10]. The operation of our algorithm required us to know which constituency each ED belonged to, and in fact this data was not included in the original datasets. We instead had to find this manually, by using the `.representative_point` method of `geopandas` to find a point inside each ED, and then using a spatial join (`sjoin`) to merge the ED and constituency datasets according to the constituency that contained each of these points. This gave us a `GeoDataFrame` which linked EDs and constituencies, and contained a `geometry` column which allowed the EDs to be plotted on a map.

Since it was necessary to quantify the extent to which a configuration breached county boundaries, we also needed geographical data on counties. This was also obtained from a 2019 dataset made available by Tailte Éireann [11]. Finally, we also needed to know the population of each ED. This data was obtained from the Central Statistics Office. At the beginning of the project, only 2018 population data was available, but data from the 2022 census became available in August 2023 [12], and it was this data that was ultimately merged with the geographical data to find the final dataframe. There was no population data for some EDs, so these were assigned zero population.

An example of what was recorded in the dataframe for the ED Bohernabreena is shown in Listing 1.1.

```
ED                                                  BOHERNABREENA
ED_ID                                                      267035
GUID                          2ae19629-1ce0-13a3-e055-000000000001
ESRI_OID                                                        8
CON                                             DUBLIN SOUTH-WEST
CON_ID                                                  260009C
SEATS                                                         5
POPULATION                                                 4496
COUNTY                                                   DUBLIN
PROVINCE                                               LEINSTER
CENTROID_X                                           711258.06
CENTROID_Y                                           720929.15
AREA                                                43938821.49
geometry      POLYGON ((708211.089 725425.627, 708215.62 725...
NEIGHBOURS    [267006, 267159, 267143, 257046, 267083, 26714...
NB_CONS                             [DUBLIN RATHDOWN, WICKLOW]
BOUNDARY                                                      1
CHANGE                                                        0
```

Listing 1.1: An example of a single row of the `GeoDataFrame`.

Some properties of each `ED` need explaining:

- `CON` and `SEATS` detail the current constituency to which `ED` belongs, and the former's

number of elected seats in the Dáil,

- `ED_ID`, `GUID` and `ESRI_ID` are all different identification systems for Irish electoral divisions,
- (`CENTROID_X`, `CENTROID_Y`), `geometry` and `AREA` contain information about the physical position, shape and area of `ED` respectively,
- `NEIGHBOURS` keeps track of the ED IDs of of the electoral divisions with which `ED` shares a border, while `NB_CONS` is the list of `ED`'s neighbouring constituencies,
- `BOUNDARY` is 1 if `ED` has a non-empty `NB_CONS` list or 0 otherwise,
- `CHANGE` is 1 if `ED` has previously been flipped as part of the evolution or 0 otherwise.

1.4.2 Finding Neighbouring EDs

The arrays of neighbouring EDs and constituencies for each ED were created by checking which shared a border using the `geopandas.GeoSeries.touches` feature. This was streamlined by defining a function `find_neighbours`, which is shown in Listing 1.2. This function was applied to the original ED dataset to obtain the `NEIGHBOURS` and `NB_CONS` columns of the dataframe shown above. It was important that each element in these columns was a `numpy` array rather than a Python list. We initially used lists, but quickly encountered errors because lists are mutable and were being incrementally altered in processes that were intended to be independent.

```
    def find_neighbours(df):
        '''
        Finds the neighbours and neighbouring CONs of each ED
        '''
        # Create column containing empty neigbours list for each ED
        df['NEIGHBOURS'] = [[] for i in  range(len(df))]
        df['NB_CONS'] = [[] for i in  range(len(df))]
        df['BOUNDARY'] = [1 for i in  range(len(df))]
        df['CHANGE'] = [0 for i in  range(len(df))]

        # Find neighbours of each ED
        for i in range(len(df)):
            t = df['geometry'][i].touches(df['geometry'].values)
            # Create column of neighbouring EDs
            df.at[i,'NEIGHBOURS'] = df['ED_ID'][t].tolist()
            # Create column of neighbouring CONs
            df.at[i,'NB_CONS'] = list(np.unique((df.loc[t,'CON']).tolist()))
            # Remove self from column
            if df.at[i,'CON'] in df.at[i,'NB_CONS']:
                df.at[i,'NB_CONS'].remove(df.at[i,'CON'])
            # Set type of ED
            if df.at[i,'NB_CONS'] == []:
                # No neighbouring CONs -> interior
                df.at[i,'BOUNDARY'] = 0
            df.at[i,'NEIGHBOURS'] = np.array(df.at[i,'NEIGHBOURS']).astype(str)
            df.at[i,'NB_CONS'] = np.array(df.at[i,'NB_CONS']) .astype(str)

    return df
```

Listing 1.2: The function used to find the EDs and constituencies which neighboured each ED.

It was then possible to identify which EDs had empty `NB_CONS` and their `BOUNDARY` value was changed from the initialised 1 to 0.

Once this function had been called on our aggregated dataframe, we had all the information we needed to proceed with the development of our algorithm.

1.5 Outline

In this project, an evolutionary algorithm was implemented in Python in order to optimise the constituency boundaries of Ireland with respect to the criteria of the Constitution. In defining the algorithm, we identified four main objectives based on our interpretation of the Constitution:

1. **Contiguity:** All constituencies should be composed of contiguous areas, except in the case of offshore islands (and other special cases discussed in Chapter 3).
2. **SER:** Each constituency should have an SER which is close to an integer greater than or equal to 3. (If the SER of an existing constituency is found to be an integer greater than 5, then the constituency can in principle be split into several smaller constituencies with SERs close to 3, 4, or 5. For example, an existing constituency with an SER close to 6 could be split into two 3-seater constituencies.)
3. **County Boundaries:** The constituencies should respect county boundaries as much as possible.
4. **Temporal Continuity:** The number of people living in EDs which have switched to different constituencies should be minimised as much as possible when attempting to satisfy the other three constraints.

Of these, the first two were considered 'primary objectives', and the second two were considered 'secondary objectives'.

The candidate states were generated and optimised using an evolutionary algorithm, which is discussed in Chapter 2. The four main criteria above were used to define a reward function, which allowed the configurations generated by the evolutionary algorithm at each generation to be evaluated and filtered via a process akin to natural selection. This is discussed in Chapter 3. Finally, in Chapter 4, we discuss the results produced by the algorithm, and possible improvements that could be made in future.

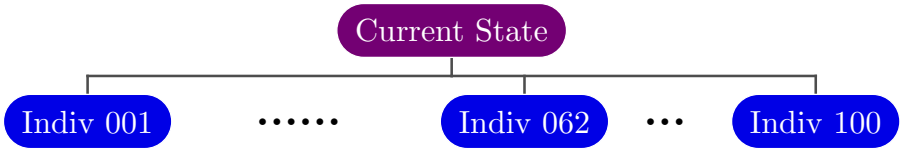
2
Evolutionary Algorithm

We seek to find a solution, i.e. a deviation from the current assignment of electoral divisions to constituencies, which simultaneously achieves several objectives. We will call the status quo the *current state*. Evolution is a similarly motivated process, from which we will take inspiration. In keeping with the lexicon of evolution, we will call an initial state the *parent state* and a member of its succeeding generation an *offspring state*.

2.1 Evolutionary Algorithm

Step 1: Mutation

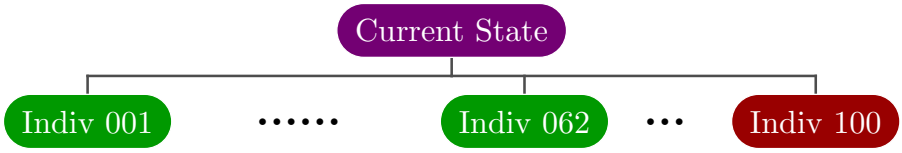
Just like randomly mutated genes introduce new traits into a species, we will trigger a random deviation from the parent state by reassigning some chosen electoral divisions. To preserve contiguity, we only consider *boundary EDs*. These are electoral divisions which share a border with at least one electoral division in a different constituency in the current state. A boundary ED is *flipped* if it is reassigned to a neighbouring constituency.



Starting with the original parent state, the current state, we produce a *generation* of, say, 100 deviated states which we call *individuals*. The first ten individuals differ from their parent by only one boundary ED being flipped. The next ten are produced by two boundary EDs being flipped. This pattern continues until the last ten, for which 10 boundary EDs were flipped. After excluding those which have already been flipped, the selection among boundary EDs is random.

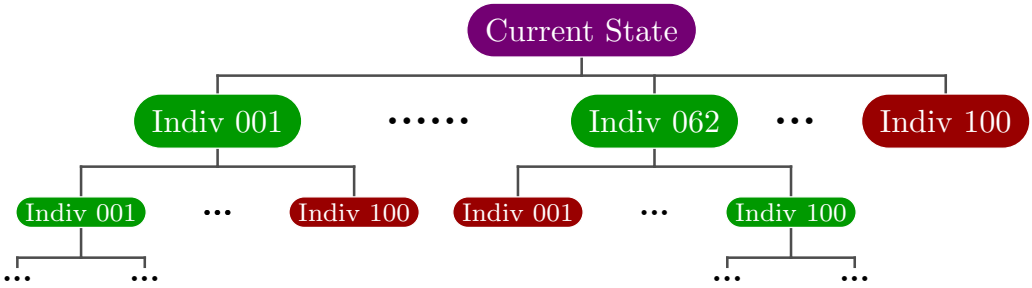
Step 2: Natural Selection

In analogy to evolution involving death of unfavourable traits via natural selection, we will only propagate individuals which tend towards our desired objectives as outlined in 1. To determine the desirability of the individuals of a generation, we first evaluate the reward function (which we will see in 3) for all individuals. Then, for example, the 10 most rewarded individuals are allowed to survive and become new parent states, while the rest are culled from the evolutionary tree.



Step 3: Evolution

This proliferation and culling is repeated until a chosen number of generations is reached. At this stage, we select the most rewarded individual from the tree; we have created a pathway for a more optimal solution. It is possible that not a single offspring of the current state is more rewarded than the latter. In this case, the current state could be a locally optimal solution.



2.2 Code Implementation

The most important step in the evolution is the first step: the flip. A flip is performed on an ED chosen at random from the pool of boundary EDs which have not previously been flipped. Ensuring that the dataframe is accurately updated for the next iteration is necessary to provide a valid description of the state before the next flip. For example, Bohernabreena, Tibradden and

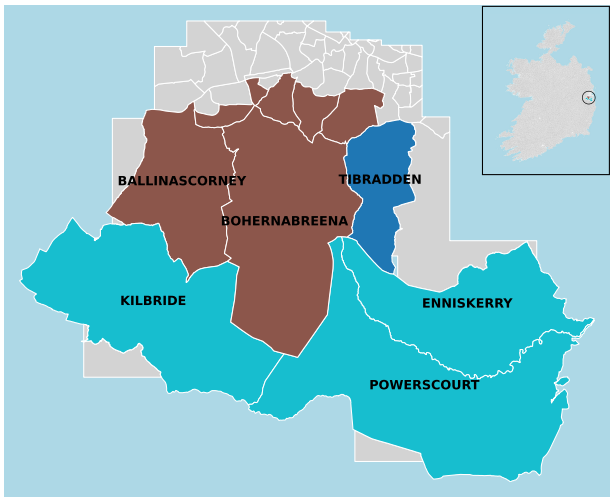


Figure 2.1: Examples of boundary EDs on the border between Dublin South-West, Dublin Rathdown, and Wicklow.

Enniskerry are boundary EDs (see Figure 2.1). In particular, Bohernabreena (Dublin South-West) is touching two other constituencies. This means it could be flipped to either the turquoise (Wicklow) or the blue (Dublin Rathdown) constituency when the current state mutates.

Once the random decision is made, the previous constituency (Dublin South-West) is added to the list of Bohernabreena's neighbouring constituency, and its new `CON` value is removed from that list.

```
    def flip(df_orig):
        '''
        Randomly swaps the CON of a boundary ED.
        '''
        # Make copy of input dataframe
        df = df_orig.copy()

        # Filter the dataframe to contain only boundary EDs
        # which have not previously changed, and have non-zero population
        pool = df[(df['BOUNDARY']!=0)&(df['CHANGE']<1)&(df['POPULATION']>0)]

        # Randomly choose one of the filtered EDs
        i = int(random.choice(pool.index.to_list()))
        print(df.loc[i]['NB_CONS'])
        # Get pre-flip CON of chosen ED
        old_con = df.at[i,'CON']
        # Choose random neighbouring CON of chosen ED
        new_con = random.choice(df.at[i,'NB_CONS'])
        # Update the CON of of the chosen ED; this is the 'flip'
        df.at[i,'CON'] = new_con
        # Update CHANGE to record that this ED has changed
        if df.at[i,'CHANGE'] == 1:
            df.at[i,'CHANGE'] = 2
        else:
            df.at[i,'CHANGE'] = 1
        # Ensure no ED has its own CON as a neighbour
        i0 = np.where(df.at[i,'NB_CONS']==new_con)
        df.at[i,'NB_CONS'] = np.delete(df.at[i,'NB_CONS'],i0)
```

Listing 2.1: The flip function used to switch boundary EDs from one constituency to another.

But we must also update Bohernabreena's neighbours, as they may have outdated `BOUNDARY` and `NB_CONS` values, which could impair the next iteration.

```
    # def flip(df_orig):
    # ----------- continued from above -----------

        # Get an array of indices of neighbouring EDs
        nb_ed_ids = df.at[i,'NEIGHBOURS']
        nb_indices = []
        for n in nb_ed_ids:
            nb_indices.append(df.loc[df['ED_ID']==n].index[0])

        # For each neighbouring ED y, if new_con is not listed in its
    neighbouring
        # CONs, then append it to the list
        for y in nb_indices:
            if new_con not in list(df.at[y,'NB_CONS']):
                df.at[y,'NB_CONS'] = np.append(df.at[y,'NB_CONS'], new_con)

```

```
            # Neighbouring EDs of y
            nb_ed_ids_y = df.at[y,'NEIGHBOURS']
            nb_indices_y = []
            for n2 in nb_ed_ids_y:
                nb_indices_y.append(df[df['ED_ID']==n2].index[0])

            # Count number of neighbours of y which are in old_con
            count = 0
            for z in nb_indices_y:
                if df.at[z,'CON'] == old_con:
                    count += 1
            # If no neighbours of y in old_con, remove old_con from the list of
            # neighbouring CONs of y
            if count == 0:
                i1 = np.where(df.at[y,'NB_CONS']==old_con)
                df.at[y,'NB_CONS'] = np.delete(df.at[y, 'NB_CONS'], i1)

        # Make sure boundary ED's are recorded properly
        for j in range(len(df)):
            if df.at[j,'NB_CONS'].size == 0:
                df.at[j,'BOUNDARY'] = 0
            else:
                df.at[j,'BOUNDARY']=1

    return df
```

Listing 2.2: Updating the neighbours of the flipped ED.

This allows us to complete the first step by producing a generation from the parent state.

```
    def reproduce(df, max_flips, kids):
        '''
        Takes in a parent dataframe df and outputs a list containing <kid> child
        dataframes on which <flips> random flips have been performed.
        '''
        offspring = []
        for j in range(kids):
            kid_data = df.copy()
            for i in range(int((j/kids)*max_flips)+1):
                kid_data = flip(kid_data)
            offspring.append(kid_data)
    return offspring
```

Listing 2.3: The function used to generate offspring states from the parent state.

Once we have rounded up the newly-born offspring, we may proceed with the second step; *culling*. To reduce the exponentially growing storage of individual states, we only kept a set of the best **<keep>** individuals of each generation.

```
    def kill(offspring, keep):
        '''
        Takes in a list of child dataframes, computes the reward function
        for each, and outputs a list with entries
        [child dataframe, corresponding reward]
        for the <keep> best children.
        '''
        chopping_block=[]
```

```
        for x in offspring:
            kid_data = x.copy()
            # Compute rewards
            r = reward(kid_data)
            chopping_block.append([x,r])
        # Sort by rewards and retain dataframes with <keep> highest rewards
        the_chosen_ones = sort_array(chopping_block)[:keep]
    return the_chosen_ones
```

Listing 2.4: The function used to cull offspring states with the lowest rewards.

The chosen, most rewarded offspring are allowed to reproduce themselves. This is the third step; evolution. To limit computation time, we looked at a three-generation example with `<kids>` individuals per generation.

```
    def evolve(df_orig, flips, kids, keep):
        '''
        Evolve original state to find improved state.
        '''
        df = df_orig.copy()
        # Create parents
        parents_and_rewards = kill(reproduce(df, flips, kids), keep)
        # Initialise global_best
        global_best = sort_array(parents_and_rewards)

        # Main evolutionary loop
        i = 1
        for parent_and_reward in parents_and_rewards:
            # Get parent df
            parent = parent_and_reward[0]
            # Find children
            children_and_rewards = kill(reproduce(parent, flips, kids), keep)
            j = 1
            for child_and_reward in children_and_rewards:
                # Update global_best
                global_best = compare(child_and_reward, global_best, keep)
                # Get child df
                child = child_and_reward[0]
                # Print status update
                print(f'Parent {i}, Child {j}')
                # Find grandchildren
                gchildren_and_rewards = kill(reproduce(child, flips, kids), keep
    )
                k = 1
                for gchild_and_reward in gchildren_and_rewards:
                    k += 1
                    # Update global_best
                    global_best = compare(gchild_and_reward, global_best, keep)
                j += 1
            i += 1

        final_states = list(map(list, zip(*global_best)))[0]
        final_rewards = list(map(list, zip(*global_best)))[1]

    return optimal_state[0:3], optimal_reward[0:3]
```

Listing 2.5: The main function used to evolve the current configuration.

The ten best (highest-reward) individuals across generations are stored in an array `global_best` until the evolutionary tree is stopped. We then return the overall three best states as a list of dataframes, for their fulfillment of the objectives from Chapter 1 to be analysed (see Chapter 4).

3
Reward Function

The configurations of constituencies generated by the evolutionary algorithm were evaluated using a reward function. This function accepted a configuration X as an input, and returned a non-negative number which described how well the configuration satisfied the constraints of the problem.

The first task of the reward function was to eliminate any configurations with non-contiguous constituencies. This constraint was imposed by defining the reward function to return zero immediately if X was found to contain any non-contiguous constituencies. Once this constraint had been imposed, the remainder of the reward function comprised three components, each of which corresponded to one of the criteria involved in the problem:

$$\text{Reward}(X) = C_{\text{SER}} f_{\text{SER}}(X) + C_{\text{cont}} f_{\text{cont}}(X) + C_{\text{CB}} f_{\text{CB}}(X), \tag{3.1}$$

where $C_{\text{SER}}, C_{\text{cont}}$ and C_{CB} are, respectively, the reward function components corresponding to the criteria for SER, temporal continuity, and respect of county boundaries, and $C_{\text{SER}}, C_{\text{cont}}$ and C_{CB} are tuneable parameters. In our algorithm, we prioritised the optimisation of the SER of all constituencies aboved the continuity and county boundary criteria, so we chose C_{SER} to be larger than C_{cont} and C_{CB}.

The implementation of each of the criteria in the reward function is discussed in detail below.

3.1 Contiguity

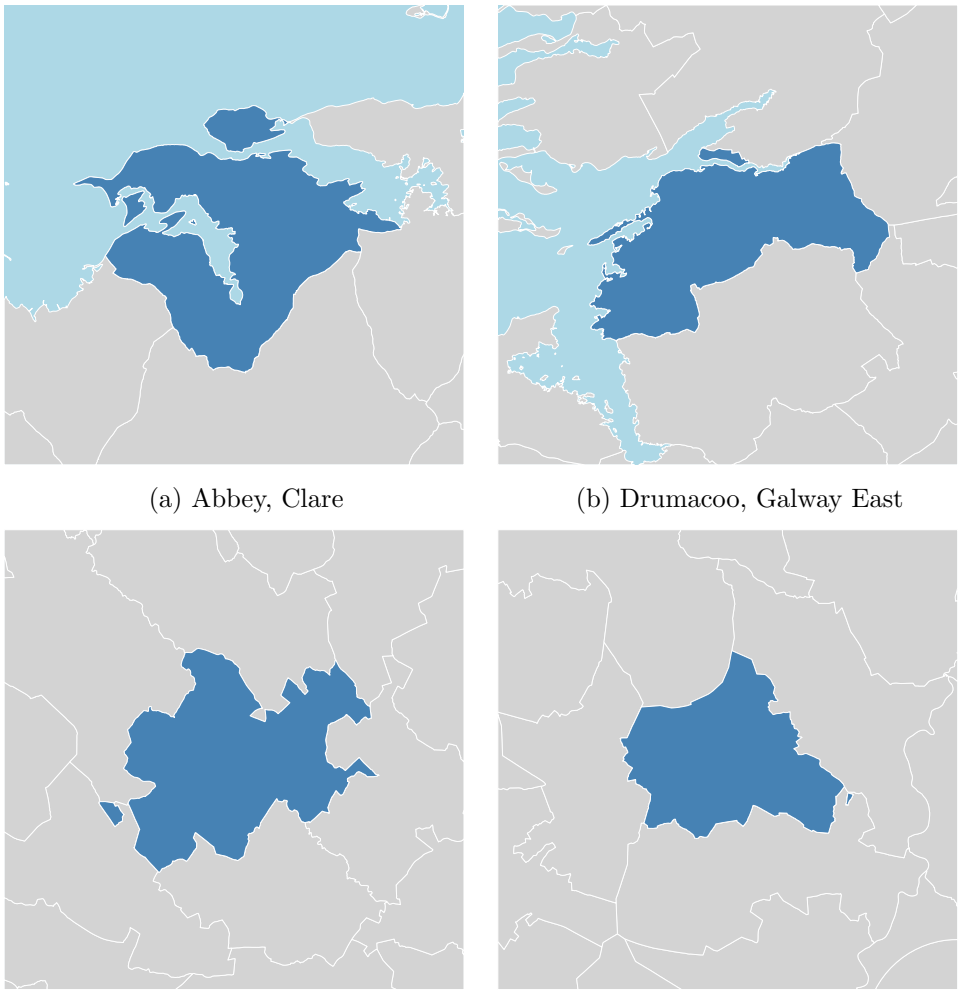
The contiguity of all constituencies was implemented as a hard constraint, meaning that configurations that contained any non-contiguous constituencies were automatically given zero reward, regardless of how well they satisfied the other three constraints of the problem.

Checking the contiguity of a constituency was one of the most challenging aspects of the definition of the reward function. Initially, a geometric check was implemented using `geopandas`, which involved taking the union of all EDs in a given constituency and checking whether the result was a single `Polygon`, in which case the constituency was contiguous, or a `MultiPolygon`, in which case the constituency was non-contiguous. In order to implement this, all islands needed to be removed from the dataset, including both the eight wholly-island EDs, and also all smaller islands which form part of EDs with components on the mainland. This was achieved by taking the union of all ED geometries in the entire dataset, and finding the largest `Polygon` in the resulting geometry. The dataset of EDs was then clipped to this polygon using an intersection method, which had the effect of removing all islands from the dataset.

However, when the geometric contiguity check was performed for the current configuration of constituencies, four constituencies were still found to fail the test. Upon further inspection, it was discovered that four boundary EDs were in fact non-contiguous. These anomalous cases are

shown in Figure 3.1. Although these constituencies are technically not contiguous, the extent of the non-contiguity is minor. Since these breaches are considered acceptable, it is desirable that the current configuration should pass the contiguity check. Each of the four offending EDs consisted of two non-contiguous components. To circumvent the problem, in each case the smallest of the two components was simply deleted from the dataset for the duration of the analysis.

This ad-hoc method was successful in allowing the program to run as intended. However, the presence of other EDs with very small non-contiguous components in the dataset may have meant that some configurations which would actually be considered acceptable were rejected on the basis of their minor non-contiguity. In addition, the geometric unions involved in this type of contiguity check were very time-consuming, and this function accounted for the vast majority of the program's execution time. It was therefore highly desirable to find a different method of testing for contiguity.



(a) Abbey, Clare

(b) Drumacoo, Galway East

(c) Moynalty, Meath East

(d) Downings, Kildare North

Figure 3.1: The four problematic non-contiguous boundary electoral divisions.

A significant speed increase was achieved by switching this contiguity check to an alternative method using the graph theory algorithms included in the library `networkx` [13]. This method involved converting the set of EDs in a given constituency to a set of nodes in a graph, where neighbouring EDs were joined by edges of the graph. The function `networkx.is_connected`

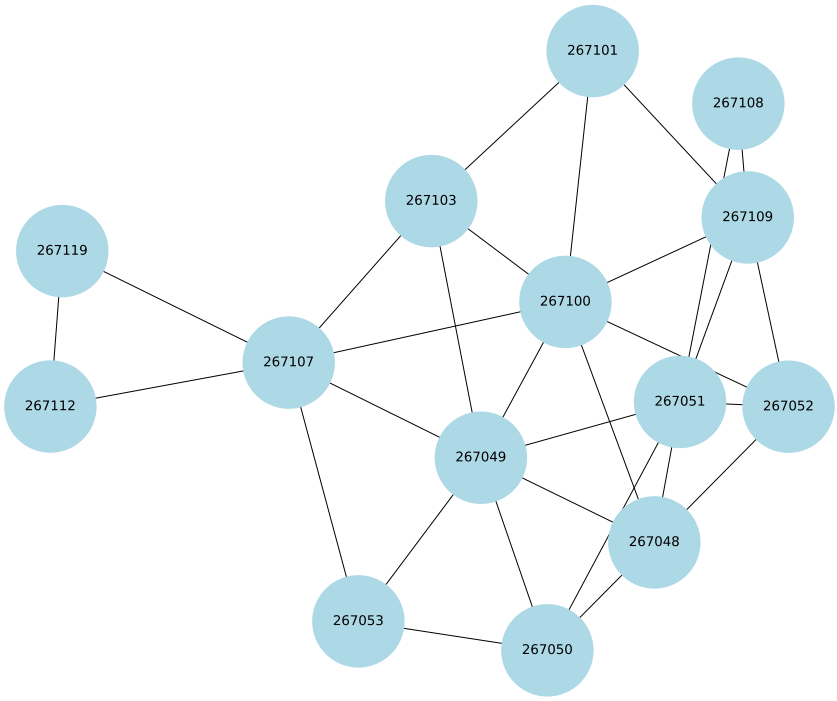


Figure 3.2: A graph of the EDs in the Dublin Mid-West constituency, with nodes labelled by ED ID and edges connecting neighbouring EDs.

was used to determine whether the graph was connected, in which case the constituency was verified to be contiguous. In this case, only the wholly-island EDs needed to be removed from the dataset, and the issues arising from the presence of non-contiguous EDs were completely avoided. This change reduced the total execution time to less than a third of its original value.

To further increase the efficiency of this component of the reward function, the contiguity check was only carried out for

1. Constituencies which had lost or gained at least one ED via a flip, and
2. Constituencies that neighboured at least one flipped ED.

Since the initial configuration contained only contiguous constituencies, checking only the contiguity of these 'changed' constituencies was sufficient to ensure global contiguity. In addition, before the creation of the `networkx` graph, a simpler preliminary check was implemented whereby the reward function instantly returned zero if any constituency contained an isolated ED, which is defined to be an ED with no neighbours in its own constituency. The whole algorithm is shown in Listing 3.1.

```
def f_contiguity(df):
    '''
    Checks whether all constituencies in the state are contiguous.
    Returns 1 if so, 0 if not.
    '''
    # Only check for changed CONs
    changed_cons = list(np.unique(df[df['CHANGE']==1]['CON']))
    # Neighbouring CONs of a flipped ED could also become discontiguous,
    # so add them to changed_cons
    for i in list(df[df['CHANGE']>0].index):
        for nc in df.loc[i, 'NB_CONS']:
```

```
            if nc not in changed_cons:
                changed_cons.append(nc)
    # Check contiguity of each changed constituency
    for c in changed_cons:
        # Filter dataframe to just EDs in constituency c
        d = df[df['CON']==c]
        # Create list of ED IDs in constituency c
        ed_ids = list(d['ED_ID'])
        # Form nested list of neighbours of each ED
        nbh_list = list([list(nbh) for nbh in d['NEIGHBOURS']])
        # Form nested list with only neighbours in constituency c
        nbhs_in_c = [[n for n in nbh_sublist if n in ed_ids]
                for nbh_sublist in nbh_list]
        if [] in nbhs_in_c: # If some ED in CON c has no neighbours in same CON
            return 0
        # Create a dictionary of ED ID-neighbour pairs,
        # but only including neighbours in same constituency
        nbh_dict = {ed_id: nbh for ed_id, nbh in
                    zip(list(d['ED_ID']), nbhs_in_c)}
        # Create a graph representing constituency c
        g = nx.Graph(nbh_dict)
        # Check whether the graph is connected, i.e. whether c is contiguous
        if not nx.is_connected(g):
            return 0
    return 1
```

Listing 3.1: The component of the reward function pertaining to contiguity.

3.2 Seat Equivalent Representation

Once the contiguity of all constituencies in the configuration had been ensured, the next priority was optimising the overall configuration with respect to the SER of each constituency. To best satisfy the provisions of the Constitution, it is desirable for the SER of each constituency to be as close as possible to an integer between 3 and 5 [3]. To apply this criterion to the algorithm, the SER component of the reward function for a configuration X was defined as

$$f_{\text{SER}}(X) = C_{\text{SER}} \sum_{\text{CONs}} g_{\text{CON}}(\text{SER}), \tag{3.2}$$

where C_{SER} is a tuneable parameter, and g_{CON} is a function which takes large values for integer SERs and near-zero values for SERs far away from integers. Several different candidates for such a function, an example of which is

$$g_{\text{CON}}(\text{SER}) = 1 - e^{-a_{\text{SER}}(\text{SER}-[\text{SER}])}, \tag{3.3}$$

where [SER] is the nearest integer to SER, and a_{SER} is a tuneable parameter. This function provides a high reward to a constituency with an SER close to an integer, and a low reward to a constituency with an SER far from an integer, and is plotted in Figure 3.3.

The definition of this component of the reward function in Python is shown in Listing 3.2.

```
def ser(df, c, national_ratio=29800):
    '''
    Returns SER of constituency c.
    '''
```

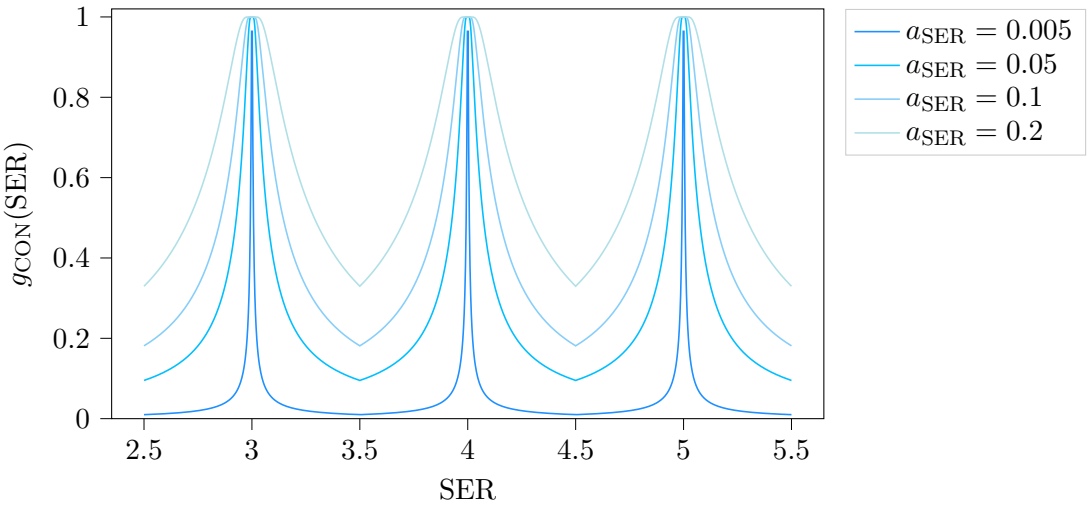


Figure 3.3: The component of the reward function related to SER for a single constituency, for various values of a_{SER}.

```
    data = df[df['CON']==c]
    pop = data['POPULATION'].sum()
    return pop/national_ratio

def f(s, d=0.05):
    '''
    Bump function for each constituency.
    '''
    # Nearest integer to s - we want the SER to be an integer
    b = round(s)
    x = np.abs(s-b)
    if x == 0:
        return 1
    return 1 - np.exp(-d/x)

def f_ser(df, a=3, national_ratio=29800):
    '''
    Checks how desirable SER of all constituencies is.
    '''
    result = 0
    for c in np.unique(df['CON'].to_list()):
        s = ser(df, c, national_ratio)
        result += f(s)
    return a*result
```

Listing 3.2: The component of the reward function pertaining to SER.

3.3 County Boundaries

To quantify the extent to which a constituency violated county boundaries, one or more 'home counties' were assigned to each constituency. For example, Cavan and Monaghan were the home counties of Cavan-Monaghan, while Dublin was the home county of Dún Laoghaire. For each constituency, the number of EDs (n_{EDs}) which were outside their home county was calculated, and the total population of these EDs (n_{people}) was found. The component of the reward function

for county boundaries was chosen to be

$$f_{\mathrm{CB}}(X) = C_{\mathrm{CB}} \sum_{\mathrm{CONs}} e^{-a_{\mathrm{CB}} n_{\mathrm{people}} - b_{\mathrm{CB}} n_{\mathrm{EDs}}}, \tag{3.4}$$

where $C_{\mathrm{CB}}, a_{\mathrm{CB}}$ and b_{CB} are tuneable parameters and we sum over all constituencies. The implementation of this method in Python is shown in Listing 3.3, where c2c is a dictionary that links constituencies to their home counties.

```
def f_exp(x, y, a, b):
    '''
    Exponential decay function.
    '''
    return np.exp(-a*x-b*y)

def f_county_boundary(df, c2c, a, b):
    '''
    Checks how much state preserves county boundaries.
    '''
    num_ed = 0
    num_ppl = 0
    for i in range(len(df)):
        if df.loc[i,'COUNTY'] not in \
            c2c[c2c['CON']==df.loc[i,'CON']]['HOME_COUNTY'].to_list():
            num_ed += 1
            num_ppl += df.loc[i,'POPULATION']
    return f_exp(num_ppl, num_ed, a, b)
```

Listing 3.3: The component of the reward function pertaining to respect for county boundaries.

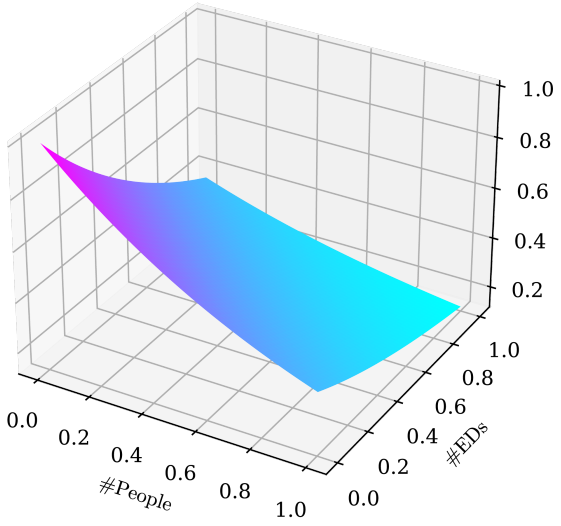


Figure 3.4: The component of the reward function pertaining to respect for county boundaries, for $C_{\mathrm{CB}} = a_{\mathrm{CB}} = b_{\mathrm{CB}} = 1$.

The decaying exponential function for each constituency is plotted in Figure 3.4. It gives a reward close to 1 when there are few breaches of county boundaries, and close to 0 when many people are in a constituency that breaches county boundaries. The value of a_{CB} was chosen to be higher than that of b_{CB}, so that we prioritised the minimisation of the number of people in a

constituency that did not match their home county over the minimisation of the number of EDs outside their home county, as this was deemed to be the more important measure of respect for county boundaries.

3.4 Temporal Continuity

To quantify the temporal continuity of a state, a similar approach was used as for county boundaries. For each state, the number of EDs (n_{EDs}) which had changed from the original state was calculated, and the total population of these EDs (n_{people}) was found. The chosen component of the reward function was

$$f_{\text{cont}}(X) = C_{\text{cont}} e^{-a_{\text{cont}} n_{\text{people}} - b_{\text{cont}} n_{\text{EDs}}}, \tag{3.5}$$

where $C_{\text{cont}}, a_{\text{cont}}$ and b_{cont} are tuneable parameters. As in the case above involving county boundaries, we prioritised the minimisation of the number of people in a changed ED over the minimisation of the number of changed EDs themselves by setting a_{cont} to be larger than b_{cont}.

3.5 The Complete Reward Function

The final definition of the reward function is shown in Listing 3.4. If the configuration contains any non-contiguous constituencies, then the reward function immediately returns zero. Otherwise, it returns a sum of the remaining three components associated with the other three primary criteria for success.

```
def reward(df, a_ser, a_cb, b_cb, a_cont, b_cont, nr=29800):
    '''
    Reward function for dataframe df.
    '''
    if not f_contiguity(df):
        return 0 # No reward if not globally contiguous
    return f_county_boundary(df, c2c, a_cb, b_cb) + \
        f_continuity(df, a_cont, b_cont) + f_ser(df, a_ser, nr)
```

Listing 3.4: The final definition of the reward function.

3.6 Additional Considerations

As well as the four main criteria discussed above, there are also several other factors which could affect the desirability of a given configuration of constituencies. Examples of such factors include the compactness of each constituency, the presence of significant geographical features, and the population density in different regions. Although these additional factors were not fully implemented in the reward function in this project, it is nevertheless interesting to consider how such factors could be taken into account.

3.6.1 Compactness

A simple way of quantifying the compactness of a constituency is provided by the convex hull test. The convex hull of a geometry is the smallest convex polygon containing all the points in the geometry, and can be computed using the `.convex_hull` attribute included in `GeoPandas`.

Figure 3.5: The convex hulls of the constituencies Kerry, Wicklow, and Laois-Offaly.

Examples of convex hulls for the constituencies of Kerry, Wicklow and Laois-Offaly are shown in Figure 3.5.

To test the compactness of a constituency, the total area of the constituency is divided by the area of its convex hull. This produces a value between 0 and 1, with values close to zero indicating low compactness, and 1 indicating maximal compactness. The compactness component of the reward function could then be computed by summing these values for each constituency in the configuration X and multiplying by a parameter C_{comp}:

$$f_{\text{comp}}(X) = C_{\text{comp}} \sum_{\text{CONs}} \frac{A_{\text{CON}}}{A_{\text{convex hull}}}. \tag{3.6}$$

Although this test is simple to implement, it is not currently included in the algorithm used in this project as it produces a substantial increase in the overall execution time.

3.6.2 Significant Physical Features

The Electoral Commission is required to have 'regard to geographic considerations including significant physical features'. This means that it is slightly undesirable for constituencies to breach major geographical features such as mountain ranges or rivers.

In order to implement this criterion into our reward function, we would need to obtain topological data for the Irish landscape, and data including the course of major rivers and waterways. We could then implement a component in the reward function which would give a higher reward to constituencies which did not breach these physical features.

3.6.3 Population Density

The Electoral Commission is also required to have regard to 'the extent of and the density of population in each constituency' when determining constituency boundaries. The distribution of population throughout a constituency can vary widely, and this can have a significant effect on the outcome of electoral processes.

To provide an example, as was pointed out in [4], all five of the currently-sitting TDs representing Wicklow reside in a subregion comprising twelve EDs concentrated in the north of the constituency, as highlighted in Figure 3.6. According to preliminary 2022 census results, the total population of Wicklow is 154,450, while that of the twelve-ED region is 71,473. This means that approximately 46% of the total population is concentrated in this twelve-ED region. This is interesting because in terms of the number of EDs involved, the twelve EDs in this subregion account for just 14.6% of the 82 EDs that make up the whole constituency. Similarly, this subregion covers an area of 216 km^2, which is just 10.7% of the total area covered by Wicklow

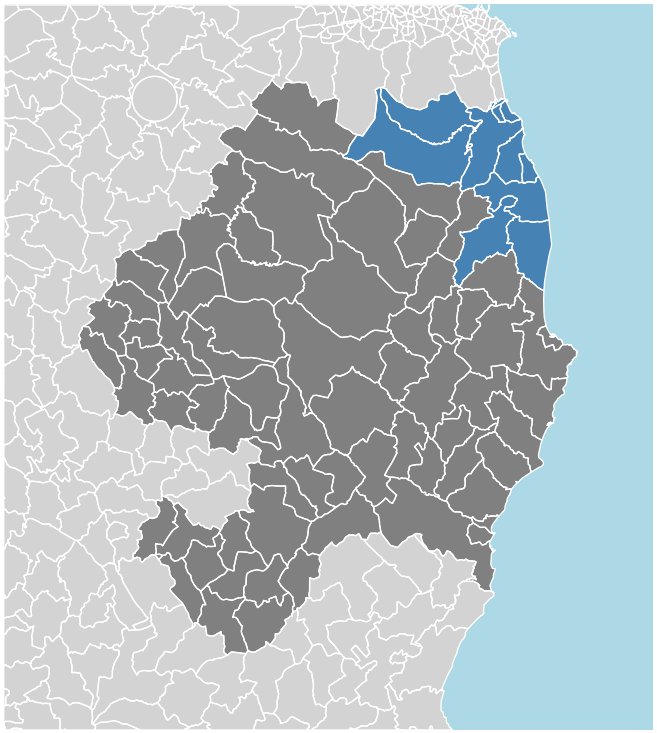


Figure 3.6: The uneven population density of Wicklow. The region shaded in blue contains approximately 46% of the total population, and is home to all five currently-sitting TDs for Wicklow.

(2,025 km^2). This means that there is a large area of Wicklow which is not represented by any local TDs in the Dáil.

It would be interesting to extend the reward function implemented in this project to take account of the population distribution across different regions of Ireland. This may help to ensure fair representation of all cohorts in the population.

4

Results and Discussion

Once the reward function described in the previous chapter was integrated into the evolutionary algorithm of Chapter 2, the process of natural selection of constituency configurations could begin. We chose to allow the configurations to evolve for three generations, with a variable number of flips per offspring state, and of offspring states and culled states per generation.

Once this algorithm had been executed, we obtained a list of three dataframes corresponding to the configurations with the three highest rewards. A list of the top three was returned since there was still a substantial benefit to be gained from human input when choosing the 'best' configuration, and this allowed us to select the most desirable configuration from the top three candidates.

In this chapter, we discuss the results produced by our algorithm, and consider some possible improvements that could be made in future.

4.1 Results

The inherent randomness involved in our algorithm meant that it produced vastly different 'optimal' solutions each time it was executed. The configurations generated by the algorithm also depended on numerous other factors, including the specific values of the parameters in the reward function, the weights assigned to each objective, the number of flips performed per offspring state, and the total number of offspring states and culled states per generation.

To provide an example of an optimised configuration, the algorithm was executed for a three-generation evolution with five flips per offspring state, ten offspring states per generation, and six culls per generation. In the notation of Listing 3.4, the weights were `a_ser=3`, `a_cb=1e-10`, `b_cb=1e-4`, `a_cont=1e-3`, and `b_cont=0.01`. For these values of the parameters, the program took approximately 22 minutes to run.

One of the resulting optimised states is shown in Figure 4.1, where the county boundaries are also plotted. This state had the second-highest reward of all states generated in this execution. This configuration will be used as an example case throughout the rest of this chapter. As required, all constituencies in this configuration are contiguous. The overlaid county boundaries also show that the breaching of county boundaries is not extreme, considering that the current configuration includes many such breaches.

The SER and VNA values of each constituency in this optimised configuration are shown in Table 4.1. Figure 4.3 shows bar charts comparing the SER and |VNA| values of constituencies in this optimised configuration, to those of the constituencies in the current configuration. It is difficult to draw conclusions from the first figure, but the second figure clearly shows that the absolute value of the VNA of most constituencies has decreased in the optimised state compared to the original state.

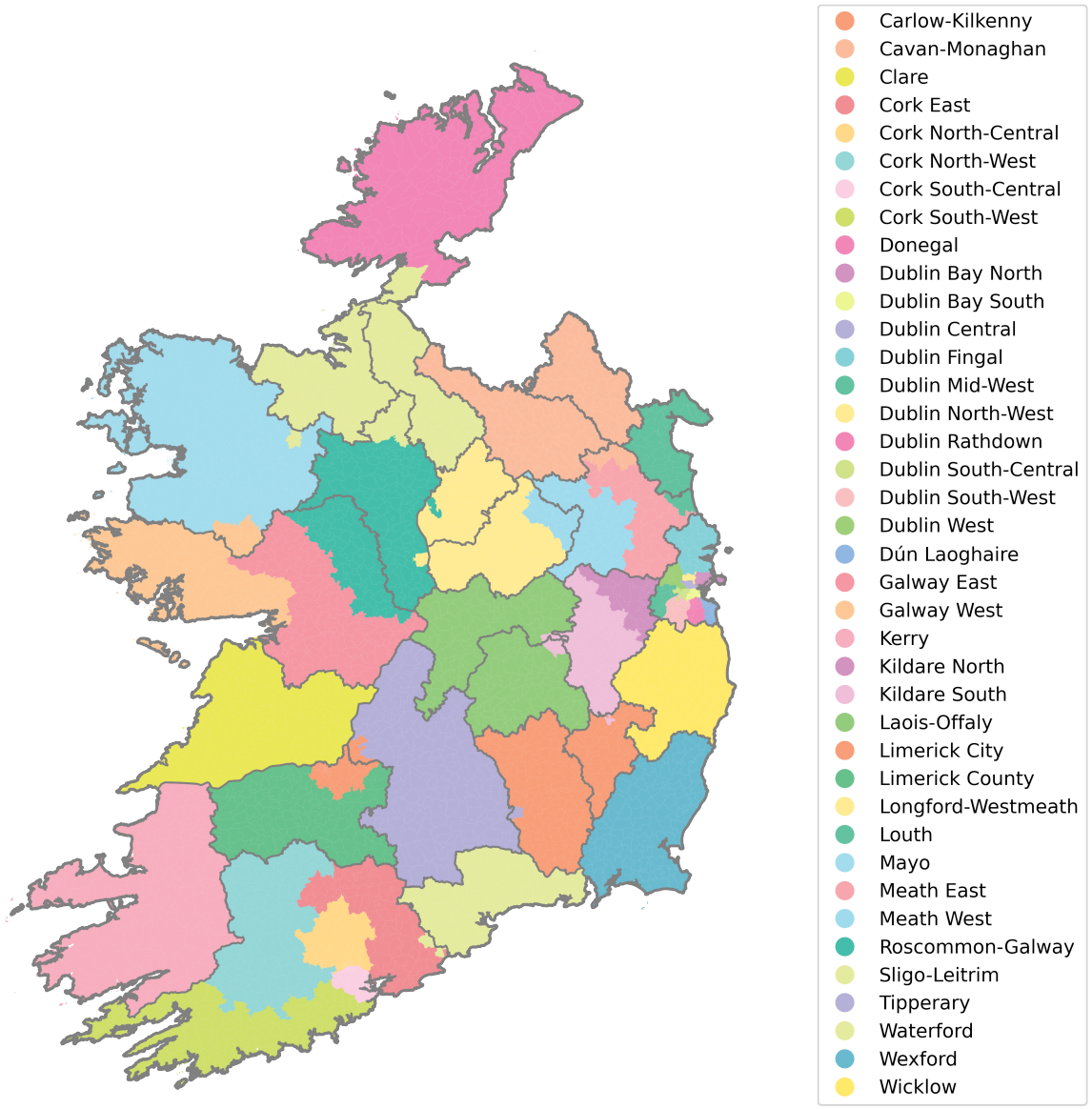


Figure 4.1: One of the top three optimal states produced by the evolutionary algorithm, overlaid with county boundaries shown as grey lines.

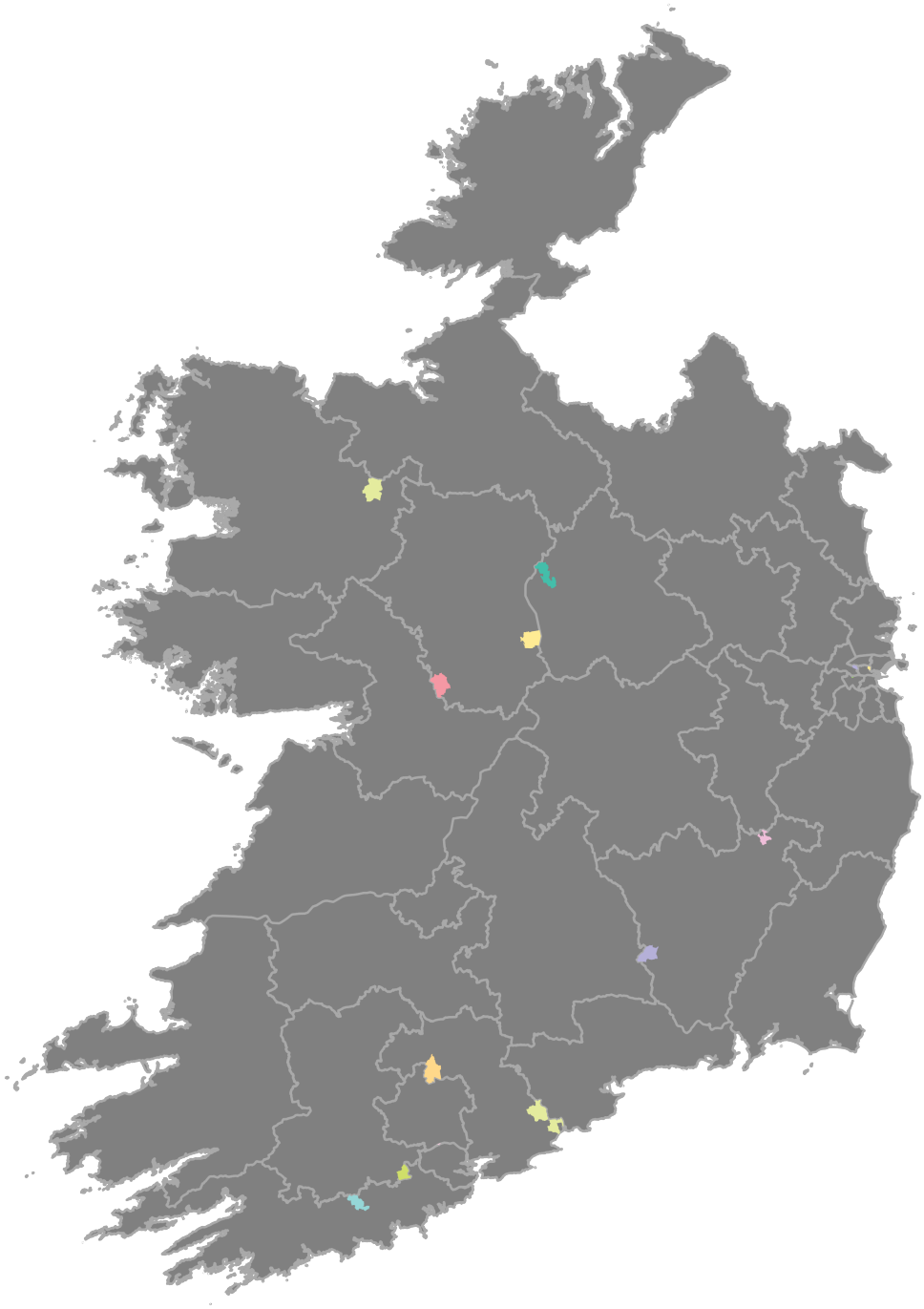


Figure 4.2: The EDs in the optimal state which had changed from one constituency to another. The colour coding here matches that shown in the legend in Figure 4.1. The lighter grey lines here represent the original constituency boundaries.

Table 4.1: The SER and VNA values for the 39 constituencies in the optimised state. Here, the number of seats is the rounded value of the SER. VNA values that lie outside the acceptable threshold of ±0.05 are shown in bold.

Constituency	Seats	SER	VNA
Carlow-Kilkenny	6	5.546	**−0.076**
Cavan-Monaghan	5	5.048	0.010
Clare	4	4.258	**0.064**
Cork East	4	3.983	−0.004
Cork North-Central	4	4.198	0.050
Cork North-West	3	2.977	−0.008
Cork South-Central	4	4.119	0.030
Cork South-West	3	2.838	**−0.054**
Donegal	5	5.286	**0.057**
Dublin Bay North	5	4.962	−0.008
Dublin Bay South	4	4.066	0.017
Dublin Central	4	4.002	0.001
Dublin Fingal	5	5.467	**0.093**
Dublin Mid-West	4	3.959	−0.01
Dublin North-West	3	2.593	**−0.136**
Dublin Rathdown	3	3.170	**0.057**
Dublin South-Central	4	4.018	0.004
Dublin South-West	5	4.917	−0.017
Dublin West	4	3.913	−0.022
Dún Laoghaire	4	4.146	0.036
Galway East	3	3.010	0.003
Galway West	5	4.866	−0.027
Kerry	5	5.234	0.047
Kildare North	5	4.523	**−0.095**
Kildare South	4	4.226	**0.056**
Laois-Offaly	5	5.399	**0.080**
Limerick City	4	4.150	0.037
Limerick County	3	3.037	0.012
Longford-Westmeath	5	4.504	**−0.099**
Louth	5	5.327	**0.065**
Mayo	4	4.377	**0.094**
Meath East	3	3.320	**0.107**
Meath West	3	3.330	**0.110**
Roscommon-Galway	3	2.972	−0.009
Sligo-Leitrim	4	4.102	0.025
Tipperary	5	5.420	**0.084**
Waterford	4	4.226	**0.057**
Wexford	5	5.464	**0.093**
Wicklow	5	5.183	0.037

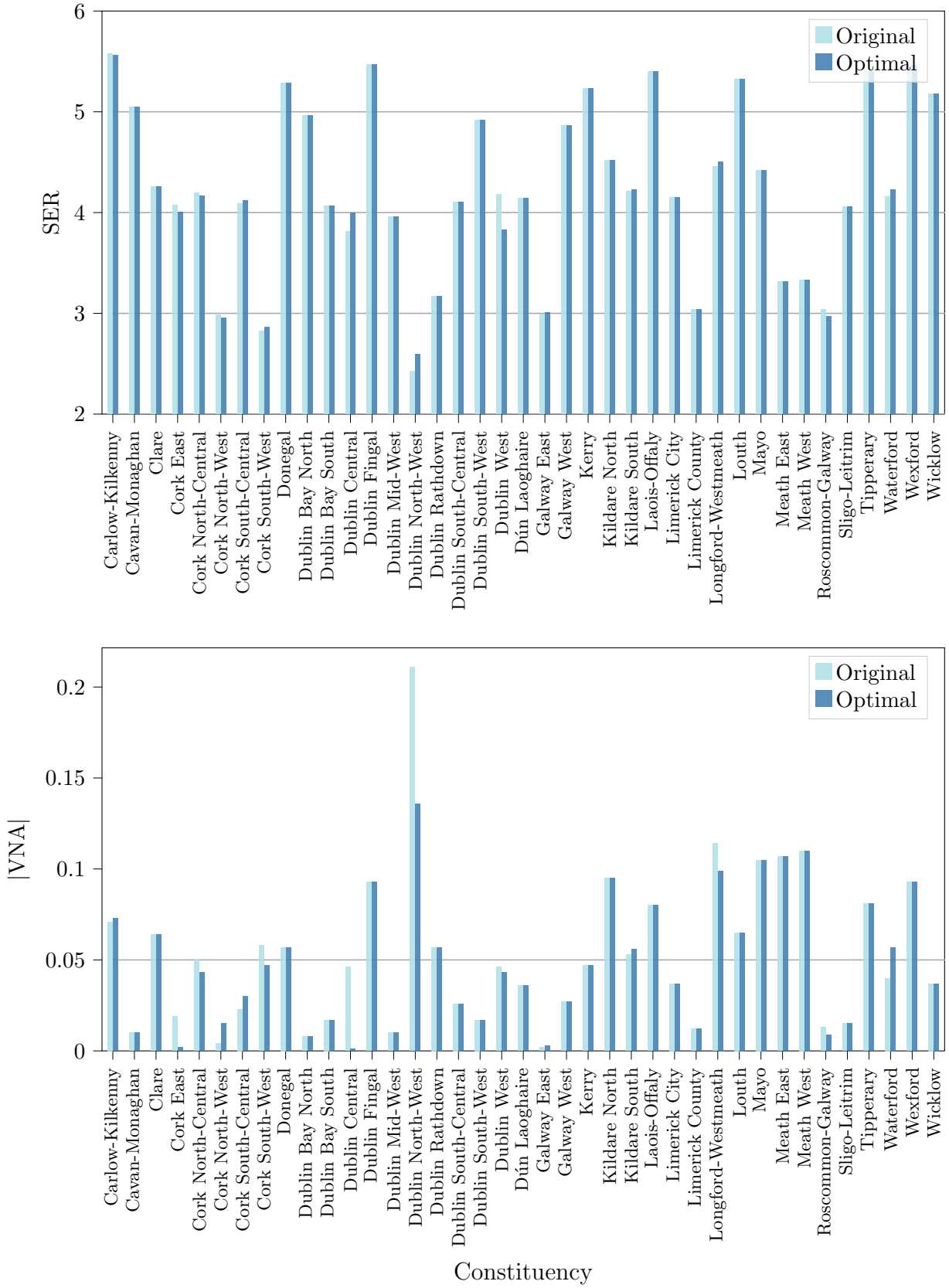


Figure 4.3: The SER and |VNA| values of each constituency in the optimal state, compared with those of the original configuration.

4.2 Discussion

To evaluate the extent to which our evolutionary algorithm is successful in optimising the constituency boundaries, it is useful to consider each of our four primary objectives in turn. Throughout the following discussion, we will frequently refer to the example optimised state shown in Figure 4.1, but many of the conclusions are generally true for most optimised states produced by the algorithm.

4.2.1 Contiguity

As is required by the Irish constitution, all configurations generated by our algorithm contain only contiguous constituencies. This is evident in Figure 4.1, and was always the case for all other tests performed with the algorithm.

4.2.2 SER

The algorithm is broadly successful at evolving the current state towards a configuration in which the SERs of most constituencies are very close to integers. This is somewhat evident in the first subfigure of Figure 4.3, where the SER of several constituencies has moved closer to the nearest integer value. The improvement is clearer in the second subfigure, which shows that the evolution has resulted in a decrease in the VNA of many constituencies. Only in the case of a single constituency, Waterford, has the algorithm caused a VNA that was originally below the threshold of 5% to rise above it.

Note that in this figure, the VNA of constituencies in both the current and optimised state have been calculated by assuming that the number of seats assigned to each constituency is equal to the rounded value of the SER, rather than the actual current number of seats. Figure 4 in the Appendix shows a similar chart, where in this case the VNA values plotted for the current configuration are computed using the current seat numbers. This chart shows that the algorithm produces a more marked improvement in VNA, assuming that additional seats can be allocated to some constituencies in the optimised configuration.

From Table 4.1, we can see that the number of seats that our model recommends be allocated to Carlow-Kilkenny is 6. This means that Carlow-Kilkenny should be divided into two three-seater constituencies. In future, it may be interesting to add the capacity for the algorithm to explicitly perform this division to create two contiguous constituencies, each with an SER close to 3.

Similar positive results were observed in many of the other optimised states produced when testing the algorithm. This suggests that the component of the reward function related to SER is effective in reducing the VNA of constituencies in the configuration.

In future, it might be helpful to explicitly include the $\pm 5\%$ VNA threshold in the definition of the reward function. This could be achieved by imposing a penalty on configurations where the VNA of any constituency rose above this threshold.

4.2.3 Respect for County Boundaries

The nature of our algorithm means that any breaching of county boundaries is likely to be minor, since it is based on a flipping of boundary EDs rather than a full redrawing of constituency boundaries. Apart from this, our algorithm prioritised the optimisation of SER values over respect for county boundaries, but it still rewarded configurations with fewer breaches of county boundaries slightly more.

The extent to which county boundaries are breached by our example optimised state is shown in Figure 4.1. There is clearly a greater breaching of county boundaries than in the current

configuration, but it is not extreme. The total number of people residing in a constituency that does not match their county is 84,177, compared to 78,188 in the current configuration. This is an increase of just 5,989 people.

4.2.4 Temporal Continuity

As in the case of respect for county boundaries, the basic operation of our algorithm meant that only small changes were made to the current configuration. This meant that temporal continuity was maintained to a large extent.

In the case of our example state, Figure 4.2 shows the EDs which have changed constituencies compared to the current configuration. Evidently, the area covered by these EDs is relatively small. The total population which had experienced a change in constituency was 28,048, which is less than 0.6% of the total population of Ireland contained in this dataset.

4.3 Possible Improvements

There are a number of possible improvements that could be made to the algorithm developed in this project. Some such improvements are discussed below.

4.3.1 Tuning of Weight Parameters

In order to determine the rewards assigned to each configuration produced at each generation of our algorithm, the weighting assigned to each objective needed to be specified. It may be the case that the desired weightings in principle may not even yield any new solutions at all. If the objectives are too strict the algorithm produces very few deviations from the current state, however this may be unavoidable if the objectives are not appropriately balanced.

We created a model where these weight parameters may be varied and thus many solutions may be generated for arbitrary sets of weights. By tuning the weight parameters, we can reassign priority to these objectives if they are not conducive to novel and usable solutions. It would therefore be useful to perform further testing of the algorithm by systematically altering the weights and observing the result on the solutions generated.

4.3.2 Stopping Criterion

There are many stopping criteria for evolution algorithms. For instance, one may keep track of the individual with the best reward function and terminate the algorithm when the newest generation no longer produces an individual with a better reward function. It is evident that this approach would get stuck in local optima. We ought not severely punish the algorithm for allowing weaker children to propagate as this may be necessary to surpass local minima in reward. Otherwise the algorithm may be dissuaded from leaving the current state at all.

Our approach was the simple choice of terminating after a fixed number of generations. However, there may be other stopping criteria which could lead to better solutions, and this is an area which could be investigated further.

4.3.3 Crossover Between Individuals

Our algorithm contrasts with typical evolutionary algorithms in that we did not implement any crossover between individuals to create the next generation. Each individual has a direct lineage back to the current state. It is not obvious whether crossover would be better or not. Crossover

has the benefit of allowing individuals to inherit multiple different rewarded features from its parents, but has the downside of the children becoming more homogeneous.

Considering how few EDs are changed compared to the total ED count, perhaps it may be advantageous to allow rewarding changes on either end of the country to propagate to the new generation via crossover as these changes would not directly interfere with each other. That is to say

$$\Delta\text{Reward(change A)} + \Delta\text{Reward(change B)} \approx \Delta\text{Reward(change A + change B)},$$

whereas if for instance two border EDs from the same constituency changed, the change to reward may be worse than the sum of their individual changes.

It would be interesting to implement crossover between individuals and test whether this improved the results.

4.3.4 Allowing EDs to Flip Multiple Times

The exclusion of already flipped EDs from the boundary pool imposes a restriction on the algorithm, since the number of candidate EDs which can change decreases monotonically between each generation. This restriction was introduced to prevent the algorithm wasting time stuck in cycles. Perhaps there may be an issue reconciling this ED exclusion with the fixed generation termination, since if the changes overshoot a candidate solution it is not permitted to backtrack and telescope towards that solution. It may therefore be useful to experiment with allowing EDs to flip two or three times before they become locked into place, as this would allow the algorithm to explore more of the solution space.

4.4 Conclusion

The evolutionary algorithm implemented in this project was broadly successful in achieving its aim of optimising the constituency boundaries of Ireland with respect to the criteria outlined in the Constitution. Considering the two primary objectives, pertaining to contiguity and SER/VNA, the algorithm produced only configurations with contiguous constituencies, and succeeded in reducing the absolute value of the VNA of most constituencies, which was the desired outcome. Turning to the two secondary objectives, relating to respect for county boundaries and maintenance of temporal continuity, the fact that the algorithm made incremental changes to the current configuration, and the definition of the reward function, ensured that the secondary objectives were also relatively well satisfied.

Overall, the algorithm could be considered a successful implementation of multi-objective optimisation. However, there are numerous improvements that could be made, foremost of which would be the addition of the capacity for crossover between configurations, as this would likely lead to a better optimised solution in a shorter time frame.

Bibliography

[1] Ajith Abraham and Robert Goldberg. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. 1st ed. Advanced Information and Knowledge Processing. Springer, 2005.

[2] Electoral Commission. *Electoral Commission Statement – Section 59(2) of the Electoral Reform Act 2022*. 2022. URL: https://www.electoralcommission.ie/constituency-reviews/.

[3] *Constitution of Ireland*. 1937. URL: http://www.irishstatutebook.ie/en/constitution/index.html.

[4] Theoretical Physics Student Association. *Boundary Submission*. May 2023. URL: https://www.electoralcommission.ie/cr-submissions/michael-mitchell/.

[5] Constituency Commission. *Constituency Commission Report*. 2017. URL: https://www.constituency-commission.ie/cc-oldreports.html.

[6] The pandas Development Team. *pandas*. Python package. Version 2.0.1. Apr. 2023. DOI: 10.5281/zenodo.3509134.

[7] Kelsey Jordahl et al. *GeoPandas*. Python package. Version 0.8.1. July 2020. DOI: 10.5281/zenodo.3946761.

[8] The matplotlib Development Team. *matplotlib: Visualisation with Python*. Python package. Version 3.7.1. Apr. 2023. DOI: 10.5281/zenodo.7697899.

[9] *Electoral Divisions - National Statutory Boundaries - 2019*. 2022. URL: https://data-osi.opendata.arcgis.com/datasets/osi::electoral-divisions-national-statutory-boundaries-2019/explore.

[10] *Constituency Boundaries Ungeneralised - National Electoral Boundaries - 2019*. 2022. URL: https://data-osi.opendata.arcgis.com/datasets/osi::constituency-boundaries-ungeneralised-national-electoral-boundaries-2017/explore.

[11] *Counties - National Statutory Boundaries - 2019*. 2022. URL: https://data-osi.opendata.arcgis.com/datasets/osi::counties-national-statutory-boundaries-2019/explore.

[12] *Census 2022 - Table F1060: Population*. 2023. URL: https://data.cso.ie/table/F1060.

[13] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart. *NetworkX*. Python package. Version 3.1. 2023. URL: https://networkx.org.

Appendix

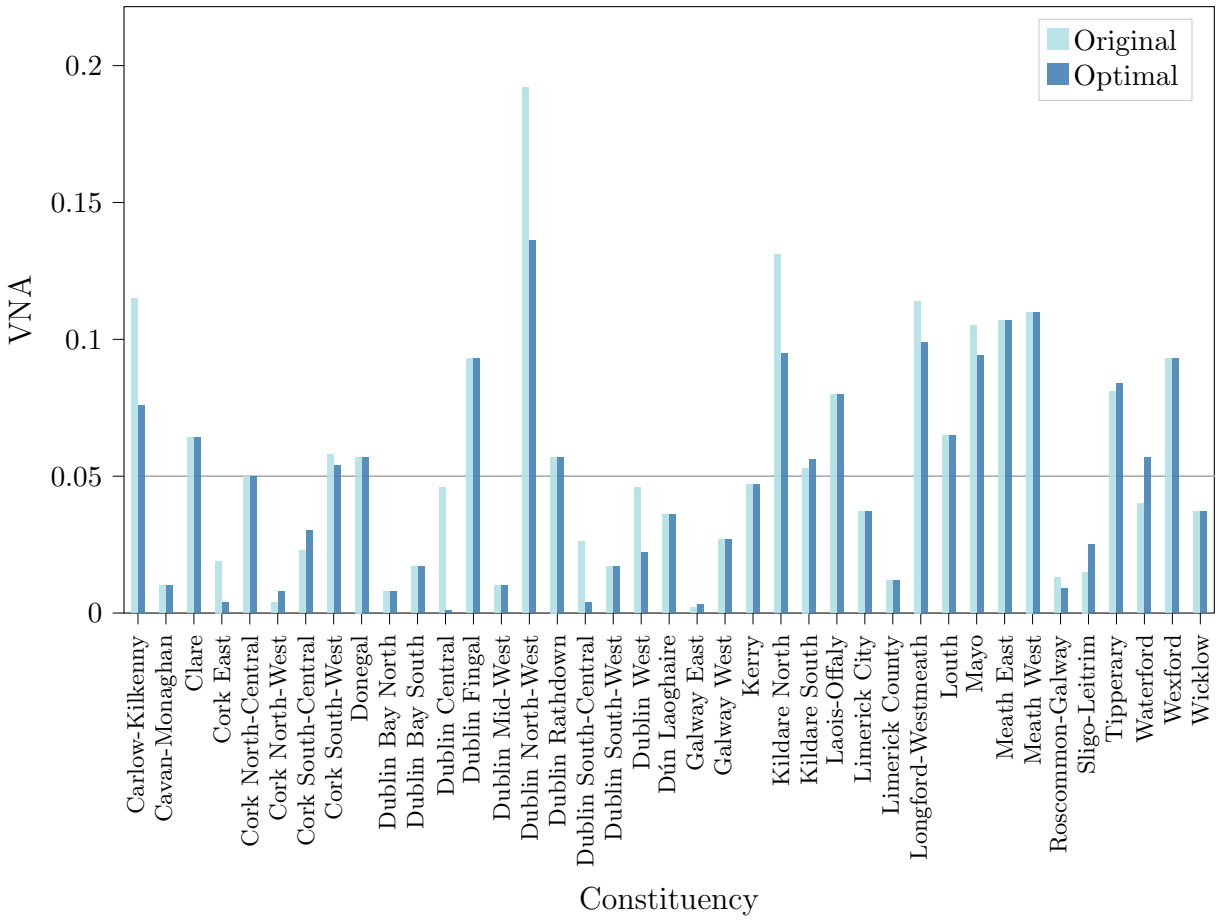


Figure 4: The absolute value of the VNA of the constituencies in the optimised configuration compared to that of constituencies in the current configuration, where the VNA calculations for the current configuration have been performed using the currently assigned seat numbers rather than the rounded value of the SER. In this case, the algorithm shows a more marked improvement in VNA, but this is to be expected because the total number of Dáil seats is not currently large enough to meet the requirements of the Constitution.