Answer Set Programming for Gerrymandering: Model, Benchmarks, and Evaluation

Claudio Bendini
claudio.bendini@studenti.unipr.it
380972

Aurora Felisari
aurora.felisari@studenti.unipr.it
379867

06 June 2025

Declarative Programming

Università degli Studi di Parma

Abstract

Gerrymandering, the strategic manipulation of electoral district boundaries, represents a significant challenge to the fairness of democratic systems. This project explores the application of Answer Set Programming (ASP), a declarative logic programming paradigm, to model and solve an abstract version of the gerrymandering problem. The objective is to maximize the number of districts won by a specific party within an $m \times n$ electoral grid by partitioning it into k contiguous districts. An ASP model was developed that defines the assignment of electoral cells to districts, ensures the contiguity of each district, computes the votes, and determines the winner of each district. For evaluation, a benchmark corpus of 100 instances was generated with varying grid sizes, numbers of districts, and vote-distribution strategies. Experiments conducted with the Clingo solver demonstrate the ASP model's ability to find optimal solutions for both parties, highlight the impact of district partitioning on election outcomes, and confirm the computational efficiency of the ASP approach for this complex combinatorial problem.

1 Introduction

Gerrymandering is a political strategy used to manipulate the boundaries of electoral districts in order to favor a particular party or group. Classical studies on gerrymandering have shown how manipulations of district boundaries can significantly influence election outcomes.

In formal terms, given an $m \times n$ grid graph whose nodes (cells) are labeled with votes 0/1, the problem consists of partitioning the nodes into k 4-connected subsets (districts) so as to maximize the number of subsets in which the vote of a specific party prevails.

In this project, we adopt an extreme abstraction: the geographical region becomes a regular $m \times n$ grid, and each cell is a "pure" voter who votes only 0 or 1. We do not consider heterogeneous population, real compactness constraints (such as polygonal shapes), or demographic criteria, but we focus exclusively on 4-connectivity and majority within each district.

It should be noted that, although extremely simplified, this model captures the combinatorial essence of gerrymandering: showing how, even with a globally balanced number of votes, partitioning into districts can alter the balance of seats.

In this section we have presented the context and motivation of the problem. In Section 2 we will provide a formal definition, while in Section 3 we will illustrate our ASP encoding.

2 Problem Definition

The gerrymandering problem addressed in this project can be formally defined as follows:

- The voting area is represented by a matrix M of size $m \times n$, where each cell corresponds to a voter.
- Each voter supports one of the two political parties, encoded as 0 or 1, based on vote predictions obtained through social media analysis.

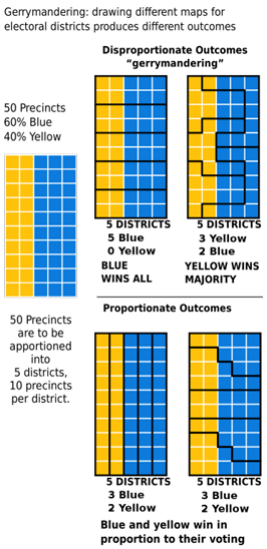

Figure 1: Qualitative example of gerrymandering in a real situation: note how, despite having an overall balanced number of votes, the blue party secures a majority of district seats.

- The area must be divided into k **districts**, each consisting of a contiguous set of adjacent cells.
- Within each district, an election is held. The party that obtains the majority of votes in a district wins that district.
- The objective is to determine the optimal partitioning of the region into k districts, so as to maximize the number of districts won by a specific party.
- Since the two parties have opposing objectives, the optimal partitioning plan is computed separately for each party.

3 ASP Encoding for the Gerrymandering Problem

This section describes in detail the logic encoding developed in Answer Set Programming (ASP) to address the gerrymandering problem. The goal is to partition a two-dimensional grid of electoral cells into a fixed number of districts so that a designated political party maximizes its number of elected representatives. ASP modeling is well suited to this type of combinatorial problem thanks to its ability to express complex constraints and optimization rules concisely and declaratively.

Input and Basic Definitions. The ASP program starts from a description of the grid and voting preferences using basic predicates:

- `cell(X,Y)`: specifies the existence of the cell at position (X, Y);
- `vote(X,Y,P)`: defines that the cell (X, Y) votes for party $P \in \{0, 1\}$;
- `grid_size`(M, N): indicates the dimensions of the grid (rows M, columns N);
- `num_districts`(K): number of districts K to create;
- `district_id`(D) and `party_id`(P): auxiliary identifiers for iteration (D and P).

The party to optimize is specified via a constant passed at execution time:

```
#const party_to_optimize = 0.
```

Listing 1: Target Party Definition (Clingo)

Assignment of Cells to Districts. Each cell is assigned to exactly one district:

```
{ district(X,Y,D) : district_id(D) } = 1 :- cell(X,Y).
```

Listing 2: Unique Assignment

This ensures that each cell participates in a single district election.

Definition of Adjacency and Connectivity Constraint. To guarantee that each district is connected, it is necessary to define the notion of adjacency:

```
adjacent(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2),
                         |X1-X2| + |Y1-Y2| = 1.
```

Listing 3: Adjacency Between Cells

Next, a root node is chosen for each district:

```
1 { root(X,Y,D) : cell(X,Y) } 1 :- district_id(D).
:- root(X,Y,D), not district(X,Y,D).
```

Listing 4: District Root

From this root, reachability is propagated:

```
reachable(X,Y,D) :- root(X,Y,D).
reachable(X2,Y2,D) :- reachable(X1,Y1,D),
                      district(X2,Y2,D),
                      adjacent(X1,Y1,X2,Y2).
:- district(X,Y,D), not reachable(X,Y,D).
```

Listing 5: Reachability and Connectivity Constraint

This ensures that each district forms a connected subgraph.

Vote Counting and Winner Determination. Votes are counted within each district:

```
district_votes(D,P,N) :- district_id(D), party_id(P),
    N = #count{ X,Y : district(X,Y,D), vote(X,Y,P) }.
```

Listing 6: Vote Counting by Party

The winner is determined by majority:

```
district_winner(D,P) :- district_votes(D,P,N1),
                        district_votes(D,1-P,N2),
                        N1 > N2.
```

Listing 7: Winner Determination

In the event of a tie, victory is assigned to party 0:

```
district_winner(D,0) :- district_votes(D,0,N),
                        district_votes(D,1,N).
```

Listing 8: Tie-break Handling

Objective Function. Maximize the number of districts won by the party being optimized:

```
representatives_for_optimized_party(N) :-
    N = #count{ D : district_winner(D,party_to_optimize) }.

#maximize { N : representatives_for_optimized_party(N) }.
```

Listing 9: Representation Optimization

Guarantees Provided by the Model. The ASP program ensures:

- unique assignment of cells to a district;
- spatial connectivity of each district;
- deterministic election of the winner with a tie-break rule;
- objective function focused on partisan representation.

This approach is flexible and modular, allowing for future extensions such as size or shape constraints on districts.

4 Benchmark Generation

To test and validate the ASP model under varying conditions, a corpus of 100 benchmark instances was constructed using the Python script `generate_benchmarks.py`. The instances were saved as ASP files containing facts about the grid, votes, and districts.

File Structure. Each file contains:

- `grid_size(`M, N`)`: dimensions of the grid;
- `num_districts(`K`)`: number of districts to create;
- `cell(`R, C`)`: coordinates of existing cells;
- `vote(`R, C, P`)`: vote in cell (R, C).

Structural Parameters. To evaluate the performance of the ASP model under diverse conditions, benchmark instances were generated over a wide range of grid sizes. These grids were classified into three categories based on their spatial dimensions:

- **Small grids**: e.g., 3×3, 4×3, and 4×4 — used to validate correctness and quickly test logical behavior.
- **Medium grids**: such as 5×5, 6×5, 7×6, and 8×8 — designed to balance expressiveness and computational effort.
- **Large grids**: up to 10×10 — used to stress-test the solver's scalability and to approximate realistic scenarios.

The number of districts K for each instance was carefully selected to maintain a meaningful ratio between the number of districts and the total number of cells. This ensures that:

- Each district contains a sufficient number of cells to allow for internal variation and competitive outcomes;
- The partitioning task remains non-trivial, i.e., neither too simple (e.g., one district per row) nor too complex (e.g., one cell per district);
- The solver is challenged with realistic configurations that reflect potential electoral scenarios.

This structural variation allows for a robust evaluation of the model's behavior across different spatial scales and optimization complexities.

Vote Generation Strategies.

- **Random**: votes assigned independently with uniform probability;
- **Clustered**: one party has a higher concentration in local areas;
- **Checkerboard**: votes alternate regularly to create high heterogeneity.

These strategies allow simulation of urban, rural, and highly polarized scenarios.

Generation Objectives. The benchmark was designed to:

- evaluate the correctness of the model;
- analyze the effect of vote distribution on the final outcome;
- test scalability relative to grid size and complexity.

This enabled a robust and varied analysis of the ASP model's capabilities.

5 Experimental Evaluation

Experiments were conducted on a machine equipped with an **AMD Ryzen 7 7700X** processor (8 physical cores, 16 threads), **32 GB of RAM**, and running Windows 11. The CPU features a multi-level cache architecture consisting of **512 KB L1**, **8 MB L2**, and **32 MB L3** cache, which contributes to efficient memory access and reduced latency during computation.

The solver used was **Clingo 5.7.1**, executed through a dedicated Python script developed to automate the entire benchmarking process, including timeout management, data extraction, and result collection.

Each benchmark instance was solved twice: once optimizing for party 0 and once for party 1. Executions were launched with Clingo's `default` configuration, without specifying additional heuristics or options.

For each run, the following metrics were recorded:

- total execution time in seconds (`TimeSec`);
- solver status (`Status`);
- number of models found (`Models`);
- number of districts won by the optimized party (`MaxReps`).

A maximum time limit of **5 minutes** (300 seconds) was imposed for each resolution. The results of all runs were saved in a `CSV` file and subsequently analyzed to produce performance evaluation graphs.

6 Results and Discussion

6.1 CSV Column Descriptions

Instance Identifier of the benchmark instance (e.g., `instance_001_m3n3k2_random.lp`). It refers to a specific grid, vote configuration, and districting problem.

PartyOptimized Indicates which party the solver aimed to favor during optimization. Values are either `0` or `1`, corresponding to Party 0 or Party 1.

Solver Strategy Most experiments used Clingo's default configuration, which applies general-purpose heuristics designed to balance performance and generality. Optionally, the solver can be configured with custom strategies like `HeuristicA` or `GreedyLocal`, which influence variable selection, branching priorities, and conflict resolution. For example:

- `HeuristicA` prioritizes atoms with higher constraint frequency;
- `GreedyLocal` favors decisions that yield short-term optimization gains.

While not actively used in this project, such strategies are relevant for future studies on heuristic impact.

MaxReps The maximum number of districts won by the optimized party in that run. For example, `MaxReps = 2` means the solver found a configuration where the favored party secured 2 districts.

TimeSec Total runtime in seconds (with decimals). For example, `0.009` means the problem was solved in 9 milliseconds.

Status Indicates how the solver terminated:

- `OPTIMUM_FOUND`: The optimal solution was proven;
- `SAT`: A feasible solution was found (but not proven optimal);
- `UNSAT`: No valid solution exists;

- `TIMEOUT`: The solver reached the time limit (300 seconds);
- `ERROR`: An internal error occurred.

Models Number of distinct models (i.e., improving solutions) found during the optimization loop. A low number means early convergence to the optimum.

ClingoReturnCode Numerical exit code returned by Clingo:

- `0`: Successful termination;
- `10`: SAT solution found;
- `20`: Problem proven UNSAT;
- `30`: Optimization complete (optimum found);
- Others: Unexpected errors.

RawOptValue The raw (negative) objective value computed by Clingo. Since Clingo minimizes by default, district wins are encoded as negative numbers:

- `-2` means the party won 2 districts;
- `-1` means it won 1 district.

6.1.1 Example Partition for `instance_001_m3n3k2_random.lp` (Party 1)

The CSV row

```
instance_001_m3n3k2_random.lp,1,default,2,0.009,OPTIMUM_FOUND,1,30,-2
```

indicates that, when optimizing for *Party 1*, the solver found in 0.009 s (`TimeSec`) an optimal partition (`OPTIMUM_FOUND`) in which Party 1 wins both districts (`RawOptValue` = –2). It was enough to generate a single model (`Models = 1`), even though `MaxReps = 2` allowed up to two iterations.

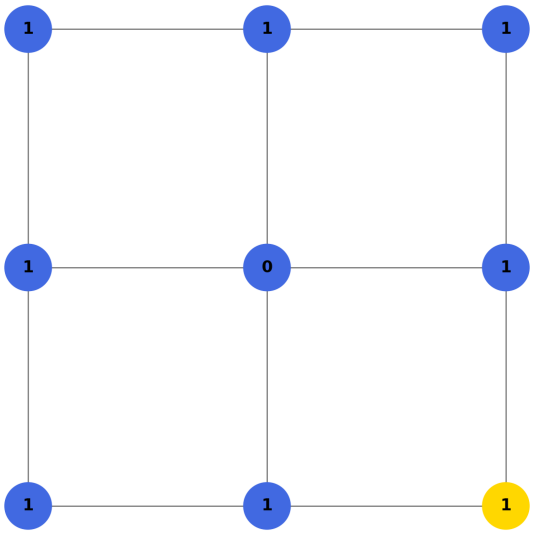

Figure 2: Optimal partition for `instance_001_m3n3k2_random.lp` when optimizing for Party 1. Each cell displays its vote (0 or 1), while the two colored districts—yellow and blue—are both won by Party 1.

Figure 2 provides a concrete visualization of the computed optimal partition. The grid represents a 3×3 voting map where each cell corresponds to a voter, casting a vote for either Party 0 (`0`) or Party 1 (`1`). The blue and yellow regions represent the two districts: both are spatially connected and contain a local majority of votes for Party 1.

Despite the presence of a vote for the opposing party, the solver successfully found a configuration in which both districts contain a sufficient majority for Party 1. This highlights how the model exploits spatial flexibility to concentrate votes and maximize representation—mimicking realistic gerrymandering strategies.

The fast execution time (0.009 s) and low model count (`Models = 1`) indicate that Clingo quickly reached and verified the optimum using the default configuration. This example serves as a qualitative validation of the model's correctness and effectiveness.

6.2 Average Resolution Time by Party

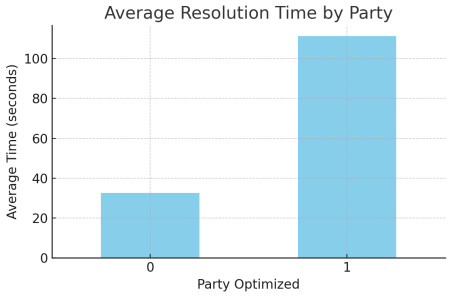

Figure 3: Average resolution time for each optimized party.

Figure 3 shows the average execution time required by the ASP solver when optimizing for each party. As illustrated, there is a substantial gap: optimizing for party 0 takes on average about 30 seconds, whereas for party 1 it takes around 110–115 seconds. This disparity indicates that, from a purely computational standpoint, the encoding is not entirely neutral—solving the same model with party 1 as the objective entails a significantly larger search effort. Such asymmetry may stem from differences in the structural characteristics of the generated instances (vote distribution, connectivity constraints, tie-breaking conditions, etc.) and should be taken into account when assessing the fairness and bilateral balance of the encoding.

6.3 Number of Representatives Obtained

Figure 4 offers a detailed comparison of the number of districts won by each party across all benchmark instances. Each pair of bars reflects the results of optimizing the district configuration in favor of Party 0 (blue) and Party 1 (yellow) respectively, under the constraint of identical vote distributions.

Despite the symmetric vote totals, several instances show one party securing a significantly greater number of districts than the other. This phenomenon reveals the impact of strategic district design and demonstrates how the same electoral landscape can yield drastically different political outcomes depending on how districts are drawn.

The visualization emphasizes the effectiveness of the ASP model in capturing this disparity, supporting the notion that gerrymandering can systematically benefit one party even in balanced scenarios. The absence of x-axis labels avoids visual clutter, allowing the reader to focus on the distributional trends and outcome disparities across instances.

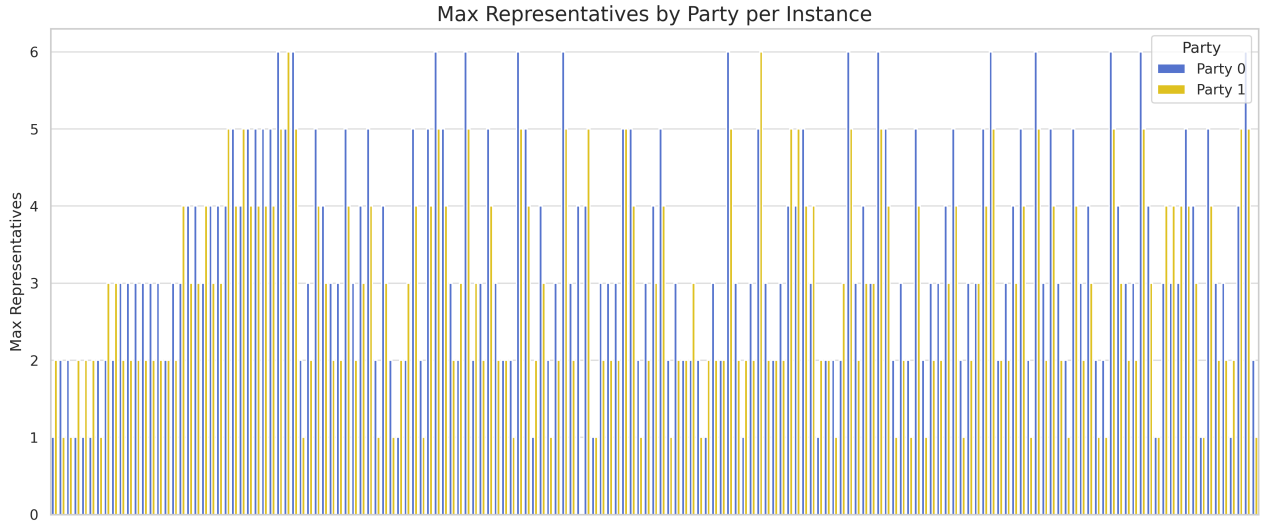

Figure 4: Number of districts won by each party in every benchmark instance.

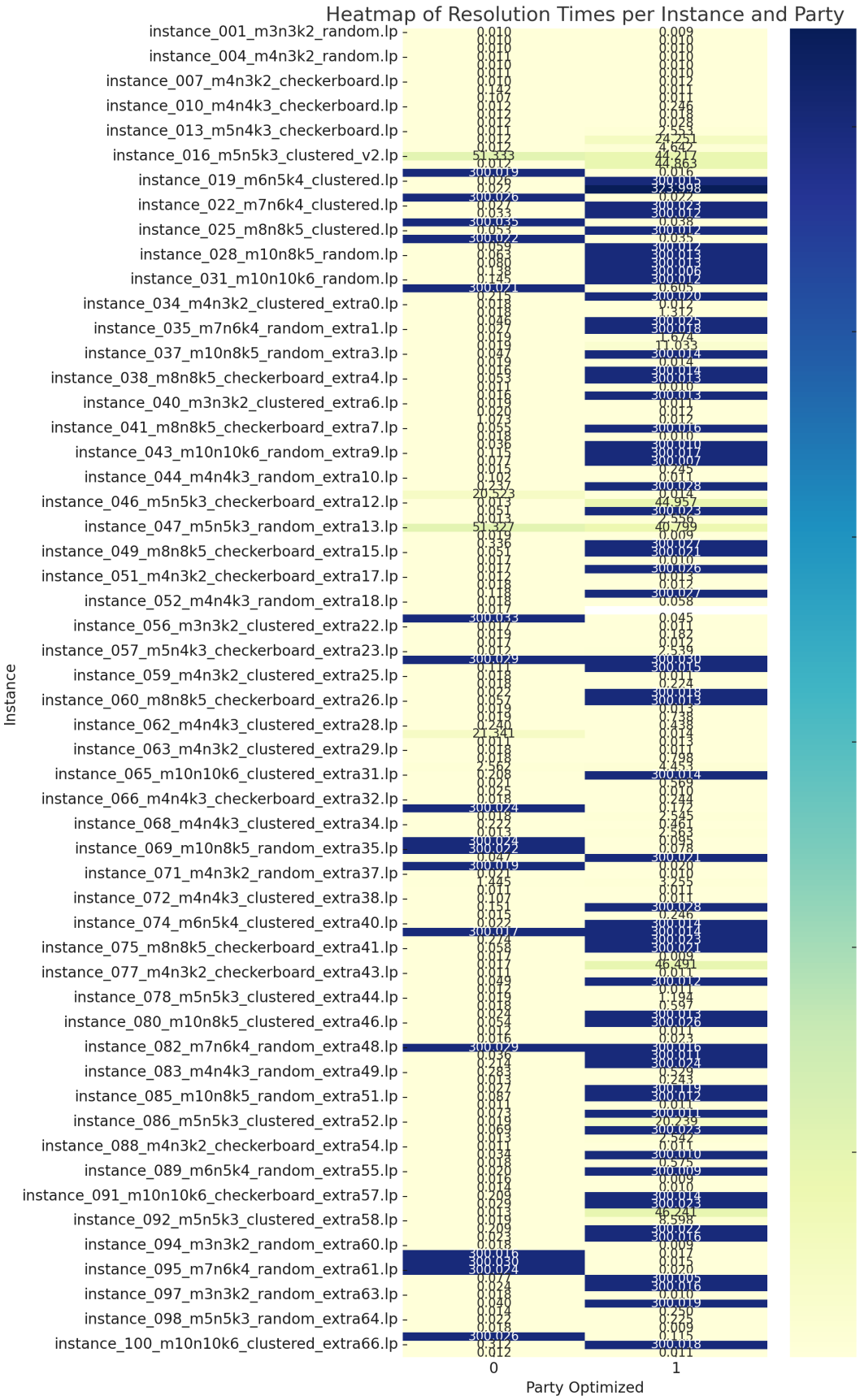

Figure 5: Heatmap of resolution time (in seconds) by instance and party.

6.4 Instance-by-Instance Time Analysis

The heatmap shown in Figure 5 offers a fine-grained view of computational performance across all benchmark instances. It enables the identification of patterns such as particularly hard cases (e.g., highly balanced voting distributions or larger grids) and the consistency of solver performance. Most instances are solved in less than 0.05 seconds, demonstrating the efficiency of the ASP encoding. A few instances approach higher runtimes but remain within the 5-minute timeout, confirming the scalability of the model.

6.5 Limitations and Future Improvements

The current model does not include constraints on uniform district size or compactness. Additionally, it does not consider multi-party dynamics or more complex demographic distributions. A possible extension is to add population constraints or geometric penalties to the objective function, in order to make the model more realistic. These enhancements would further increase the applicability of the ASP model in real-world electoral districting analyses.

7 Conclusions

This project aimed to investigate the gerrymandering problem through the lens of logic programming, using Answer Set Programming (ASP) as the primary tool for modeling and solving. A comprehensive ASP model was developed to represent a simplified yet meaningful version of the problem, in which a grid of voters is partitioned into contiguous districts with the goal of maximizing the representation of a given party. Model validation was supported by creating a large benchmark suite and conducting a rigorous experimental evaluation.

The obtained results confirm the adequacy and effectiveness of the ASP approach. The model proved correct in satisfying all imposed constraints, such as unique assignment of cells, district contiguity, and winner determination according to the majority rule. From a computational standpoint, the Clingo solver was able to solve most instances—including medium-sized ones—in very short times, and the more complex instances within the established timeout, demonstrating the practical feasibility of the approach. Notably, the experiments highlighted how district configuration can drastically influence election outcomes, allowing a party to secure a majority of seats even without an absolute majority of votes at the global level, thereby confirming the very nature of the gerrymandering problem. A symmetry in resolution times was also observed, regardless of which party was being optimized.

Despite these positive results, the current model has some limitations that open the way for future research directions. First, explicit constraints on district compactness or uniformity in size (e.g., in terms of number of voters) were not included—crucial aspects in real-world discussions of gerrymandering. Additionally, the model considers a two-party scenario and does not capture the complexity of multiparty systems or diverse demographic distributions within the electorate.

For the future, it would be interesting to extend the ASP model to incorporate such constraints, for example by introducing compactness measures (such as the area-to-perimeter ratio) or population-balancing requirements among districts. One could also explore modeling more complex objectives, such as creating competitive districts or minimizing "partisan bias." Another extension could involve analyzing scenarios with more than two political parties or integrating richer demographic data to simulate more realistic conditions. Finally, to tackle even larger instances, it may be useful to investigate integrating specialized heuristics or problem decomposition techniques within the ASP framework.