Answer Set Programming per Gerrymandering: Modellazione, Benchmark and Valutazione

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

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

06 Giugno 2025

Programmazione Dichiarativa
Università degli Studi di Parma

Abstract

Il *gerrymandering*, ovvero la manipolazione strategica dei confini dei distretti elettorali, rappresenta una sfida rilevante per l'equità dei sistemi democratici. Questo progetto esplora l'applicazione della *Answer Set Programming* (ASP), un paradigma di programmazione logica dichiarativa, per modellare e risolvere una versione astratta del problema del gerrymandering. L'obiettivo è massimizzare il numero di distretti vinti da un determinato partito all'interno di una griglia elettorale $m \times n$, suddividendola in k distretti contigui.

È stato sviluppato un modello ASP che definisce l'assegnazione delle celle elettorali ai distretti, garantisce la contiguità di ciascun distretto, calcola i voti e determina il vincitore di ogni distretto. A fini di valutazione, è stato generato un corpus di benchmark composto da 100 istanze, variando dimensioni della griglia, numero di distretti e strategie di distribuzione del voto.

Gli esperimenti condotti con il risolutore Clingo dimostrano la capacità del modello ASP di trovare soluzioni ottimali per entrambi i partiti, evidenziano l'impatto della suddivisione distrettuale sugli esiti elettorali e confermano l'efficienza computazionale dell'approccio ASP per questo complesso problema combinatorio.

1 Introduzione

Il *gerrymandering* è una strategia politica utilizzata per manipolare i confini dei distretti elettorali al fine di favorire un determinato partito o gruppo. Studi classici sul gerrymandering hanno mostrato come la manipolazione dei confini possa influenzare in modo significativo i risultati elettorali.

Formalmente, dato un grafo a griglia $m \times n$ i cui nodi (celle) sono etichettati con voti 0/1, il problema consiste nel suddividere i nodi in k sottoinsiemi 4-connessi (distretti), in modo da massimizzare il numero di sottoinsiemi in cui prevale il voto di un partito specifico.

In questo progetto adottiamo un'astrazione estrema: la regione geografica è modellata come una griglia regolare $m \times n$, e ogni cella rappresenta un elettore "puro" che vota solo 0 o 1. Non consideriamo popolazioni eterogenee, vincoli reali di compattezza (come forme poligonali) o criteri demografici, ma ci concentriamo esclusivamente sulla 4-connettività e sulla maggioranza all'interno di ciascun distretto.

È importante notare che, sebbene estremamente semplificato, questo modello cattura l'essenza combinatoria del gerrymandering: mostra come, anche in presenza di un numero bilanciato di voti su scala globale, la suddivisione in distretti possa alterare significativamente la distribuzione dei seggi.

In questa sezione abbiamo presentato il contesto e le motivazioni del problema. Nella Sezione 2 forniremo una definizione formale, mentre nella Sezione 3 illustreremo la nostra codifica ASP.

2 Definizione del Problema

Il problema di gerrymandering affrontato in questo progetto può essere definito formalmente come segue:

- L'area di voto è rappresentata da una matrice M di dimensione $m \times n$, in cui ogni cella corrisponde a un elettore.

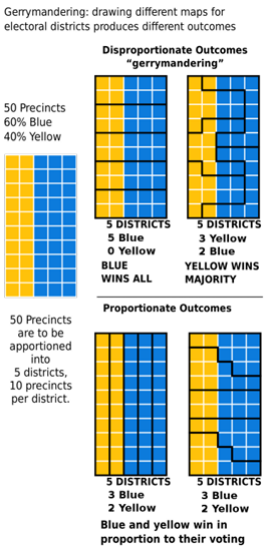

Figure 1: Esempio qualitativo di gerrymandering in una situazione reale: si noti come, nonostante un numero complessivo bilanciato di voti, il partito blu riesca a ottenere la maggioranza dei distretti.

- Ogni elettore supporta uno dei due partiti politici, codificati come 0 o 1, sulla base di previsioni di voto ottenute tramite l'analisi dei social media.
- L'area deve essere suddivisa in k **distretti**, ciascuno costituito da un insieme contiguo di celle adiacenti.
- All'interno di ogni distretto si tiene un'elezione. Il partito che ottiene la maggioranza dei voti in un distretto vince quel distretto.
- L'obiettivo è determinare una suddivisione ottimale della regione in k distretti, in modo da massimizzare il numero di distretti vinti da un determinato partito.
- Poiché i due partiti hanno obiettivi contrapposti, il piano di suddivisione ottimale viene calcolato separatamente per ciascun partito.

3 Codifica ASP per il Problema del Gerrymandering

Questa sezione descrive nel dettaglio la codifica logica sviluppata in Answer Set Programming (ASP) per affrontare il problema del gerrymandering. L'obiettivo è suddividere una griglia bidimensionale di celle elettorali in un numero fisso di distretti in modo tale che un determinato partito politico massimizzi il numero di rappresentanti eletti. La modellazione ASP si adatta bene a questo tipo di problema combinatorio grazie alla sua capacità di esprimere in modo conciso e dichiarativo vincoli complessi e regole di ottimizzazione.

Input e Definizioni di Base. Il programma ASP parte da una descrizione della griglia e delle preferenze di voto mediante i seguenti predicati:

- `cell(X,Y)`: specifica l'esistenza della cella nella posizione (X, Y);
- `vote(X,Y,P)`: indica che la cella (X, Y) vota per il partito $P \in \{0, 1\}$;
- `grid_size`(M, N): definisce le dimensioni della griglia (righe M, colonne N);
- `num_districts`(K): numero K di distretti da creare;
- `district_id`(D) e `party_id`(P): identificatori ausiliari per iterazione (D e P).

Il partito da ottimizzare è specificato tramite una costante passata in fase di esecuzione:

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

Listing 1: Definizione del Partito da Ottimizzare (Clingo)

Assegnazione delle Celle ai Distretti. Ogni cella è assegnata a un solo distretto:

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

Listing 2: Assegnazione Unica

Definizione di Adiacenza e Vincolo di Connettività. Per garantire che ogni distretto sia connesso, si definisce il concetto di adiacenza:

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

Listing 3: Adiacenza tra Celle

Si sceglie poi un nodo radice per ogni distretto:

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

Listing 4: Radice del Distretto

Dal nodo radice si propaga la raggiungibilità:

```
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: Raggiungibilità e Connettività

Conteggio dei Voti e Determinazione del Vincitore. I voti sono contati per partito all'interno di ciascun distretto:

```
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: Conteggio Voti per Distretto

Il vincitore è determinato per maggioranza:

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

Listing 7: Determinazione del Vincitore

In caso di parità, la vittoria è assegnata al partito 0:

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

Listing 8: Gestione della Parità

Funzione Obiettivo. Massimizzare il numero di distretti vinti dal partito ottimizzato:

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

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

Listing 9: Ottimizzazione della Rappresentanza

Proprietà Garantite dal Modello. Il programma ASP assicura:

- assegnazione univoca delle celle a un distretto;
- connettività spaziale dei distretti;
- determinazione univoca del vincitore (con regola di parità);
- ottimizzazione focalizzata sulla rappresentanza di un partito.

Questo approccio è modulare e flessibile, e può essere esteso facilmente per includere vincoli sulla dimensione o forma dei distretti.

4 Generazione dei Benchmark

Per testare e validare il modello ASP in condizioni diverse, è stato costruito un corpus di 100 istanze di benchmark utilizzando lo script Python `generate_benchmarks.py`. Le istanze sono salvate in file ASP contenenti fatti su griglia, voti e distretti.

Struttura dei File. Ogni file contiene:

- `grid_size`(M, N): dimensioni della griglia;
- `num_districts`(K): numero di distretti;
- `cell`(R, C): coordinate delle celle esistenti;
- `vote`(R, C, P): voto nella cella (R, C).

Parametri Strutturali. Per valutare il comportamento del modello ASP in contesti diversi, sono state generate istanze con griglie di varie dimensioni, suddivise in tre categorie:

- **Griglie piccole**: es. 3×3, 4×3, 4×4 — utili per test di correttezza e debugging.
- **Griglie medie**: es. 5×5, 6×5, 7×6, 8×8 — bilanciano complessità e realismo.
- **Griglie grandi**: fino a 10×10 — testano la scalabilità e simulano scenari realistici.

Il numero di distretti K è stato scelto per mantenere un rapporto significativo tra celle totali e distretti, in modo da:

- garantire che ogni distretto abbia abbastanza celle per permettere una competizione;
- evitare che il problema diventi banale (es. un distretto per riga) o eccessivamente frammentato;
- fornire configurazioni realistiche che riflettano vere situazioni elettorali.

Strategie di Generazione dei Voti.

- **Casuale**: voti assegnati con distribuzione uniforme;
- **Clusterizzata**: un partito ha maggiore concentrazione in certe aree;
- **A scacchiera**: alternanza regolare per massima eterogeneità.

Queste strategie simulano contesti urbani, rurali e polarizzati.

Obiettivi della Generazione. Il benchmark è stato progettato per:

- valutare la correttezza del modello;
- analizzare l'effetto della distribuzione dei voti sul risultato;
- testare la scalabilità rispetto a dimensione e complessità.

In questo modo è stata possibile un'analisi ampia e significativa del comportamento del modello ASP.

5 Valutazione Sperimentale

Gli esperimenti sono stati eseguiti su una macchina dotata di processore **AMD Ryzen 7 7700X** (8 core fisici, 16 thread), **32 GB di RAM**, e sistema operativo Windows 11. La CPU dispone di un'architettura cache multilivello composta da **512 KB di L1**, **8 MB di L2** e **32 MB di L3**, contribuendo a un accesso efficiente alla memoria e a una latenza ridotta durante l'elaborazione.

Il risolutore utilizzato è stato **Clingo 5.7.1**, eseguito tramite uno script Python dedicato sviluppato per automatizzare l'intero processo di benchmarking, inclusa la gestione dei timeout, l'estrazione dei dati e la raccolta dei risultati.

Ogni istanza di benchmark è stata risolta due volte: una volta ottimizzando per il partito 0 e una volta per il partito 1. Le esecuzioni sono state avviate con la configurazione `default` di Clingo, senza specificare euristiche o opzioni aggiuntive.

Per ciascuna esecuzione sono state registrate le seguenti metriche:

- tempo totale di esecuzione in secondi (`TimeSec`);
- stato del risolutore (`Status`);
- numero di modelli trovati (`Models`);
- numero di distretti vinti dal partito ottimizzato (`MaxReps`).

A ciascuna risoluzione è stato imposto un limite massimo di tempo di **5 minuti** (300 secondi). I risultati di tutte le esecuzioni sono stati salvati in un file `CSV` e successivamente analizzati per produrre i grafici di valutazione delle prestazioni.

6 Risultati e Discussione

6.1 Descrizione delle Colonne del CSV

Instance Identificatore dell'istanza di benchmark (es. `instance_001_m3n3k2_random.lp`). Si riferisce a una specifica configurazione di griglia, voti e distretti.

PartyOptimized Indica quale partito è stato favorito dal risolutore durante l'ottimizzazione. I valori possibili sono `0` o `1`, corrispondenti al Partito 0 o Partito 1.

Solver Strategy La maggior parte degli esperimenti ha utilizzato la configurazione predefinita di Clingo, che applica euristiche generaliste pensate per bilanciare prestazioni e generalità. Facoltativamente, Clingo può essere configurato con strategie personalizzate come `HeuristicA` o `GreedyLocal`, che influenzano la selezione delle variabili, le priorità di ramificazione e la risoluzione dei conflitti. Ad esempio:

- `HeuristicA` dà priorità agli atomi con maggiore frequenza di vincoli;
- `GreedyLocal` favorisce decisioni che portano a guadagni immediati.

Sebbene non siano state utilizzate attivamente in questo progetto, tali strategie sono rilevanti per studi futuri sull'impatto delle euristiche.

MaxReps Numero massimo di distretti vinti dal partito ottimizzato. Ad esempio, `MaxReps = 2` indica che il partito favorito ha ottenuto 2 distretti.

TimeSec Tempo totale di esecuzione in secondi (anche decimali). Ad esempio, `0.009` indica una soluzione trovata in 9 millisecondi.

Status Indica come si è conclusa l'esecuzione del risolutore:

- `OPTIMUM_FOUND`: è stata trovata e dimostrata la soluzione ottima;
- `SAT`: è stata trovata una soluzione valida (ma non necessariamente ottima);
- `UNSAT`: nessuna soluzione valida esiste;
- `TIMEOUT`: raggiunto il limite massimo di tempo;
- `ERROR`: si è verificato un errore interno.

Models Numero di modelli distinti (soluzioni migliorative) trovati. Un numero basso indica una rapida convergenza all'ottimo.

ClingoReturnCode Codice numerico restituito da Clingo:

- `0`: terminazione corretta;
- `10`: soluzione SAT trovata;
- `20`: problema dimostrato UNSAT;
- `30`: ottimo trovato;
- Altri: errori inaspettati.

RawOptValue Valore grezzo (negativo) della funzione obiettivo calcolato da Clingo. Poiché Clingo minimizza per impostazione predefinita, le vittorie di distretto sono codificate come numeri negativi:

- `-2` indica che il partito ha vinto 2 distretti;
- `-1` indica 1 distretto vinto.

6.1.1 Esempio di Partizionamento per `instance_001_m3n3k2_random.lp` (Partito 1)

La riga CSV

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

indica che, ottimizzando per il *Partito 1*, il risolutore ha trovato in 0.009 s (`TimeSec`) una partizione ottimale (`OPTIMUM_FOUND`) in cui il Partito 1 vince entrambi i distretti (`RawOptValue` = –2). È stato sufficiente generare un solo modello (`Models = 1`), anche se `MaxReps = 2` consentiva fino a due iterazioni.

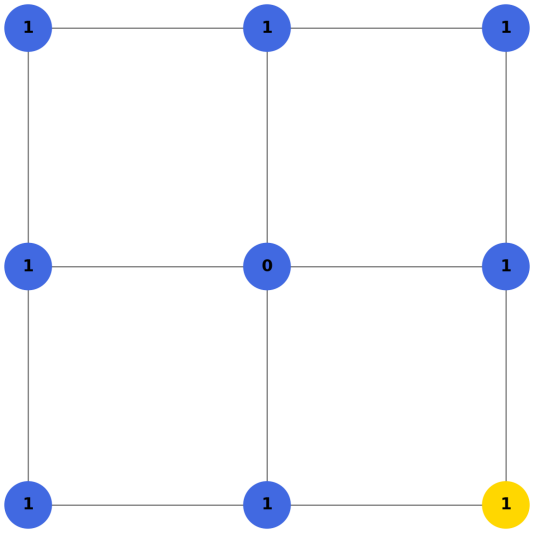

Figure 2: Partizionamento ottimale per `instance_001_m3n3k2_random.lp` con ottimizzazione per il Partito 1. Ogni cella mostra il suo voto (0 o 1), mentre i due distretti colorati—giallo e blu—sono entrambi vinti dal Partito 1.

La Figura 2 mostra una visualizzazione concreta della partizione ottimale calcolata. La griglia rappresenta una mappa di voto 3×3 dove ogni cella corrisponde a un elettore che vota per il Partito 0 (`0`) o il Partito 1 (`1`). Le regioni blu e gialla rappresentano i due distretti: entrambi sono connessi e contengono una maggioranza di voti per il Partito 1.

Nonostante la presenza di voti per l'altro partito, il risolutore ha trovato una configurazione in cui entrambi i distretti sono vinti dal Partito 1. Questo evidenzia come il modello sfrutti la flessibilità spaziale per concentrare i voti e massimizzare la rappresentanza, simulando realisticamente le strategie di gerrymandering.

Il tempo di esecuzione ridotto (0.009 s) e il numero minimo di modelli (`Models = 1`) confermano che Clingo ha identificato rapidamente l'ottimo utilizzando la configurazione predefinita. Questo esempio costituisce una validazione qualitativa dell'efficacia del modello.

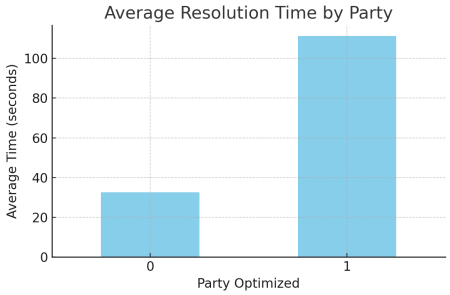

Figure 3: Tempo medio di risoluzione per ciascun partito ottimizzato.

6.2 Tempo Medio di Risoluzione per Partito

Figura 3 mostra il tempo medio di esecuzione richiesto dal solver ASP quando si ottimizza per ciascun partito. Come si può osservare, esiste una differenza marcata: ottimizzare per il partito 0 richiede mediamente circa 30 secondi, mentre per il partito 1 si sale a circa 110–115 secondi. Questa discrepanza indica che, da un punto di vista strettamente computazionale, l'encoding non è del tutto neutrale: risolvere lo stesso modello con target il partito 1 comporta un carico di ricerca significativamente maggiore. Tale asimmetria potrebbe derivare da differenze nella struttura delle istanze generate (distribuzione dei voti, vincoli di connessione, gestione dei pareggi...) e andrebbe tenuta in conto quando si valuta l'equità e la bilateralità del modello.

6.3 Numero di Rappresentanti Ottenuti

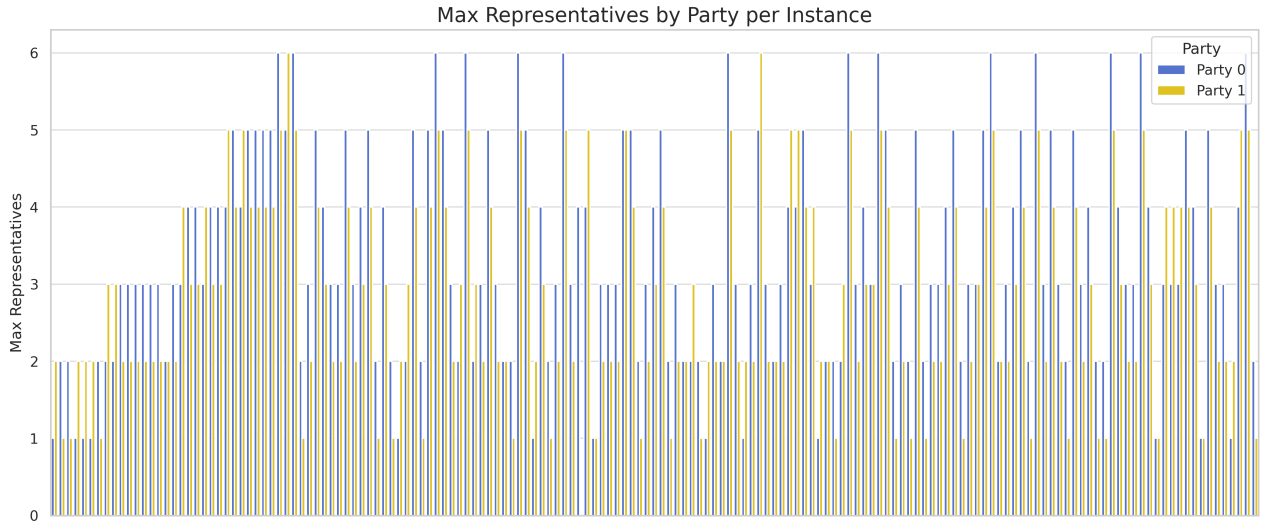

Figure 4: Numero di distretti vinti da ciascun partito in ogni istanza di benchmark.

La Figura 4 fornisce un confronto dettagliato tra il numero di distretti vinti da ciascun partito in tutte le istanze. Ogni coppia di barre riflette il risultato ottenuto ottimizzando la partizione a favore del Partito 0 (blu) e del Partito 1 (giallo), mantenendo costante la distribuzione dei voti.

Sebbene i voti totali siano simmetrici, diverse istanze mostrano un partito che ottiene significativamente più distretti dell'altro. Questo fenomeno dimostra l'impatto della suddivisione strategica dei distretti e come lo stesso scenario elettorale possa produrre risultati politici drasticamente diversi a seconda della configurazione.

La visualizzazione evidenzia l'efficacia del modello ASP nel catturare tale disparità, supportando l'idea che il gerrymandering possa sistematicamente avvantaggiare un partito anche in situazioni di equilibrio.

6.4 Analisi dei Tempi per Istanze

La heatmap della Figura 5 fornisce una vista dettagliata delle prestazioni computazionali per ogni istanza. Permette di identificare pattern come i casi particolarmente complessi (es. distribuzioni di voto equilibrate o griglie grandi) e la consistenza delle prestazioni del risolutore. La maggior parte delle istanze è risolta in meno

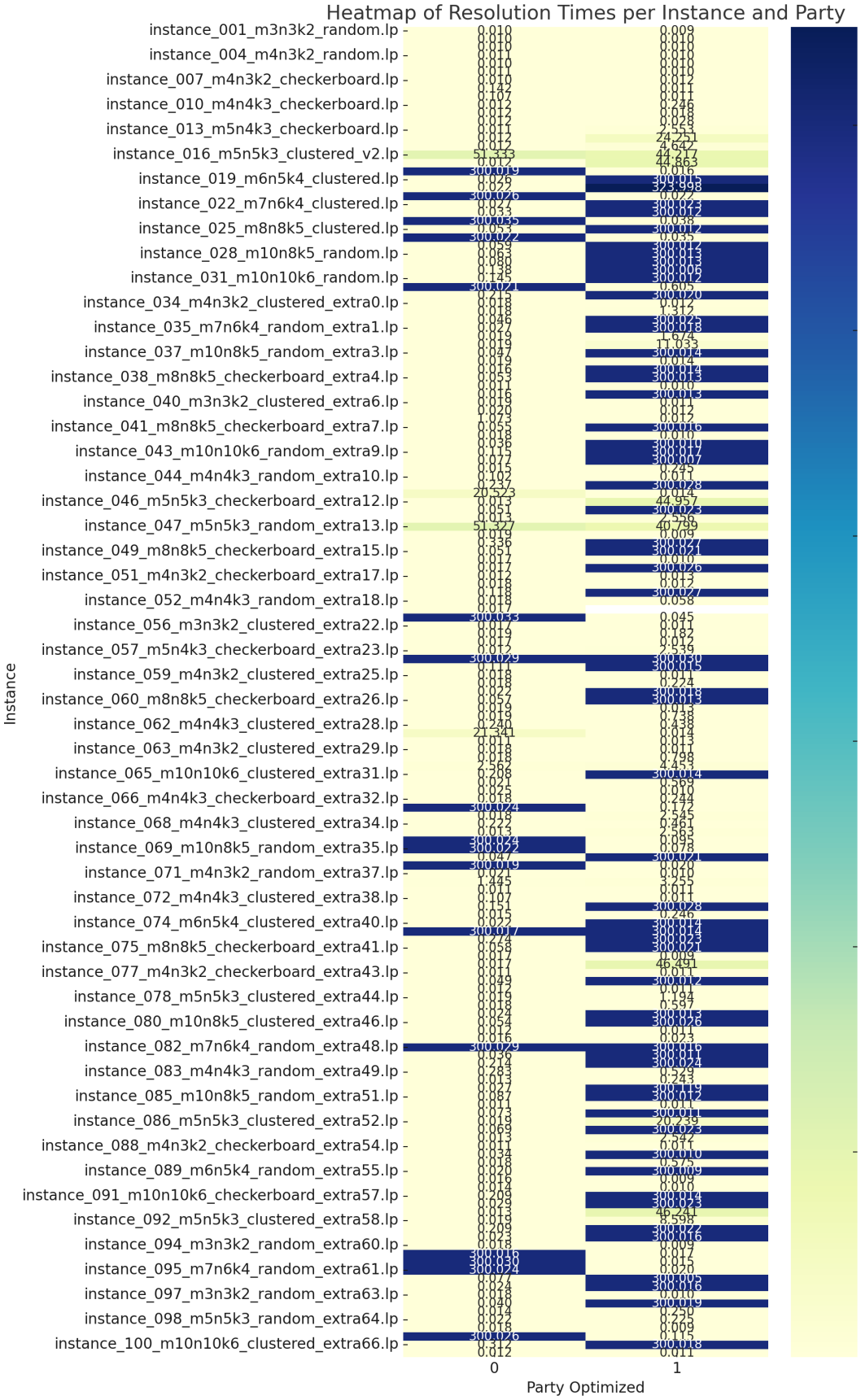

Figure 5: Heatmap dei tempi di risoluzione (in secondi) per istanza e partito.

di 0.05 secondi, dimostrando l'efficienza della codifica ASP. Alcune istanze si avvicinano al tempo massimo, ma restano comunque entro il timeout, confermando la scalabilità del modello.

6.5 Limitazioni e Miglioramenti Futuri

Il modello attuale non impone vincoli sulla dimensione uniforme o sulla compattezza dei distretti. Inoltre, non considera dinamiche multipartitiche o distribuzioni demografiche più complesse. Un'estensione possibile consiste nell'aggiungere vincoli sulla popolazione o penalità geometriche nella funzione obiettivo, rendendo il modello più realistico e applicabile a contesti elettorali reali.

7 Conclusioni

Questo progetto si è posto l'obiettivo di analizzare il problema del gerrymandering attraverso il paradigma della programmazione logica, utilizzando l'Answer Set Programming (ASP) come strumento principale per modellare e risolvere il problema. È stato sviluppato un modello ASP completo per rappresentare una versione semplificata ma significativa del problema, in cui una griglia di elettori viene suddivisa in distretti contigui con lo scopo di massimizzare la rappresentanza di un determinato partito. La validazione del modello è stata supportata dalla creazione di un ampio insieme di benchmark e da una rigorosa valutazione sperimentale.

I risultati ottenuti confermano l'adeguatezza e l'efficacia dell'approccio ASP. Il modello si è dimostrato corretto nel soddisfare tutti i vincoli imposti, come l'assegnazione univoca delle celle, la contiguità dei distretti e la determinazione dei vincitori secondo la regola di maggioranza. Dal punto di vista computazionale, il risolutore Clingo è stato in grado di risolvere la maggior parte delle istanze — comprese quelle di dimensione media — in tempi molto brevi, e le istanze più complesse entro il timeout prefissato, dimostrando la fattibilità pratica dell'approccio. In particolare, gli esperimenti hanno evidenziato come la configurazione dei distretti possa influenzare drasticamente l'esito delle elezioni, consentendo a un partito di ottenere la maggioranza dei seggi anche senza una maggioranza assoluta dei voti a livello globale, confermando così la natura intrinseca del problema del gerrymandering. È stata inoltre osservata una simmetria nei tempi di risoluzione, indipendentemente dal partito ottimizzato.

Nonostante questi risultati positivi, il modello attuale presenta alcune limitazioni che aprono la strada a futuri sviluppi. Innanzitutto, non sono stati inclusi vincoli espliciti sulla compattezza o sull'uniformità dei distretti (es. in termini di numero di elettori), aspetti cruciali nel dibattito reale sul gerrymandering. Inoltre, il modello considera uno scenario bipartitico e non cattura la complessità dei sistemi multipartitici o la diversità demografica all'interno dell'elettorato.

In futuro, sarebbe interessante estendere il modello ASP per includere tali vincoli, ad esempio introducendo misure di compattezza (come il rapporto area/perimetro) o requisiti di bilanciamento della popolazione tra i distretti. Si potrebbero inoltre esplorare obiettivi più complessi, come la creazione di distretti competitivi o la minimizzazione del "partisan bias". Un'altra possibile estensione riguarda l'analisi di scenari con più di due partiti politici o l'integrazione di dati demografici più ricchi per simulare condizioni più realistiche. Infine, per affrontare istanze di dimensioni ancora maggiori, potrebbe essere utile indagare l'integrazione di euristiche specializzate o tecniche di decomposizione del problema all'interno del framework ASP.