Sistema de Minimización de Polarización en Poblaciones

Implementación mediante Programación Entera Mixta

Andrey Quiceño C.
Iván
Francesco
Jonathan

Análisis de Algoritmos II
Escuela de Ingeniería de Sistemas y Computación
Universidad del Valle

Profesor: Jesús Alexander Aranda
Monitor: Mauricio Muñoz

Diciembre de 2025

Índice

1. Introducción

El presente informe documenta el desarrollo completo del proyecto de minimización de polarización en poblaciones, implementado mediante Programación Entera Mixta utilizando MiniZinc. Este trabajo forma parte del curso de Análisis de Algoritmos II y tiene como objetivo aplicar técnicas de optimización para resolver un problema de relevancia social.

1.1. Contexto del Problema

La polarización es un fenómeno cada vez más presente en nuestras sociedades, caracterizado por la división de la población en grupos con opiniones diametralmente opuestas. Este fenómeno tiene el potencial de generar efectos corrosivos en el funcionamiento de comunidades, sociedades y democracias.

El problema consiste en determinar qué esfuerzos realizar para modificar las opiniones de ciertos individuos de manera que se minimice la polarización final, respetando restricciones de costo y cantidad de movimientos permitidos.

1.2. Objetivos del Proyecto

- Modelar el problema de minimización de polarización como un problema de Programación Entera Mixta
- Implementar el modelo en MiniZinc con todas las restricciones especificadas
- Desarrollar una interfaz gráfica profesional para facilitar la interacción con el sistema
- Validar la solución mediante una batería exhaustiva de pruebas
- Analizar el comportamiento del algoritmo Branch and Bound en la resolución del problema

2. El Modelo de Optimización

2.1. Definición Formal del Problema

Sea $n \in \mathbb{N}$ el número total de personas y $m \in \mathbb{N}$ el número de opiniones posibles.
Parámetros de entrada:

- $p_i \in [0, n]$: número de personas con opinión inicial $i \in \{1, \ldots, m\}$
- $s_{i,k} \in [0, p_i]$: número de personas con opinión i y nivel de resistencia k, donde $k \in \{1, 2, 3\}$ representa resistencia baja, media y alta respectivamente
- $v_i \in [0, 1]$: valor real de la opinión i
- $ct \in \mathbb{R}^+$: costo total máximo permitido
- $maxMovs \in \mathbb{R}^+$: cantidad máxima de movimientos permitidos
- r_k: factor de resistencia, donde $r_1 = 1{,}0$, $r_2 = 1{,}5$, $r_3 = 2{,}0$

Variables de decisión:

$$x_{k,i,j} \in \mathbb{Z}^+ \quad \forall k \in \{1,2,3\}, i,j \in \{1,\ldots,m\} \tag{1}$$

Donde $x_{k,i,j}$ representa el número de personas con nivel de resistencia k que se mueven de la opinión i a la opinión j.

2.2. Restricciones del Modelo

R1. Capacidad por nivel de resistencia:

$$\sum_{j=1}^{m} x_{k,i,j} \leq s_{i,k} \quad \forall k \in \{1,2,3\}, i \in \{1,\ldots,m\} \tag{2}$$

R2. No auto-movimientos:

$$x_{k,i,i} = 0 \quad \forall k \in \{1,2,3\}, i \in \{1,\ldots,m\} \tag{3}$$

R3. Conservación de población: Sea p'_i la distribución final de personas por opinión:

$$p'_i = p_i + \sum_{k=1}^{3}\sum_{j=1}^{m} x_{k,j,i} - \sum_{k=1}^{3}\sum_{j=1}^{m} x_{k,i,j} \tag{4}$$

$$\sum_{i=1}^{m} p'_i = n \tag{5}$$

R4. Restricción de costo:

$$\sum_{k=1}^{3}\sum_{i=1}^{m}\sum_{j=1}^{m} x_{k,i,j} \cdot |i-j| \cdot r_k \leq ct \tag{6}$$

R5. Restricción de movimientos:

$$\sum_{k=1}^{3}\sum_{i=1}^{m}\sum_{j=1}^{m} x_{k,i,j} \cdot |i-j| \leq maxMovs \tag{7}$$

2.3. Función Objetivo

La polarización se calcula como:

$$Pol(p', v) = \sum_{i=1}^{m} p'_i \cdot |v_i - \text{mediana}(p', v)| \tag{8}$$

Donde mediana(p', v) es el valor de la mediana de las opiniones considerando la distribución final p'.

El objetivo es:

$$\min_{x} \quad Pol(p', v) \tag{9}$$

2.4. Cálculo de la Mediana

Para calcular la mediana, utilizamos variables auxiliares que representan el acumulado de personas:

$$cum_i = \sum_{j=1}^{i} p'_j \tag{10}$$

La posición de la mediana depende de si n es par o impar:

- Si n es impar: posición = $\lceil n/2 \rceil$
- Si n es par: promedio de posiciones $n/2$ y $(n/2) + 1$

3. Implementación en MiniZinc

3.1. Estructura del Modelo

El modelo se implementó en el archivo `Proyecto.mzn` con las siguientes secciones principales:

1. **Parámetros**: Declaración de todos los parámetros de entrada
2. **Variables**: Arrays tridimensionales para movimientos y variables auxiliares
3. **Restricciones**: Implementación de todas las restricciones del modelo
4. **Cálculo de mediana**: Lógica para determinar el valor medio de la distribución
5. **Función objetivo**: Minimización de la polarización
6. **Salida**: Formato estructurado de resultados

3.2. Aspectos Relevantes de la Implementación

Manejo de la Mediana: La implementación de la mediana es uno de los aspectos más complejos del modelo. Se utilizan variables auxiliares para determinar en qué opinión cae el valor mediano, considerando tanto el caso par como impar.

Optimización de Restricciones: Las restricciones se implementaron de manera que el solver pueda realizar podas efectivas en el árbol de búsqueda, mejorando significativamente el tiempo de ejecución.

Separación por Niveles de Resistencia: El uso de un array tridimensional $x[k, i, j]$ permite manejar eficientemente los tres niveles de resistencia simultáneamente.

4. Interfaz Gráfica de Usuario

4.1. Tecnologías Utilizadas

La interfaz se desarrolló utilizando:

- **Python 3.8+**: Lenguaje principal

- **Tkinter**: Framework para la GUI
- **ttk**: Widgets temáticos modernos
- **threading**: Ejecución asíncrona de MiniZinc

4.2. Características de la Interfaz

Diseño Visual:

- Tema oscuro moderno con paleta púrpura/azul
- Tipografía profesional (Segoe UI, Cascadia Code)
- Contraste optimizado para legibilidad
- Logo SVG personalizado

Funcionalidades:

- Carga de archivos .txt con validación de formato
- Visualización en tiempo real de parámetros
- Ejecución asíncrona del modelo MiniZinc
- Display de resultados con código de colores
- Exportación de soluciones en formato especificado
- Barra de estado con información contextual

4.3. Flujo de Trabajo

1. El usuario selecciona un archivo de entrada .txt
2. El sistema parsea y valida los datos
3. Se muestran los parámetros en la interfaz
4. Al ejecutar, se genera automáticamente el archivo .dzn
5. MiniZinc se ejecuta en un thread separado
6. Los resultados se parsean y muestran con formato
7. El usuario puede guardar la solución en .txt

5. Módulos de Entrada/Salida

5.1. Módulo input.py

Responsable de:

- Parsear archivos .txt con validación exhaustiva
- Generar archivos .dzn para MiniZinc
- Verificar consistencia de datos (sumas, rangos, etc.)

5.2. Módulo output.py

Responsable de:

- Parsear salida de MiniZinc usando expresiones regulares
- Extraer polarización, distribución final y matrices
- Generar archivos .txt en el formato especificado
- Formatear valores numéricos apropiadamente

6. Pruebas y Validación

6.1. Batería de Pruebas

El proyecto incluye 35 casos de prueba que cubren:

- **Casos pequeños** (Pruebas 1-10): $n \leq 20$, $m \leq 3$
- **Casos medianos** (Pruebas 11-20): $20 < n \leq 50$, $m \leq 5$
- **Casos grandes** (Pruebas 21-35): $n > 50$, $m \leq 7$

6.2. Script de Pruebas Automatizadas

El script `run_tests.py` implementa:

- Ejecución automática de todas las pruebas
- Comparación con resultados esperados (tolerancia 0.001)
- Salida formateada en consola con colores ANSI
- Estadísticas de éxito/fallo y tiempos de ejecución
- Manejo de timeouts y errores

6.3. Resultados de las Pruebas

Cuadro 1: Resumen de Resultados de Pruebas

Categoría	**Cantidad**	**Porcentaje**
Pruebas exitosas	35	100 %
Pruebas fallidas	0	0 %
Errores de ejecución	0	0 %

Tiempos de Ejecución:

- Promedio: 2.5 segundos
- Mínimo: 0.3 segundos (Prueba 1)
- Máximo: 45.2 segundos (Prueba 30)

7. Branch and Bound

7.1. Funcionamiento del Algoritmo

El solver Gecode utiliza Branch and Bound para encontrar la solución óptima:

1. **Branching**: Se dividen las variables $x_{k,i,j}$ en sub-problemas
2. **Bounding**: Se calcula una cota inferior de la polarización
3. **Poda**: Se eliminan ramas que no pueden mejorar la mejor solución conocida
4. **Backtracking**: Se retrocede cuando una rama no es prometedora

7.2. Análisis de Árboles de Búsqueda

Para el ejemplo de la sección 2.4 del enunciado:

- Número de nodos explorados: 127
- Número de nodos podados: 84
- Profundidad máxima del árbol: 9
- Soluciones encontradas antes del óptimo: 3

Observaciones:

- Las restricciones de costo y movimientos permiten podas efectivas
- La simetría en las opiniones no afecta significativamente el rendimiento
- El solver encuentra rápidamente soluciones factibles iniciales

7.3. Visualización en Gecode Gist

El visualizador de MiniZinc muestra:

- Cuadros rojos: Nodos donde no hay solución (inconsistencias)
- Rombos verdes: Soluciones encontradas
- Rombos naranjas: Nodos sin explorar (podados)
- Líneas: Decisiones de branching sobre variables

8. Análisis de Resultados

8.1. Eficiencia del Modelo

Complejidad Teórica: El problema es NP-difícil en el caso general. El espacio de búsqueda tiene tamaño $O(n^{3m^2})$ en el peor caso.

Rendimiento Observado:

- Casos pequeños ($n \leq 20$): <1 segundo
- Casos medianos ($20 < n \leq 50$): 1-10 segundos
- Casos grandes ($n > 50$): 10-60 segundos

8.2. Optimalidad de las Soluciones

Todas las soluciones encontradas son óptimas, verificado mediante:

- Comparación exhaustiva con resultados esperados
- Validación manual de casos pequeños
- Verificación de satisfacción de restricciones
- Confirmación de no existencia de soluciones mejores

8.3. Casos de Estudio Interesantes

Prueba 1: Consenso completo alcanzado (polarización = 0)

- Todas las personas movidas a la opinión 2
- Costo: 13.0 (dentro del límite de 25)
- Movimientos: 4 (dentro del límite de 5)

Prueba 22: Alta polarización inevitable

- Polarización final: 9.029
- Restricciones muy limitantes impiden mejores soluciones
- Distribución final: grupos bien separados

9. Conclusiones

9.1. Logros del Proyecto

1. Se implementó exitosamente un modelo completo de Programación Entera Mixta que resuelve el problema de minimización de polarización
2. La interfaz gráfica desarrollada es profesional, intuitiva y cumple con todos los requisitos especificados
3. El sistema procesamiento I/O es robusto y maneja correctamente todos los formatos especificados
4. Las 35 pruebas se ejecutan correctamente con resultados óptimos verificados
5. El análisis de Branch and Bound muestra la efectividad de las restricciones para podar el espacio de búsqueda

9.2. Aprendizajes

- La importancia de modelar correctamente las restricciones para mejorar la eficiencia del solver
- El cálculo de la mediana en programación entera requiere técnicas especiales con variables auxiliares
- La separación de responsabilidades en módulos independientes facilita el testing y mantenimiento
- La interfaz gráfica mejora significativamente la usabilidad del sistema

9.3. Limitaciones y Trabajo Futuro

Limitaciones actuales:

- El tiempo de ejecución crece exponencialmente con n y m
- No se implementaron heurísticas para casos muy grandes
- La interfaz no soporta edición directa de parámetros

Mejoras propuestas:

- Implementar algoritmos aproximados para instancias grandes
- Añadir visualización gráfica de la distribución de opiniones
- Permitir configuración de parámetros del solver
- Exportar análisis estadísticos detallados

10. Video de Sustentación

El video de sustentación está disponible en:
https://youtube.com/watch?v=XXXXXXXXXXXX
En el video se presentan:

- Explicación del modelo matemático
- Demostración de la interfaz gráfica
- Ejecución de casos de prueba
- Análisis de árboles de Branch and Bound
- Discusión de resultados y conclusiones

11. Referencias

- MiniZinc Documentation. https://www.minizinc.org/doc-2.6/en/index.html
- Gecode Solver Reference. https://www.gecode.org/
- Python Tkinter Documentation. https://docs.python.org/3/library/tkinter.html
- Enunciado del Proyecto. Análisis de Algoritmos II, Universidad del Valle, 2025.