Moderando el conflicto interno de opiniones en una red social

Calderón Prieto Brandon (2125974) Cely Archila Juleipssy Daianne (2122036)
Fonseca Idarraga Juan David (2323942)

25 de julio de 2025

Índice general

0.1. Modelo genérico 1
0.1.1. Parámetros 1
0.1.2. Variables 1
0.1.3. Restricciones 1
0.1.4. Función objetivo 2
0.1.5. Clasificación 2
0.2. Implementación 2
0.3. Análisis de Branch and Bound 2
0.3.1. Descripción del mecanismo 2
0.3.2. Análisis de árboles generados 3
0.4. Instancias y pruebas 5
0.4.1. Instancias de prueba provistas 5
0.4.2. Instancias adicionales generadas 5
0.5. Análisis de resultados 5
0.6. Conclusiones 5

0.1. Modelo genérico

0.1.1. Parámetros

- $n \in \mathbb{N}$: número total de personas.
- $m \in \mathbb{N}$: número total de opiniones.
- $p \in \mathbb{N}^m$: vector con la distribución de personas por opinión, donde p_i es el número de personas que inicialmente tienen la opinión $i \in 1 \dots m$, $\sum_{i=1}^{m} p_i = n$.
- $e \in [0,1]^m$: vector con los valores de extremismo de las opiniones, donde $e_i \in [0,1]$ es el valor de extremismo de la opinión $i \in 1 \dots m$.
- c: matriz de costes, donde $c_{i,j} \in \mathbb{R}^+$ es el coste de mover una persona de la opinión i a la opinión j, para $i, j \in 1 \dots m$ ($c_{i,i} = 0$).
- ce: vector de coste extra, donde $ce_i \in \mathbb{R}^+$ es el coste adicional de mover una persona a la opinión i si esa opinión estaba inicialmente vacía, para $i \in 1 \dots m$.
- $ct \in \mathbb{R}^+$: coste total permitido.
- $M \in \mathbb{N}$: número máximo de movimientos permitidos.

0.1.2. Variables

Una matriz s, donde $s_{i,j} \in \mathbb{N}$ es el número de personas movidas de la opinión i a la opinión j, para $i, j \in 1 \dots m$.

Esta matriz es de dimensiones $m \times m$ y debe cumplir las siguientes restricciones:

- $\sum_{j=1}^{m} s_{i,j} = p_i$: para cada opinión inicial i, la suma de personas que se mueven desde esa opinión hacia todas las demás (incluida ella misma) debe ser igual al número de personas que originalmente tenían la opinión i. $\sum_{i=1}^{m} \sum_{j=1}^{m} s_{i,j} = n$.
- $s_{i,j} \geq 0$ para todo $i, j \in 1 \dots m$.

0.1.3. Restricciones

Numero de movimientos

$$\sum_{i=1}^{m} \sum_{j=1}^{m} s_{i,j} \cdot |j - i| \leq M \tag{1}$$

Coste total

$$\sum_{i=1}^{m} \sum_{j=1}^{m} c_{i,j} \left(1 + \frac{p_i}{n}\right) * s_{i,j} + \delta_{p_j,0} \cdot ce_j * s_{i,j} \leq ct \tag{2}$$

$\delta_{p_j,0}$ es una función indicadora que vale 1 si $p_j = 0$ y 0 en caso contrario.

0.1.4. Función objetivo

La idea es minimizar el extremismo, que se calcula con la siguiente fórmula:

$$E(p', e) = \sum_{i=1}^{m} {p'}_i * e_i \tag{3}$$

- p': vector con la distribución de personas tras aplicar los movimientos de s. $p'[j] = \sum_{i=1}^{m} s[i, j]\ \forall j = 1 \dots m$.
- e: vector con los valores de extremismo de las opiniones.

0.1.5. Clasificación

Aunque todas las variables de decisión son enteras, para modelar las restricciones es necesario usar variables de tipo `float`. Esto hace que el modelo sea un *Programación Lineal Entera Mixta*.

0.2. Implementación

Gracias a las instrucciones de modelado de MiniZinc y la naturaleza del problema, las restricciones y la función objetivo se pueden expresar en pocas líneas de código.

La implementación puede describirse en los siguientes pasos:

- Computación de distancias: para evitar el cálculo repetido de las distancias entre opiniones, se crea una matriz d donde $d_{i,j} = |i - j|$.
- Conservación de flujos ($\sum_{j=1}^{m} s_{i,j} = p_i$): se hace con el fin de que nadie "desaparezca". También se añade la restricción $s_{i,j} \leq p_i$ para acotar dominios y acelerar la búsqueda.
- Cálculo de la distribución final: para calcular la distribución final se usó la fórmula $p'[j] = \sum_{i=0}^{m} s[i, j]$ $\forall j = 1 \dots m$, personas que "terminan.$^{\text{en}}$ la opinion j.
- Integración con Python: se utiliza la librería minizinc para ejecutar el modelo de MiniZinc desde Python, permitiendo una mayor flexibilidad en la gestión de instancias y resultados.

Obsérvese que `s`, `total_moves` y `p´` son variables enteras, mientras que `total_cost` y `extremism` son continuas. Por ello, el modelo es de tipo *Programación Lineal Entera Mixta*.

0.3. Análisis de Branch and Bound

0.3.1. Descripción del mecanismo

Branch and Bound es un algoritmo utilizado para resolver problemas de optimización entera y combinatoria. Su objetivo es hallar la solución óptima sin evaluar todas las combinaciones posibles, lo cual lo hace eficiente en espacios de búsqueda grandes y discretos.

- **Ramificación**: se divide el problema original en subproblemas más pequeños al restringir los valores de ciertas variables, generando un árbol de decisión.
- **Acotamiento**: se calcula una cota (superior o inferior) para cada nodo del árbol, que indica el mejor valor posible que se puede alcanzar desde ese nodo.
- **Poda**: se descartan nodos cuyo valor no puede superar a la mejor solución encontrada hasta el momento o que no tienen solución factible.

Gracias a este enfoque, el algoritmo reduce significativamente el número de soluciones que debe explorar, encontrando la solución óptima de manera más eficiente.

0.3.2. Análisis de árboles generados

El árbol de decisión permite visualizar de manera estructurada el proceso de búsqueda de soluciones dentro de un espacio factible. A través del uso de distintos símbolos y colores, se representan los diversos tipos de nodos involucrados en el análisis, facilitando la interpretación de las estrategias aplicadas, como la exploración y la poda. A continuación, se explican los elementos principales del gráfico:

- **Círculos azules**: representan nodos internos o intermedios del árbol. Son puntos de decisión donde aún es posible realizar divisiones, es decir, donde se evalúan variables y se generan nuevas ramas.
- **Triángulos rojos**: corresponden a nodos que fueron podados, es decir, no continuaron su expansión. Esto ocurre cuando se determina, mediante el uso de cotas (superior o inferior), que seguir explorando esa rama no aportará mejores soluciones. Este tipo de poda es característico de métodos como *Branch and Bound*.
- **Rombos verdes**: indican nodos que representan soluciones factibles, es decir, alternativas que cumplen con todas las restricciones del problema. Estos nodos suelen ubicarse al final de cada rama (hojas del árbol).
- **Cuadrados rojos, naranjas y otros**: Representan nodos que también fueron podados, pero por otras razones, como violaciones de restricciones, inconsistencias o límites establecidos por heurísticas. Por ejemplo:
  - **Cuadrado rojo**: señala una poda por factibilidad, cuando el nodo no cumple con las restricciones.
  - **Cuadrado naranja**: indica una poda temprana, aplicada por criterios adicionales como límites de tiempo, profundidad o guías heurísticas.

En la parte inferior del gráfico se presenta una leyenda numérica que resume el comportamiento del árbol:

- Depth: Profundidad máxima alcanzada.
- Número de soluciones factibles encontradas.
- Nodos que fueron podados.
- Nodos abiertos o explorados durante el proceso.

Este tipo de representación resulta útil para comprender el alcance de la exploración, la eficiencia del algoritmo y el impacto de las estrategias de poda aplicadas.

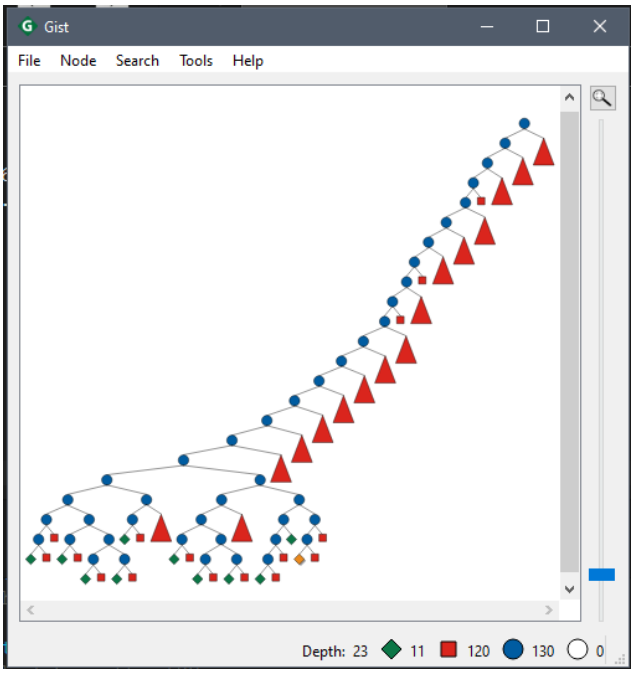

Figura 1: Árbol de decisión - prueba número 1

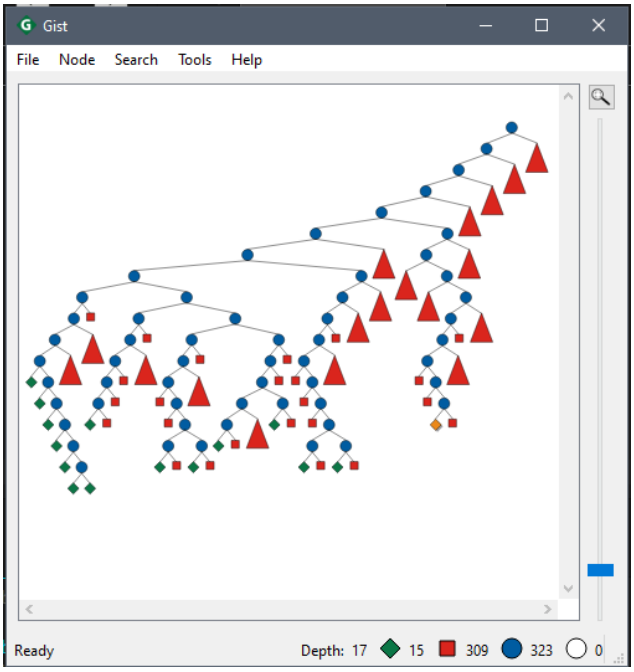

Figura 2: Árbol de decisión - prueba número 2

0.4. Instancias y pruebas

0.4.1. Instancias de prueba provistas

Se nos asignó una batería de pruebas compuesta por 30 instancias. En todos los casos se logró encontrar soluciones óptimas. Para la ejecución del modelo, se utilizó el solver HiGHS a través de la biblioteca MiniZinc integrada en Python. En la tabla que se presenta a continuación, se pueden observar los resultados obtenidos para algunas de estas pruebas.

Cuadro 1: Resumen de resultados - Pruebas provistas

Prueba	Costo	Soluciones	Tiempo	Nodos	Fallos	Valor Óptimo
0	6.3	16	562ms	649	309	6.3
2	4.582	11	0.019ms	235	107	4.582
3	0.261	12	0.101ms	163	69	0.261
4	1.06	38	118ms	1738	831	1.06
6	9.583	8	288ms	4876	2430	9.583

0.4.2. Instancias adicionales generadas

Creamos un conjunto de 5 instancias personalizadas, cuyos resultados se detallan a continuación:

Cuadro 2: Resumen de resultados - Pruebas generadas

Prueba	Costo	Soluciones	Tiempo	Nodos	Fallos	Valor Óptimo
1	11.9	13	10.39s	250981	125478	17.38
2	35.0	6	390ms	109	49	5.3
3	55.6	21	2.3ms	107	0	12.326
4	145	35	11.08ms	163,369	81,650	9.7
5	16.9985	15	1.78s	35883	17927	13.521

Las instancias utilizadas son adecuadas para evaluar el modelo, ya que presentan una variedad de condiciones que permiten analizar su desempeño de forma integral. Se incluye diversidad en tamaños y extremismos (p y e), lo cual permite observar cómo el modelo responde ante elementos con demandas muy distintas. Además, se emplean matrices de costos (c) no uniformes, es decir, estructuras en las que los valores de transporte entre elementos varían significativamente en lugar de mantenerse constantes o simétricos. Esto obliga al modelo a buscar rutas realmente eficientes en lugar de elegir caminos evidentes o arbitrarios.

Asimismo, las restricciones impuestas (ct y M) fueron definidas para evitar soluciones triviales, exigiendo que el modelo realice un proceso de optimización más riguroso. Finalmente, al mantener constante el número de elementos y variar otros parámetros, se facilita el análisis del impacto que generan pequeños cambios en los datos sobre la solución final.

0.5. Análisis de resultados

0.6. Conclusiones