Universidad del Valle
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación

Moderando el Conflicto Interno de Opiniones en una Red Social

Anderson Johan Alban Angulo - 202310006
Andrés Felipe Asprilla Urrutia - 202224101
Andrés Mauricio Ortiz Bermúdez - 202110330
Carlos Mauricio Tovar Parra - 201741699

Profesor

Jesús Alexander Aranda Bueno Ph.D.

Curso

Análisis y Diseño de Algoritmos II (750020C)

19 de abril de 2025

Índice

1. Introducción **6**

2. Funciones auxiliares **7**

3. Algoritmo Fuerza Bruta **9**
3.1. Descripción . . . 9
3.2. Implementación . . . 9
3.3. Pseudocódigo . . . 10
3.4. Complejidad . . . 10
3.4.1. Complejidad Temporal . . . 10
3.4.2. Complejidad Espacial . . . 11
3.5. Corrección . . . 11
3.6. Análisis . . . 12
3.7. Impacto en el Rendimiento . . . 14

4. Algoritmo Dinámico **16**
4.1. Descripción . . . 16
4.2. Pasos Programación Dinámica . . . 16
4.2.1. Caracterizando la Estructura de una Solución Óptima . . . 16
4.2.2. Definiendo Recursivamente el Valor de la Solución Óptima . . . 18
4.2.3. Calculando el Valor de la Solución Óptima . . . 19
4.2.4. Construyendo la Solución Óptima . . . 19
4.2.5. ¿Cuántos Subproblemas Hay? . . . 19
4.3. Implementación . . . 20
4.4. Pseudocódigo . . . 22
4.5. Complejidad . . . 23
4.5.1. Complejidad Temporal . . . 23
4.5.2. Complejidad Espacial . . . 23
4.5.3. ¿Es Útil en la Práctica? . . . 23
4.6. ¿El Algoritmo Siempre da la Respuesta Correcta al Problema? . . . 26
4.7. Análisis . . . 27
4.8. Impacto en El Rendimiento . . . 28

5. Algoritmo Voraz **29**
5.1. Descripción . . . 29
5.2. Definición de la heurística . . . 29
5.3. ¿Entendimos el Algoritmo? . . . 30
5.3.1. Solución del Ejercicio 2.3.1 . . . 30
5.3.2. Solución del Ejercicio 2.3.2 . . . 32

5.4. Implementación 35
5.5. Pseudocódigo 36
5.6. Complejidad 36
5.6.1. Complejidad temporal 36
5.6.2. Complejidad espacial 37
5.7. Corrección 37
5.7.1. ¿Cuándo da la Respuesta Correcta? 38
5.7.2. ¿Cuándo no da la Respuesta Correcta? 39
5.8. Análisis 39

6. Comparación de Algoritmos **41**
6.1. Solución Óptima 41
6.2. Tiempos de Ejecución 45
6.3. Ventajas y Desventajas 49
6.3.1. Fuerza Bruta 49
6.3.2. Programación Dinámica 49
6.3.3. Voraz 50

7. Casos de Prueba **51**
7.1. Caso de Prueba 1 51
7.2. Caso de Prueba 2 52
7.3. Caso de Prueba 3 52
7.4. Caso de Prueba 4 53
7.5. Caso de Prueba 5 53

8. Conclusiones **54**

9. Ejecución del proyecto **55**

Índice de cuadros

1. Datos entrada: análisis fuerza bruta 12
2. Combinaciones posibles fuerza bruta 13
3. Resultados obtenidos: análisis fuerza bruta. 14
4. Resultados ordenados por k y tamaño de entrada n. 25
5. Tiempos de ejecución mínimo y máximo por número de grupos 28
6. Datos de entrada de la red social $\mathcal{RS}_1$ 30
7. Utilidades y costos unitarios de la red social $\mathcal{RS}_1$ 30
8. Cálculo del conflicto interno con la estrategia E_V para $\mathcal{RS}_1$ 31
9. Datos de entrada de la red social $\mathcal{RS}_2$ 32
10. Utilidades y costos unitarios de la red social $\mathcal{RS}_2$ 33
11. Cálculo del conflicto interno con la estrategia E_V para $\mathcal{RS}_2$ 34
12. Comparación de los resultados obtenidos por los distintos algoritmos para los diferentes archivos de prueba 41
13. Tiempos de ejecución para los distintos algoritmos en cada archivo de prueba 45
14. Comparación general entre algoritmos. 48

Índice de figuras

1. Gráfica del conflicto interno generado por fuerza bruta. 42
2. Gráfica del conflicto interno generado por programación dinámica. 43
3. Gráfica del conflicto interno generado por algoritmo voraz. 43
4. Gráfica del conflicto interno comparando los tres metodos. 44
5. Gráfica comparativa del tiempo de respuesta obtenido por cada algoritmo. 46
6. Gráfica del tiempo de respuesta obtenido por fuerza bruta. 46
7. Gráfica del tiempo de respuesta obtenido por programación dinámica. 47
8. Gráfica del tiempo de respuesta obtenido por algoritmo voraz. 47
9. Caso de Prueba 1 51
10. Caso de Prueba 2 52
11. Caso de Prueba 3 52
12. Caso de Prueba 4 53
13. Caso de Prueba 5 53

Lista de Algoritmos

1. modciFB: Estrategia óptima por fuerza bruta 10
2. modciPD: Estrategia óptima con programación dinámica 22
3. modciPV: Estrategia voraz basada en utilidad heurística 36

1. Introducción

En este proyecto se aborda el problema de la moderación del conflicto interno de opiniones en una red social ficticia, denominada ModCI. Este fenómeno se presenta cuando un mismo individuo sostiene opiniones inconsistentes frente a dos afirmaciones relacionadas con un mismo tema, lo que refleja una falta de coherencia interna. Esta incoherencia puede ser indicativa de una comprensión superficial o confusa del tema, y su presencia generalizada en una red social puede afectar la calidad del debate y la estabilidad de las posturas colectivas.

El objetivo principal del proyecto es diseñar e implementar soluciones algorítmicas que permitan reducir el conflicto interno de la red social, modificando las opiniones de algunos de sus agentes de manera estratégica, teniendo en cuenta el nivel de rigidez de cada uno y un presupuesto limitado de esfuerzo disponible. Para ello, se utilizaron y compararon tres enfoques clásicos del diseño de algoritmos: fuerza bruta, algoritmos voraces y programación dinámica.

Cada enfoque será aplicado al problema ModCI para identificar qué agentes deben ser priorizados en el proceso de ajuste de opinión, con el fin de maximizar la reducción del conflicto interno total en la red. Además de implementar los algoritmos, se realizará un análisis comparativo considerando la optimalidad de las soluciones y la eficiencia computacional de cada estrategia.

Este proyecto busca que los estudiantes pongan en práctica las técnicas aprendidas para enfrentar problemas de naturaleza combinatoria, y que sean capaces de evaluar críticamente las ventajas y limitaciones de diferentes enfoques algorítmicos, tanto desde el punto de vista teórico como práctico.

2. Funciones auxiliares

Se implementaron varias funciones auxiliares que permiten calcular de manera modular el esfuerzo de moderación aplicado a una red social y el conflicto interno resultante. Estas funciones son esenciales para evaluar cualquier estrategia de solución (fuerza bruta, voraz o dinámica).

$$\begin{aligned}
\textbf{CI} &: \text{Conflicto Interno} \\
\mathcal{RS} &: \text{Red Social} \\
o_{i,1} &: \text{Primera opinión del grupo } i \\
o_{i,2} &: \text{Segunda opinión del grupo } i \\
r_i &: \text{Rigidez del grupo } i \\
E &: \text{Estrategia de moderación}
\end{aligned}$$

1. `calcular_conflicto`

Esta función estima el nivel de conflicto interno de una red social, considerando una posible estrategia de moderación. Si no se proporciona una estrategia, se asume que todos los agentes conservan sus opiniones originales. El cálculo sigue la definición de conflicto interno:

$$CI(\mathcal{RS}) = \frac{\sum_{i=0}^{n-1} \left(n_i^{\mathcal{RS}} * \left(o_{i,1}^{\mathcal{RS}} - o_{i,2}^{\mathcal{RS}} \right)^2 \right)}{n}$$

donde e_i es la cantidad de agentes moderados del grupo i. El conflicto se promedia respecto al número total de grupos

- **Entrada:** `red_social` (lista de tuplas), `estrategia` (lista opcional).
- **Salida:** Valor del conflicto interno (real).

2. `calcular_esfuerzo`

Calcula el esfuerzo total requerido para aplicar una estrategia de moderación sobre los grupos de la red. El esfuerzo se calcula como:

$$Esfuerzo(\mathcal{RS}, E) = \sum_{i=0}^{n-1} \left[\left| o_{i,1}^{\mathcal{RS}} - o_{i,2}^{\mathcal{RS}} \right| * r_i^{\mathcal{RS}} * e_i \right]$$

- **Entrada:** `red_social`, `estrategia`.
- **Salida:** Esfuerzo total requerido (entero).

3. `calcular_ci_esfuerzo`

Función compuesta que retorna simultáneamente el conflicto interno y el esfuerzo total asociado a una estrategia. Es usada como función de evaluación principal por los tres enfoques de solución.

- **Entrada:** `red_social`, `estrategia`.
- **Salida:** Tupla `(conflicto, esfuerzo)`.

4. `extraer_datos_red`

Facilita la manipulación de la red social al extraer sus atributos en listas separadas.

- **Entrada:** `red_social`.
- **Salida:** Cuádrupla de listas: `(n, opiniones_1, opiniones_2, rigidez)`.

3. Algoritmo Fuerza Bruta

3.1. Descripción

El enfoque de fuerza bruta para resolver el problema ModCI consiste en examinar exhaustivamente todas las posibles formas en que puede distribuirse el esfuerzo disponible entre los grupos de agentes de una red social, con el fin de minimizar su conflicto interno. Este enfoque parte de la premisa de considerar cada combinación factible de cambios en las opiniones de los agentes, evaluando en cada caso el nivel de conflicto resultante. De todas las estrategias que respetan el límite de esfuerzo permitido, se elige aquella que produce el menor conflicto interno. Al contemplar el espacio completo de soluciones, este método asegura encontrar la solución óptima. No obstante, su principal desventaja radica en el elevado costo computacional que implica, lo que restringe su uso práctico a redes de tamaño reducido.

3.2. Implementación

La implementación del algoritmo de fuerza bruta se realizó empleando una representación estructurada de la red social como una lista de tuplas. Cada tupla modela un grupo de agentes e incluye cuatro elementos: la cantidad de individuos en el grupo, sus opiniones frente a dos afirmaciones específicas, y su nivel de rigidez al cambio de opinión.

El proceso se estructura en dos fases principales:

1. **Generación Exhaustiva de Estrategias:** Se construye el conjunto completo de combinaciones posibles de moderación sobre los grupos de agentes. Para cada grupo, se consideran todos los valores enteros desde 0 hasta el número total de agentes del grupo, lo cual representa cuántos de sus miembros podrían modificar su opinión. Al combinar estas posibilidades para todos los grupos, se genera el espacio total de estrategias viables.S

2. **Evaluación y Selección de la Mejor Estrategia:** Cada combinación es evaluada mediante la función auxiliar 3. `calcular_ci_esfuerzo`, la cual calcula tanto el conflicto interno como el esfuerzo requerido para esa estrategia. Si el esfuerzo total no supera el máximo permitido $\mathtt{R}_{\max}$ y el conflicto interno es menor al mejor encontrado hasta el momento, se actualiza la mejor solución.

El resultado final es una tupla que contiene: el conflicto interno mínimo alcanzado, el esfuerzo total utilizado y la estrategia óptima de asignación de esfuerzos.

3.3. Pseudocódigo

Algoritmo 1: modciFB: Estrategia óptima por fuerza bruta

Input: Una red social RS como lista de tuplas $(n_i, op1_i, op2_i, rigidez_i)$ para cada grupo i, y un esfuerzo máximo permitido R_{max}

Output: Tupla $(CI_{\text{mín}}, Esfuerzo_{\text{total}}, Estrategia_{\text{óptima}})$

$mejorCI \leftarrow \infty$; // Inicializa el conflicto mínimo como infinito

$mejorEsfuerzo \leftarrow 0$; // Inicializa el esfuerzo asociado

$mejorEstrategia \leftarrow []$; // Inicializa la mejor estrategia vacía

$estrategias \leftarrow [[]]$; // Lista inicial con una estrategia vacía

foreach *grupo* g_i *en* RS **do**

  $nuevasEstrategias \leftarrow []$

  foreach *estrategia en estrategias* **do**

    for $e \leftarrow 0$ **to** n_i **do**

      $nuevasEstrategias$.append$(estrategia + [e])$

  $estrategias \leftarrow nuevasEstrategias$; // Actualizar todas las combinaciones

foreach *estrategia* E *en estrategias* **do**

  $(CI, esfuerzo) \leftarrow calcular\_ci\_esfuerzo(RS, E)$

  if $esfuerzo \leq R_{max}$ ***and*** $CI < mejorCI$ **then**

    $mejorCI \leftarrow CI$

    $mejorEsfuerzo \leftarrow esfuerzo$

    $mejorEstrategia \leftarrow E$

return $(mejorCI, mejorEsfuerzo, mejorEstrategia)$

3.4. Complejidad

3.4.1. Complejidad Temporal

El algoritmo de fuerza bruta tiene una complejidad temporal exponencial, debido a que genera y evalúa todas las posibles combinaciones de esfuerzos para los grupos de la red. Si hay n grupos y cada grupo tiene k_i personas, entonces el número total de combinaciones posibles es:

$$O\left(\prod_{i=1}^{n}(k_i + 1)\right)$$

En el peor de los casos, si todos los grupos tienen un tamaño máximo de k, la complejidad se puede expresar como:

$$O\left((k+1)^n\right)$$

Además, para cada combinación se evalúa el conflicto interno y el esfuerzo total, lo que implica

un recorrido lineal sobre los n grupos. Por lo tanto, la complejidad final del algoritmo es:

$$O\left(n(k+1)^n\right)$$

Esto lo convierte en un algoritmo extremadamente costoso computacionalmente, ya que el tiempo de ejecución crece exponencialmente con el número de grupos y el tamaño de cada uno. Aunque garantiza una solución óptima, este enfoque no es adecuado para instancias grandes del problema.

En términos prácticos, esta complejidad significa que incluso para valores moderados de n y k, el número total de combinaciones puede volverse inmanejable. Por ejemplo, si se tienen apenas 10 grupos de tamaño 10, el algoritmo tendría que evaluar $(10+1)^{10} = 25937424601$ combinaciones posibles.

3.4.2. Complejidad Espacial

La complejidad espacial también es considerable. El algoritmo almacena en memoria todas las combinaciones posibles de esfuerzos antes de evaluarlas. Por tanto, en el peor de los casos, el número de combinaciones almacenadas es:

$$O\left((k+1)^n\right)$$

donde n es la cantidad de grupos y k el número máximo de agentes por grupo. Esto se debe a que se construye la lista completa de estrategias (vectores de tamaño n), en la cual cada componente puede tomar un valor entre 0 y k.

Este costo espacial es resultado del uso de listas auxiliares que mantienen todas las estrategias posibles en memoria antes de su evaluación, lo que puede saturar rápidamente la memoria en problemas con muchos grupos o agentes.

3.5. Corrección

El algoritmo de fuerza bruta desarrollado para el problema ModCI garantiza la obtención de la solución óptima, ya que evalúa exhaustivamente todas las combinaciones posibles de distribución de esfuerzos entre los grupos de la red social. Para cada combinación generada, calcula el nivel de conflicto interno resultante y verifica que el esfuerzo total no supere el límite máximo permitido. Posteriormente, selecciona la estrategia que logra minimizar el conflicto interno dentro de las restricciones establecidas.

Sin embargo, aunque el algoritmo es correcto desde el punto de vista funcional, su ejecución implica una búsqueda exhaustiva de soluciones que conlleva una complejidad exponencial. A medida que aumenta el número de grupos o el tamaño de estos, el número de combinaciones crece de forma drástica, lo cual impacta negativamente en el rendimiento. Esto limita su aplicabilidad a redes de pequeño tamaño, donde aún es viable computacionalmente. Por lo tanto, si bien este enfoque asegura una solución óptima y es útil como referencia o para validar la calidad de soluciones aproximadas, no es práctico para redes sociales grandes, y por ello es necesario acudir a técnicas

más eficientes como los algoritmos voraces o la programación dinámica para abordar instancias más complejas del problema.

3.6. Análisis

A continuación se presenta un análisis concreto que permite ilustrar el funcionamiento paso a paso del algoritmo de fuerza bruta

Datos de entrada:

Grupo	Agentes	Opinión 1	Opinión 2	Rigidez
1	1	-4	2	0.1
2	4	1	-1	0.2
3	3	3	0	0.5

$R_{\max} = 5$

Cuadro 1: Datos entrada: análisis fuerza bruta

Análisis del proceso

El paso a paso para generar la solución en el algoritmo de fuerza bruta es el siguiente:

1. **Generación de combinaciones posibles**: Se generan todas las combinaciones de esfuerzos posibles para los grupos, respetando el rango desde 0 hasta el número de agentes por grupo.

   En este caso:

   - Grupo 1: 0 a 1 (2 opciones)
   - Grupo 2: 0 a 4 (5 opciones)
   - Grupo 3: 0 a 1 (2 opciones)

   Total de combinaciones: $2 \times 5 \times 2 = 20$

2. **Cálculo del conflicto interno (CI)**: Se calcula el conflicto interno para cada combinación válida, considerando cuántos agentes permanecen sin moderar. Usando la función 3. `calcular_ci_esfuerzo`.

3. **Cálculo del esfuerzo total por combinación**: Se calcula el esfuerzo total requerido usando la misma función auxiliar.

4. **Filtrado por esfuerzo máximo permitido ($R_{\max}$)**: Se descartan combinaciones cuyo esfuerzo supere el valor de $R_{\max} = 5$.

5. **Selección de la mejor estrategia**: De las combinaciones válidas, se selecciona la que genere el menor conflicto interno.

Combinaciones posibles (20 en total):

Combinación (G1 - G2 - G3)	Esfuerzo Total	Conflicto Interno	Válida
0 - 0 - 0	0	45.3333	✓
0 - 0 - 1	3	33.3333	✓
0 - 1 - 0	2	40.0000	✓
0 - 1 - 1	5	28.0000	✓
0 - 2 - 0	4	34.6667	✓
0 - 2 - 0	7	22.6667	✗
0 - 3 - 0	5	29.3333	✓
0 - 3 - 1	8	17.3333	✗
0 - 4 - 0	7	24.0000	✗
0 - 4 - 1	10	12.0000	✗
1 - 0 - 0	1	33.3333	✓
1 - 0 - 1	**4**	**21.3333**	★
1 - 1 - 0	3	28.0000	✓
1 - 1 - 1	6	16.0000	✗
1 - 2 - 0	5	22.6667	✓
1 - 2 - 1	8	10.6667	✗
1 - 3 - 0	6	17.3333	✗
1 - 3 - 1	9	5.3333	✗
1 - 4 - 0	8	12.0000	✗
1 - 4 - 1	11	0.0000	✗

Cuadro 2: Combinaciones posibles fuerza bruta

Resultados Finales del Algoritmo

Métrica	Valor
Estrategia óptima	[1, 0, 1]
Conflicto interno	21.3333
Esfuerzo total	4
Tiempo de ejecución	0.0039 segundos

Cuadro 3: Resultados obtenidos: análisis fuerza bruta.

Este ejemplo ilustra cómo el algoritmo de fuerza bruta explora exhaustivamente todas las posibles combinaciones de esfuerzos para encontrar la que minimice el conflicto interno. Aunque este enfoque garantiza encontrar la solución óptima, tiene una complejidad exponencial, ya que el número de combinaciones crece rápidamente con cada grupo o incremento de agentes.

En esta prueba con solo 3 grupos y 20 combinaciones, el tiempo de ejecución fue casi instantáneo, lo que demuestra que para problemas pequeños es muy efectivo. Sin embargo, en problemas reales con muchos grupos o agentes, este enfoque se vuelve inviable por el alto costo computacional, siendo recomendable en esos casos el uso de algoritmos voraces o programación dinámica.

3.7. Impacto en el Rendimiento

A pesar de su simplicidad, el enfoque de fuerza bruta presenta importantes limitaciones prácticas. Una de las más críticas es el impacto que tiene sobre el rendimiento a medida que crece el tamaño de la red social o el número de agentes por grupo.

Supongamos que la red social está compuesta por n grupos, y que cada grupo i tiene a_i agentes. Como las estrategias posibles para cada grupo van desde moderar 0 hasta a_i agentes, entonces el número de combinaciones de estrategias posibles es:

$$\prod_{i=1}^{n}(a_i + 1)$$

Este crecimiento es exponencial en el número de grupos, y además está influenciado por el tamaño de cada grupo. Incluso si cada grupo tuviera pocos agentes, al multiplicar las posibilidades de todos los grupos, el total de combinaciones puede escalar rápidamente.

Por ejemplo, para un conjunto de grupos con tamaños heterogéneos, como:

$$\{a_1, a_2, \ldots, a_n\}$$

...el total de estrategias a evaluar será:

$$(a_1 + 1) \cdot (a_2 + 1) \cdot \ldots \cdot (a_n + 1)$$

Cada una de estas combinaciones debe ser evaluada para calcular su conflicto interno y su esfuerzo total, lo que implica un costo computacional adicional por iteración.

Advertencia sobre rendimiento

Aunque el algoritmo garantiza encontrar la solución óptima, el tiempo de ejecución crece de forma multiplicativa con la cantidad de grupos y el tamaño de cada uno, lo que puede llevar a tiempos de ejecución inaceptables en redes sociales grandes.
En estos casos, se vuelve recomendable recurrir a algoritmos de programación dinámica, que también garantiza la solución óptima en menor tiempo o voraces que sacrifican algo de precisión a cambio de un rendimiento mucho más eficiente.

4. Algoritmo Dinámico

4.1. Descripción

El algoritmo de programación dinámica aplicado en este proyecto busca minimizar el conflicto interno dentro de una red social, bajo la restricción de un presupuesto limitado de esfuerzo (denotado como $R_{\max}$). Este conflicto surge de las diferencias de opinión entre los agentes de distintos grupos, y puede ser mitigado parcialmente moderando a algunos de estos agentes, a costa de cierto esfuerzo.

A diferencia de enfoques exhaustivos como la fuerza bruta, que evalúan todas las combinaciones posibles de moderación, este algoritmo divide el problema en subproblemas más pequeños y los resuelve de manera progresiva y óptima, reutilizando resultados previamente calculados para construir la solución global.

El proceso se fundamenta en el principio de *optimalidad de Bellman*, que asegura que una solución óptima para el problema completo puede ser construida a partir de soluciones óptimas de sus subproblemas.

A cada paso, el algoritmo evalúa todas las opciones posibles de moderación para un grupo específico —desde no moderar a ningún agente hasta moderar a todos— y calcula el conflicto y esfuerzo que implicaría esa decisión. Luego, actualiza de forma eficiente una estructura de datos que almacena, para cada cantidad de esfuerzo acumulado, la mejor configuración encontrada hasta ese punto.

Al finalizar, se selecciona la estrategia que, sin superar el esfuerzo disponible, logra el menor conflicto interno posible.

Este enfoque garantiza que la solución obtenida es óptima dentro de las restricciones definidas, y resulta mucho más escalable que evaluar explícitamente todas las combinaciones posibles, especialmente en redes sociales con un número elevado de grupos o agentes.

4.2. Pasos Programación Dinámica

Sabemos que para calcular la solución óptima para un algoritmo dinámico se deben cumplir ciertos aspectos:

1. Caracterizar la solución óptima (Subestructura óptima)
2. Definir recursivamente el valor de la solución óptima
3. Calcular el valor de la solución óptima (algoritmo)
4. Construir la solución óptima (algoritmo)

4.2.1. Caracterizando la Estructura de una Solución Óptima

Para resolver el problema de minimizar el conflicto interno en una red social bajo una restricción de esfuerzo máximo permitido ($R_{\max}$), es necesario comprender cómo se construye la solución

óptima a partir de decisiones locales en cada grupo de agentes.

- **Definición del problema original:**

  Denotamos el problema general como:

  $$\text{MinCI}(Grupos_n, R_{\max})$$

  donde:

  - $Grupos_n$: conjunto de n grupos sociales.
  - $R_{\max}$: esfuerzo total máximo permitido.
  - MinCI: equivale a la función objetivo del problema, que llamamos `ModCI`.

- **Subproblemas:**

  El enfoque de programación dinámica divide el problema en subproblemas de la forma:

  $$\text{MinCI}(Grupos_i, R_j)$$

  donde:

  - i representa los primeros i grupos, con $1 \leq i \leq n$
  - R_j representa el esfuerzo disponible en ese momento, con $0 \leq j \leq R_{\max}$

- **Recurrencia y decisiones:**

  Para cada grupo i, se evalúan todas las decisiones posibles de moderación: desde moderar a 0 hasta moderar a n_i agentes (cantidad total del grupo). A cada opción le corresponde un coste de esfuerzo e y un conflicto potencial.

  La decisión óptima para el subproblema $\text{MinCI}(Grupos_i, R_j)$ se construye a partir de:

  - Las decisiones óptimas de los subproblemas anteriores $(i-1)$.
  - Comparando todas las combinaciones posibles de moderación:

    $$e \in [0, n_i]$$

  - Solo se consideran las combinaciones que cumplen:

    $$Costo(e) \leq R_j$$

  Para cada combinación válida, se calcula:

  $$CI_{\text{acumulado}} = CI_{\text{previo}} + (n_i - e)(o_{1i} - o_{2i})^2$$

  donde:

- $CI_{\text{acumulado}}$: conflicto total acumulado hasta el grupo actual i, considerando la estrategia de esfuerzo aplicada.
- CI_{previo}: conflicto acumulado hasta el grupo $i-1$, correspondiente a una asignación de esfuerzo previa.
- n_i: número total de agentes en el grupo i.
- e: número de agentes moderados en el grupo i.
- $n_i - e$: número de agentes no moderados en el grupo i.
- $(o_{1i} - o_{2i})$: diferencia cuadrática entre las dos opiniones del grupo. Refleja la intensidad del conflicto si no se modera ese grupo.

- **Estructura óptima:**

  Así, la solución óptima para el problema completo:

$$\text{MinCI}(Grupos_n, R_{\max})$$

  se obtiene reutilizando las mejores soluciones de los subproblemas previos, considerando todas las configuraciones posibles de esfuerzo y moderación por grupo.

Esto permite:

- Mantener un historial de las mejores decisiones hasta cada punto.
- Evitar recalcular combinaciones ya exploradas.
- Garantizar una solución óptima dentro del límite de esfuerzo.

4.2.2. Definiendo Recursivamente el Valor de la Solución Óptima

Una vez caracterizada la subestructura óptima, necesitamos definir una fórmula recursiva que describa cómo se calcula el valor de la solución óptima a partir de los subproblemas.

- **Variables:**
  - i: índice del grupo actual (0-indexado)
  - e: número de agentes moderados en el grupo i $(0 \leq e \leq n_i)$
  - $c(i, e)$: esfuerzo necesario para moderar e agentes en el grupo i
  - (o_{1i}, o_{2i}): distancia cuadrática de opiniones en el grupo i
  - $S(i, j)$: valor mínimo del conflicto interno al considerar los primeros i grupos con un total de esfuerzo j
- **Recurrencia:**

$$S(i,j) = \min_{0 \leq e \leq n_i,\, c(i,e) \leq j} \{S(i-1, j-c(i,e)) + (n_i - e) \cdot d_i\}$$

- **Explicación:**
  - Para cada grupo i, evaluamos todas las posibles cantidades de agentes e que podrían moderarse.
  - El costo en esfuerzo por moderar e agentes se calcula como:

$$c(i, e) = \lceil |o_{1i} - o_{2i}| \cdot rigidez_i \cdot e \rceil$$

  - Si ese esfuerzo no supera el total disponible j, entonces actualizamos el conflicto acumulado:
    - Se agrega el impacto de los $n_i - e$ agentes no moderados, que contribuyen al conflicto interno con el factor d_i.
  - De todas las combinaciones viables, nos quedamos con la que minimice el conflicto total.

- **Condiciones triviales:**

$$S(-1, 0) = 0$$

$$S(-1, j) = \infty, \quad \text{si } j > 0$$

Esto indica que al inicio, sin haber procesado ningún grupo, solo se puede tener un conflicto acumulado 0 si no se ha usado esfuerzo. Todo lo demás es inválido.

4.2.3. Calculando el Valor de la Solución Óptima

La construcción de la solución óptima se fundamenta en el almacenamiento y la actualización de la lista de decisiones (**decisions**). Cada vez que se actualiza el estado en $dp[i + 1]$, se guarda la decisión tomada para el grupo i. Al finalizar el proceso, la función **seleccionar_mejor_estrategia** recorre los estados finales y recupera la secuencia de decisiones que produjo el menor conflicto, dando como resultado la reconstrucción de la solución óptima.

4.2.4. Construyendo la Solución Óptima

4.2.5. ¿Cuántos Subproblemas Hay?

El número de subproblemas esta ligado a:

- La cantidad de grupos (n)
- El esfuerzo total acumulado **used_effort** el cual será R_{max}
- El número total de agentes no moderados restantes **total_rem** el cual se denominara como T

Entonces una buena estimación superior para la cantidad de subproblemas será:

$$\text{subproblemas} \leq n \cdot R_{\text{alcanzado}} \cdot k_{\text{alcanzado}}$$

Donde:

- $R\_alcanzado$: cantidad de valores de $used\_effort$ que realmente aparecen.
- $k\_alcanzado$: cantidad de valores distintos de $total\_rem$ generados por esfuerzo.

Fórmula general para subproblemas calculados:

$$\#\text{subproblemas} \approx \sum_{i=0}^{n} \sum_{\text{used\_effort} \in \text{dp}[i]} |\text{dp}[i][\text{used\_effort}]|$$

Esto se traduce a:

- Recorro n etapas
- Para cada etapa, cuento cuántos esfuerzos ($used\_effort$) hay
- Y para cada uno, cuántos valores de $total\_rem$ (subproblemas reales) contiene

Resumen:

- Fórmula estimada de subproblemas:

$$O(n \cdot R_{\text{real}} \cdot k_{\text{real}})$$

- Acotado superiormente por la complejidad espacial e inferiormente por la temporal.

4.3. Implementación

Respecto a la implemenctación de **modciPD.py** es importante resaltar que esta utiliza una estrategia bottom-up dado a que la solución se genera a partir de las soluciones de los subproblemas más pequeños hasta el problema más grande, a su vez esta sigue los siguientes pasos:

1. **Inicialización de DP:**
   - Se crea la tabla *dp* como una lista de diccionarios, la cual será la encargada de guardar los pasos que lleven a la solución óptima.
2. **Actualización de estados:**

- Para cada grupo dentro de la red, se recorren las posibles decisiones (pasos) y estas son almacenadas solo si cumplen con el criterio de que al aplicar las mismas no se puede superar el esfuerzo máximo proporcionado por la red.

3. **Selección de la estrategia óptima:**
   - Una vez procesados todos los grupos, se analiza el último nivel de la tabla dp (el cual corresponde a $dp[n]$, allí se encuentran todas las estrategias posibles que llegaron hasta el final (significa que tomaron una decisión válida para todos los grupos respetando el esfuerzo máximo (R_{max})).
   - Cada una de estas estrategias esta asociada a un esfuerzo utilizado y un número total de personas que no fueron moderadas.
   - La función **seleccionar_mejor_estrategia** revisa todas las estrategias anteriormente mencionadas y para cada una de ellas recupera el valor acumulado del conflicto y las decisiones tomadas en cada grupo.
   - Luego, se calcula un conflicto promedio de cada estrategia (a partir de la división del conflicto acumulado entre el número de grupos) y se selecciona la estrategia que tenga el conflicto más bajo.
   - En esta etapa se determina cuál fue la mejor estrategia y se reconstruyen las decisiones que llevaron hasta ese punto.

Finalmente, se utiliza la función **calcular_ci_esfuerzo** para obtener el valor real del conflicto interno de la estrategia seleccionada y se retorna una tupla compuesta por: el conflicto interno total alcanzado por la estrategia, el esfuerzo total requerido para aplicarla y la lista de decisiones tomadas para moderar a los agentes.

4.4. Pseudocódigo

Algoritmo 2: modciPD: Estrategia óptima con programación dinámica

Input: Una red social RS como lista de tuplas $(n_i, op1_i, op2_i, rigidez_i)$ para cada grupo i, y un esfuerzo máximo permitido R_{max}

Output: Tupla $(CI_{\text{mín}}, Esfuerzo_{\text{total}}, Estrategia_{\text{óptima}})$

$n \leftarrow$ cantidades por grupo, $op1$, $op2$, $rigidez$ extraídos de RS

$d_i \leftarrow (op1_i - op2_i)^2$ para cada grupo i ; `// Desacuerdo cuadrático`

Inicializar $dp[0][0][0] \leftarrow (0,0,[])$; `// Suma` S`, decisiones`

for $i \leftarrow 0$ **to** $|RS| - 1$ **do**

  foreach $usedEffort$ *en* $dp[i]$ **do**

    foreach $rem, (S, decisions)$ *en* $dp[i][usedEffort]$ **do**

      for $e \leftarrow 0$ **to** n_i **do**

        $cost \leftarrow \lceil |op1_i - op2_i| \cdot rigidez_i \cdot e \rceil$

        $newEffort \leftarrow usedEffort + cost$

        if $newEffort \leq R_{max}$ **then**

          $newRem \leftarrow n_i - e$

          $newTotal \leftarrow rem + newRem$

          $newS \leftarrow S + newRem \cdot d_i$

          $newDecisions \leftarrow decisions + [e]$

          if $newEffort$ *no está en* $dp[i+1]$ **then**

            Crear $dp[i+1][newEffort] \leftarrow \{\}$

          if $newTotal$ *no está en* $dp[i+1][newEffort]$ ***or*** $newS < dp[i+1][newEffort][newTotal].S$ **then**

            $dp[i+1][newEffort][newTotal] \leftarrow (newS, newDecisions)$

$mejorCI \leftarrow \infty$, $mejorEstrategia \leftarrow$ `None`

foreach $usedEffort$ *en* $dp[num\_grupos]$ **do**

  foreach $totalRem, (S, decisions)$ *en* $dp[num\_grupos][usedEffort]$ **do**

    $conflict \leftarrow S/num\_grupos$

    if $conflict < mejorCI$ **then**

      $mejorCI \leftarrow conflict$

      $mejorEstrategia \leftarrow decisions$

if $mejorEstrategia =$ `None` **then**

  return $(\infty, 0, [])$

else

  return calcular_ci_esfuerzo(RS, $mejorEstrategia$) $+(mejorEstrategia)$

4.5. Complejidad

4.5.1. Complejidad Temporal

Las operaciones que principalmente realiza **modciPD** dentro del bucle el cual se encarga de recorrer todos los grupos n y que serán de ayuda para entender su complejidad temporal serán:

- actualizar_dp: $O(k)$ el recorrido que esta hace en su bucle interno depende del número de personas dentro de la red social (k).
- seleccionar_mejor_estrategia: $O(R_{max}+1)$ el recorrido que esta hace en su bucle interno depende de R_{max}

$$O\Big(n \times (k+1) \times (R_{max}+1)\Big)$$

4.5.2. Complejidad Espacial

- El tamaño de la matriz dp el cual es $(n+1) \times (R_{max}+1)$, siendo n la cantidad de grupos y R_{max} el esfuerzo máximo, a estos se les suma $+1$ para la construcción de los casos triviales.
- Dentro de la matriz existe un diccionario cuya función es almacenar la decisión tomada en cada subproblema, siendo de tamaño n.
- $k+1$ siendo k el total de agentes dentro de la red social y a este valor se le añade 1 para la construcción del caso trivial.

Esto implica una complejidad espacial del orden de:

$$O\Big((n+1) \times (R_{max}+1) \times n \times (k+1)\Big)$$

De manera simplificada esto se puede representar como:

$$O(n^2 \times R_{max} \times k)$$

4.5.3. ¿Es Útil en la Práctica?

El enfoque de programación dinámica es útil para reducir drásticamente el número de combinaciones evaluadas en comparación con una busqueda exhaustiva, especialmente en instancias donde el número de grupos n y el esfuerzo máximo R_{max} son moderados.

Hipótesis: La reutilización de resultados parciales mediante PD permite calcular la solución óptima sin tener que explorar todo el campo de de las posibles estrategias que se puedan aplicar.

Teniendo en cuenta lo anterior, se realiza un análisisi teórico de cómo sería el uso de capacidad y tiempo aproximado para distintas instancias del problema. Teniendo en cuenta lo siguiente:

- **K:** Mayor cantidad de agentes en un grupo dentro de la red social (sería el peor caso, donde no se modera ningún agente en un grupo).
- **n:** Cantidad de grupos de la red social. Esta en terminos de k y se representa de la manera $n = 2^k$.
- $R_{\mathbf{max}}$**:** Esfuerzo máximo permitido.
- **Tamaño de celda:** El cual esta expresado como un tamaño de 4 Bytes (por celda)
- μ**:** Rendimiento de la máquina en operaciones por segundo, el cual se establecio como 3×10^8 en minutos (ajustado a $\frac{3\times10^8}{60} = 5 \times 10^6$ para pasarlo a segundos).
- **Cual es la formula del espacio teorico?** La formula utilizada en este caso es la de la complejidad espacial multiplicado por el tamaño de celda, el cual quedaria de la siguiente manera:

$$n^2 \cdot R_{max} \cdot k \cdot 4bytes$$

- **Cual es la formula del tiempo teorico?** La formula utilizada en este caso es la de la complejidad temporal dividido entre el rendimiento de la maquina en segundos (μ) , el cual quedaria de la siguiente manera:

$$\frac{n \cdot (k+1) \cdot (R_{max}+1)}{5 \times 10^6}$$

  Los cuales nos darian como resultante la siguiente tabla:

# Prueba	k	$n = (2^k)$	R_{max} (Esfuerzo)	Espacio teórico	Tiempo teórico
1	4	16	1000	4.10 MB	0.02 s
2	4	16	10000	40.96 MB	0.16 s
3	4	16	50000	204.80 MB	0.80 s
4	9	512	1000	9.44 GB	1.02 s
5	9	512	10000	94.37 GB	10.24 s
6	9	512	50000	471.86 GB	51.20 s
7	14	16384	1000	15.03 TB	49.15 s
8	14	16384	10000	150.32 TB	8.19 min
9	14	16384	50000	751.62 TB	40.96 min
10	19	524288	1000	20.89 PB	34.95 min
11	19	524288	10000	208.91 PB	5.83 h
12	19	524288	50000	1.04 EB	1.21 d
13	24	16777216	1000	27.02 EB	23.30 h
14	24	16777216	10000	270.22 EB	9.71 d
15	24	16777216	50000	1351.08 EB	1.62 mes

Cuadro 4: Resultados ordenados por k y tamaño de entrada n.

Conclusión Hipótesis Temporal: Cuando el número de grupos y el límite R_{max} no son excesivamente altos, el algoritmo de programación dinámica consigue una solución óptima en un tiempo razonable. Sin embargo, para valores muy grandes, la cantidad de estados crece de forma considerable, y el tiempo de ejecución puede llegar a ser elevado. Aun así, representa una mejora sustancial frente a enfoques como fuerza bruta.

> Por ejemplo, con $n = 512$ y $R_{\max} = 1\,000$ el tiempo teórico estimado es $\approx 1{,}02$ s; con $n = 16\,384$ y $R_{\max} = 10\,000$ asciende a $\approx 8{,}19$ min; y para $n = 524\,288$ y $R_{\max} = 50\,000$ el tiempo estimado llega a $\approx 1{,}21$ días.

- Para valores de k entre 4 y 9, el tiempo de resolución es bueno, ya que varía de fracciones de segundo a pocos segundos, sin superar el minuto. Para k = 14 y $R_{\max} = 1000$, el tiempo de respuesta sigue siendo bueno aunque asciende casi al minuto.
- Para $k = 14$ y $R_{\max}$ mayores, el tiempo de respuesta se vuelve aceptable, siendo de varios minutos sin superar la hora. Para k=19 y $R_{\max}$ bajos, el tiempo de respuesta sigue siendo aceptable
- Para los casos de k=19 y $R_{\max}$ medianos y grandes, el tiempo de respuesta se vuelve inaceptable. Los valores de k 19 tienen tiempos de respuesta de varias horas, incluso días o meses lo cual los vuelve completamente inaceptables

Conclusión Hipótesis Espacial: El uso de memoria adicional para almacenar la tabla *dp* es el principal factor en el costo espacial del método. Sin embargo, en la práctica, la poda de estados ineficientes y la restricción basada en el esfuerzo máximo (R_{max}) disminuyen notablemente el consumo de memoria, haciéndolo viable incluso en redes de gran tamaño.

> Por ejemplo, para $n = 512$ y $R_{\max} = 10\,000$ la tabla ocupa $\approx$ 94,37 GB; con $n = 16\,384$ y $R_{\max} = 50\,000$ llega a $\approx$ 751,62 TB; y para $n = 16\,777\,216$ con $R_{\max} = 10\,000$ se requieren hasta $\approx$ 270,22 EB.

- Para $k = 4$ el espacio utilizado es bueno, sin superar unos cuantos MB
- Para $k = 9$ y $R_{\max}$ pequeños y medianos, el espacio utilizado es aceptable, sin superar las 100GB
- Para $k = 9$ y $R_{\max}$ grandes, el espacio utilizado ascende casi a la media TB, lo cual se vuelve inaceptable. Para $k > 9$ el espacio utilizado es completamente inaceptable, yendo desde descenas de TB hasta miles de EB.

Impacto de R_{max}

Dos configuraciones con el mismo k y n pueden diferir significativamente en tiempo según el valor de $R_{\text{máx}}$, lo que sugiere que este parámetro afecta directamente la cantidad de subproblemas o combinaciones evaluadas por el algoritmo.

Aunque para valores altos de Rmax se disponga de mayor libertad para moderar a más agentes, lo que intuitivamente sugiere que el problema se vuelve menos complejo, se observa que para casos en los que se tiene el mismo k pero aumenta Rmax el rendimiento empeora. Esto debido a que un valor más alto de Rmax implica evaluar aún más estados factibles para encontrar la solución optima.

Esto indica que $R_{\text{máx}}$ afecta el tamaño del espacio de soluciones factibles y por tanto modificando el costo tanto temporal como espacial. Sin embargo, el impacto de $R_{\text{máx}}$ no es completamente predecible ni proporcional: aumentar este parámetro no siempre implica un incremento lineal del tiempo. Esto se debe a que su efecto depende del comportamiento interno del algoritmo, es decir, de cuántos subproblemas adicionales son realmente considerados y almacenados como consecuencia de permitir soluciones más amplias.

4.6. ¿El Algoritmo Siempre da la Respuesta Correcta al Problema?

Sí. Al basarse en programación dinámica, este método garantiza la obtención de la solución óptima para el problema de moderar el conflicto interno en una red social, siempre que se cumplan las siguientes condiciones:

- **Propiedad de subestructura óptima:** El problema puede descomponerse en subproblemas cuya solución óptima se construye a partir de las soluciones óptimas de instancias más pequeñas. Cada decisión de moderar un grupo de agentes se basa en resultados parciales previamente calculados, de modo que ninguna combinación viable queda fuera del análisis.
- **Cobertura exhaustiva de las estrategias viables:** Se consideran, para cada grupo, todas las cantidades posibles de agentes a moderar (desde cero hasta el tamaño del grupo), respetando siempre el límite de esfuerzo disponible. Esto asegura que no se descarte ninguna estrategia factible que pudiera conducir al mínimo conflicto.
- **Comparación y selección global:** Una vez evaluadas todas las combinaciones válidas, se selecciona aquella cuyo nivel de conflicto interno final es el más bajo. Al procesar sistemáticamente todos los subproblemas y reconstruir la mejor secuencia de moderación, la solución resultante es, por definición, la óptima.

Limitaciones Prácticas

En teoría, la corrección está garantizada mientras se exploren y almacenen todas las combinaciones de subproblemas. En la práctica, a medida que crecen el número de grupos y el presupuesto de esfuerzo, la cantidad de estados intermedios puede aumentar y exigir grandes recursos de memoria o tiempo. No obstante, en instancias de tamaño moderado, estas exigencias siguen siendo manejables y la estrategia conserva su garantía de optimalidad.

4.7. Análisis

El método de programación dinámica, aplicado al problema de moderación de opiniones, presenta las siguientes características de comportamiento:

- **Crecimiento polinómico controlado:** El tiempo de cómputo crece de forma suave al aumentar el número de grupos, manteniéndose dentro de una escala práctica (desde milisegundos hasta minutos). Esto refleja que, aunque se exploran múltiples estados, la fusión de soluciones parciales evita la explosión exponencial.
- **Estabilidad frente a variaciones de presupuesto:** Para un mismo tamaño de red, la variación del límite de esfuerzo provoca oscilaciones en el tiempo (mayor esfuerzo $\rightarrow$ más estados a evaluar), pero nunca aumenta desproporcionadamente. Esto evidencia que la estructura de DP maneja eficientemente incluso presupuestos altos.
- **Escalabilidad hasta instancias moderadas–grandes:**
  - **Pequeñas ($n \leq 10$):** tiempos imperceptibles ($<$ 1 ms–500 ms).

- **Medianas ($10 < n \leq 25$):** siempre por debajo de 13 s, adecuado para uso interactivo.
- **Grandes ($25 < n \leq 50$):** de decenas a centenares de segundos, aceptable en procesos batch o análisis offline.
- **Muy grandes ($n \approx 100$):** alrededor de 5 min, puede considerarse límite práctico sin optimizaciones adicionales.

- **Equilibrio espacio-tiempo:** El algoritmo sacrifica algo de memoria para almacenar la tabla de estados, pero a cambio evita recomputaciones y ofrece un rendimiento más predecible frente a heurísticos voraces o fuerza bruta.

4.8. Impacto en El Rendimiento

A continuación, los tiempos observados en diversas pruebas con presupuestos variados, medidos en segundos:

Grupos	**Mínimo (s)**	**Máximo (s)**
5	0,0021	0,0073
10	0,0029	0,5037
15	0,0402	5,1563
20	3,3455	8,9823
25	5,4816	12,4681
50	29,3461	174,4527
100	—	296,0398

Cuadro 5: Tiempos de ejecución mínimo y máximo por número de grupos

- **Umbral interactivo:** hasta 15 grupos se mantiene por debajo de 5 s incluso en el peor caso.
- **Procesamiento batch:** entre 20 y 50 grupos, cabe en un marco de decenas de segundos a pocos minutos.
- **Límite práctico:** alrededor de 100 grupos, el tiempo se acerca a los 5 min, recomendando optimizaciones o partición de la red si se precisa mayor escala.

5. Algoritmo Voraz

5.1. Descripción

El algoritmo voraz implementado aborda el problema de minimizar el conflicto interno dentro de una red social, sujeto a una restricción presupuestal de esfuerzo (denotada como R_{max}). El conflicto surge a partir de las discrepancias entre las opiniones que los agentes expresan sobre dos afirmaciones, y puede ser reducido moderando a ciertos agentes, aunque dicha acción implica un costo en términos de esfuerzo.

A diferencia del enfoque de fuerza bruta, que explora exhaustivamente todas las combinaciones posibles de moderación, y del enfoque de programación dinámica, que construye soluciones óptimas a partir de subproblemas más pequeños, este enfoque se basa en una estrategia de decisión secuencial que busca aprovechar el esfuerzo disponible de la forma más eficiente posible. Para ello, cada grupo de agentes es evaluado según un criterio heurístico que estima la utilidad relativa de intervenir sobre él.

Este enfoque no garantiza encontrar la solución óptima al problema, ya que no explora todo el espacio de posibilidades. Sin embargo, en muchos casos ofrece soluciones de buena calidad con un costo computacional mucho menor, lo que lo convierte en una opción atractiva para redes sociales de mayor tamaño o para situaciones donde el tiempo de cómputo es limitado.

5.2. Definición de la heurística

La toma de decisiones del algoritmo voraz se fundamenta en una heurística que asigna una utilidad a cada grupo de agentes, con el propósito de priorizar aquellos cuya moderación podría ser más efectiva en términos de reducción del conflicto interno por unidad de esfuerzo.

La utilidad asignada a cada grupo se define como:

$$\text{utilidad}_i = \frac{|op1_i - op2_i|}{\text{máx}(rigidez_i, \varepsilon)}$$

donde:

- $op1_i$ y $op2_i$ son las opiniones del grupo i frente a las dos afirmaciones del problema,
- $rigidez_i$ es el nivel de resistencia al cambio de opinión del grupo i,
- ε es una constante positiva muy pequeña que evita divisiones por cero.

Esta función favorece a los grupos que presentan una alta discrepancia de opiniones y baja rigidez, es decir, aquellos en los que intervenir podría tener mayor impacto y menor costo relativo.

Una vez calculadas las utilidades para todos los grupos, se ordenan en orden descendente y se asigna el esfuerzo disponible comenzando por los grupos más rentables, hasta agotar el presupuesto permitido R_{max} o no poder realizar más asignaciones viables.

5.3. ¿Entendimos el Algoritmo?

5.3.1. Solución del Ejercicio 2.3.1

Consideramos la red social $\mathcal{RS}_1 = (\langle 3, -100, 50, 0{,}5\rangle, \langle 1, 100, 80, 0{,}1\rangle, \langle 1, -10, 0, 0{,}5\rangle, R_{max} = 80)$, compuesta por tres grupos de agentes. La información detallada de los grupos es la siguiente:

Grupo	n_i (agentes)	o_{i1}	o_{i2}	r_i (rigidez)
0	3	-100	50	0.5
1	1	100	80	0.1
2	1	-10	0	0.5

Cuadro 6: Datos de entrada de la red social $\mathcal{RS}_1$

Paso 1: Cálculo de utilidad y costo unitario

Calculamos para cada grupo su utilidad según la **??**) y el costo de moderar a un solo agente:

Grupo	Utilidad heurística	Costo unitario por agente
0	$\frac{\lvert -100-50\rvert}{0{,}5} = 300$	$\lceil 150 \cdot 0{,}5 \rceil = 75$
1	$\frac{\lvert 100-80\rvert}{0{,}1} = 200$	$\lceil 20 \cdot 0{,}1 \rceil = 2$
2	$\frac{\lvert -10-0\rvert}{0{,}5} = 20$	$\lceil 10 \cdot 0{,}5 \rceil = 5$

Cuadro 7: Utilidades y costos unitarios de la red social $\mathcal{RS}_1$

Paso 2: Orden de prioridad por utilidad

Ordenamos los grupos en orden descendente según su utilidad:

1. Grupo 0: utilidad 300
2. Grupo 1: utilidad 200
3. Grupo 2: utilidad 20

Paso 3: Asignación de esfuerzo

Inicializamos $R_{\max} = 80$.

- **Grupo 0**:
  - Costo unitario: 75
  - Máximo posible: $\lfloor \frac{80}{75} \rfloor = 1$ agente
  - Se asigna: 1 agente
  - Nuevo $R_{\max} = 80 - 75 = 5$

- **Grupo 1**:
  - Costo unitario: 2
  - Máximo posible: $\left\lfloor \frac{5}{2} \right\rfloor = 2$, pero el grupo tiene solo 1 agente
  - Se asigna: 1 agente
  - Nuevo $R_{\max} = 5 - 2 = 3$
- **Grupo 2**:
  - Costo unitario: 5
  - Máximo posible: $\left\lfloor \frac{3}{5} \right\rfloor = 0$
  - No se asignan agentes

Resultado parcial de la estrategia

- **Estrategia aplicada (E_V):** $[1, 1, 0]$
- **Esfuerzo total utilizado:** $75 + 2 = 77$

Cálculo del conflicto interno

La siguiente tabla muestra, grupo por grupo, cómo se calcula el conflicto interno de la red tras aplicar la estrategia $E_V = [1, 1, 0]$:

Grupo	n_i	e_i	$n_i - e_i$ (no moderados)	$(o_{i1} - o_{i2})^2$	Aporte al CI
0	3	1	2	$(-100 - 50)^2 = 22500$	$2 \cdot 22500 = 45000$
1	1	1	0	$(100 - 80)^2 = 400$	$0 \cdot 400 = 0$
2	1	0	1	$(-10 - 0)^2 = 100$	$1 \cdot 100 = 100$
				Suma total	45100

Cuadro 8: Cálculo del conflicto interno con la estrategia E_V para $\mathcal{RS}_1$

Conflicto interno resultante: $\frac{45100}{3} \approx 15033{,}33$

Resultado final

- **Estrategia aplicada:** $[1, 1, 0]$
- **Esfuerzo total utilizado:** 77
- **Conflicto interno resultante:** $\approx 15033{,}33$

Análisis final y respuestas a las preguntas del ejercicio

Para la red social $\mathcal{RS}_1$ se compararon tres estrategias:

- **Estrategia** E_1: $[0, 1, 1]$, con esfuerzo total 7 y conflicto interno 22500
- **Estrategia** E_2: $[1, 0, 0]$, con esfuerzo total 75 y conflicto interno 15166,66
- **Estrategia** E_V: $[1, 1, 0]$, con esfuerzo total 77 y conflicto interno $\approx$ 15033,33

A partir de estos resultados, respondemos las preguntas planteadas:

- **¿Habrá mejores soluciones que E_2?**
  Sí. La estrategia E_V encontrada por el algoritmo voraz logra un menor nivel de conflicto interno ($\approx$ 15033,33) que E_2 (15166,66), y sigue siendo aplicable dentro del esfuerzo permitido ($77 \leq 80$). Además, esta estrategia fue verificada con el enfoque de fuerza bruta, confirmando que es la solución óptima para esta instancia.
- **¿Habrá estrategias de moderación no aplicables?**
  Sí. Por ejemplo, la estrategia $[2, 1, 1]$ (es decir, moderar a 2 agentes del grupo 0, 1 del grupo 1 y 1 del grupo 2) resulta en un esfuerzo total de:

$$\lceil 150 \cdot 0{,}5 \cdot 2 \rceil + \lceil 20 \cdot 0{,}1 \cdot 1 \rceil + \lceil 10 \cdot 0{,}5 \cdot 1 \rceil = 150 + 2 + 5 = 157$$

  lo cual excede el límite $R_{max} = 80$. Por tanto, no es aplicable.

5.3.2. Solución del Ejercicio 2.3.2

Consideramos la red social $RS2 = (\langle 3, -100, 100, 0{,}8 \rangle, \langle 2, 100, 80, 0{,}5 \rangle, \langle 4, -10, 10, 0{,}5 \rangle, R_{max} = 400)$, compuesta por tres grupos de agentes. La información detallada de los grupos es la siguiente:

Grupo	n_i (agentes)	o_{i1}	op_{i2}	r_i (rigidez)
0	3	-100	100	0.8
1	2	100	80	0.5
2	4	-10	10	0.5

Cuadro 9: Datos de entrada de la red social $\mathcal{RS}_2$

Paso 1: Cálculo de utilidad y costo unitario

Calculamos para cada grupo su utilidad según la Definición de la heurística) y el costo de moderar a un solo agente:

Grupo	Utilidad heurística	Costo unitario por agente
0	$\frac{\lvert -100-100\rvert}{0{,}8} = 250$	$\lceil 200 \cdot 0{,}8 \rceil = 160$
1	$\frac{\lvert 100-80\rvert}{0{,}5} = 40$	$\lceil 20 \cdot 0{,}5 \rceil = 10$
2	$\frac{\lvert -10-10\rvert}{0{,}5} = 40$	$\lceil 20 \cdot 0{,}5 \rceil = 10$

Cuadro 10: Utilidades y costos unitarios de la red social $\mathcal{RS}_2$

Paso 2: Orden de prioridad por utilidad

Ordenamos los grupos en orden descendente según su utilidad:

1. Grupo 0: utilidad 250
2. Grupo 1: utilidad 40
3. Grupo 2: utilidad 40

Paso 3: Asignación de esfuerzo

Inicializamos $R_{\max} = 400$.

- **Grupo 0**:
  - Costo unitario: 160
  - Máximo posible: $\lfloor \frac{400}{160} \rfloor = 2$, pero el grupo tiene solo 3 agentes
  - Se asigna: 2 agentes
  - Nuevo $R_{\max} = 400 - 2 \cdot 160 = 80$
- **Grupo 1**:
  - Costo unitario: 10
  - Máximo posible: $\lfloor \frac{80}{10} \rfloor = 8$, pero el grupo tiene solo 2 agentes
  - Se asigna: 2 agentes
  - Nuevo $R_{\max} = 80 - 2 \cdot 10 = 60$
- **Grupo 2**:
  - Costo unitario: 10
  - Máximo posible: $\lfloor \frac{60}{10} \rfloor = 6$, pero el grupo tiene solo 4 agentes
  - Se asigna: 4 agentes
  - Nuevo $R_{\max} = 60 - 4 \cdot 10 = 20$

Resultado parcial de la estrategia

- **Estrategia aplicada (E_V):** $[2, 2, 4]$
- **Esfuerzo total utilizado:** $2 \cdot 160 + 2 \cdot 10 + 4 \cdot 10 = 320 + 20 + 40 = 380$

Cálculo del conflicto interno

La siguiente tabla muestra, grupo por grupo, cómo se calcula el conflicto interno de la red tras aplicar la estrategia $E = [2, 2, 4]$:

Grupo	n_i	e_i	$n_i - c_i$ (no moderados)	$(o1_i - o2_i)^2$	Aporte al CI
0	3	2	1	$(-100 - 100)^2 = 40000$	$1 \cdot 40000 = 40000$
1	2	2	0	$(100 - 80)^2 = 400$	0
2	4	4	0	$(-10 - 10)^2 = 400$	0
				Suma total	40000

Cuadro 11: Cálculo del conflicto interno con la estrategia E_V para $\mathcal{RS}_2$

Conflicto interno resultante: $\frac{40000}{3} \approx 13333{,}33$

Resultado final

- **Estrategia aplicada:** $[2, 2, 4]$
- **Esfuerzo total utilizado:** 380
- **Conflicto interno resultante:** $\approx 13333{,}33$

Análisis final y respuestas a las preguntas del ejercicio

En este ejercicio se compararon tres estrategias de moderación aplicables a la red social $RS2$:

- **Estrategia** E_1: $[0, 1, 2]$ con esfuerzo total 30 y conflicto interno 40400
- **Estrategia** E_2: $[2, 2, 2]$ con esfuerzo total 360 y conflicto interno 13600
- **Estrategia** E_V: $[2, 2, 4]$ con esfuerzo total 380 y conflicto interno $\approx 13333{,}33$

A partir de esta comparación, respondemos a las preguntas planteadas:

- **¿Habrá mejores soluciones que E_2?**
  Sí. La estrategia encontrada mediante el algoritmo voraz E_V logra un conflicto interno menor ($\approx 13333{,}33$ frente a 13600) y cumple con el límite de esfuerzo ($380 \leq 400$). Además, se verificó que esta estrategia también fue identificada por el algoritmo de fuerza bruta como la solución óptima, lo que confirma su calidad.

- **¿Habrá estrategias de moderación no aplicables?**
  Sí. Existen muchas estrategias que no son viables porque requieren más esfuerzo del disponible. Por ejemplo, la estrategia $[3, 2, 4]$, que intenta moderar completamente a todos los grupos:

$$\lceil 200 \cdot 0{,}8 \cdot 3 \rceil + \lceil 20 \cdot 0{,}5 \cdot 2 \rceil + \lceil 20 \cdot 0{,}5 \cdot 4 \rceil = 480 + 20 + 40 = 540$$

  lo cual supera el presupuesto de $R_{max} = 400$, y por tanto no es aplicable.

5.4. Implementación

La implementación del algoritmo voraz se realizó empleando una representación estructurada de la red social como una lista de tuplas. Cada tupla representa un grupo de agentes e incluye cuatro elementos: la cantidad de individuos en el grupo, sus opiniones frente a dos afirmaciones específicas y su nivel de rigidez al cambio de opinión.

El proceso se estructura en dos fases principales:

1. **Cálculo de Utilidades y Priorización:** Para cada grupo, se calcula una utilidad basada en una heurística que considera la diferencia entre sus opiniones y su rigidez al cambio. Esta utilidad refleja cuán conveniente sería moderar a un grupo determinado en términos de impacto por unidad de esfuerzo. Luego, todos los grupos son ordenados en orden descendente según esta utilidad.

2. **Asignación Voraz de Esfuerzo:** Siguiendo el orden de prioridad determinado, se asigna a cada grupo la cantidad máxima posible de agentes a moderar, respetando el esfuerzo restante disponible. Esta asignación se realiza considerando el costo unitario de moderar un agente en cada grupo, el cual depende de su rigidez y diferencia de opiniones. El proceso continúa hasta agotar el presupuesto de esfuerzo R_{max} o no poder realizar más asignaciones viables.

Al finalizar, la función auxiliar `calcular_ci_esfuerzo` se utiliza para evaluar el conflicto interno y el esfuerzo total correspondiente a la estrategia construida. El resultado final es una tupla que contiene el conflicto interno resultante, el esfuerzo total utilizado y la estrategia voraz de asignación de esfuerzos.

5.5. Pseudocódigo

Algoritmo 3: modciPV: Estrategia voraz basada en utilidad heurística

Input: Una red social RS como lista de tuplas $(n_i, op1_i, op2_i, rigidez_i)$ para cada grupo i, y un esfuerzo máximo permitido R_{max}

Output: Tupla $(CI, Esfuerzo_{\text{total}}, Estrategia)$

$estrategia \leftarrow [0, 0, \ldots, 0]$; `// Inicializar estrategia vacía de tamaño` $|RS|$

$esfuerzoRestante \leftarrow R_{max}$

$utilidades \leftarrow []$; `// Lista de utilidades heurísticas`

foreach *grupo* g_i *en* RS **do**

  if $n_i = 0$ **then**

    continue

  $utilidad \leftarrow utilidad\_base(op1_i, op2_i, rigidez_i)$

  $costoUnitario \leftarrow \max(1, \lceil |op1_i - op2_i| \cdot rigidez_i \rceil)$

  $utilidades$.append($(utilidad, i, costoUnitario)$)

$utilidades \leftarrow$ ordenar_descendente(utilidades, clave=utilidad)

foreach $(utilidad, i, costoUnitario)$ *en* $utilidades$ **do**

  if $esfuerzoRestante \leq 0$ **then**

    break

  $maxAsignable \leftarrow \min(n_i, \lfloor \frac{esfuerzoRestante}{costoUnitario} \rfloor)$

  $estrategia[i] \leftarrow maxAsignable$

  $esfuerzoRestante \leftarrow esfuerzoRestante - maxAsignable \cdot costoUnitario$

$(CI, esfuerzoTotal) \leftarrow calcular\_ci\_esfuerzo(RS, estrategia)$

return $(CI, esfuerzoTotal, estrategia)$

5.6. Complejidad

5.6.1. Complejidad temporal

El algoritmo voraz `modciPV` realiza las siguientes operaciones principales:

- Recorre todos los grupos de la red social (de tamaño n) para calcular una utilidad heurística y el esfuerzo unitario por agente. Esto toma $O(n)$.
- Ordena la lista de grupos por utilidad en orden descendente, usando la función `sort` de python la cual implementa el algoritmo `Timsort` (Python Software Foundation, 2025) que tiene una complejidad de $O(n \log n)$ en el peor de los casos Auger et al., 2018.
- Itera sobre la lista ordenada para asignar agentes a moderar mientras haya esfuerzo disponible. En el peor caso, esta operación es lineal: $O(n)$.

- Finalmente, calcula el conflicto interno total y el esfuerzo utilizado, operación que, puede tomar $O(n)$.

Por lo tanto, la complejidad temporal total está dominada por el ordenamiento de la lista de grupos, lo cual nos da una complejidad de:

$$O(n \log n)$$

5.6.2. Complejidad espacial

Respecto al uso de memoria, el algoritmo:

- Almacena información de utilidad por grupo en una lista auxiliar de tamaño n.
- Genera una lista de estrategias de tamaño n (uno por grupo).

No se utilizan estructuras de datos anidadas ni matrices adicionales, por lo que el uso de espacio crece linealmente con el número de grupos.

La complejidad espacial del algoritmo es entonces:

$$O(n)$$

5.7. Corrección

El algoritmo voraz implementado para el problema ModCI proporciona una estrategia de moderación factible dentro del límite de esfuerzo permitido. Para ello, calcula una utilidad heurística para cada grupo de la red social y, en función de este valor, prioriza la asignación de recursos a aquellos grupos que se consideran más rentables.[en] términos de impacto por unidad de esfuerzo.

En cada paso, el algoritmo garantiza que:

- El esfuerzo total asignado no excede el presupuesto disponible R_{max}.
- Las asignaciones de agentes a moderar respetan los límites de cada grupo.
- La estrategia final generada es válida y puede ser evaluada con la función de conflicto interno.

Desde el punto de vista funcional, el algoritmo es correcto, ya que produce soluciones viables que cumplen con las restricciones del problema. No obstante, al tratarse de una heurística, no explora todo el espacio de soluciones posibles, por lo que no garantiza la obtención de la solución óptima global.

A pesar de esta limitación, el enfoque voraz resulta especialmente útil en contextos donde la eficiencia computacional es prioritaria. Gracias a su bajo costo computacional y a su capacidad para generar soluciones razonables en tiempos reducidos, se convierte en una alternativa práctica y escalable frente a enfoques más costosos como la fuerza bruta.

Para evaluar la calidad de sus soluciones, se ejecutó la batería de pruebas comparando los resultados del algoritmo voraz con las soluciones óptimas obtenidas por fuerza bruta o programación dinámica. En cada caso, se calculó el valor relativo de la diferencia mediante la fórmula:

$$\frac{CI_{\text{voraz}}}{CI_{\text{óptima}}} - 1$$

A partir de estas pruebas, se obtuvo un promedio de aproximadamente 0,0162, lo cual indica que, en promedio, la solución voraz fue apenas un 1,62 % peor que la óptima. Este bajo margen de desviación sugiere que el algoritmo voraz, aunque no exacto, puede ser altamente competitivo para resolver instancias del problema de forma eficiente.

Teniendo en cuenta las complejidades de los tres enfoques se tiene que el enfoque voraz ofrece una excelente relación entre eficiencia y calidad de solución, especialmente cuando se busca un equilibrio entre rendimiento y precisión en redes sociales de gran tamaño.

5.7.1. ¿Cuándo da la Respuesta Correcta?

Existen ciertos escenarios en los que la estrategia voraz garantiza la obtención de la solución óptima. Estos casos son estructuralmente simples y no requieren decisiones complejas de distribución del esfuerzo. A continuación, se describen los más relevantes:

- **Red social con un único grupo.** Si la red contiene un solo grupo de agentes, cualquier cantidad de esfuerzo se destinará únicamente a él. Como no hay necesidad de priorizar entre grupos, la estrategia voraz y la óptima coinciden necesariamente.
- **El esfuerzo disponible permite moderar completamente a todos los grupos.** Cuando el presupuesto de esfuerzo es suficiente para aplicar el máximo nivel de moderación posible en todos los grupos, no es necesario priorizar. En este caso, todos los algoritmos —incluido el voraz— aplicarán la misma estrategia completa y óptima.
- **Escenario favorable para la elección voraz.** En ciertos escenarios, el presupuesto disponible coincide de forma precisa con el esfuerzo necesario para moderar completamente a los grupos más eficientes, es decir, aquellos con mayor utilidad y menor costo por agente. En estos casos, el enfoque voraz —que asigna esfuerzo de forma secuencial comenzando por los grupos más rentables— toma las mismas decisiones que una estrategia óptima. Por ejemplo, si moderar tres grupos de alta utilidad consume exactamente el esfuerzo total, y los grupos restantes son menos eficientes o demasiado costosos, no hay necesidad de considerar combinaciones parciales. La estrategia voraz, al seguir su ordenamiento local, construye la solución correcta sin evaluaciones adicionales.

  Ejemplo: Este comportamiento se observa en la red de la Prueba 5 de la batería de pruebas: 5 grupos con esfuerzo disponible 460. El algoritmo voraz aplica 456 unidades de esfuerzo en

los grupos 2, 4 y 1, que presentan la mejor utilidad por agente, logrando la misma solución que un algoritmo óptimo. No es necesario distribuir el esfuerzo parcialmente entre grupos, ni dejar fuera algún grupo más eficiente.

Sin embargo, este caso no es representativo de una generalidad. El hecho de que el esfuerzo disponible encaje exactamente con la cantidad necesaria para cubrir completamente los grupos más eficientes es una coincidencia estructural que no se presenta con frecuencia y no es facilmente replicable. En la mayoría de instancias reales, es común que el esfuerzo óptimo deba distribuirse entre varios grupos de forma parcial, situación en la que la estrategia voraz no suele ser óptima, aunque se acerca bastante.

5.7.2. ¿Cuándo no da la Respuesta Correcta?

Aunque el algoritmo voraz ofrece una solución válida y eficiente al problema ModCI, su naturaleza heurística implica que no garantiza la obtención del conflicto interno mínimo en todos los casos. A continuación, se detallan algunos escenarios típicos en los que esta estrategia puede generar soluciones subóptimas:

- **Distribución óptima del esfuerzo requiere intervenir en varios grupos de forma parcial.** El enfoque voraz concentra el esfuerzo en el grupo que ofrece la mayor utilidad individual, incluso si esto implica intervenir completamente en un solo grupo. Sin embargo, en muchos casos, distribuir parcialmente ese mismo esfuerzo entre varios grupos puede reducir más el conflicto general. Por ejemplo, usar todo el presupuesto en un grupo con solo dos agentes con mayor utilidad deja sin cubrir a grupos más numerosos donde podrían haberse moderado más individuos con el mismo costo total.

5.8. Análisis

El algoritmo voraz implementado para el problema ModCI constituye una solución computacionalmente eficiente que, en la mayoría de casos, entrega resultados cercanos al óptimo. Su lógica de priorización basada en utilidad permite obtener estrategias factibles de forma rápida, lo cual es especialmente valioso en redes sociales de gran tamaño o cuando el tiempo de cómputo es una restricción.

A pesar de su eficiencia, el algoritmo presenta limitaciones estructurales debido a su enfoque local. Al no evaluar el impacto global de sus decisiones, puede generar soluciones subóptimas en escenarios donde el esfuerzo debe ser distribuido cuidadosamente entre múltiples grupos.

No obstante, la calidad de sus soluciones fue evaluada sistemáticamente mediante una métrica relativa de comparación con soluciones óptimas de la batería de pruebas. Los resultados mostraron que, en promedio, la solución generada por el algoritmo voraz se alejó apenas un 1,62 % de la óptima. Esta baja desviación sugiere que, a pesar de no explorar todo el espacio de soluciones, el enfoque voraz logra capturar buena parte de la estructura del problema y construir respuestas competitivas.

En resumen, el algoritmo ofrece un equilibrio notable entre velocidad y calidad de solución. Si bien no garantiza optimalidad, su comportamiento consistente y su bajo margen de error lo hacen una herramienta efectiva y escalable para abordar el problema ModCI en contextos prácticos.

Advertencia sobre la correctitud

La principal debilidad del enfoque voraz radica en que realiza asignaciones de forma secuencial y local, sin considerar la interacción global entre grupos y el impacto acumulado que puede tener una distribución alternativa del esfuerzo. Por ello, si bien en muchos casos ofrece soluciones cercanas al óptimo, existen configuraciones donde se aleja de la mejor estrategia posible.

6. Comparación de Algoritmos

Haremos una comparativa de los tres algoritmos con las pruebas que se nos han proporcionado. La idea es entender mejor cómo se comporta cada uno, qué tan eficiente es y qué tipo de resultado podemos esperar en distintos escenarios.

6.1. Solución Óptima

Prueba	N. Grupos de Agentes	Valor Solución (CI)	Fuerza Bruta	Dinamica	Voraz	Diferencia de las soluciones
Prueba1.txt	5	0	0.0	0.000	0.000	0
Prueba2.txt	5	19616.4	19616.4	19616.400	19667.600	0.002610061
Prueba3.txt	5	1621.6	1621.6	1621.600	1720.200	0.060804144
Prueba4.txt	5	5103.8	5103.8	5103.800	5219.000	0.022571417
Prueba5.txt	5	14369	14369.0	14369.000	14369.000	0
Prueba6.txt	10	31002.7	31002.7	31002.700	32040.200	0.033464827
Prueba7.txt	10	2762.9	2762.9	2762.900	2844.800	0.029642767
Prueba8.txt	10	21948.9	X	21948.900	22165.900	0.0098866
Prueba9.txt	10	75544.6	X	75544.600	75934.600	0.005162513
Prueba10.txt	10	0	X	0.000	0.000	0
Prueba11.txt	15	19391.933	X	19391.933	19639.933	0.012788823
Prueba12.txt	15	16558.266	X	16558.266	16689.733	0.00793966
Prueba13.txt	15	43861	X	43861.000	43931.533	0.001608103
Prueba14.txt	15	22694.266	X	22694.266	23016.466	0.014197419
Prueba15.txt	15	68944.399	X	68944.399	69209.000	0.00383789
Prueba16.txt	20	0	X	0.000	0.000	0
Prueba17.txt	20	771.15	X	771.150	916.950	0.189068275
Prueba18.txt	25	28058.12	X	28058.120	28093.720	0.001268795
Prueba19.txt	25	0	X	0.000	0.000	0
Prueba20.txt	25	12431.4	X	12431.400	12683.600	0.020287337
Prueba21.txt	50	0	X	0.000	0.000	0
Prueba22.txt	50	0	X	0.000	0.000	0
Prueba23.txt	50	0	X	0.000	0.000	0
Prueba24.txt	50	0	X	0.000	0.000	0
Prueba25.txt	50	0	X	0.000	0.000	0
Prueba26.txt	50	28988.24	X	28988.240	29189.540	0.006944195
Prueba27.txt	50	25102.36	X	25102.360	25306.720	0.008141067
Prueba28.txt	100	930.02	X	930.020	982.060	0.055955786
Prueba29.txt	100	0	X	X	0.000	0
Prueba30.txt	100	-	X	X	0.000	0

Cuadro 12: Comparación de los resultados obtenidos por los distintos algoritmos para los diferentes archivos de prueba

Al evaluar la precisión de los algoritmos, se compararon sus resultados con el valor de la solución (CI) obtenido para cada prueba y la comparación con los resultados de cada algoritmo permite medir su exactitud.

- **Fuerza bruta:**

- Garantiza la solución óptima siempre que logra completarse. En todas las pruebas donde arrojó un resultado, este coincidió exactamente con el valor de la solución óptima esperada.
- Sin embargo, falló en 23 de las 30 pruebas, marcadas con "X", lo que indica que no logró encontrar una solución dentro del tiempo razonable (especialmente a partir de 10 o más grupos de agentes).

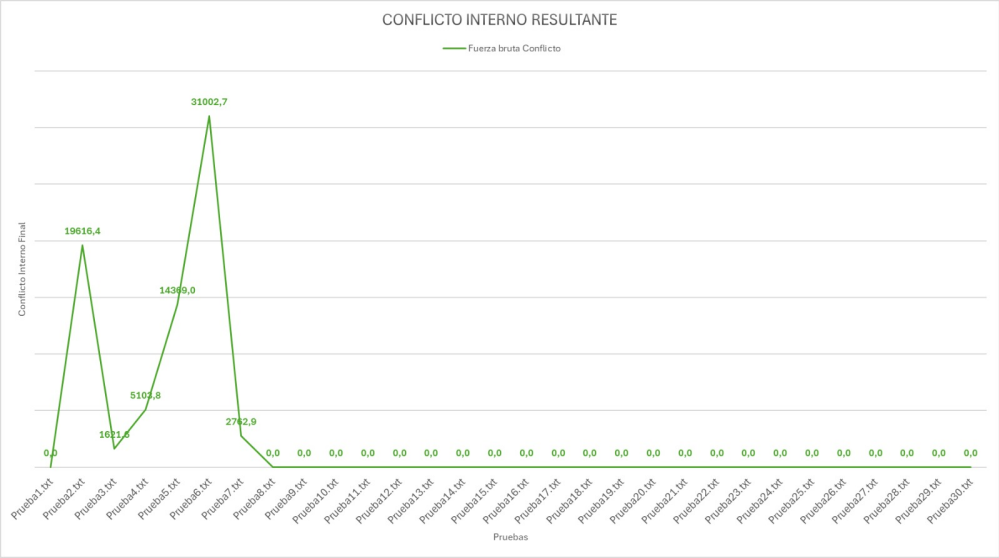

Figura 1: Gráfica del conflicto interno generado por fuerza bruta.

- **Programación dinámica:**
  - Siempre encontró la solución óptima, incluso en instancias donde la fuerza bruta no completó.
  - Solo falló en 2 pruebas, no retornando respuesta en Prueba29.txt, Prueba30.txt.

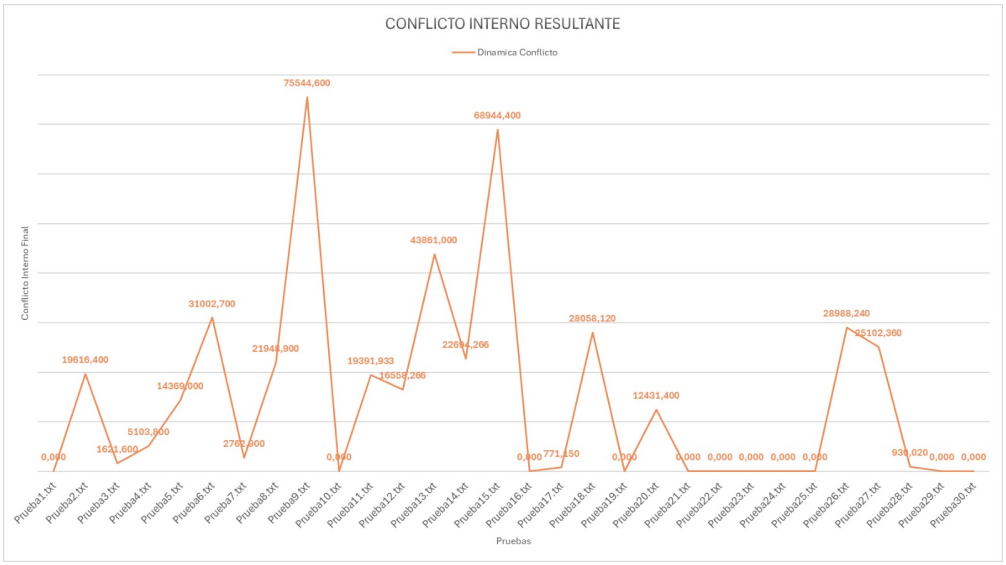

Figura 2: Gráfica del conflicto interno generado por programación dinámica.

- **Algoritmo voraz:**
  - Fue el más rápido en todas las pruebas, con tiempos del orden de milisegundos.
  - No garantiza solución óptima: en todas las pruebas arrojó una solución, pero esta no siempre coincidió con el CI mínimo real, sin embargo, en promedio difiere de la respuesta óptima que arroja la programación dinámica en un 1,62 %.

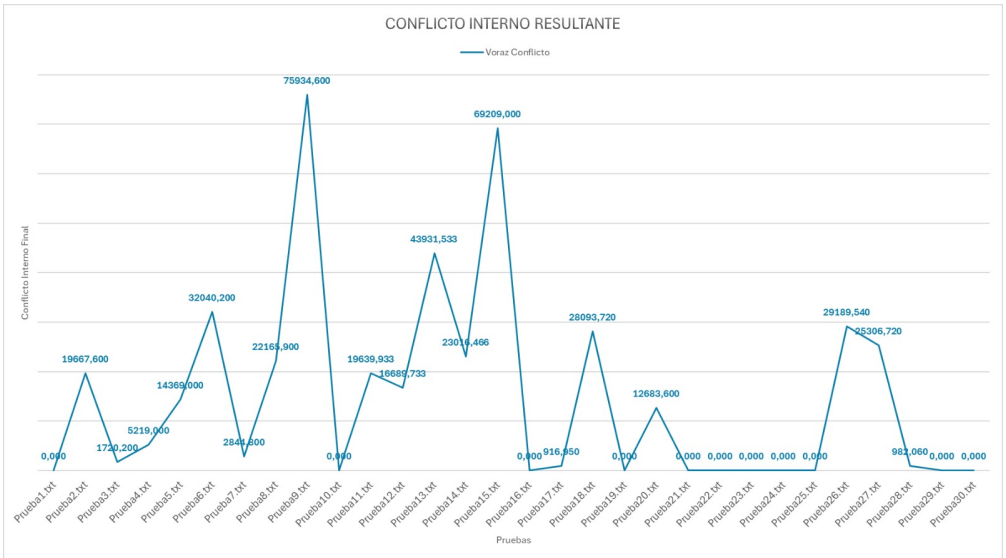

Figura 3: Gráfica del conflicto interno generado por algoritmo voraz.

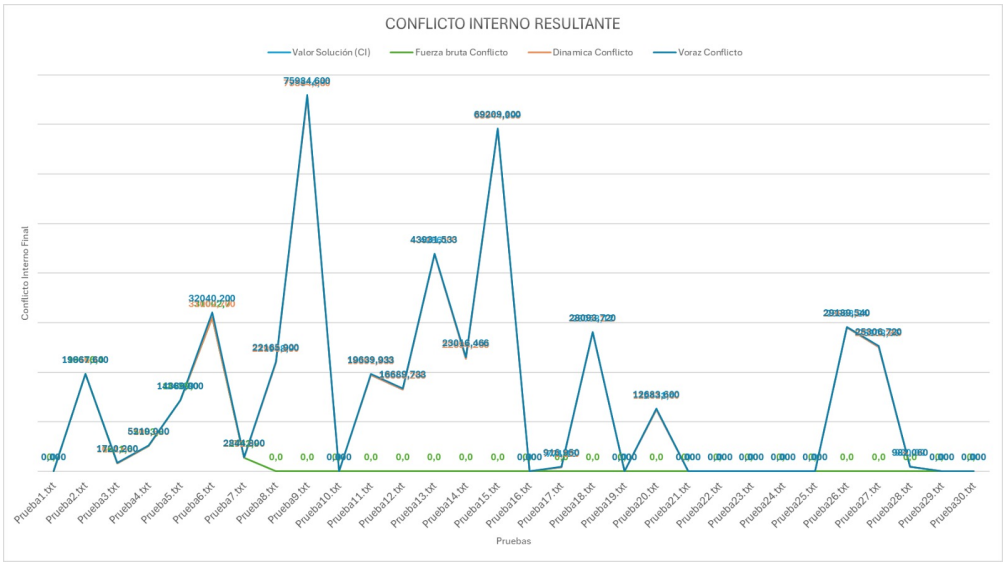

Figura 4: Gráfica del conflicto interno comparando los tres metodos.

En resumen:

- Fuerza bruta y dinámica aseguran la precisión máxima, pero no siempre pueden ejecutarse eficientemente.
- Voraz, aunque no garantiza optimalidad, ofrece soluciones rápidas y suficientemente cercanas a la óptima, incluso en escenarios con muchos agentes.

6.2. Tiempos de Ejecución

Prueba	N. Grupos de Agentes	Fuerza Bruta Tiempo seg	Dinamica Tiempo seg	Voraz Tiempo seg
Prueba1.txt	5	0,0051	0,0037	0,0000
Prueba2.txt	5	0,0358	0,0065	0,0001
Prueba3.txt	5	0,0117	0,0073	0,0000
Prueba4.txt	5	0,0065	0,0021	0,0000
Prueba5.txt	5	0,0421	0,0041	0,0000
Prueba6.txt	10	115,9665	0,0029	0,0000
Prueba7.txt	10	35,1900	0,0559	0,0000
Prueba8.txt	10	X	0,5037	0,0001
Prueba9.txt	10	X	0,4903	0,0000
Prueba10.txt	10	X	0,9360	0,0000
Prueba11.txt	15	X	0,9360	0,0000
Prueba12.txt	15	X	0,0402	0,0000
Prueba13.txt	15	X	0,5627	0,0000
Prueba14.txt	15	X	5,1563	0,0000
Prueba15.txt	15	X	0,7574	0,0000
Prueba16.txt	20	X	3,3455	0,0000
Prueba17.txt	20	X	8,9823	0,0000
Prueba18.txt	25	X	5,4816	0,0000
Prueba19.txt	25	X	10,1032	0,0001
Prueba20.txt	25	X	12,4681	0,0000
Prueba21.txt	50	X	29,3461	0,0001
Prueba22.txt	50	X	25,0628	0,0001
Prueba23.txt	50	X	160,8041	0,0001
Prueba24.txt	50	X	174,4527	0,0001
Prueba25.txt	50	X	131,9559	0,0001
Prueba26.txt	50	X	71,8948	0,0001
Prueba27.txt	100	X	553,2430	0,0002
Prueba28.txt	100	X	296,0398	0,0002
Prueba29.txt	100	X	X	0,0002
Prueba30.txt	100	X	X	0,0002

Cuadro 13: Tiempos de ejecución para los distintos algoritmos en cada archivo de prueba

En cuanto a los tiempos, es claro en la tabla que los tiempos del algoritmo voraz son mucho menores que de los demás algoritmos. A Continuación una representación gráfica de los tiempos de los tres algoritmos:

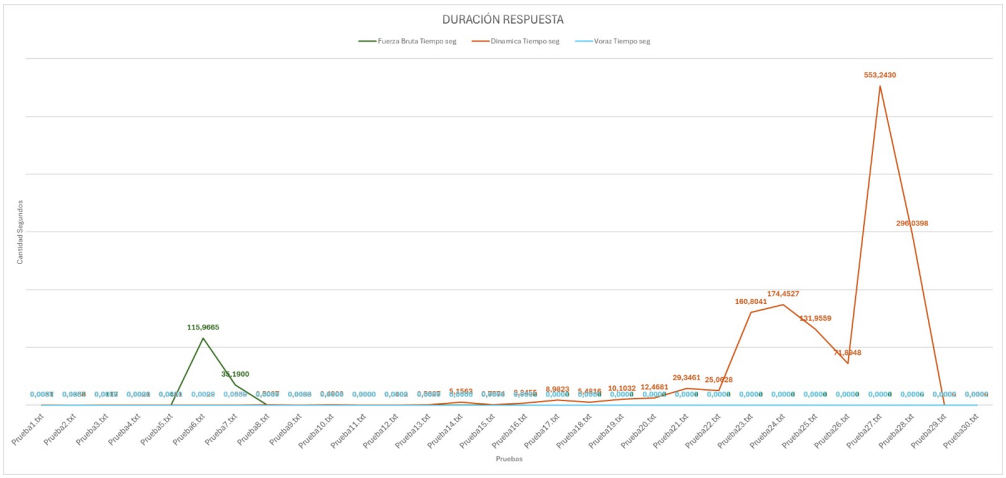

Figura 5: Gráfica comparativa del tiempo de respuesta obtenido por cada algoritmo.

- **Fuerza bruta:**
  - Su tiempo de ejecución crece exponencialmente con el número de agentes.
  - Aunque fue rápido para pequeños casos (5 o 10 grupos), superó los 100 segundos en algunas pruebas (Prueba6.txt) y no logró 73

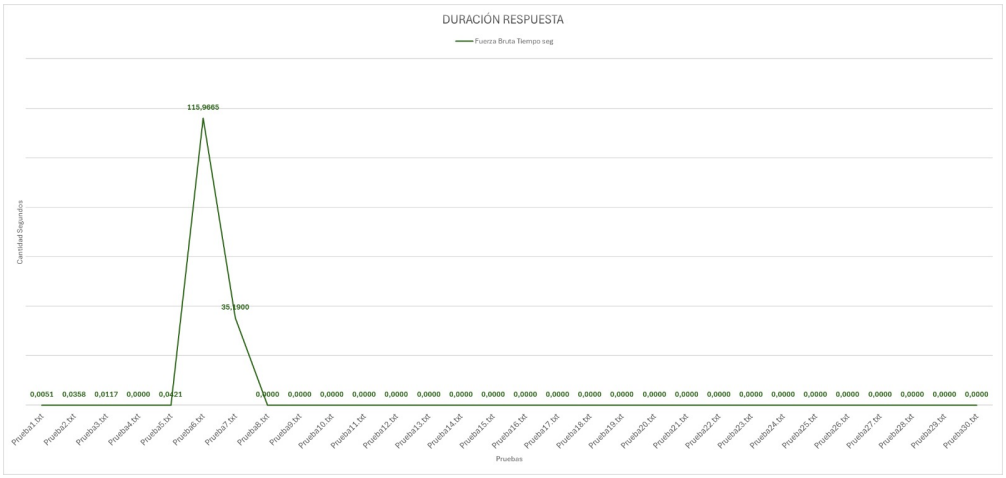

Figura 6: Gráfica del tiempo de respuesta obtenido por fuerza bruta.

- **Programación dinámica:**
  - Presenta un crecimiento más moderado, pero aún significativo en problemas grandes.
  - Tiempos mayores a 1 minuto se observaron en los archivos con más de 50 grupos.
  - Solo falló en 2 pruebas (Prueba29.txt, Prueba30.txt), mostrando una alta escalabilidad en comparación con fuerza bruta.

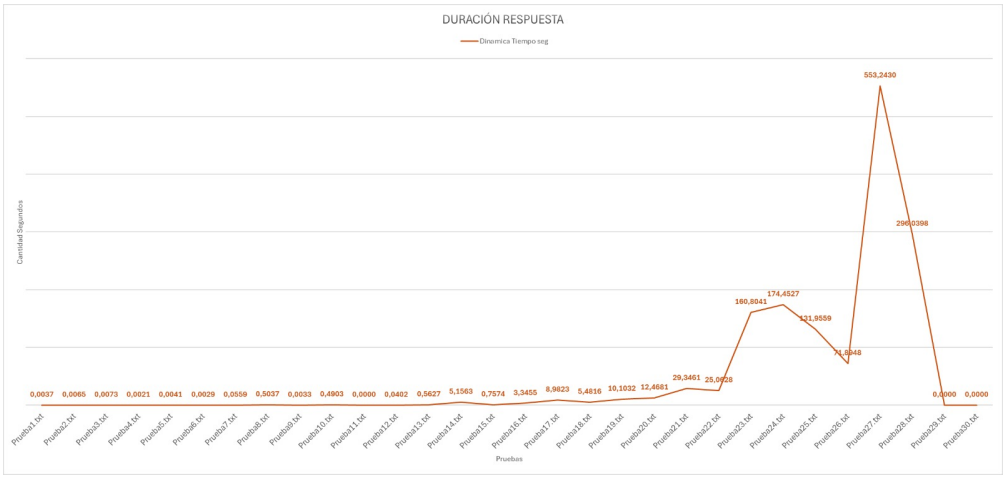

Figura 7: Gráfica del tiempo de respuesta obtenido por programación dinámica.

- **Algoritmo voraz:**
  - Extremadamente eficiente en tiempo. Todos los tiempos son inferiores a 0.0003 segundos.

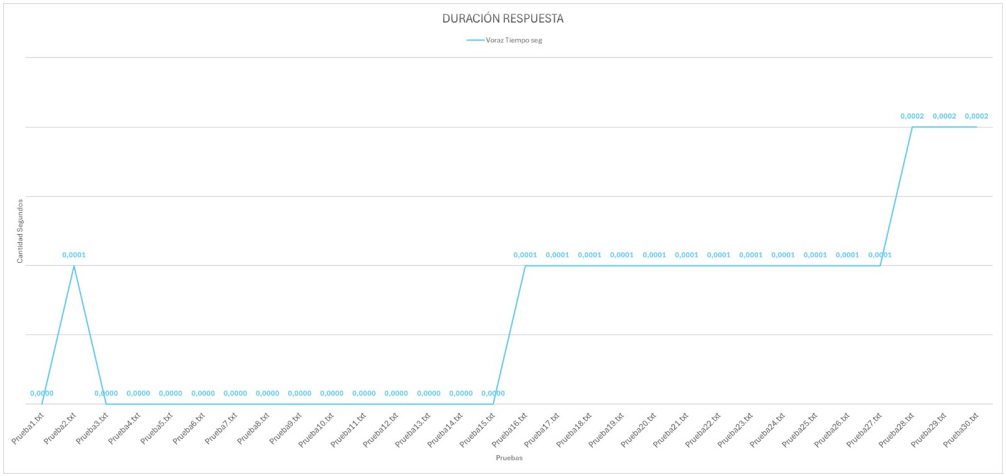

Figura 8: Gráfica del tiempo de respuesta obtenido por algoritmo voraz.

Conclusión:

- Si se prioriza el tiempo de ejecución, el algoritmo voraz es ideal, resolviendo virtualmente al instante.
- Para situaciones donde la calidad de la solución es crítica, puede preferirse programación dinámica, aunque con tiempos más elevados.
- Fuerza bruta solo es viable para instancias pequeñas, siendo impráctico para escenarios reales por su excesivo tiempo.

La siguiente tabla presenta una comparación general entre los tres enfoques implementados: fuerza bruta, programación dinámica y algoritmo voraz. Se analizan aspectos clave como eficiencia, escalabilidad y adecuación al problema:

Criterio	**Fuerza Bruta**	**Programación Dinámica**	**Algoritmo Voraz**
Complejidad Temporal	$O(n(k+1)^n)$	$O(n \cdot R_{\text{real}} \cdot k_{\text{real}})$	$O(n \log n)$
Complejidad Espacial	$O((k+1)^n)$	$O(n^2 \times R_{max} \times k)$	$O(n)$
¿Siempre da la solución óptima?	Sí	Sí	No
¿Es capaz de manejar gran cantidad de agentes?	No	Moderada	Alta
Aplicación recomendada	Problemas pequeños	Problemas moderados	Problemas grandes
Eficiencia en Tiempo	Muy baja (crece exponencialmente)	Moderada (tiempo polinomial)	Alta (tiempo lineal o logarítmico)
Eficiencia en Espacio	Muy baja, requiere mucho almacenamiento	Moderada, requiere una cantidad de almacenamiento moderada	Alta, requiere poco espacio adicional

Cuadro 14: Comparación general entre algoritmos.

Análisis general:

- **Fuerza Bruta** es útil como referencia o verificación para instancias pequeñas. Su exactitud es total, pero su alto costo computacional limita su uso práctico a casos muy reducidos.
- **Programación Dinámica** logra mantener la precisión y mejorar la eficiencia respecto a la fuerza bruta. Aun así, su uso intensivo de memoria y tiempo en casos grandes representa una limitación.
- **Algoritmo Voraz**, aunque no garantiza la solución óptima, ofrece resultados de buena calidad en tiempos extremadamente bajos, por lo que es ideal para escenarios donde se necesita una respuesta rápida y lo más cercana a la óptima.

6.3. Ventajas y Desventajas

El análisis de los tres algoritmos propuestos para resolver el problema de moderación del conflicto interno en redes sociales permite establecer comparaciones fundamentales en cuanto a precisión, eficiencia y aplicabilidad práctica. A continuación, se sintetizan las ventajas y desventajas más relevantes de cada enfoque:

6.3.1. Fuerza Bruta

- **Ventajas:**
  - Garantiza encontrar la solución óptima, al evaluar todas las posibles combinaciones.
  - Es útil como referencia o punto de comparación para validar otros algoritmos aproximados.
  - Implementación directa y sencilla desde el punto de vista conceptual.
- **Desventajas:**
  - Su complejidad temporal es exponencial, lo que lo vuelve inviable para instancias grandes.
  - Requiere un tiempo de ejecución considerable incluso para tamaños de entrada moderados.
  - No escala con eficiencia a redes sociales reales con cientos o miles de agentes.

6.3.2. Programación Dinámica

- **Ventajas:**
  - Debido a que se divide el problema en subproblemas y se resuelve cada uno, se garantiza encontrar la solución óptima, en las pruebas realizadas, la solución siempre es la óptima como se puede ver.
  - La complejidad espacial es reducida significativamente a diferencia de fuerza bruta.
  - Se evita hacer cálculos repetidos ya que utiliza los que ya ha hecho en otros subproblemas, haciendolá más adecuada para redes medianas y grandes donde se requiere precisión y rendimiento.
- **Desventajas:**
  - Aunque su complejidad temporal mejora, se ve afectado el espacio en memoria, la complejidad espacial, con el fin de almacenar las soluciones parciales

6.3.3. Voraz

- **Ventajas:**
  - Su ejecución es rápida y consume pocos recursos, ideal para entornos donde el tiempo de respuesta es crítico.
  - La implementación es sencilla y generalmente más comprensible.
  - Escala bien para redes sociales grandes.
- **Desventajas:**
  - No garantiza obtener la solución óptima, ya que toma decisiones locales sin evaluar el impacto global.

Comparación Global

El algoritmo de fuerza bruta actúa como una línea base teórica inigualable en precisión, pero ineficiente. La programación dinámica logra un balance adecuado entre eficiencia y exactitud, mientras que el enfoque voraz se justifica principalmente cuando se requiere velocidad y se puede tolerar una aproximación. La elección del algoritmo adecuado dependerá del tamaño del problema, los recursos disponibles y el nivel de precisión requerido por la aplicación práctica.

7. Casos de Prueba

Se realizan casos de prueba para los diferentes tipos de algoritmos (fuerza bruta, programación dinámica y voraz) con el objetivo de evaluar su eficiencia, exactitud y comportamiento en diversas condiciones. Esto permite comparar qué algoritmo es más eficiente en determinados casos, poniendo a prueba su desempeño en situaciones donde el uso de memoria y capacidad de procesamiento son claves para su correcto funcionamiento. Se han llevado a cabo los siguientes casos de pruebas:

7.1. Caso de Prueba 1

Nombre Del Caso De Prueba	**Caso De Prueba 1**	
Descripción	Red Social (5 grupos) Personas: 5, Opinión 1: -6, Opinión 2: -94, Terquedad: 0.062 Personas: 6, Opinión 1: -84, Opinión 2: -7, Terquedad: 0.378 Personas: 1, Opinión 1: -52, Opinión 2: 33, Terquedad: 0.073 Personas: 4, Opinión 1: 77, Opinión 2: -47, Terquedad: 0.626 Personas: 4, Opinión 1: -75, Opinión 2: 75, Terquedad: 0.718 Conflicto Interno: 46604.60 Esfuerzo Disponible: 4044	
Pasos	**Resultados Esperados del Sistema**	**Resultados Obtenidos del Sistema**
1. Subir el archivo	Se visualiza la tabla que describe el contenido del archivo de prueba	Se visualiza la tabla que describe el contenido del archivo de prueba
	Se habilitan los botones para resolver la prueba por los diferentes metodos	Se habilitan los botones para resolver la prueba por los diferentes metodos
2. Click en el algoritmo a ejecutar	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.
	Se visualiza la tabla que describe el contenido despues de moderar la red social	Se visualiza la tabla que describe el contenido despues de moderar la red social
Algoritmo	**Resultados del Conflicto Interno Esperado**	**Resultados del Conflicto Interno Obtenido**
Fuerza Bruta	0	0
Dinámico	0	0
Voraz	0	0

Figura 9: Caso de Prueba 1

7.2. Caso de Prueba 2

Nombre Del Caso De Prueba	Caso De Prueba 2	
Descripción	Red Social (5 grupos) Personas: 7, Opinión 1: -52, Opinión 2: 87, Terquedad: 0.372 Personas: 3, Opinión 1: -44, Opinión 2: -27, Terquedad: 0.965 Personas: 6, Opinión 1: -13, Opinión 2: 100, Terquedad: 0.439 Personas: 6, Opinión 1: 24, Opinión 2: 40, Terquedad: 0.45 Personas: 5, Opinión 1: 37, Opinión 2: -38, Terquedad: 0.18 Conflicto Interno: 48477.80 Esfuerzo Disponible: 388	
Pasos	**Resultados Esperados del Sistema**	**Resultados Obtenidos del Sistema**
1. Subir el archivo	Se visualiza la tabla que describe el contenido del archivo de prueba	Se visualiza la tabla que describe el contenido del archivo de prueba
	Se habilitan los botones para resolver la prueba por los diferentes metodos	Se habilitan los botones para resolver la prueba por los diferentes metodos
2. Click en el algoritmo a ejecutar	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.
	Se visualiza la tabla que describe el contenido despues de moderar la red social	Se visualiza la tabla que describe el contenido despues de moderar la red social
Algoritmo	**Resultados del Conflicto Interno Esperado**	**Resultados del Conflicto Interno Obtenido**
Fuerza Bruta	19616.4	19616.4
Dinámico	19616.4	19616.4
Voraz	19616.4	19667.6

Figura 10: Caso de Prueba 2

7.3. Caso de Prueba 3

Nombre Del Caso De Prueba	Caso De Prueba 3	
Descripción	Red Social (5 grupos) Personas: 1, Opinión 1: -67, Opinión 2: -8, Terquedad: 0.41 Personas: 5, Opinión 1: -73, Opinión 2: -96, Terquedad: 0.73 Personas: 6, Opinión 1: -26, Opinión 2: -20, Terquedad: 0.703 Personas: 7, Opinión 1: 88, Opinión 2: -3, Terquedad: 0.068 Personas: 7, Opinión 1: 73, Opinión 2: -3, Terquedad: 0.608 Conflicto Interno: 20948.20 Esfuerzo Disponible: 365	
Pasos	**Resultados Esperados del Sistema**	**Resultados Obtenidos del Sistema**
1. Subir el archivo	Se visualiza la tabla que describe el contenido del archivo de prueba	Se visualiza la tabla que describe el contenido del archivo de prueba
	Se habilitan los botones para resolver la prueba por los diferentes metodos	Se habilitan los botones para resolver la prueba por los diferentes metodos
2. Click en el algoritmo a ejecutar	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.
	Se visualiza la tabla que describe el contenido despues de moderar la red social	Se visualiza la tabla que describe el contenido despues de moderar la red social
Algoritmo	**Resultados del Conflicto Interno Esperado**	**Resultados del Conflicto Interno Obtenido**
Fuerza Bruta	1621.6	1621.6
Dinámico	1621.6	1621.6
Voraz	1621.6	1720.2

Figura 11: Caso de Prueba 3

7.4. Caso de Prueba 4

Nombre Del Caso De Prueba	Caso De Prueba 4	
Descripción	Red Social (5 grupos) Personas: 2, Opinión 1: -17, Opinión 2: 25, Terquedad: 0.309 Personas: 4, Opinión 1: -54, Opinión 2: 88, Terquedad: 0.339 Personas: 3, Opinión 1: -4, Opinión 2: 75, Terquedad: 0.365 Personas: 3, Opinión 1: -87, Opinión 2: --63, Terquedad: 0.317 Personas: 7, Opinión 1: -99, Opinión 2: -40, Terquedad: 0.968 Conflicto Interno: 25800.40 Esfuerzo Disponible: 315	
Pasos	**Resultados Esperados del Sistema**	**Resultados Obtenidos del Sistema**
1. Subir el archivo	Se visualiza la tabla que describe el contenido del archivo de prueba	Se visualiza la tabla que describe el contenido del archivo de prueba
	Se habilitan los botones para resolver la prueba por los diferentes metodos	Se habilitan los botones para resolver la prueba por los diferentes metodos
2. Click en el algoritmo a ejecutar	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.
	Se visualiza la tabla que describe el contenido despues de moderar la red social	Se visualiza la tabla que describe el contenido despues de moderar la red social
Algoritmo	**Resultados del Conflicto Interno Esperado**	**Resultados del Conflicto Interno Obtenido**
Fuerza Bruta	5103.8	5103.8
Dinámico	5103.8	5103.8
Voraz	5103.8	5219

Figura 12: Caso de Prueba 4

7.5. Caso de Prueba 5

Nombre Del Caso De Prueba	Caso De Prueba 5	
Descripción	Red Social (5 grupos) Personas: 6, Opinión 1: 57, Opinión 2: 10, Terquedad: 0.537 Personas: 9, Opinión 1: 63, Opinión 2: 98, Terquedad: 0.749 Personas: 4, Opinión 1: -36, Opinión 2: 39, Terquedad: 0.636 Personas: 9, Opinión 1: -24, Opinión 2: 52, Terquedad: 0.984 Personas: 4, Opinión 1: 49, Opinión 2: -69, Terquedad: 0.452 Conflicto Interno: 30891.80 Esfuerzo Disponible: 460	
Pasos	**Resultados Esperados del Sistema**	**Resultados Obtenidos del Sistema**
1. Subir el archivo	Se visualiza la tabla que describe el contenido del archivo de prueba	Se visualiza la tabla que describe el contenido del archivo de prueba
	Se habilitan los botones para resolver la prueba por los diferentes metodos	Se habilitan los botones para resolver la prueba por los diferentes metodos
2. Click en el algoritmo a ejecutar	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.	Se visualiza los resultados obtenidos despues de ejecutado el metodo seleccionado.
	Se visualiza la tabla que describe el contenido despues de moderar la red social	Se visualiza la tabla que describe el contenido despues de moderar la red social
Algoritmo	**Resultados del Conflicto Interno Esperado**	**Resultados del Conflicto Interno Obtenido**
Fuerza Bruta	14369	14369
Dinámico	14369	14369
Voraz	14369	14369

Figura 13: Caso de Prueba 5

8. Conclusiones

- **El algoritmo de fuerza bruta**: garantiza encontrar la solución óptima al problema, ya que explora exhaustivamente todas las combinaciones posibles de esfuerzo entre los grupos. Sin embargo, su principal desventaja es la complejidad exponencial, lo que lo hace inviable para redes con más de unos pocos grupos. En la práctica, su uso queda restringido a casos muy pequeños, donde se puede usar como referencia o para validar otros métodos.

- **El algoritmo de programación dinámica**: representa una mejora significativa en cuanto a eficiencia respecto al enfoque de fuerza bruta. Se basa en subproblemas más pequeños y permite reutilizar resultados intermedios, logrando una solución óptima dentro del presupuesto de esfuerzo. A pesar de que su complejidad temporal y espacial sigue siendo considerable, su rendimiento es aceptable en redes medianas, incluso con restricciones de tiempo o memoria. Permite alcanzar un equilibrio entre exactitud, rendimiento y escalabilidad.

- **El algoritmo voraz**: es el más rápido de los tres, ya que se basa en una heurística que prioriza grupos con alta discrepancia de opiniones y baja rigidez. Aunque no garantiza una solución óptima, ofrece resultados suficientemente buenos en tiempos muy bajos. Su bajo costo computacional lo hace ideal para redes grandes o situaciones donde se requiere una respuesta rápida. Sin embargo, su eficacia depende fuertemente de la calidad de la heurística utilizada, y puede no funcionar bien en ciertos casos.

- **En conclusión**: el análisis comparativo evidencia que cada enfoque tiene su nicho de aplicabilidad. La fuerza bruta es útil solo para instancias pequeñas y como referencia. La programación dinámica es preferible cuando se necesita precisión y aún se dispone de recursos moderados. El enfoque voraz es el más adecuado cuando se busca eficiencia y rapidez en contextos de gran escala. La elección del algoritmo debe estar guiada por el tamaño de la red, el presupuesto disponible y el nivel de precisión requerido.

9. Ejecución del proyecto

La ejecución de este proyecto se apoya en contenedores Docker para garantizar un entorno aislado y reproducible. A continuación, se detallan los pasos necesarios para clonar, construir y lanzar la aplicación, tanto en sistemas Windows como Linux/macOS.

1. Clonar el repositorio

Abra una terminal o consola de comandos y ejecute:

```
git clone https://github.com/Ajred96/ProyectoIADAII.git
cd ProyectoIADAII
```

2. Instalar Docker y Docker Compose

Antes de continuar, asegúrese de tener instalado Docker en su sistema.

- **Windows:** Instalar desde https://www.docker.com/products/docker-desktop
- **Linux/macOS:** Seguir instrucciones en https://docs.docker.com/engine/install/

Una vez instalado, puede verificar la instalación ejecutando:

```
docker --version
docker compose version
```

3. Construir y ejecutar el proyecto

Desde la raíz del proyecto, ejecute el siguiente comando para construir e iniciar los contenedores:

```
docker compose up --build
```

Este comando descargará las imágenes necesarias y levantará automáticamente los servicios de la aplicación.

4. Acceder a la aplicación

Una vez que todos los servicios estén corriendo, la interfaz estará disponible en su navegador en:

```
http://localhost:4200
```

La aplicación quedará lista para ser utilizada sin necesidad de configuración adicional.

Referencias

Auger, N., Juge, V., Nicaud, C., & Pivoteau, C. (2018). On the Worst-Case Complexity of TimSort. *Proceedings of the 26th Annual European Symposium on Algorithms (ESA 2018)*, *112*, 4:1-4:12. https://doi.org/10.4230/LIPIcs.ESA.2018.4

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd). MIT Press.

DeepSeek. (2025). DeepSeek Chat [Modelo de lenguaje de gran escala] [Consultado en de abril de 2025]. https://www.deepseek.com

OpenAI. (2025). ChatGPT: Modelo de lenguaje de gran escala de OpenAI [Consultado en de abril de 2025]. https://chat.openai.com

Python Software Foundation. (2025). *Sorting HOW TO — Documentación de Python*. Consultado el 12 de abril de 2025, desde https://docs.python.org/es/3.13/howto/sorting.html