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 abril de 2025

Índice general

0.1. Programación voraz

Durante el desarrollo de soluciones heurísticas para el problema de moderación de opiniones en redes sociales (ModCI), se planteó como objetivo diseñar una estrategia **voraz** que, si bien no garantizara la solución óptima, pudiera ofrecer resultados cercanos al óptimo en tiempos razonables. Este enfoque es especialmente valioso para redes de gran tamaño, donde las soluciones exactas resultan computacionalmente costosas.

0.1.1. Variantes de estrategias voraces evaluadas

El proceso comenzó con la definición de **métricas de priorización** para determinar el orden en que deberían moderarse los grupos de agentes. Cada variante compartía una estructura general similar:

1. **Ordenar los grupos** según una métrica específica.
2. **Recorrer secuencialmente los grupos ordenados**, moderando agentes en cada uno hasta agotar el presupuesto de esfuerzo disponible.

Las métricas evaluadas para ordenar los grupos incluyeron:

- **Discrepancia absoluta**: prioriza grupos con mayor desacuerdo interno entre opiniones. $|o_1 - o_2|$
- **Radio discrepancia/rigidez**: favorece grupos donde la discrepancia es alta y la rigidez es baja, buscando así un mayor impacto en la reducción del conflicto interno a un menor costo. $\frac{|o_1-o_2|}{r}$
- **Reducción conflicto por esfuerzo necesario**: se toma el aporte que hace cada grupo a la reducción del conflicto interno (numerador) y se divide por el esfuerzo necesario para moderar un solo agente de ese grupo (denominador), dando una idea de la eficiencia en la reducción del conflicto interno por grupo, teniendo en cuenta su costo. $\frac{(o_1-o_2)^2}{|o_1-o_2|r}$

Cada una de estas métricas fue implementada en su correspondiente estrategia voraz, siguiendo la estructura común, y se llevaron a cabo pruebas sistemáticas para evaluar su desempeño sobre distintos casos de prueba. Estas pruebas compararon inicialmente el **conflicto interno resultante** y su cercanía a la solución óptima. Esto dio como resultado la siguiente tabla:

Prueba	Discrepancy-rigidity	Efficiency	Discrepancy-rigidity-one-at-a-time	Discrepancy	Óptimo
1	0	0	0	0	0,0
2	19616,4	32090	19667,6	21428,4	19616,4
3	1621,6	5776	1727,4	2423,6	1621,6
4	5103,8	13491	5219	5924,6	5103,8
5	14369	20981,6	14369	16287	14369,0
6	31002,7	36483,3	32086,1	40174,1	31002,7
7	2762,9	8941,4	3012,9	2827,7	2762,9
8	22057	31613,9	22178	23647,2	21948,9
9	75544,6	77092	75954,2	81186,2	75544,6
10	0	0	0	0	0,0
11	19615,9	29174,8	20911,733	28848,133	19391,9
12	16558,3	18361,2	16696,933	20843,6	16558,3
13	43861	69317,2	44072,6	44072,6	43861,0
14	22732,3	37460,333	23286,466	40054,6	22694,3
15	68944,4	97733,6	69209	70267,4	68944,4
16	0	0	0	0	0,0
17	771,15	8527,8	919,4	2552,55	771,2
18	28062,6	34011,56	28423,96	33411,84	28058,1
19	0	0	0	0	0,0
20	12572,6	22393,28	13295,48	14685	12431,4
21	0	0	0	0	0,0
22	0	0	0	0	0,0
23	0	0	0	0	0,0
24	0	0	0	0	0,0
25	0	0	0	0	0,0
26	28988,2	35850,88	29283,48	33048,16	28988,2
27	25110,6	39758,06	25392,42	29556,6	25102,4
28	937,23	3129,11	991,18	1248,81	930,0
29	0	0	0	0	0,0
30	0	0	0	0	0,0

0.1.2. Selección de la mejor estrategia

En los resultados de la tabla anterior, se puede apreciar que en todos los casos la solución basada en el **radio discrepancia/rigidez** es mejor en comparación con las demás, obteniendo el resultado óptimo en una gran mayoría de los casos. Teniendo esto en cuenta, se calculó qué tan cercana era esta solución a la solución óptima, utilizando la siguiente fórmula:

$$\left(\frac{\text{valor solución voraz}}{\text{valor solución óptima}} - 1\right) * 100 \tag{1}$$

Este valor se calculó para cada caso de prueba, y se obtuvo que la calidad de la solución encontrada en comparación a la solución óptima es en promedio de 0,1258 %. De esta manera, se concluyó que la métrica basada en el **radio discrepancia/rigidez** ofrecía la mejor relación entre efectividad y simplicidad. Esta métrica permite una interpretación intuitiva: prioriza los grupos donde moderar es más rentable.[en] términos de esfuerzo invertido frente a reducción del conflicto interno.

Vale la pena aclarar que se probaron dos enfoques distintos para el caso del **radio discrepancia/rigidez**:

1. Moderar los agentes uno por uno en cada iteración.
2. Moderar en cada iteración sobre un grupo la máxima cantidad posible de agentes.

Según los resultados, es más adecuado moderar en cada iteración la máxima cantidad de agentes posible, y así, iterar sobre cada grupo una sola vez. Las diferencias en los resultados obtenidos con cada enfoque se pueden explicar por la función techo presente en la función que calcula el esfuerzo requerido para moderar un determinado número de agentes de un grupo. Moderar cada agente uno por uno no da como resultado el mismo esfuerzo que si se moderaran todos a la vez. Adicionalmente, resulta más eficiente el segundo enfoque porque de esta manera la complejidad de la solución es ajena al número de agentes por grupo y solo se ve afectada por el número de grupos en la red.

0.1.3. Optimización de la estrategia seleccionada

Una vez elegida la mejor estrategia voraz, se enfocaron esfuerzos en **mejorar su eficiencia temporal**, considerando que había oportunidad de mejorar especialmente en el proceso de ordenamiento de los grupos.

Se implementaron dos variantes para esta optimización:

1. **Versión basada en montículo (heap):** Utiliza una estructura de heap máximo para seleccionar iterativamente el grupo con mayor radio, permitiendo una complejidad temporal de $O(n \log n)$.
2. **Versión basada en ordenamiento lineal:** Aplica **Radix Sort** sobre valores enteros escalados del radio, ordenando dígito por dígito cada valor. Grupos con valores de rigidez muy bajos, se ordenan con base en su discrepancia solamente utilizando **Counting sort**, acotando así el número de dígitos del radio discrepancia/rigidez. De este manera, se alcanza una complejidad teórica de $O(n)$.

0.1.4. Consideraciones prácticas

Aunque desde el punto de vista teórico la versión con ordenamiento lineal debería ser más eficiente, las pruebas empíricas mostraron que la versión basada en la utilización de un montículo (heap) resultaba **más rápida en la práctica**. Se realizaron pruebas con redes sociales de tamaños muy grandes, variando el número de grupos en cada caso, y se tomaron los respectivos tiempos de ejecución. En la siguiente tabla podemos ver los resultados:

Tamaño de la prueba	Con montículo	Con ordenamiento lineal
1000	0,0012	0,0046
10000	0,0149	0,0807
100000	0,2716	3,5861
1000000	1,7902	22,7417
10000000	17,0956	233,7900

Como se puede apreciar, la solución con ordenamiento lineal está por debajo de la solución con heap en todos los casos, incluso en los de mayor tamaño.

Aunque es difícil determinar específicamente por qué sucede esto, puede haber varias posibles razones. Para empezar, en la solución con montículos se utiliza la estructura **heapq**, incluida de manera nativa en Python, pero que internamente está implementado en C. Además, la notación $O(n)$ ignora las constantes multiplicativas. En la práctica, **Radix** y **Counting sort** pueden tener constantes grandes debido a:

- Múltiples pasadas sobre los datos (una por cada dígito o valor).
- Operaciones adicionales como **escalado**, **creación de buckets**, y **conversión de tipos**.
- Uso intensivo de estructuras auxiliares (listas, arrays de conteo).

De esta manera, en la práctica puede resultar mucho más eficiente utilizar la versión con **heapq**. Sin embargo, como nos interesa en mayor medida la complejidad teórica, nos quedaremos con la versión de ordenamiento lineal, considerando también que su implementación es más interesante y vale la pena ver más a detalle.

0.1.5. Descripción del algoritmo voraz

Este algoritmo implementa una **estrategia voraz** para moderar agentes en una red social con el objetivo de minimizar el conflicto interno, sin exceder un presupuesto de esfuerzo $r_{\text{máx}}$. Se basa en **ordenar previamente** los grupos de agentes de manera eficiente usando algoritmos de **ordenamiento lineal** (Counting sort y Radix sort), de modo que las decisiones de moderación puedan hacerse rápidamente en orden de prioridad.

Métrica de decisión

Cada grupo de agentes G se caracteriza por:

- Un número de agentes n.
- Dos opiniones o_1, o_2.
- Un nivel de rigidez r.

La métrica utilizada para priorizar los grupos es el **radio discrepancia/rigidez** dado por:

$$\text{radio}(G) = \frac{|o_1 - o_2|}{r} \tag{2}$$

La estrategia voraz elige moderar primero los grupos que ofrecen **mayor discrepancia relativa por unidad de esfuerzo**, es decir, aquellos con mayores valores para la métrica.

Para garantizar una mejor eficiencia y un tratamiento especial de casos extremos, los grupos se dividen en dos categorías:

1. **Grupos prioritarios**: aquellos con rigidez $r < 10^{-6}$. En estos, el esfuerzo requerido por agente es tan bajo que conviene moderarlos primero, ordenándolos simplemente por discrepancia con **Counting sort**.
2. **Grupos normales**: se ordenan en orden descendente según la métrica definida, utilizando **Radix sort** para lograr una eficiencia del orden $O(n)$.

Pasos del algoritmo

1. **Evaluación rápida de caso trivial**:
   - Si el esfuerzo total requerido para moderar todos los agentes es menor o igual al presupuesto $r_{\text{máx}}$, se modera a todos los agentes de todos los grupos.
2. **Separación de grupos**:
   - Se dividen los grupos en **prioritarios** y **normales**, según su rigidez.
3. **Ordenamiento eficiente**:
   - Se ordenan los grupos prioritarios por discrepancia (de mayor a menor) usando **Counting sort**.
   - Se ordenan los grupos normales de acuerdo con su radio discrepancia/rigidez (de mayor a menor) usando **Radix sort**. Para esto, se convierten los valores del radio a enteros utilizando un factor de escalamiento (10^6) que permita conservar precisión, y se ordenan dígito por dígito internamente con **Counting sort**.
   - Finalmente, se concatenan ambos grupos, dejando primero al grupo prioritario.
4. **Moderación secuencial**:
   - Se recorren los grupos ordenados, moderando la mayor cantidad posible de agentes en cada uno, sin exceder el presupuesto de esfuerzo.
5. **Finalización**:
   - El proceso se detiene cuando se agota el presupuesto de esfuerzo disponible o no hay más grupos por moderar.

¿Entendimos el algoritmo?

1. **Entrada:** $RS_1 = (\langle 3, -100, 50, 0{,}5\rangle, \langle 1, 100, 80, 0{,}1\rangle, \langle 1, -10, 0, 0{,}5\rangle, 80)$

   Descripción de los grupos:

   - Grupo 0: 3 agentes, discrepancia 150, rigidez 0,5 → ratio = $\frac{150}{0{,}5} = 300$.
   - Grupo 1: 1 agente, discrepancia 20, rigidez 0,1 → ratio = $\frac{20}{0{,}1} = 200$.
   - Grupo 2: 1 agente, discrepancia 10, rigidez 0,5 → ratio = $\frac{10}{0{,}5} = 20$.

   Orden tras ordenamiento: Grupo 0, luego Grupo 1, luego Grupo 2.

   Moderación:

   - Grupo 0:
     - Esfuerzo por agente = $150 * 0{,}5 = 75$.
     - Puede moderar solo 1 agente (porque $75 \leq 80$).
     - Presupuesto restante = $80 - 75 = 5$.
   - Grupo 1:
     - Esfuerzo por agente = $20 * 0{,}1 = 2$.
     - Puede moderar 2 agentes, pero el grupo solo tiene 1, así que modera 1.
     - Presupuesto restante = $5 - 2 = 3$.
   - Grupo 2:
     - Esfuerzo por agente = $10 * 0{,}5 = 5$.
     - No puede moderar (presupuesto insuficiente).

   Resultado final: [1, 1, 0] → Es decir, 1 agente moderado del grupo 0 y 1 del grupo 1.

   Conflicto interno resultante:

$$CI\left(RS_1{}'\right) = \frac{1 * (-150)^2 + 0 + (-10)^2}{3} = \frac{22500 + 0 + 100}{3} = \frac{22600}{3} = 15033{,}33 \tag{3}$$

2. **Entrada:** $RS_2 = (\langle 3, -100, 100, 0{,}8\rangle, \langle 2, 100, 80, 0{,}5\rangle, \langle 4, -10, 10, 0{,}5\rangle, 400)$

   Descripción de los grupos:

   - Grupo 0: 3 agentes, discrepancia 200, rigidez 0,8 → ratio = $\frac{200}{0{,}8} = 250$.
   - Grupo 1: 2 agentes, discrepancia 20, rigidez 0,5 → ratio = $\frac{20}{0{,}5} = 40$.
   - Grupo 2: 4 agentes, discrepancia 20, rigidez 0,5 → ratio = $\frac{20}{0{,}5} = 40$.

   Orden tras ordenamiento: Grupo 0, luego Grupo 1, luego Grupo 2.

   Moderación:

   - Grupo 0:
     - Esfuerzo por agente = $200 * 0{,}8 = 160$.
     - Puede moderar 2 agentes (320 esfuerzo).
     - Presupuesto restante = $400 - 320 = 80$.
   - Grupo 1:
     - Esfuerzo por agente = $20 * 0{,}5 = 10$.
     - Puede moderar 2 agentes (20 esfuerzo).
     - Presupuesto restante = $80 - 20 = 60$.
   - Grupo 2:
     - Esfuerzo por agente = $20 * 0{,}5 = 10$.
     - Puede moderar 4 agentes (40 esfuerzo).
     - Presupuesto restante = $60 - 40 = 20$.

Resultado final: [2, 2, 4] → Es decir, 2 agentes moderados del grupo 0, 2 del grupo 1 y 4 del grupo 2.
Conflicto interno resultante:

$$CI\left(RS_2{}'\right) = \frac{1 * (-200)^2 + 0 + 0}{3} = \frac{40000}{3} = 13333{,}33 \tag{4}$$

3. **Entrada:** $RS_3 = (\langle 2, 40, -60, 0{,}25\rangle, \langle 3, -50, 50, 0{,}1\rangle, 75)$
Descripción de los grupos:
   - Grupo 0: 2 agentes, discrepancia 100, rigidez 0,25 → ratio = $\frac{100}{0{,}25} = 400$.
   - Grupo 1: 3 agentes, discrepancia 100, rigidez 0,1 → ratio = $\frac{100}{0{,}1} = 1000$.

   Orden tras ordenamiento: Grupo 1, luego Grupo 0.
   Moderación:
   - Grupo 1:
     - Esfuerzo por agente = $100 * 0{,}1 = 10$.
     - Puede moderar 3 agentes (30 esfuerzo).
     - Presupuesto restante = $75 - 30 = 45$.
   - Grupo 0:
     - Esfuerzo por agente = $100 * 0{,}25 = 25$.
     - Puede moderar 1 agente (25 esfuerzo).
     - Presupuesto restante = $45 - 25 = 20$.

   Resultado final: [1, 3] → Es decir, 1 agente moderado del grupo 0 y 3 del grupo 1.
   Conflicto interno resultante:

$$CI\left(RS_3{}'\right) = \frac{1 * (100)^2 + 0}{2} = \frac{10000}{2} = 5000 \tag{5}$$

4. **Entrada:** $RS_4 = (\langle 1, 100, -100, 0{,}01\rangle, \langle 2, 50, -50, 0{,}5\rangle, 5)$
Descripción de los grupos:
   - Grupo 0: 1 agente, discrepancia 200, rigidez 0,01 → ratio = $\frac{200}{0{,}01} = 20000$.
   - Grupo 1: 2 agentes, discrepancia 100, rigidez 0,5 → ratio = $\frac{100}{0{,}5} = 200$.

   Orden tras ordenamiento: Grupo 0, luego Grupo 1.
   Moderación:
   - Grupo 0:
     - Esfuerzo por agente = $200 * 0{,}01 = 2$.
     - Puede moderar 1 agente (2 esfuerzo).
     - Presupuesto restante = $5 - 2 = 3$.
   - Grupo 1:
     - Esfuerzo por agente = $100 * 0{,}5 = 50$.
     - No puede moderar (presupuesto insuficiente).

   Resultado final: [1, 0] → Es decir, 1 agente moderado del grupo 0 y 0 del grupo 1.
   Conflicto interno resultante:

$$CI\left(RS_4{}'\right) = \frac{0 + 2 * (100)^2}{2} = \frac{20000}{2} = 10000 \tag{6}$$

Los resultados se resumen en la siguiente tabla:

Caso	Estrategia	Conflicto interno	¿Óptima?
RS_1	[1,1,0]	15033,33	Sí
RS_2	[2,2,4]	13333,33	Sí
RS_3	[1,3]	5000	Sí
RS_4	[1,0]	10000	Sí

0.1.6. Complejidad temporal teórica

Algorithm 1 Greedy Moderation with Radix and Counting Sort

if calculate_max_effort(social_network) $\leq$ social_network.r_max **then**
  return [*group.n* for each *group* $\in$ social_network.groups]
end if
$n \leftarrow$ length of social_network.groups
strategy $\leftarrow [0, 0, \ldots, 0]$ of size n
remaining_r $\leftarrow$ social_network.r_max
r_min $\leftarrow 10^{-6}$
priority_groups $\leftarrow$ [ ]
normal_groups $\leftarrow$ [ ]
for all *group* $\in$ social_network.groups **do**
  if *group.r* $<$ *r_min* **then**
    append *group* to *priority_groups*
  else
    append *group* to *normal_groups*
  end if
end for
sorted_priority_groups $\leftarrow$ CountingSortByDiscrepancy(*priority_groups*)
sorted_normal_groups $\leftarrow$ RadixSortByRatio(*normal_groups*)
sorted_groups $\leftarrow$ Concatenate(*sorted_priority_groups*, *sorted_normal_groups*)
group_to_index $\leftarrow \{\}$
for $i \leftarrow 0$ **to** $n - 1$ **do**
  group_to_index[id(*social_network.groups*[i])] $\leftarrow i$
end for
for all *group* $\in$ *sorted_groups* **do**
  if *remaining_r* ≤ 0 **then**
    break
  end if
  index $\leftarrow$ *group_to_index*[id(*group*)]
  max_agents $\leftarrow$ *group.n*
  $effort\_per\_agent \leftarrow |group.o_1 - group.o_2| \times group.r$
  if $effort\_per\_agent = 0$ **or** $effort\_per\_agent > remaining\_r$ **then**
    continue
  end if
  max_possible $\leftarrow$ floor($remaining\_r / effort\_per\_agent$)
  agents_to_moderate $\leftarrow$ mín(*max_agents*, *max_possible*)
  if *agents_to_moderate* > 0 **then**
    strategy[*index*] $\leftarrow$ *strategy*[*index*] + *agents_to_moderate*
    $remaining\_r \leftarrow remaining\_r - \lceil agents\_to\_moderate \times effort\_per\_agent \rceil$
  end if
end for
return *strategy*

Como se puede apreciar, cada sección del código tiene como máximo un costo de orden lineal $O(n)$. Luego, la suma de todos los costos parciales da como resultado también un orden lineal. Ahora, como se parte de la idea de que Counting sort y Radix sort se implementan correctamente y estos dos son algoritmos de ordenamiento lineal, se puede determinar que el ordenamiento de los grupos se da en tiempo lineal. Sin embargo, es necesario justificar por qué se garantiza la linealidad en ambos casos.

Counting sort

Tiene una complejidad teórica $O(n+k)$, donde:

- n es el número de elementos a ordenar.
- k es el rango máximo de los valores enteros posibles.

En este algoritmo, usamos Counting sort para ordenar grupos con **rigidez muy baja** ($r < r_{\text{mín}}$) por su **discrepancia**, es decir, por $o_1 - o_2$. Como las opiniones están acotadas entre -100 y 100, la discrepancia máxima es 200 y la mínima es 0. Esto significa que el rango de discrepancia es **constante** (a lo sumo 201 valores posibles), por lo tanto, la complejidad de Counting sort en este caso es:

$$O(n + 201) = O(n) \tag{7}$$

Radix sort

Tiene una complejidad teórica $O(d * (n + b))$, donde:

- n es el número de elementos.
- b es la base usada para descomponer los números (usualmente base 10).
- d es el número de dígitos máximos del valor más grande.

En este algoritmo, Radix Sort se aplica sobre la métrica:

$$\text{radio}(G) = \frac{|o_1 - o_2|}{r} \tag{8}$$

Aunque esta métrica es real (**float**), es escalada previamente a un número entero (en este caso, multiplicando por 10^6) para preservar el orden relativo con suficiente precisión. Dado que $o_1 - o_2$ está acotado y $r \geq 10^{-6}$, el valor máximo posible de la métrica escalada es:

$$\text{radio}_{\text{máx}} = \frac{200}{10^{-6}} * 10^6 = 200 * 10^6 \tag{9}$$

Esto significa que los valores del radio tienen como máximo 15 **dígitos decimales**, es decir, $d = 15$. Luego, la complejidad de Radix sort en este caso es:

$$O(15 * (n + 10)) = O(15n + 150) = O(n) \tag{10}$$

Porque d y b son constantes, la complejidad es de orden lineal.

0.1.7. Corrección del algoritmo

El algoritmo no garantiza encontrar siempre la solución óptima al problema de minimizar el conflicto interno en una red social bajo una restricción de esfuerzo. Esto se debe a que se trata de una estrategia voraz, que en cada paso toma decisiones locales que parecen óptimas (moderando primero los grupos con menor esfuerzo relativo o discrepancia alta), pero sin considerar el efecto global de esas decisiones sobre la red completa.

La estrategia voraz no explora todas las posibles combinaciones de asignación de esfuerzo, sino que selecciona, de forma determinista y ordenada, aquellos grupos que según una métrica predefinida (como discrepancia o discrepancia/rigidez) parecen más beneficiosos. Por tanto, puede quedar atrapada en óptimos locales, sin alcanzar el mínimo conflicto interno global posible.

¿Cuándo sí da la respuesta óptima?

El algoritmo sí produce la solución óptima en ciertos casos particulares, por ejemplo:

- Cuando el presupuesto de esfuerzo es suficiente para moderar completamente todos los grupos (caso trivial): en ese caso, la estrategia modera todos los agentes, lo cual es la solución óptima.
- Cuando los grupos son homogéneos (misma discrepancia y rigidez), y el esfuerzo se distribuye equitativamente.
- Cuando la métrica voraz coincide con el impacto real que tiene moderar un grupo en la reducción del conflicto interno, por ejemplo, si todos los grupos tienen rigidez muy baja (donde la discrepancia domina el cálculo del conflicto interno), o cuando el presupuesto es tan pequeño que solo se puede moderar un único grupo y la elección más beneficiosa es evidente.

¿Cuándo puede no dar la respuesta óptima?

El algoritmo puede fallar al encontrar la mejor solución en los siguientes escenarios:

- Cuando hay una combinación de grupos donde moderar parcialmente varios grupos produce menos conflicto interno que moderar completamente uno solo (aunque este tenga mejor puntuación según la métrica voraz).
- Cuando el impacto acumulativo de moderar ciertos grupos no se alinea con la métrica usada para ordenarlos (por ejemplo, la discrepancia/rigidez no siempre refleja la mejor inversión del esfuerzo si hay casos con rigidez baja pero poco impacto en el conflicto interno).

0.2. Fuerza Bruta

0.2.1. Complejidad

El algoritmo de fuerza bruta implementado en el proyecto presenta una complejidad temporal de orden exponencial. Esta complejidad se deriva del hecho de que el algoritmo explora de manera exhaustiva todas las posibles combinaciones de estrategias de moderación para los grupos que conforman la red social.

Dado un conjunto de k grupos, donde cada grupo i posee n_i agentes, el algoritmo genera el producto cartesiano de los posibles valores de moderación para cada grupo, los cuales varían desde 0 hasta n_i. Como resultado, el número total de combinaciones evaluadas es:

$$O\left(\prod_{i=1}^{k}(n_i+1)\right) \tag{11}$$

Esta naturaleza combinatoria implica que el tiempo de ejecución del algoritmo crece de forma exponencial respecto al número de grupos y al número de agentes por grupo. En consecuencia, este enfoque resulta computacionalmente costoso para redes sociales de tamaño medio o grande, siendo únicamente viable en escenarios reducidos donde la cantidad de grupos y agentes es limitada.

En el caso particular en que todos los grupos tengan el mismo número de agentes, es decir, $n_i = m$ para todo i, la expresión anterior se simplifica a:

$$O\left((m+1)^k\right) \tag{12}$$

$$= O(m^k) \tag{13}$$

Esta forma resume la naturaleza exponencial del algoritmo, destacando cómo su complejidad se incrementa significativamente al aumentar el número de grupos (k) o la cantidad de agentes por grupo (m).

0.2.2. Corrección

El algoritmo de fuerza bruta implementado en el proyecto garantiza la obtención de la solución óptima para el problema planteado. Este algoritmo evalúa todas las posibles combinaciones de moderación entre los grupos de agentes de la red social.

Este enfoque tiene como objetivo garantizar que:

- Se consideren todas las estrategias viables, es decir, todas las formas posibles de moderar entre 0 y n_i agentes en cada grupo i.
- Se verifique, para cada combinación, si el esfuerzo total requerido no excede el límite permitido $R_{\text{máx}}$.
- Se calcule el conflicto interno resultante para cada estrategia válida.
- Finalmente, se elija la combinación que minimiza el conflicto interno, cumpliendo así con el objetivo del problema.

Este procedimiento asegura la corrección del resultado, ya que no se omite ninguna posibilidad y se selecciona óptimamente la estrategia más beneficiosa dentro de las restricciones dadas. Sin embargo, esta exactitud tiene como costo una alta complejidad computacional, lo cual hace que el algoritmo solo sea viable para instancias pequeñas o moderadas del problema.

0.3. Programación dinámica

0.3.1. Caracterización de la subestructura óptima

Definimos la notación $\mathrm{IC}(i, r)$ como el problema de minimizar el conflicto interno utilizando los primeros i grupos de agentes con un esfuerzo disponible r. La notación $\mathrm{modIC}(i, r)$ denota la secuencia de decisiones óptimas (es decir, el número de agentes a moderar en cada grupo) para los grupos i hasta n utilizando r de esfuerzo, donde la secuencia vacía $\langle\rangle$ indica que no se realizan más moderaciones.

$$\mathrm{modIC}(i, r) = \begin{cases} \langle\rangle & i = n + 1 \\ k_i :: \mathrm{modIC}(i + 1, r - \mathrm{e\_effort}(i, k_i)) & i \neq n \end{cases} \tag{14}$$

$$\mathrm{IC}(i, R_{\mathrm{máx}}) = \mathrm{modIC}(1, R_{\mathrm{máx}}) \tag{15}$$

0.3.2. Valor recursivo de la solución óptima

- e_effort(i, k): es el esfuerzo requerido para **moderar** k agentes del grupo i ($\lceil |o_{i,1} - o_{i,2}| \cdot r_i \cdot k \rceil$).
- conflict(i, k): es la contribución al conflicto interno del grupo i luego de **moderar** k agentes ($(n_i - k) * (o_{i,1} - o_{i,2})^2$).

Normalización en cada paso

$$\mathrm{recIC}_1(i, r) = \begin{cases} 0 & i = 0 \\ \mathrm{mín}_{0 \leq k_i \leq n_i} \left\{ \frac{(i-1) \cdot \mathrm{recIC}_1(i-1, r - \mathrm{e\_effort}(i,k_i)) + \mathrm{conflict}(i,k_i)}{i} \right\} & i > 0 \end{cases} \tag{16}$$

$$\mathrm{valueIC}(n, R_{\mathrm{máx}}) = \mathrm{recIC}_1(n, R_{\mathrm{máx}}) \tag{17}$$

Acumulación total y normalización final

$$\mathrm{recIC}_2(i, r) = \begin{cases} 0 & i = 0 \\ \mathrm{mín}_{0 \leq k_i \leq n_i} \{\mathrm{recIC}_2(i - 1, r - \mathrm{e\_effort}(i, k_i)) + \mathrm{conflict}(i, k_i)\} & i > 0 \end{cases} \tag{18}$$

$$\mathrm{valueIC}(n, R_{\mathrm{máx}}) = \frac{\mathrm{recIC}_2(n, R_{\mathrm{máx}})}{n} \tag{19}$$

Justificación equivalencia

Sea $F(i, r)$ la suma total del conflicto acumulado en los i primeros grupos.
Ambas formulaciones calculan el mismo valor $\mathrm{valueIC}(n, R_{\mathrm{máx}}) = \frac{1}{n} \sum_{i=0}^{n-1} \mathrm{conflict}(i, k_i)$.

- Caso base: para $i = 0$, ambas definiciones retornan 0.
- Paso inductivo: supongamos que para $i - 1$, $\mathrm{recIC}_1(i, r') = \frac{F(i-1,r')}{i-1}$ y $\mathrm{recIC}_2(i, r') = F(i - 1, r')$.
-
  - recIC_1.

$$\mathrm{recIC}_1(i, r) = \frac{(i - 1) \cdot \mathrm{recIC}_1(i - 1, r - \mathrm{e\_effort}(i, k_i)) + \mathrm{conflict}(i, k_i)}{i} \tag{20}$$

$$= \frac{F(i - 1, r') + \mathrm{conflict}(i, k_i)}{i} \tag{21}$$

  - recIC_2.

$$\mathrm{recIC}_2(i, r) = \mathrm{recIC}_2(i - 1, r - \mathrm{e\_effort}(i, k_i)) + \mathrm{conflict}(i, k_i) \tag{22}$$

$$= F(i - 1, r') + \mathrm{conflict}(i, k_i) \tag{23}$$

Al final $\mathrm{valueIC}(n, R_{\mathrm{máx}}) = \frac{\mathrm{recIC}_2(n, R_{\mathrm{máx}})}{n} = \frac{F(n, R_{\mathrm{máx}})}{n}$, que coincide con $\mathrm{recIC}_1(n, R_{\mathrm{máx}})$.
Lo cual tiene sentido, minimizar para $\frac{F(i-1,r') + \mathrm{conflict}(i,k_i)}{i}$ es equivalente a minimizar $F(i-1, r') + \mathrm{conflict}(i, k_i)$ porque dividir por una constante positiva (i) no cambia la elección del k_i que da el mínimo valor.

0.3.3. Algoritmo para calcular el costo de una solución óptima

Para calcular el costo óptimo mediante programación dinámica, utilizamos un bottom-up que construye sistemáticamente la solución a partir de los casos base.

Definimos una matriz *storage* donde $storage[i][r]$ representa el mínimo conflicto interno posible considerando los primeros i grupos con un esfuerzo disponible r. El algoritmo completa esta matriz iterativamente:

Algorithm 2 cálculo del costo óptimo mediante programación dinámica (bottom-up)

```
n ← número de grupos
r_max ← esfuerzo máximo disponible
inicializar storage[0][r] = 0 para todo r ∈ [0, r_máx]
inicializar storage[i][r] = ∞ para todo i ∈ [1, n], r ∈ [0, r_máx]
for i = 1 hasta n do
    obtener grupo actual: a_{i−1} = ⟨n_i, o_{i,1}, o_{i,2}, r_i⟩
    conflict_per_agent ← (o_{i,1} − o_{i,2})^2
    effort_per_agent ← |o_{i,1} − o_{i,2}| · r_i
    for r = 0 hasta r_máx do
        for k = 0 hasta n_i do
            required_effort ← ⌈effort_per_agent · k⌉
            if required_effort ≤ r then
                remaining_conflict ← (n_i − k) · conflict_per_agent
                total_conflict ← storage[i − 1][r − required_effort] + remaining_conflict
                if total_conflict < storage[i][r] then
                    storage[i][r] ← total_conflict
                end if
            end if
        end for
    end for
end for
return storage[n][r_máx]
```

El valor $storage[n][r_{\text{máx}}]$ representa la suma total del conflicto mínimo posible. Para obtener el conflicto interno normalizado según la definición del problema, debemos dividir este valor por n:

$$\text{valueIC}(n, R_{\text{máx}}) = \frac{storage[n][r_{\text{máx}}]}{n} \tag{24}$$

0.3.4. Algoritmo para calcular una solución óptima

Mientras que la sección anterior nos permite calcular el valor mínimo de conflicto interno, también necesitamos determinar la estrategia óptima que produce dicho valor. Para reconstruir esta estrategia, utilizamos una matriz adicional durante el proceso que registra las decisiones óptimas.

Definimos una matriz *decisions* donde $decisions[i][r]$ representa el número óptimo de agentes a moderar del grupo i cuando se dispone de un esfuerzo r. Esta matriz se completa durante el mismo proceso iterativo en que calculamos el costo mínimo.

Algorithm 3 registro de decisiones (extensión del algoritmo anterior)

```
inicializar decisions[i][r] = 0 para todo i ∈ [0, n], r ∈ [0, r_máx]
...
if total_conflict < storage[i][r] then
    storage[i][r] ← total_conflict
    decisions[i][r] ← k                          ▷ registramos la decisión óptima
end if
...
```

Una vez completadas las matrices *storage* y *decisions*, podemos reconstruir la estrategia óptima mediante un proceso de retroceso:

Algorithm 4 reconstrucción de la estrategia óptima

```
n ← número de grupos
r_máx ← esfuerzo máximo disponible
inicializar optimal_strategy como un arreglo de n ceros
remaining_effort ← r_máx
for i = n hasta 1 (en orden descendente) do
    k ← decisions[i][remaining_effort]                ▷ número óptimo de agentes a moderar
    optimal_strategy[i − 1] ← k
    obtener grupo actual: a_{i−1} = ⟨n_i, o_{i,1}, o_{i,2}, r_i⟩
    effort_per_agent ← |o_{i,1} − o_{i,2}| · r_i
    required_effort ← ⌈effort_per_agent · k⌉
    remaining_effort ← remaining_effort − required_effort
end for
return optimal_strategy
```

El resultado *optimal_strategy* es un arreglo donde la posición i indica cuántos agentes deben ser moderados del grupo i para lograr el mínimo conflicto interno posible. Esta estrategia satisface la restricción de que su esfuerzo total no excede $R_{\text{máx}}$.

La estrategia resultante $E = \langle e_0, e_1, \ldots, e_{n-1} \rangle$ representa nuestra solución final al problema, donde cada e_i indica exactamente cuántos agentes del grupo i deben ser moderados para minimizar el conflicto interno de la red social.

0.3.5. Complejidad

Rellenando la matriz *storage* **y** *decisions*

Algorithm 5 estructura del cálculo del costo óptimo mediante programación dinámica (bottom-up)

```
...
for i = 1 hasta n do
    ...
    for r = 0 hasta r_máx do
        for k = 0 hasta n_i do
            ...
        end for
    end for
end for
return storage[n][r_máx]
```

- Temporal: $O(n \cdot R_{\text{máx}} \cdot \text{máx}(n_i))$, donde n es el número de grupos, $R_{\text{máx}}$ es el esfuerzo máximo disponible y máx(n_i) es el número máximo de agentes en cualquier grupo.
- Espacial: $O(n \cdot R_{\text{máx}})$, donde n es el número de grupos, $R_{\text{máx}}$ es el esfuerzo máximo disponible.

  Esto es válido porque en el ciclo interior, las operaciones que se hacen son asignaciones, es decir, tienen complejidad $O(1)$.

Recuperando la solución de la matriz *decisions*

Algorithm 6 estructura de la reconstrucción de la estrategia óptima

```
...
inicializar optimal_strategy como un arreglo de n ceros
...
for i = n hasta 1 (en orden descendente) do
    ...
end for
return optimal_strategy
```

- Temporal: $O(n)$, donde n es el número de grupos.

- Espacial: $O(n)$, donde n es el número de grupos.

  Esto es válido porque en el ciclo interior, las operaciones que se hacen son asignaciones, es decir, tienen complejidad $O(1)$.

Por lo tanto, la complejidad temporal del algoritmo es $O(n \cdot R_{\text{máx}} \cdot \text{máx}(n_i) + n)$ y la espacial es $O(n \cdot R_{\text{máx}} + n)$.

0.4. Comparación entre estrategias

Cuadro 1: Comparación de complejidad entre estrategias de moderación

Estrategia	**Complejidad temporal**	**Complejidad espacial**	**¿Óptima?**
Voraz	$O(n)$	$O(n)$	No
Fuerza bruta	$O\left(\prod_{i=1}^{k}(n_i+1)\right)$	$O(k)$	Sí
Programación dinámica	$O(n \cdot R_{\text{máx}} \cdot \text{máx}(n_i))$	$O(n \cdot R_{\text{máx}})$	Sí

Cada una de las estrategias analizadas presenta ventajas y desventajas que deben ser consideradas al momento de seleccionar la más adecuada para un escenario real:

- **Voraz:**
  - **Ventajas:** Su principal fortaleza es la eficiencia, ya que puede manejar redes sociales de gran tamaño en tiempos muy reducidos y con bajo consumo de memoria. Es fácil de implementar y ajustar, y en la práctica suele obtener soluciones muy cercanas al óptimo.
  - **Desventajas:** No garantiza la obtención de la solución óptima en todos los casos.
- **Fuerza bruta:**
  - **Ventajas:** Garantiza encontrar la solución óptima, ya que explora exhaustivamente todas las combinaciones posibles. Es útil como referencia para validar la calidad de otras estrategias en instancias pequeñas.
  - **Desventajas:** Su complejidad exponencial lo hace inviable para redes sociales medianas o grandes, ya que el tiempo y la memoria requeridos crecen rápidamente con el número de grupos y agentes.
- **Programación dinámica:**
  - **Ventajas:** Ofrece una solución óptima con una complejidad mucho menor que la fuerza bruta, siendo factible para instancias de tamaño moderado. Permite un balance entre exactitud y eficiencia.
  - **Desventajas:** Aunque mejora respecto a la fuerza bruta, sigue siendo costosa en términos de tiempo y memoria para redes muy grandes o cuando el presupuesto de esfuerzo es alto, ya que la complejidad depende linealmente de ambos factores.

En aplicaciones reales donde la escala de la red social es grande y se requiere una respuesta rápida, la estrategia voraz es la opción más práctica, sacrificando una mínima cantidad de optimalidad a cambio de eficiencia. Para instancias pequeñas o cuando se requiere la mejor solución posible, la programación dinámica es preferible. La fuerza bruta queda relegada a propósitos de validación o análisis teórico en casos muy reducidos.