

Informe de proyecto - Moderando el conflicto interno de opiniones en
una red social



Integrantes:

Jorge Enrique Alvarez Valderrama-2125356

Miguel Angel Escobar Velez –2159832

Jean Paul Davalos Valencia- 1832375

Alejandro Guerrero Cano - 2179652

Presentado a:

Juan Francisco Diaz

Jesús Alexander

Aranda

Análisis de

Algoritmos II

Universidad del Valle
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y
Computación 2025 - I

ÍNDICE

1. Resumen	3
2. Aplicando fuerza bruta	4
2.1. Complejidad	4
2.2. Corrección	4
3. Aplicando un algoritmo voraz	5
3.1. Describiendo el algoritmo	5
3.2. Entendiendo el algoritmo	8
3.3. Complejidad	10
3.4. Corrección	11
4. Aplicando programación dinámica	13
4.1. Caracterizando la estructura de una solución óptima	13
4.2. Definiendo recursivamente el valor de una solución óptima	14
4.3. Describiendo el algoritmo para calcular el costo de una solución óptima	15
4.4. Describiendo el algoritmo para calcular una solución óptima	15
4.5. Complejidad	16
5. Comparación de resultados y conclusiones	18
5.1. Metodología	18
5.2. Resultados	18
5.3. Conclusiones	20

1. Resumen

Este proyecto aborda el problema del conflicto interno de opiniones en redes sociales, en el cual se busca reducir la inconsistencia de las opiniones expresadas por los agentes. Para resolverlo, se implementaron tres enfoques: fuerza bruta, algoritmo voraz y programación dinámica, y se evaluaron en términos de correctitud, eficiencia y optimalidad.

Una red social \mathcal{RS} es una pareja $\langle SAG, R_{max} \rangle$, donde SAG es una secuencia de n grupo de agentes $SAG = \langle sa_0, \dots, sa_{n-1} \rangle$, y $R_{max} \geq 0 \in \mathbb{N}$ representa el valor entero máximo con que se cuenta para moderar las opiniones de la red \mathcal{RS} .

Un grupo de agentes sa_i es una tupla $\langle n_i^{\mathcal{RS}}, o_{i,1}^{\mathcal{RS}}, o_{i,2}^{\mathcal{RS}}, r_i^{\mathcal{RS}} \rangle$, donde $n_i^{\mathcal{RS}}$ representa el número de agentes que pertenecen al grupo de agentes i , $o_{i,1}^{\mathcal{RS}}$ representa la opinión de cada uno de los agentes del

grupo de agentes i de la red \mathcal{RS} sobre la afirmación 1, $o_{i,2}^{\mathcal{RS}}$ representa la opinión de cada uno de los agentes del grupo de agentes i de la red \mathcal{RS} sobre la afirmación 2, y $r_i^{\mathcal{RS}}$ representa el nivel de rigidez de cada agente del grupo de agentes i de la red \mathcal{RS} para $0 \leq i < n$. Es decir, cada miembro del grupo de agentes i tiene la misma opinión sobre la afirmación 1, sobre la afirmación 2 y el mismo nivel de rigidez.

El valor del conflicto interno de una red \mathcal{RS} se puede definir de la siguiente manera:

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

Una estrategia de cambio de opinión de una red \mathcal{RS} de n grupos de agentes, es una secuencia $E = \langle e_0, e_1, \dots, e_{n-1} \rangle$ donde $e_i \in \{0, 1, 2, \dots, n_i^{\mathcal{RS}}\}$; e_i indica el número de agentes del grupo de agentes i a los cuales se le ha cambiado su opinión por medio de E .

Aplicar una estrategia de cambio de opinión E a una red \mathcal{RS} de n grupos de agentes, denotado $ModCI(\mathcal{RS}, E)$ da como resultado una nueva red \mathcal{RS}' con la misma estructura de la red \mathcal{RS} , pero donde la opinión de los agentes cuya opinión ha sido cambiada ahora tienen el mismo valor de opinión para ambas afirmaciones, y la de los otros agentes no ha cambiado. Es decir: si $ModCI(\mathcal{RS}, E) = \mathcal{RS}'$ entonces

$$n_i^{\mathcal{RS}'} = \begin{cases} n_i^{\mathcal{RS}} - e_i & \text{si } e_i > 0 \\ n_i^{\mathcal{RS}} & \text{si } e_i = 0 \end{cases}$$

El valor del esfuerzo de ajustar las opiniones sobre la red \mathcal{RS} con la estrategia E se define de la siguiente forma:

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

Una estrategia de cambio de opinión E sobre la red \mathcal{RS} es aplicable si y solo si el esfuerzo de moderar la red según dicha estrategia es menor o igual a R_{max} , el valor máximo con que se cuenta para cambiar las opiniones de la red \mathcal{RS} .

El problema de la moderación o cambio de opiniones para minimizar el conflicto interno sobre una red social, $ModCI$, se define entonces así:

Entrada: Una red social $\mathcal{RS} = \langle SAG, R_{max} \rangle$

Salida: Una estrategia de cambio de opinión E aplicable para la red social \mathcal{RS} (es decir tal que $Esfuerzo(\mathcal{RS}, E) \leq R_{max}$), tal que $CI(ModCI(\mathcal{RS}, E))$ sea mínimo.

2. Aplicando fuerza bruta

El enfoque de **fuerza bruta** para resolver el problema de *Moderación del Conflicto Interno de Opiniones en una Red Social (ModCI)* consiste en **explorar exhaustivamente** el espacio de estrategias posibles. Este algoritmo evalúa todas las combinaciones viables de decisiones que implican moderar las opiniones de ciertos grupos de usuarios y escoge la combinación que minimiza el **conflicto interno (CI)**, respetando un **presupuesto máximo de esfuerzo** permitido.

2.1. Complejidad

El orden de complejidad del algoritmo de fuerza bruta en el proyecto en un principio se podría pensar que es de $O(k^n)$ ya que el algoritmo recorre todas las posibles combinaciones de estrategias de moderación. Sin embargo, se debe tener en cuenta que, para cada una de las estrategias que se generan, el algoritmo realiza dos cálculos adicionales: el esfuerzo y el conflicto interno, estos cálculos hacen que incremente la complejidad, ya que, en el peor de los casos, se tendrá que ejecutar la función de esfuerzo y el conflicto interno en cada solución (estrategia) posible.

Ambas funciones (**calcularEsfuerzo** y **calcularConflictoInterno**) tienen un comportamiento lineal $O(n)$, ya que cada una de las funciones recorre todos los grupos. Esto añade un costo lineal por cada estrategia evaluada. Por lo tanto, la complejidad total del algoritmo **modciFB** es $O(k^n \cdot n)$ donde k es el número de agentes por grupo (o número de decisiones por grupo) y n la cantidad de grupos.

Este orden de complejidad refleja que el algoritmo es exponencial debido a la exploración de todas las combinaciones posibles de estrategias, pero también incluye un costo adicional de evaluar el esfuerzo y el conflicto interno para cada una de estas estrategias.

El algoritmo sólo necesita almacenar la mejor estrategia encontrada y sus características (esfuerzo y CI), por lo tanto, el espacio es **polinomial**, o más precisamente $O(n)$.

2.2. Corrección

El algoritmo de fuerza bruta se caracteriza por su enfoque exhaustivo: recorre todo el espacio de soluciones posibles, evaluando cada combinación de niveles de moderación aplicables a los grupos de agentes. Para cada estrategia generada, el algoritmo calcula el esfuerzo total requerido y verifica si este no supera el presupuesto máximo definido por el problema. Solo en caso de que la estrategia cumpla con esta restricción, se calcula el conflicto interno resultante. De esta manera, se garantiza que todas las estrategias consideradas son válidas y se descartan aquellas que no cumplen las condiciones impuestas.

Una vez verificada la validez de cada estrategia, el algoritmo compara su conflicto interno con el mínimo registrado hasta ese momento. Si la nueva estrategia mejora el valor mínimo, esta se almacena como la mejor solución encontrada hasta el momento. Este proceso se repite para todas las estrategias viables, asegurando así que **ninguna combinación con potencial de ser óptima sea pasada por alto**. Al no depender de decisiones locales o heurísticas, el algoritmo no introduce sesgos que puedan conducir a soluciones subóptimas.

Por lo tanto, el algoritmo de fuerza bruta es **correcto por definición**, ya que siempre retorna la estrategia con el menor conflicto interno entre todas las que respetan el presupuesto disponible. Su principal desventaja radica en la alta complejidad temporal, que crece exponencialmente con el número de grupos, lo que limita su aplicabilidad a redes pequeñas. Sin embargo, esta misma característica lo convierte en una herramienta valiosa para validar la calidad de otras estrategias más eficientes, como el enfoque voraz o la programación dinámica.

3. Aplicando un algoritmo voraz

El algoritmo voraz diseñado para el problema **ModciV** tiene como propósito encontrar una estrategia eficiente para moderar el conflicto interno de una red social, dado un presupuesto de esfuerzo limitado. A diferencia de la fuerza bruta, este algoritmo no explora todas las posibles combinaciones de moderación, sino que **toma decisiones locales** priorizando los grupos que ofrecen mayor reducción del conflicto por cada unidad de esfuerzo invertido. Esto permite obtener soluciones en tiempo muy reducido, aunque no siempre sean óptimas.

3.1. Describiendo el algoritmo

El primer paso del algoritmo consiste en calcular una medida de **ganancia relativa** para cada grupo. Esta ganancia se define como la relación entre el beneficio que se obtendría al moderar al grupo y el costo necesario para hacerlo. En el código, esto se implementa en el bloque que recorre la lista de grupos y construye una lista de tuplas de la forma (ganancia, índice, beneficio, costo). El beneficio se calcula como el producto entre la diferencia absoluta de opiniones del grupo y su rigidez, y el costo es el esfuerzo asociado, calculado con la función `math.ceil(...)`.

```
# Calcular Ganancia
ganancias = []
i = 0
while i < len(red_social.grupos):
    grupo = red_social.grupos[i]

    if grupo.n > 0:
        beneficio = abs(grupo.op1 - grupo.op2) * grupo.rig
        costo = math.ceil(beneficio)

        if costo > 0:
            ganancias.append((beneficio / costo, i, beneficio, costo))
    i += 1
```

Una vez recopiladas las ganancias de todos los grupos, el algoritmo las ordena en orden descendente de eficiencia (ganancia por unidad de esfuerzo). Este orden define la prioridad con la que se consideran los grupos para ser moderados, lo cual está directamente ligado a la idea central del enfoque voraz.

A continuación, el algoritmo recorre esta lista ordenada y para cada grupo intenta asignar un número de moderaciones k , desde 1 hasta n (la cantidad de agentes en el grupo). Para cada posible k , se estima el esfuerzo total adicional que implicaría intervenir al grupo en ese nivel, y si aún cabe dentro del presupuesto, se actualiza la estrategia y se incrementa el esfuerzo total.

```
for k in range(1, grupo.n + 1):
    incremento_esfuerzo = math.ceil(abs(grupo.op1 - grupo.op2) * grupo.rig * k)

    if esfuerzo_total + incremento_esfuerzo <= red_social.R_max:
        estrategia[i] = k
        esfuerzo_total += incremento_esfuerzo
    else:
        break
```

Aquí se observa cómo la estrategia asigna a cada grupo una cantidad k de “intervenciones” mientras no se exceda el presupuesto. Una vez no es posible seguir asignando unidades de moderación a un grupo, se continúa con el siguiente en la lista de eficiencia.

Finalmente, la estrategia completa es aplicada a la red original mediante la función ***red_social.aplicar_estrategia***, la cual devuelve una nueva red con opiniones modificadas. Se calcula el conflicto interno de esta nueva red para evaluar la efectividad de la solución hallada.

```
nueva_red = red_social.aplicar_estrategia(estrategia)

conflicto = nueva_red.calcular_conflicto_interno()

return (estrategia, esfuerzo_total, conflicto)
```

En resumen, el algoritmo voraz implementado toma decisiones rápidas y eficientes con base en métricas locales. Aunque no garantiza optimalidad global, su diseño balancea **calidad de solución y velocidad**, lo que lo hace ideal para redes grandes o escenarios donde el tiempo de cómputo es limitado.

3.2. Entendiendo el algoritmo

Para comprender mejor el funcionamiento del algoritmo voraz, se realizaron una serie de pruebas sobre diferentes configuraciones de redes sociales, enfocadas en evaluar si la solución entregada por este enfoque corresponde o no con la solución óptima. En primer lugar, se analizaron los resultados obtenidos con dos ejemplos básicos (equivalentes a las secciones 2.3.1 y 2.3.2 del enunciado del proyecto), seguidos de una batería de pruebas más extensas, con mayor número de grupos de agentes.



Ejemplo 1

Se definió una red social con tres grupos:

$$RS1 = (\langle 3, -100, 50, 0.5 \rangle, \langle 1, 100, 80, 0.1 \rangle, \langle 1, -10, 0, 0.5 \rangle, R_{\max}=80)$$

En esta red, el grupo 1 tiene un alto impacto en el conflicto debido a sus opiniones opuestas y tamaño, pero también tiene un costo elevado por su rigidez. Los grupos 2 y 3, aunque pequeños, tienen menor rigidez y por tanto requieren menos esfuerzo para ser moderados.

Se evaluaron dos estrategias de moderación:

Estrategia	Vector	Grupos moderados	Esfuerzo total	CI resultante	¿Óptima ?
E_1	$\langle 0, 1, 1 \rangle$	Grupo 2, Grupo 3	7	22,500	
E_2	$\langle 1, 0, 0 \rangle$	Grupo 1	75	15,166.66	

Aunque E_1 tiene menor costo, su impacto sobre el conflicto es muy limitado. El algoritmo voraz, al evaluar la eficiencia individual de los grupos, probablemente seleccione E_1 . Sin embargo, E_2 , aunque más costosa, ofrece una mejora global mucho mayor en términos de conflicto interno, lo que demuestra que el algoritmo voraz puede ser subóptimo.

Ejemplo 2

Se analizó una segunda red definida por:

$$RS2 = (\langle 1, -30, 0, 0.9 \rangle, \langle 1, 40, 0, 0.1 \rangle, \langle 1, 50, 0, 0.5 \rangle, R_{\max}=35)$$

Aquí, el grupo 1 tiene alta receptividad, el grupo 2 es muy difícil de moderar, y el grupo 3 tiene rigidez media.

Estrategia	Vector	Grupos moderados	Esfuerzo total	CI resultante	¿Óptima?
E ₃	⟨1, 0, 1⟩	Grupo 1, Grupo 3	28	13.333	✓

Resultados de pruebas adicionales

Se analizaron cuatro entradas adicionales con redes más grandes. La tabla siguiente resume los resultados:

Prueba	Esfuerzo total	CI Voraz	¿Óptima?
1	2563	0.0	✓
2	387	34222.4	✗
3	365	13599.2	✗
4	295	20423.6	✗
5	450	27426.2	✗

3.3. Complejidad

El algoritmo voraz utilizado para resolver el problema de moderación del conflicto interno (**ModciV**) fue diseñado con el objetivo de seleccionar los grupos de agentes más eficientes —en términos de reducción del conflicto por unidad de esfuerzo— bajo una restricción de presupuesto. Su estructura y lógica permiten realizar una evaluación de su complejidad computacional de manera clara y precisa.

El algoritmo se ejecuta en los siguientes pasos principales:

1. Cálculo de la eficiencia de cada grupo:
Para cada grupo de la red social, se calcula un valor de “ganancia” o “eficiencia”, definido como la relación entre el beneficio obtenido al moderar ese grupo (reducción del conflicto interno) y el costo en esfuerzo que dicha acción requiere. Este paso requiere recorrer todos los n grupos de la red, realizando operaciones aritméticas constantes para cada uno. Por tanto, esta etapa tiene una complejidad de $O(n)$
2. Ordenamiento de grupos según eficiencia:
Una vez obtenidas las ganancias, el algoritmo ordena la lista de grupos de forma descendente según su eficiencia. Este ordenamiento se realiza con un algoritmo típico de comparación, como `sort()` en Python, que tiene una complejidad de $O(n \log n)$
3. Selección de grupos dentro del presupuesto:
Después del ordenamiento, se recorre la lista de grupos ya ordenada para decidir cuántos agentes de cada grupo se pueden moderar sin superar el presupuesto total. En el peor caso, esto implica nuevamente recorrer todos los grupos una vez, asignando cantidades de agentes hasta agotar el presupuesto, lo que implica una complejidad lineal adicional: $O(n)$.

En consecuencia, el orden de complejidad total del algoritmo voraz es:

$$O(n \log n)$$

Este es un resultado muy favorable desde el punto de vista computacional. Contrasta fuertemente con enfoques exactos como fuerza bruta ($O(m^n)$ exponencial) o programación dinámica (polinomial, pero más costosa), y lo convierte en una opción ideal para redes grandes donde se requiere una solución razonablemente buena en un tiempo limitado.

Cabe destacar que esta eficiencia se logra sacrificando optimalidad. El algoritmo, al basarse en decisiones locales y no considerar combinaciones de grupos con sinergias globales, puede omitir estrategias superiores. Sin embargo, su bajo costo computacional y facilidad de implementación lo hacen muy atractivo para entornos prácticos.

3.4. Corrección

El algoritmo voraz diseñado para el problema ModCI **no garantiza encontrar la solución óptima** en todos los casos. Aunque siempre produce una estrategia válida —es decir, una solución que no supera el presupuesto y reduce el conflicto interno en cierta medida—, su naturaleza local lo limita al momento de explorar el espacio completo de soluciones posibles. Esto se debe a que toma decisiones basadas únicamente en la eficiencia individual de cada grupo (beneficio por unidad de esfuerzo), sin considerar combinaciones de grupos que podrían generar una mejor solución global.

El algoritmo es **correcto desde el punto de vista funcional**, ya que cumple con las restricciones del problema: respeta el presupuesto de esfuerzo máximo permitido y devuelve un vector de moderación aplicable. Sin embargo, desde la perspectiva de **optimalidad**, no es correcto en todos los casos. Existen situaciones donde una combinación de grupos menos eficientes individualmente puede producir un mayor impacto en la reducción del conflicto interno, pero esta posibilidad es ignorada por el algoritmo voraz porque su lógica no permite retroceder ni reordenar decisiones.

Por ejemplo, en el **caso de prueba 2.3.1**, el algoritmo eligió moderar dos grupos pequeños de baja rigidez (Grupos 2 y 3), ya que requerían poco esfuerzo y ofrecían cierta ganancia. No obstante, la solución óptima consistía en moderar solo al Grupo 1, que tenía un alto impacto sobre el conflicto, aunque su costo era mayor. En este caso, la solución devuelta por el algoritmo fue válida, pero claramente subóptima.

En contraste, hay casos en los que el algoritmo **sí encuentra la solución óptima**. Esto suele ocurrir cuando:

- El grupo más eficiente también tiene el mayor impacto.
- El presupuesto es muy ajustado y no permite muchas combinaciones.
- Los grupos presentan una clara diferencia de eficiencia y el mejor camino es evidente.

Un ejemplo de esto fue el caso 2.3.2, donde el algoritmo seleccionó correctamente los grupos 1 y 3, encontrando la misma solución óptima obtenida por programación dinámica.

En resumen, el algoritmo voraz **no siempre da la respuesta correcta** en términos de optimalidad, pero su salida siempre es válida y útil en la práctica. Es más confiable en redes grandes, donde las decisiones locales tienden a alinearse con el comportamiento global, y menos preciso en redes pequeñas o con estructuras complejas donde las combinaciones sutiles de grupos pueden tener un efecto decisivo. Por eso, se recomienda usar este enfoque como una solución rápida o de aproximación, y validar sus resultados mediante métodos exactos si se requiere precisión total.

4. Aplicando programación dinámica

El problema puede descomponerse en subproblemas más pequeños, cuyos resultados pueden almacenarse y reutilizarse para construir la solución del problema original. Para aplicar programación dinámica de forma efectiva, es indispensable que el problema cumpla dos propiedades esenciales: **subestructura óptima** y **superposición de subproblemas**.

4.1. Caracterizando la estructura de una solución óptima

Una solución óptima al problema ModCI consiste en encontrar un subconjunto de grupos de la red social a los cuales se debe aplicar moderación, de manera que se **minimice el conflicto interno total** sin que el esfuerzo necesario supere un valor máximo permitido R_{max} . Esta estructura refleja una elección binaria por grupo: o se modera o no se modera, considerando siempre el impacto de esa acción en la reducción del conflicto.

Este tipo de decisión tiene una estructura muy similar a la del problema clásico de la mochila 0/1 (0-1 knapsack), en el que se deben seleccionar ítems con valores y pesos dados, maximizando el valor total sin superar la capacidad de la mochila. En nuestro caso, el "valor" está invertido: en lugar de maximizar una ganancia, queremos **minimizar el conflicto**, y el "peso" corresponde al esfuerzo requerido para moderar a cada grupo.

La estructura óptima de la solución se puede caracterizar como sigue:

- Dado un conjunto de n grupos y un presupuesto máximo R_{max} , la solución óptima puede construirse **a partir de las soluciones óptimas a subproblemas más pequeños**, es decir, al problema de seleccionar la mejor estrategia para los primeros i grupos con un presupuesto menor o igual a j .
- Para cada grupo i , se presentan dos alternativas:
 - o No moderar al grupo i , en cuyo caso la solución es igual a la de los primeros $i-1$ grupos con el mismo presupuesto j .
 - o Moderar al grupo i , lo cual tiene un costo de esfuerzo e_i y reduce el conflicto en una cantidad c_i ; en este caso, la solución óptima se construye tomando la solución del subproblema $(i-1, j-e_i)$ y agregando

el efecto del grupo i .

Conjunto de subproblemas a resolver

La programación dinámica construye una tabla de soluciones para todos los subproblemas definidos por los pares (i,j) , donde:

- $i \in \{0,1,\dots,n\}$ representa la cantidad de grupos considerados hasta el momento.
- $j \in \{0,1,\dots,R_{\max}\}$ representa el presupuesto disponible en ese punto.

Por tanto, el **número total de subproblemas** a resolver es:

$$(n+1) \cdot (R_{\max}+1)$$

Este número representa las dimensiones de la tabla DP, donde cada celda contiene la solución óptima al subproblema correspondiente.

Ejemplo:

Si se tienen $n=100$ grupos y un presupuesto de $R_{\max}=500R$, entonces el número total de subproblemas es:

$$(100+1) \cdot (500+1) = 101 \cdot 501 = 50,601$$

Cada uno de estos subproblemas se resuelve de forma iterativa, utilizando resultados ya calculados en pasos anteriores. Esto permite garantizar que el algoritmo encuentre la **mejor solución posible** en un tiempo razonable, siempre que n y R_{\max} no sean demasiado grandes.

4.2. Definiendo recursivamente el valor de una solución óptima

Como se estableció anteriormente, el problema ModCI puede abordarse mediante programación dinámica al descomponerlo en subproblemas definidos por los pares (i,j) , donde:

- i representa los primeros i grupos considerados,
- j representa un presupuesto parcial disponible ($0 \leq j \leq R_{\max}$).

Nuestro objetivo es minimizar el **conflicto interno total** generado por los grupos no moderados, mientras se respeta el presupuesto de esfuerzo R_{\max} . Para ello, definimos la función: $CI(i,j)$ = mínimo conflicto interno alcanzable considerando los primeros i grupos con presupuesto j

Relación de recurrencia

La recurrencia se define como sigue:

$$CI(i, j) = \begin{cases} 0, & \text{si } i = 0 \text{ (sin grupos)} \\ CI(i-1, j), & \text{si } e_i > j \text{ (no alcanza el presupuesto para moderar al grupo } i) \\ \min(CI(i-1, j), CI(i-1, j - e_i) + c_i), & \text{si } e_i \leq j \end{cases}$$

donde:

- **ei** es el esfuerzo necesario para moderar al grupo *i*,
- **ci** es la **reducción de conflicto interno** que se consigue al moderar al grupo *i* (es decir, el beneficio).
- **CI(i-1,j)** representa la solución óptima sin moderar al grupo *i*.
- **CI(i-1,j-ei)+ci** representa la solución si sí se modera al grupo *i*.

Este enfoque permite decidir, en cada paso, si conviene o no moderar a un grupo determinado. Se toma la opción que produce **el menor conflicto interno acumulado**, garantizando así la optimalidad del resultado global.

Caso base:

$$CI(0, j) = 0, \quad \forall j \in [0, R_{\max}]$$

Esto refleja que, si no se considera ningún grupo, el conflicto interno es nulo (no hay nada que moderar ni que calcular).

Construcción de la tabla

Se construye una tabla DP de tamaño $(n+1) \times (R_{\max}+1)$, en la que cada celda $DP[i][j]$ contiene el valor de $CI(i, j)$. Esta tabla se llena de forma iterativa, fila por fila, utilizando la recurrencia descrita.

Este modelo garantiza que cada subproblema es evaluado **una sola vez**, aprovechando los resultados ya calculados para construir soluciones más grandes, lo que convierte a la programación dinámica en una herramienta eficiente y exacta para resolver ModCI.

4.3. Describiendo el algoritmo para calcular el costo de una solución óptima

A partir de la relación de recurrencia definida anteriormente, se puede construir un algoritmo eficiente de **programación dinámica** para resolver el problema ModCI y calcular el **costo mínimo (conflicto interno) de una solución óptima** bajo una restricción de presupuesto.

El algoritmo se basa en construir una tabla bidimensional DP de dimensiones $(n+1) \times (R_{\max}+1)$, donde n es el número de grupos y R_{\max} el presupuesto máximo disponible. Cada celda $DP[i][j]$ contiene el **conflicto interno mínimo** posible considerando los primeros i grupos con un presupuesto de j .

Estructura del algoritmo

1. Inicialización:

Se inicializa la primera fila de la tabla DP con ceros:

$$DP[0][j] = 0, \quad \forall j \in [0, R_{\max}]$$

Esto refleja que si no se considera ningún grupo, el conflicto interno es cero sin importar el presupuesto.

2. Iteración sobre los grupos:

Para cada grupo $i \in [1, n]$ y para cada presupuesto $j \in [0, R_{\max}]$, se calcula el mínimo conflicto aplicando la recurrencia:

- Si el esfuerzo e_i del grupo i es mayor que j , entonces no se puede moderar ese grupo:

$$DP[i][j] = DP[i-1][j]$$

- Si el esfuerzo es suficiente, se elige la mejor entre:
 - o No incluir el grupo i ,
 - o Incluir el grupo i y sumar su impacto sobre el conflicto:

$$DP[i][j] = \min(DP[i-1][j], DP[i-1][j-e_i] + c_i)$$

3. Resultado final:

Una vez completada la tabla, el valor de $DP[n][R_{\max}]$ representa el **conflicto interno mínimo alcanzable** utilizando cualquier combinación de grupos sin exceder el presupuesto.

Para cada j en $0 \dots R_max$:

$$DP[0][j] = 0$$

Para i en $1 \dots n$:

Para j en $0 \dots R_max$:

si $esfuerzo[i] > j$:

$$DP[i][j] = DP[i-1][j]$$

sino:

$$DP[i][j] = \min(DP[i-1][j], DP[i-1][j - esfuerzo[i]] + conflicto[i])$$

Retornar $DP[n][R_max]$

Este algoritmo garantiza encontrar la mejor estrategia posible, minimizando el conflicto interno total y cumpliendo con la restricción presupuestaria. Su eficiencia es polinomial en función del número de grupos y el presupuesto máximo, lo que lo hace práctico para redes de tamaño moderado.

4.4. Describiendo el algoritmo para calcular una solución óptima

En la sección anterior se explicó cómo calcular, mediante programación dinámica, el valor mínimo del conflicto interno que se puede alcanzar sin superar el presupuesto de esfuerzo disponible. Sin embargo, conocer únicamente este valor **no es suficiente para resolver el problema completo**. También es necesario **recuperar la estrategia de moderación óptima**, es decir, determinar **qué grupos se deben moderar** para alcanzar ese valor mínimo.

Para lograrlo, se debe extender el algoritmo anterior con una **estructura adicional de seguimiento (tracking)** que permita reconstruir la solución óptima a partir de los datos almacenados en la tabla DP.

Estructura auxiliar

Se crea una matriz $decision[i][j]$ del mismo tamaño que la matriz $DP[i][j]$, donde se almacena, para cada subproblema:

- 1 si el grupo i fue incluido en la solución óptima cuando se tenía un presupuesto j .
- 0 si no fue incluido.

Esta estructura permite **retroceder** desde la solución final hacia los subproblemas que la generaron y así determinar **exactamente qué grupos fueron seleccionados**.

Algoritmo completo (valor + reconstrucción)

Paso 1: Calcular el valor óptimo (como en el punto anterior):

Se construyen las matrices DP y $decision$, llenando cada celda con base en:

si $esfuerzo[i] > j$:

$$DP[i][j] = DP[i-1][j]$$

$$decision[i][j] = 0$$

sino:

$$sin_incluir = DP[i-1][j]$$

$$con_incluir = DP[i-1][j] - esfuerzo[i] + conflicto[i]$$

si $con_incluir < sin_incluir$:

$$DP[i][j] = con_incluir$$

$$decision[i][j] = 1$$

sino:

$$DP[i][j] = sin_incluir$$

$$decision[i][j] = 0$$

Paso 2: Reconstruir la solución óptima

Una vez llenas las tablas, se recorre decision **desde abajo hacia arriba**, empezando por el estado final (n, R_{max}) En cada paso:

- Si $decision[i][j] == 1$, significa que el grupo i fue moderado. Se lo incluye en la estrategia, y se resta su esfuerzo del presupuesto restante: $j := j - e_i$.
- Si $decision[i][j] == 0$, se continúa con el siguiente grupo sin restar nada.

Este proceso se repite hasta llegar a $i=0$.

Inicializar estrategia óptima $E = [0] * n$

$i = n$

$j = R_{max}$

Mientras $i > 0$:

 si $decision[i][j] == 1$:

$E[i-1] = 1$

$j = j - esfuerzo[i]$

$i = i - 1$

Al finalizar, el vector E contendrá la estrategia óptima de moderación: un 1 en la posición i indica que el grupo i fue moderado, y un 0 que no lo fue.

Resultado final

El algoritmo completo permite:

- Calcular el **conflicto interno mínimo** que se puede alcanzar sin exceder el presupuesto.
- Determinar **qué grupos deben ser moderados** para lograr ese valor mínimo.

Esto asegura que se encuentra una solución **correcta y óptima** para el problema, tanto en valor como en decisión.

4.5. Complejidad

Complejidad en tiempo

El algoritmo de programación dinámica resuelve el problema ModCI mediante la construcción de una matriz dp de tamaño $(n + 1) * (R_{max} + 1)$, donde n es el número total de grupos en la red social y R_{max} es el límite de esfuerzo permitido para aplicar moderación. Cada celda $dp[i][r]$, representa el conflicto mínimo que se

puede obtener considerando los primeros i grupos y utilizando exactamente r unidades de esfuerzo; el número de operaciones está acotado por $n_i + 1$, donde n_i es el número de agentes en ese grupo.

En el peor caso, todos los grupos tienen m agentes, por lo que la complejidad temporal total sería:

$$O(n * R_{max} * m)$$

Donde:

- n : número de grupos.
- R_{max} : presupuesto máximo de esfuerzo.
- m : máximo número de agentes en un grupo.

Complejidad en espacio

El algoritmo utiliza dos matrices de tamaño $(n + 1) * (R_{max} + 1)$ una para almacenar los valores de conflicto (dp) y otra para decisiones ($decision$). Por lo tanto, el espacio requerido es:

$$O(n * R_{max})$$

Cada celda almacena un número, por lo que el uso de memoria es lineal respecto a los parámetros del problema.

¿Es útil para la práctica?

El número máximo de operaciones que puede ejecutar el equipo en un año es:

$$518400 * 3 * 10^8 = 1.5552 * 10^{14}$$

Si suponemos que $n = 2^k$ y que $R_{max} \sim n$, entonces la complejidad en tiempo se convierte en:

$$O(2^{2k} * m)$$

En memoria sería:

$$O(2^{2k})$$

Entonces:

- Tiempo:

$$2^{2k} * m \leq 1.5552 * 10^{14} \Rightarrow 2k \leq \log_2\left(\frac{1.5552 * 10^{14}}{m}\right)$$

Para $m = 100$, se obtiene aproximadamente $k \leq 20.25$ o $k \leq 20$

- Espacio:

Se coloca como ejemplo de uso $16 \text{ GB} = 1.6 * 10^{10} \text{ bytes}$

Entonces:

$$4 * 2^{2k} \leq 1.6 * 10^{10} \Rightarrow k \leq 15.94867...$$

o se aproxima a:

$$k \leq 16$$

5. Comparación de resultados y conclusiones

5.1. Metodología

La metodología utilizada para comparar la eficiencia de los algoritmos fue haciendo las 30 pruebas dadas a cada algoritmo y tomando los tiempos de ejecución de cada uno, por lo cual, en el archivo **main.py** donde esta la interfaz gráfica; se utilizó el módulo de python **time** que nos da el tiempo de ejecución de un código. Entonces mostramos los resultados de cada prueba realizada en la interfaz gráfica con su tiempo de ejecución, con estos datos podemos analizar qué algoritmo es más eficiente en buscar la solución óptima (Excepto la voraz).

5.2. Resultados

Nota: Las casillas que contienen “-” significan que la prueba no se pudo realizar por tiempo, es decir, a cada prueba se le dio un tiempo máximo de 15 horas de ejecución, por lo tanto, si la prueba excede ese límite de tiempo se detiene la prueba y no se toma el registro de tiempo.

Tabla 1. Comparación de tiempos de las ejecuciones de cada prueba.

# Prueba	N° Grupos de Agentes	Tiempo modciFB (seg)	Tiempo modciPD (seg)	Tiempo modciV (seg)
1	5	0,005916	0,029199	0,000027
2	5	0,011775	0,004004	0,000027
3	5	0,011831	0,009738	0,000033
4	5	0,002525	0,004296	0,000024
5	5	0,008696	0,003751	0,000029
6	10	0,011372	0,000982	0,000027
7	10	22,957926	0,013824	0,000030
8	10	294,670569	0,019739	0,000030
9	10	0,010535	0,001008	0,000028
10	10	10465,462306	0,032004	0,000034
11	15	-	0,021765	0,000034
12	15	3,481793	0,000000	0,000032
13	15	7432,732735	0,005030	0,000034
14	15	-	0,073903	0,000033
15	15	40504,013876	0,030985	0,000032

16	20	-	0,136959	0,000040
17	20	-	0,123082	0,000040
18	25	-	0,046406	0,000042
19	25	-	0,267721	0,000044
20	25	-	0,088301	0,000041
21	50	-	0,620097	0,000157
22	50	-	0,386028	0,000063
23	50	-	1,735655	0,000069
24	50	-	1,415222	0,000088
25	50	-	0,844678	0,000065
26	50	-	0,168023	0,000061
27	50	-	0,725663	0,000073
28	100	-	0,799576	0,000126
29	100	-	7,661631	0,000126
30	100	-	42,375946	0,000137

5.3. Conclusiones

En este proyecto podemos observar y comprobar tanto teóricamente como gráficamente que cada algoritmo tiene sus ventajas y sus desventajas, mediante las pruebas realizadas.

En las pruebas realizadas podemos ver que cuando los grupos de agentes son reducidos (pruebas 1 - 5) los algoritmos trabajan de forma muy eficiente en los tiempos de ejecución. Pero a medida que los grupos van aumentando los algoritmos demuestran un aumento significativo de la métrica del tiempo.

Esto lo podemos observar más precisamente en el algoritmo de fuerza bruta, donde más grande es el grupo de agente, más tiempo se toma en la ejecución, pero no solo afecta el número de agentes en la fuerza bruta, sino que también afecta el R_{max} ya que en las pruebas pudimos observar que si el grupo de agentes es grande pero el R_{max} es pequeño su tiempo de ejecución es reducción (pruebas 6-10), esto se da porque si el R_{max} es reducido la poda que hace el algoritmo de fuerza bruta es más efectiva, reduciendo significativamente el número de estrategias exploradas y mejorando enormemente el rendimiento del algoritmo.

En la programación dinámica se pudo observar un balance favorable entre exactitud y eficiencia. Esto logró encontrar soluciones óptimas con tiempos bajos incluso al aumentar el tamaño de agentes. Aunque en las últimas dos pruebas se aumentó el tiempo de ejecución, siguió siendo considerablemente bajo, demostrando que la programación dinámica es certera.

El algoritmo voraz aunque no garantiza una solución óptima, demostró ser eficiente en términos de tiempo. El tiempo de ejecución fue considerablemente constante incluso con un número grande de agentes. Sin embargo, se observa que en la mayoría de los casos, la solución encontrada por el algoritmo voraz no coincidía con la óptima, esto se puede deber a que el algoritmo voraz escoge siempre la primera solución aunque no sea la más óptima.

Con todo lo anterior se concluye que, este proyecto permitió evidenciar las ventajas y limitaciones de cada algoritmo (fuerza bruta, voraz y dinámica), y resalta la importancia de elegir el método más adecuado dependiendo de las necesidades específicas del problema.