.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1

Fecha de entrega: 24/11/2023

- Gauler, Ian Benjamin, Padrón: 109437
- Jauregui, Melina Belén, Padrón: 109524
- Tourne Passarino, Patricio, Padrón: 108725

Índice

1. Introducción

El propósito de este informe radica en llevar a cabo un análisis de los distintos enfoques y diseños aplicados para abordar la problemática presentada por Scaloni. Este desafío se centra en la determinación del conjunto mínimo de jugadores, denotado como C, requerido para satisfacer las demandas de cada medio, asegurando al menos la presencia de un jugador favorito por medio. Es importante destacar que este problema se enmarca como un caso específico del 'Hitting Set Problem'. En consecuencia, se realizará una evaluación detallada de múltiples soluciones con el fin de resolver este último. A lo largo de este informe, se expondrán y analizarán los distintos criterios empleados para abordar esta cuestión, proporcionando una justificación fundamentada en el análisis teórico y práctico de las soluciones propuestas.

1.1. Información de la problemática

El enunciado propuesto, nos da la siguiente información:

- Scaloni tiene a su disposición el conjunto A de $n = 43$ jugadores $a_1, a_2, \ldots, a_{43}$.
- Existen m medios, cada uno con un grupo de jugadores favorito $B_1, B_2, \ldots, B_m$, ($B_i \in A \forall i$)
- Se quiere el subconjunto $C \in A$ de menor tamaño tal que C tenga al menos un elemento de cada B_i (es decir, $C \cap B_i$)
- Scaloni necesita obtener el grupo C más pequeño de jugadores de tal forma que cada medio B_i tenga al menos un jugador en el equipo.

2. Análisis de la complejidad del problema

El problema planteado por Scaloni es un caso específico del problema del Hitting-Set, cuya definición formal se presenta de la siguiente manera: dado un conjunto A compuesto por n elementos y m subconjuntos $B_1, B_2, \ldots, B_m$ pertenecientes a A ($B_i \subseteq A \forall i \in \mathbb{N}_m$), se busca encontrar un conjunto $C \subseteq A$ tal que para cada subconjunto C, donde $C \subseteq A / \forall j \in \mathbb{N}_m, C \cap B_j \neq \emptyset$.

Además de su formulación básica, el problema del Hitting-Set también presenta una versión de decisión: dada la colección de un conjunto A con n elementos y m subconjuntos $B_1, B_2, \ldots, B_m$ de A ($B_i \subseteq A$ para cada i), junto con un parámetro numérico k, se plantea el interrogante sobre la existencia de un conjunto $C \subseteq A$ que cumpla con dos condiciones fundamentales:

- En primer lugar, que la cardinalidad de C sea menor o igual a k ($|C| \leq k$)
- En segundo lugar, que para cada subconjunto B_j, la intersección entre C y B_j no sea vacía ($C \cap B_j \neq \emptyset$) para todo j perteneciente al conjunto de números naturales hasta m.

El problema del Hitting-Set se sitúa en la clase de complejidad NP debido a su capacidad de verificación en tiempo polinomial, lo que implica una verificación eficiente de su solución propuesta. La verificación de la solución se reduce a confirmar dos condiciones fundamentales: Primero, se debe comprobar que la cardinalidad del conjunto propuesto C es menor o igual a k, donde k es un parámetro dado. Segundo, es necesario verificar que para cada conjunto B_j dentro de una colección de subconjuntos $B_1, B_2, \ldots, B_m$, C contenga al menos un elemento de B_j.

Para realizar esta verificación, se llevan a cabo dos operaciones clave que definen la complejidad del proceso. La primera operación, relacionada con la verificación de la cardinalidad de C ($|C| \leq k$), tiene una complejidad constante $O(1)$, ya que implica simplemente obtener el número de elementos en C y verificar que $|C| \leq k$.

La segunda operación, que implica verificar la inclusión de al menos un elemento de C en cada conjunto B_j, presenta una complejidad de $O(n \times m)$. Esto se debe a que para cada conjunto B_j (j perteneciente al conjunto de números naturales hasta m) (operación $O(m)$), se debe recorrer, en el peor de los casos, todo el conjunto C ($O(n)$) para verificar la pertenencia de al menos un elemento de este en B_j (operación con complejidad $O(1)$ si tanto B_j como C se implementan como un conjunto 'set').

La clasificación del problema del Hitting-Set como NP-Completo se establece mediante la demostración de su reducibilidad polinómica a partir de otros problemas ya catalogados como NP-Completo.

En nuestro análisis, nos hemos propuesto abordar esta demostración de dos maneras distintas, empleando estrategias de reducción que ilustran la naturaleza NP-Completa del Hitting-Set Problem:

- Reducción de Vertex Cover a Dominating-Set Problem -¿Reducción de Dominating-Set Problem a Hitting-Set Problem.
- Reducción de Set Cover a Hitting-Set Problem.

Es importante destacar que la pertenencia de Vertex Cover y Set Cover a NP-Completo fue demostrada en clases anteriores.

- Vertex Cover [link]
- Set Cover [link]

2.1. Reducción Vertex Cover a Dominating-Set Problem

La reducción Vertex-Cover $\leq_p$ Dominating-Set consta en lo siguiente: Dado del grafo G con n vértices y m aristas del problema Vertex Cover, para cada par de vértices adyacentes $v - w$, se agregan ambos al grafo G' (del problema Dominating Set) junto a su arista y se agrega un tercer vértice auxiliar vw, adyacente a los otros dos. A será el conjunto de vértices auxiliares.

Luego, del conjunto V' de k vértices solución de Dominating Set, se debe agregar a V (solución de Vertex Cover) todos los vértices de V' que no estén en el conjunto de auxiliares y, para cada vértice en $V' \cap A$ se agrega a V cualquiera de sus adyacentes. La primera parte de la reducción es $O(m)$ y la segunda es $O(k), k \leq n$, por lo que la complejidad total es $O(m + k)$, que es polinomial.

$$\text{Vertex-Cover} \leq_p \text{Dominating-Set}$$

Finalmente:

$$\begin{matrix}\text{Vertex-Cover} \leq_p \text{Dominating-Set} \\ \text{Dominating-Set} \leq_p \text{Hitting-Set}\end{matrix} \quad \overset{\text{por transitividad}}{\Longrightarrow} \quad \text{Vertex-Cover} \leq_p \text{Hitting-Set}$$

$$\Longrightarrow \text{Hitting-Set} \in \text{NP-Completo}$$

2.2. Reducción Dominating-Set Problem a Hitting-Set Problem

Recordemos el problema de Dominating Set: Dado un grafo G, se busca un conjunto de vértices C tal que, para todo vértice $v \in G$, este esté contenido en C o existe al menos un vértice en C adyacente de v.

La reducción Dominating-Set $\leq_p$ Hitting-Set consta en lo siguiente: Dado un grafo G de n vértices del problema Dominating Set, para cada vértice v_i, se construye un grupo B_i con el

mismo y todos sus vértices adyacentes. Entonces, el grupo resultado del Hitting Set con los subconjuntos $B_1, B_2, \ldots, B_m$ será también el grupo resultado del Dominating Set. *Nota: Para cada k-clique dentro del grafo habrá hasta k sets iguales.* La complejidad de esta reducción depende de la implementación del grafo: Si los vértices desconocen a sus adyacentes, es $O(n^2)$ porque para cada vértice v_i se debe recorrer todos los vértices de G para verificar si son adyacentes a v_i; en cambio, si los vértices tiene referencia a sus adyacentes, la complejidad es $O(n \times o)$, siendo o el promedio del grado entre vértices, que puede variar entre 0 y m. En ambos casos, se trata de una complejidad polinomial.

$$\text{Dominating-Set} \leq_p \text{Hitting-Set}$$

La reducción Vertex-Cover $\leq_p$ Dominating-Set consta en lo siguiente: Dado del grafo G con n vértices y m aristas del problema Vertex Cover, para cada par de vértices adyacentes $v - w$, se agregan ambos al grafo G' (del problema Dominating Set) junto a su arista y se agrega un tercer vértice auxiliar vw, adyacente a los otros dos. A será el conjunto de vértices auxiliares. Luego, del conjunto V' de k vértices solución de Dominating Set, se debe agregar a V (solución de Vertex Cover) todos los vértices de V' que no estén en el conjunto de auxiliares y, para cada vértice en $V' \cap A$ se agrega a V cualquiera de sus adyacentes. La primera parte de la reducción es $O(m)$ y la segunda es $O(k), k \leq n$, por lo que la complejidad total es $O(m + k)$, que es polinomial.

$$\text{Vertex-Cover} \leq_p \text{Dominating-Set}$$

Finalmente:

$$\begin{matrix}\text{Vertex-Cover} \leq_p \text{Dominating-Set} \\ \text{Dominating-Set} \leq_p \text{Hitting-Set}\end{matrix} \xRightarrow{\text{por transitividad}} \text{Vertex-Cover} \leq_p \text{Hitting-Set}$$

$$\implies \text{Hitting-Set} \in \text{NP-Completo}$$

3. Programación Lineal

3.1. Programación Lineal Entera

También se puede reducir Hitting-Set Problem a Programación Lineal de la siguiente manera:

1. Para cada elemento i del universo U, se crea una variable y_i que puede tomar los valores 0 o 1.
2. Para cada set S_j, se crea una función $f_j = \sum_{e \in S_j} e$ y se acota $1 \leq f_j \leq |S_j|$.
3. Se busca minimizar la función $f = \sum_{i=0}^{n} y_i$.

Luego, el conjunto solución C será

$$C = \{e_i \in U \mid y_i = 1\}$$

La complejidad de esta reducción es $O(n + m)$, que es polinomial, y la complejidad de Programación Lineal Entera es exponencial.

3.2. Programación Lineal Continua

También se puede aproximar la solución de Hitting-Set Problem con Programación Lineal continua. La reducción es similar, con la única diferencia de que las variables y_i pueden tomar valores reales entre 0 y 1.

Luego, el conjunto solución C será

$$C = \left\{ e_i \in U \mid y_i \geq \frac{1}{2} \right\}$$

La complejidad de esta reducción es $O(n + m)$, que es polinomial, y la complejidad de Programación Lineal Continua es $O(n^9)$.

4. Greedy

Proponemos dos aproximaciones extra. La primera es un algoritmo greedy que calcula la cantidad de los m subconjuntos B en la que aparece cada jugador y luego agrega el jugador con más apariciones de cada subconjunto. La primera operación tiene complejidad $O(k \times m)$, con k el promedio de jugadores por subconjunto ($k \leq n$). Con grupos más chicos, el algoritmo tiende a lineal ($O(m)$) y con grupos más grandes, a $O(m \times n)$. La segunda operación tiene la misma complejidad. Entonces, la complejidad total es $O(k \times m)$ y resulta pseudopolinomial.

Esta resulta ser una m-aproximación, ya que este algoritmo puede elegir m jugadores, cuando la solución óptima conta con uno solo. Esto se presenta cuando se tienen m grupos que comparen m jugadores ($|B_1 \cap B_2 \cdots \cap B_m| = m$). Todos los jugadores dentro de la intersección pertenencen a m grupos y el algoritmo propuesto elige, para cada grupo, el jugador que pertenece a más grupos. Como cada grupo tiene m jugadores con el mismo puntaje, podría elegir a cualquiera, y si para cada grupo elige uno distinto, terminaríamos con un resultado de tamaño m, mientras que bastaba con elegir uno solo.

La segunda aproximación comienza de la misma manera, calculando la cantidad de apariciones de cada jugador en cada subconjunto. Luego agrega el jugador con más apariciones entre todos y quita las apriciones de los subconjuntos que ya cubre al resto de los jugadores. Realiza esta operación hasta quedarse sin jugadores o que el jugador encontrado tenga cero apariciones restantes. Ya vimos quie la complejidad de la primera operación es $O(k \times m)$. La segunda tiene complejidad $O(j \times g \times m)$, con j la cantidad de jugadores de la solución, $j \leq m$, y g la cantidad promedio de grupos que cubre cada jugador $g \leq k$. Entonces, la complejidad total es $O(k \times m + j \times g \times m)$.

5. Mediciones

Para las mediciones, creamos listas de compilados de ejemplo de forma aleatoria. Decidimos que el tiempo de análisis de un compilado fuera un valor aleatorio entre 1 y 99'999, basándonos en los datos de ejemplo provistos por la cátedra.

Medimos el tiempo de ejecución de nuestro algoritmo para ciertas cantidades de compilados.

Notamos que los procesos de fondo del sistema nos generaba distorciones en los tiempos de ejecución de una misma muestra. Atacamos este problema realizando varias mediciones para el mismo escenario de compilados y tomando el promedio de los tiempos obtenidos.

Además, como definimos cada escenario arbitrariamente, en una cantidad de compilados x, este escenario puede ser poco favorable (puede estar muy desordenado), mientras que en una cantidad de compilados $x + 1$ puede ser muy favorable. Por lo cual la diferencia de tiempos va a ser muy grande a pesar de que la cantidad de compilados solamente difiere en una unidad (TimSort es una combinación de inserción y mergeSort, por lo cual el desorden inicial toma gran relevancia en el rendimiento final del algorítmo). Para solucionar esto, decidimos hacer varios escenarios distintos para la misma cantidad de compilados.

Con el objetivo de comparar los tiempos de ejecución de nuestro algoritmo con la complejidad teórica, optamos por realizar tanto un análisis de regresión lineal como uno de regresión lineal logarítmica que se ajustara a nuestros datos. Para evaluar qué curva se ajusta mejor, usamos la

raíz del error cuadrático medio (RMSE). Realizamos este análisis en un intervalo con tamaños pequeños (hasta 1'000) y en un intervalo con tamaños grandes (hasta 10'000). Ver figuras **??** y **??**

Se puede observar que en valores pequeños, el tiempo de ejecución del algoritmo sigue una clara curva lineal logarítmica. Sin embargo, a medida que aumenta el tamaño de la muestra de compilados, su comportamiento se aproxima más a una complejidad lineal.

Esto ocurre debido a la naturaleza de la función lineal logarítmica, que presenta una pendiente logarítmica. Cuando se trabajan con valores grandes, esta pendiente se vuelve casi constante, lo que la asemeja a una función lineal. Esto se debe a que el logaritmo es una función monótona creciente, lo que significa que su derivada es siempre positiva para valores mayores que cero. Sin embargo, el crecimiento de esta función se vuelve cada vez más lento a medida que los valores aumentan, indicando que su segunda derivada es siempre negativa. En consecuencia, para valores muy grandes, la función lineal logarítmica se estabiliza y, aunque no es completamente constante, su comportamiento se asemeja a una función lineal.

De esta manera, mediante el gráfico y la validación a través del error cuadrático medio, hemos comprobado que la curva lineal logarítmica se adapta de manera más precisa a nuestros datos.

De esta forma pudimos comprobar empíricamente que la complejidad tiende a $O(n \log n)$.

Nótese que graficamos también dos curvas que demarcan una estimación de los cuantiles verticales 0,1 y 0,9. Esta estimación se realizó calculando los cuantiles para un grupo pequeño centrado en cada punto. Estas curvas nos ayudan a dimencionar cómo la varianza del tiempo de ordenamiento crece con el aumento del tamaño de la información de entrada.

Adicionalmente, graficamos la densidad de los tiempos que también brinda una visualización de cómo aumenta la varianza con el aumento del tamaño de los datos de entrada. Ver figuras **??** y **??**

6. Conclusiones

Finalmente, consideramos que la solución óptima para abordar la problemática de Scaloni sería que éste analizara los compilados en función del tiempo requerido por los asistentes para visualizar cada uno, organizándolos en orden descendente. Esta estrategia permitiría resolver el problema con una complejidad algorítmica de orden $O(n \log n)$. Este enfoque garantiza la máxima eficiencia en la visualización de los compilados y permite que el tiempo invertido por Scaloni y sus asistentes se administre de manera óptima.