FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Teoría de Algoritmos I

Curso de verano 2024

Trabajo Práctico 3: NP-Completos

Alumno	Padrón	E-mail
Juan Ignacio Pérez Di Chiazza	109.887	jperezd@fi.uba.ar
Joaquín Parodi	100.752	jparodi@fi.uba.ar
Máximo Utrera	109.651	mutrera@fi.uba.ar

Índice

Introducción

Scaloni ya está armando la lista de 43 jugadores que van a ir al mundial 2026. Hay mucha presión por parte de la prensa para bajar línea de cuál debería ser el 11 inicial. Lo de siempre. Algunos medios quieren que juegue Roncaglia, otros quieren que juegue Mateo Messi, y así. Cada medio tiene un subconjunto de jugadores que quiere que jueguen. A Scaloni esto no le importa, no va a dejar que la prensa lo condicione, pero tiene jugadores jóvenes a los que esto puede afectarles.

Justo hay un partido amistoso contra Burkina Faso la semana que viene. Oportunidad ideal para poner un equipo que contente a todos, baje la presión y poder aislar al equipo.

El problema es, ¿cómo elegir el conjunto de jugadores que jueguen ese partido (entre titulares y suplentes que vayan a entrar)? Además, Scaloni quiere poder usar ese partido para probar cosas aparte. No puede gastar el amistoso para contentar a un periodista mufa que habla mal de Messi, por ejemplo. Quiere definir el conjunto más pequeño de jugadores necesarios para contentarlos y poder seguir con la suya. Con elegir un jugador que contente a cada periodista/medio, le es suficiente.

Ante este problema, Bilardo se sentó con Scaloni para explicarle que en realidad este es un problema conocido (viejo zorro como es, ya se comió todas las operetas de prensa así que se conoce este problema de memoria). Se sirvió una copa de Gatorei y le comentó:

'Esto no es más que un caso particular del Hitting-Set Problem. El cual es: Dado un conjunto A de n elementos y m subconjuntos $B_1, B_2, \ldots, B_m$ de A ($B_i \subseteq A \forall i$) , queremos el subconjunto $C \subseteq A$ de menor tamaño tal que C tenga al menos un elemento de cada B_i (es decir, $C \cap B_i \neq \emptyset$). En nuestro caso, A son los jugadores convocados, los B_i son los deseos de la prensa, y C es el conjunto de jugadores que deberían jugar contra Burkina Faso si o si'.

Bueno, ahora con un poco más claridad en el tema, Scaloni necesita de nuestra ayuda para ver si obtener este subconjunto se puede hacer de forma eficiente (polinomial) o, si no queda otra, con qué alternativas contamos.

Consigna

Para los primeros dos puntos, considerar la versión de decisión del Hitting-Set Problem:

Dado un conjunto de elemento A de n elementos, m subconjuntos $B_1, B_2, \ldots, B_m$ de A $(B_i \subseteq A \forall i)$ y un número k, ¿existe un subconjunto $C \subseteq A$ con $|C| \leq k$ tal que C tenga al menos un elemento de cada B_i (es decir, $C \cap B_i \neq \emptyset$)?

1. Demostrar que el Hitting-Set Problem se encuentra en NP.
2. Demostrar que el Hitting-Set Problem es, en efecto, un problema NP-Completo.
3. Escribir un algoritmo que, por backtracking, obtenga la solución óptima al problema. Generar sets de datos para corroborar su correctitud, así como tomar mediciones de tiempos.
4. Implementar alguna otra aproximación (u algoritmo greedy) que les parezca de interés. Realizar mediciones y comparaciones con la solución óptima con sets de datos propios. (incluso para valores que el algoritmo del punto anterior fuera inmanejable). Indicar y justificar su complejidad. Obtener una cota empírica de aproximación.
5. Agregar cualquier conclusión que parezca relevante.

1. Análisis del problema

El problema involucra la toma de decisiones de Scaloni para elegir el equipo en un partido amistoso contra Burkina Faso. Tiene como restricción elegir al menos un jugador pedido por cada periodista.

Para resolver el problema, se lo considerará como un caso particular del problema de Hitting-Set el cual plantea que:

"Dado un conjunto A de n elementos y m subconjuntos $B_1, B_2, \ldots, B_m$ de A ($B_i \subseteq A \forall i$) , se quiere obtener el subconjunto $C \subseteq A$ de menor tamaño tal que C tenga al menos un elemento de cada B_i (es decir, $C \cap B_i \neq \emptyset$)."

En este caso:

- A son los jugadores convocados.
- Los B_i contienen los jugadores que cada periodista quiere que estén.
- C es el conjunto de jugadores que deberían jugar contra Burkina Faso si o si.

Se observa que el conjunto A no es necesario para resolver el problema ya que solo es necesario elegir jugadores que pertenezcan a los subconjuntos de B. También se observa que uno de los subconjuntos podría pedir exactamente n jugadores, que son la totalidad de los jugadores disponibles, es decir el largo de B_i es menor o igual a n.

La cantidad de jugadores seleccionados en C sera como máximo m, en el caso que no haya intersecciones y se seleccione uno por cada subconjunto, y a su vez se puede decir que en este caso $m = n$ (ya que hay un subconjunto para cada jugador) y $C = A$.

Sin la restricción de que se quiere buscar la solución de menor largo posible entonces el problema siempre tiene solución $C = A$, de lo contrario también siempre tendrá solución pero ya no será trivial.

Se define la popularidad de un jugador como la frecuencia con la que aparece en los subconjuntos B_i. A mayor popularidad habrán mas posibilidades de que el jugador sea seleccionado.

A priori una posible solución al problema podría ser tomar a los jugadores más populares hasta que se hayan cumplido todas las peticiones, pero esta no es una opción valida, ya que como se puede ver en la siguiente figura, no siempre se minimizan los jugadores a seleccionar:

Dados esos subconjuntos de pedidos se llega a que la solución al problema sería un conjunto C que incluye a los más populares hasta cumplir todas las peticiones, y en este caso se tomarán a todos los jugadores ya que la petición 4 no estará cumplida hasta que se incluya al jugador *Alvarez* que es el jugador menos popular, y esta solución no es óptima ya que solamente tomando a 2 jugadores (por ejemplo: *Messi* y *Alvarez* o *Dybala* y *Alvarez*) se pueden abarcar todos los subconjuntos. Entonces como quedó demostrado, la popularidad no es un criterio que se puede tener en cuenta para conseguir la mejor solución al problema.

Como las peticiones están asociadas al gusto subjetivo de cada periodista no hay un criterio que se pueda tomar para encontrar la solución con los menos jugadores posibles, esto lleva a que el problema solo se pueda resolver probando todas las combinaciones de jugadores y eligiendo la óptima.

1.1. ¿Se encuentra Hitting-Set en NP?

Para determinar si este problema pertenece a la clase de complejidades NP habrá que demostrar la existencia de un verificador que chequea la correctitud de la solución al problema en tiempo polinomial o eficientemente. El algoritmo que lo verifica es el siguiente:

```
def verificar(A, B, k, sol):
    if len(sol) > k:
        return False

    for subconjunto in B:
        hit = False
        for elem in subconjunto:
            if elem in sol:
                hit = True
                break
        if not hit:
            return False

    return True
```

Este algoritmo verifica la solución en tiempo $O(m*n)$, ya que se iteran todos los m subconjuntos y por cada uno se ven sus elementos que en el peor caso serían un total de n (como se mencionó en el previo análisis). Esta complejidad es polinomial, cumpliéndose así la pertenencia de este problema a la clase NP.

1.2. ¿Es Hitting-Set un problema NP-Completo?

Para demostrar que este problema se encuentra en la clase NP-Completo se reducirá polinomialmente un problema conocido que se sabe que es NP-Completo a este. En este caso se reducirá el problema Dominating set a Hitting-Set.

En su versión de problema de decisión, el problema de Hitting Set es: Dado un conjunto A y un conjunto B, ¿existe un conjunto C de tamaño k ($k \leq$ tamaño de A), tal que la intersección entre C y todos los subconjuntos de B no sea vacía?

Por otro lado, el problema de Dominating Set en su versión de decisión es: Dado un grafo $G(V, E)$, ¿existe un subconjunto de V de tamaño k' tal que todos los vértices del grafo estén en dicho subconjunto o sean adyacentes a un elemento del subconjunto?

Se puede ver fácilmente que tanto k como k' son equivalentes, y existe una transformación para que una instancia del problema de decisión de Dominating Set, se puede resolver utilizando Hitting Set como se muestra a continuación:

- Primero para poder hacer que HS resuelva DS hay que adaptar la entrada. Dada una instancia de DS se deberá crear un conjunto A con todos los nodos del grafo G (que es parte de dicha instancia), una vez creado ese conjunto habrá que crear un nuevo conjunto de conjuntos B, en el que se pondrá un subconjunto B_i por cada nodo en G, y ese subconjunto incluirá el mismo nodo y todos sus adyacentes. Una vez creados ambos conjuntos, se le provee A, B y el k sin modificar (de la instancia de DS) a una caja negra resolvedora de Hitting set. Estas transformaciones se hacen en tiempo polinomial ya que para eso basta con realizar un recorrido de grafos como BFS.
- La caja negra se ejecutará una sola vez, y dará el resultado de Hitting set.
- Una vez obtenido el resultado de Hitting set, tendremos un conjunto de nodos que a su vez es exactamente la solución de Dominating set. Esto ocurre porque hitting set se asegurará de

seleccionar al menos un elemento de cada subconjunto en B lo que es lo mismo que decir que dado un nodo u se asegura de seleccionarlo o seleccionar al menos alguno de sus adyacentes (esto es lo que busca Dominating set).

Entonces podemos decir que $DS \leq_p HS$ y por lo tanto HS pertenece a $NP - Completo$. Adicionalmente se puede decir que como $HS \in NP - Completo$ entonces también $HS \in NP - Hard$.

A continuación se puede ver un ejemplo gráfico:

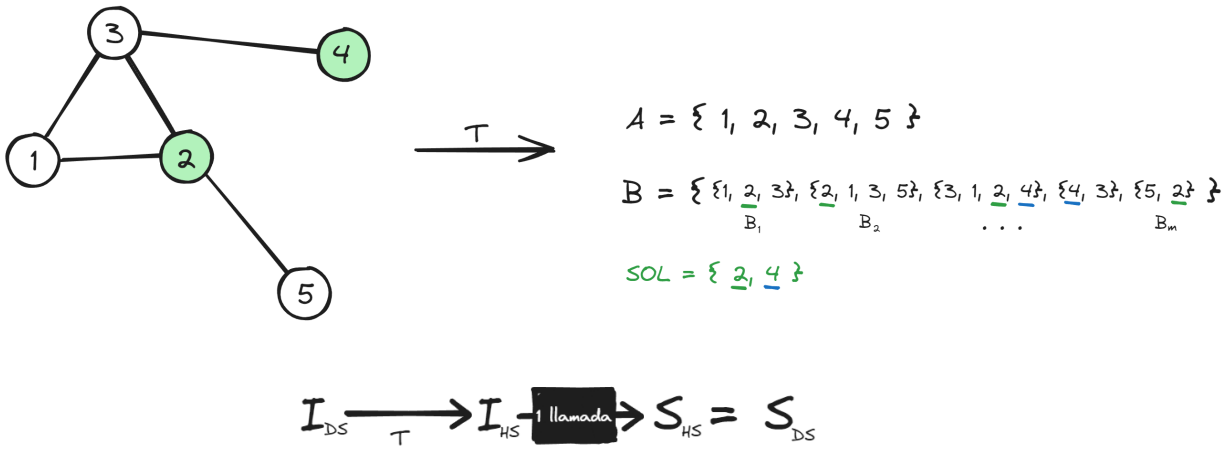


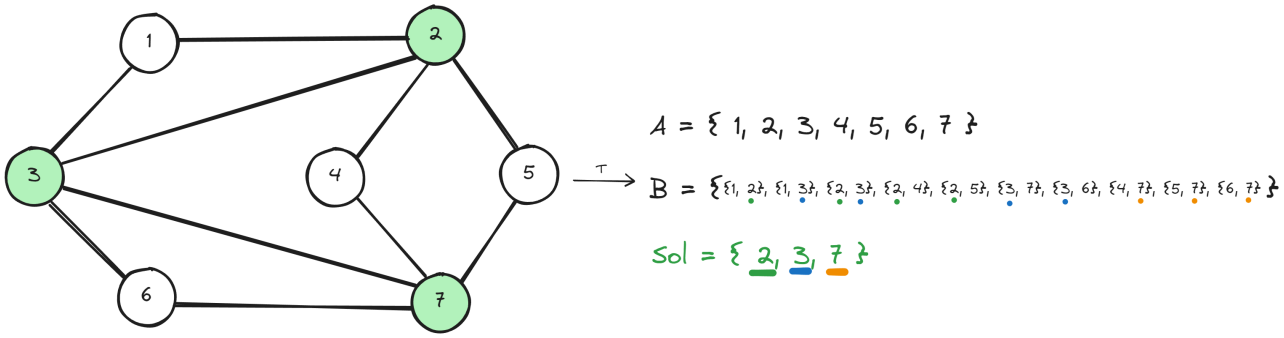
Figura 1: Donde I_{DS} y I_{HS} son Instancias del problema DS y HS respectivamente. Sus soluciones se denominan como $S_{DS/HS}$. T es la transformación previamente explicitada.

1.2.1. Reduccion polinomica de Vertex Cover

Como ahora está demostrado que este problema es NP-completo, entonces se puede decir que todos los problemas en NP pueden ser reducidos polinomicamente al mismo, incluyendo el resto de problemas de clase de complejidad NP-completo. Para verificar esto se demuestra otra reducción, en este nuevo caso será de Vertex cover a Hitting set.

Si el conjunto A es una lista con todos los vértices del grafo de entrada de Vertex Cover, y el conjunto B es una lista de todas las aristas del grafo tal que cada subconjunto B_i representa una arista, entonces es fácil corroborar que hitting set nos proporcionará una solución a vertex cover, ya que deberá cubrir todos los subconjuntos B_i con la menor cantidad de elementos de A, que es equivalente a decir que seleccionará a la menor cantidad de nodos que cubran todas las aristas, luego se puede decir que $VertexCover \leq_p HittingSet$

Por otro lado el problema de decisión de Vertex Cover es: Dado el grafo G(V,E), existe un subconjunto de tamaño j de V de forma que todas las aristas sean adyacentes a dicho subconjunto? Se puede ver en este caso también, la equivalencia directa entre j y k (mencionado anteriormente), por lo que se puede utilizar el problema de decisión de Hitting Set para obtener una respuesta válida a este problema de decisión (adaptando como se mencionó anteriormente el input para que sea un input válido en el problema de Hitting Set).



2. Planteo del algoritmo

2.1. Algoritmo óptimo

El algoritmo propuesto es uno que revisa todas las posibles combinaciones de jugadores una por una hasta terminar de explorar lo necesario para llegar a una solución de mínima longitud. Para esto se utiliza la técnica de diseño de algoritmos **Backtracking** y se toman criterios de poda para evitar explorar caminos que no contienen la solución, estos criterios son:

- Si el largo de la solución parcial supera el largo de la mejor solución encontrada hasta el momento.
- Si un subconjunto (o pedido) ya está abarcado por la solución parcial actual.

Adicionalmente a esto no se tienen en cuenta caminos donde no se llega a un Hitting set válido, para lo que se usa la función verificadora dada en **1.1. ¿Se encuentra Hitting-Set en NP?**

2.1.1. Por qué es óptimo

La razón por la cual este algoritmo de backtracking siempre encuentra la solución óptima al problema de hitting set es debido a cómo está diseñado el algoritmo:

- Poda: El algoritmo utiliza la técnica de poda para evitar explorar soluciones que no pueden conducir a la solución óptima. Por ejemplo, en este se verifica si la longitud de la solución parcial es mayor o igual a la longitud de la solución total actual. Si es así, no tiene sentido seguir explorando ese camino ya que no conducirá a una solución mejor.
- Backtracking: Si una rama de la exploración no conduce a una solución óptima, el algoritmo retrocede y explora las otras ramas. Esto asegura que se exploren todas las posibilidades.
- Verificación de validez: Antes de actualizar la solución total, se verifica si la solución parcial encontrada realmente es un hitting set válido. Esto garantiza que solo se consideren soluciones válidas.

2.1.2. Complejidad

- La complejidad temporal es exponencial ya que en cada llamada recursiva, hay dos opciones para cada jugador en los subconjuntos en B, incluirlo en la solución parcial o no incluirlo. Dado que hay m subconjuntos en B, cada uno con hasta n jugadores, esto resulta en un máximo de 2^{m*n} posibles combinaciones que deben ser exploradas en el peor caso. Entonces, la complejidad de este algoritmo es $O(2^{m*n})$ en el peor caso.
- La complejidad espacial del algoritmo viene dada primeramente por el uso de la pila de llamadas durante la recursión. En el peor caso, la profundidad de la recursión podría ser igual al número de subconjuntos en B, lo que implica una complejidad de espacio de $O(m)$, donde m es el número de subconjuntos en B. Adicionalmente se utilizan listas para guardar las soluciones parciales y totales que en el peor caso contendrán a todos los jugadores n, por lo que la complejidad final es $O(m + n)$.

2.1.3. Código en Python

```python
def obtener_hitting_set(A,B):
    sol_total = []
    _hitting_set_recursivo(B, [], sol_total, 0)
    return sol_total


def _hitting_set_recursivo(B, solucion_parcial, solucion_total, subset_actual):

    if len(solucion_parcial) >= len(solucion_total) and len(solucion_total) > 0:
        return False # Por este camino no se puede llegar a una solucion mejor (o de menor largo)

    if subset_actual == len(B) and verificar(B, solucion_parcial):
        solucion_total[:] = solucion_parcial[:]
        return True # Se llego a una solucion valida

    for elem in solucion_parcial:
        if elem in B[subset_actual]: # Salteamos el subset si ya esta hiteado
            return _hitting_set_recursivo(B, solucion_parcial, solucion_total, subset_actual + 1)

    for elem in B[subset_actual]:

        solucion_parcial.append(elem)

        if not _hitting_set_recursivo(B, solucion_parcial, solucion_total, subset_actual + 1):
                solucion_parcial.pop()
                return True # Este camino no minimiza el largo del hitting set, se vuelve uno atras en la recursion

        solucion_parcial.pop() # Ya se encontro una solucion, se sigue iterando en busca de una mejora

    return True
```

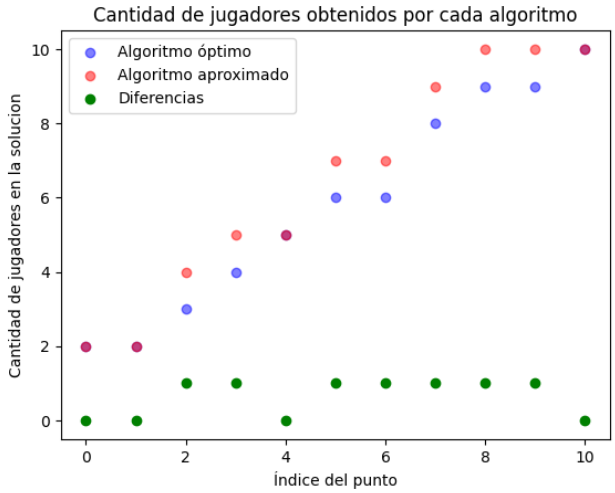
2.2. Algoritmo Aproximado y Cota Teórica

Se utilizó la idea de "Weighted Set Cover"para la realización de este algoritmo, siguiendo y adaptando cada paso a la sección 11.3 de *Algorithm Design* de Kleinberg & Tardos. Esto consiste en que cada jugador j_i tenga un peso asociado w_i. Debido a que la medida de peso es 'negativa', es decir, a mayor peso, menos conviene el jugador, lo que se utilizó fue la inversa de la frecuencia del jugador en los subconjuntos de B. Ya que el problema planteado, similar a Weighted Set Cover, en el que hay que cubrir el conjunto U con la menor cantidad de subconjuntos, y donde cada subconjunto tiene un peso asociado, si decimos que B cumple la función de U, y cada jugador del conjunto A cumple la función del subconjunto s_k, se puede demostrar que el peso obtenido por el algoritmo planteado está acotado superiormente por $\log(n) \cdot w^*$, donde w^* es el peso óptimo y n el tamaño del conjunto B.

2.2.1. Cota de aproximación empírica

Para encontrar una cota empírica se debe probar ambos algoritmos (el óptimo y el aproximado) con diversos sets de datos y medir sus diferencias, la mayor diferencia que se obtenga entre el

resultado dado por el algoritmo aproximado y el óptimo será la cota de aproximación empírica.



Dada la evidencia obtenida mediante las pruebas, se puede observar que el resultado obtenido por el algoritmo aproximado nunca supera el largo de la solución óptima por más de 1 elemento (con los sets utilizados para las pruebas), por lo tanto si llamamos f(B) a la función que nos devuelve para un set B su hitting-set de menor tamaño, y g(B) a la función que nos devuelve un hitting-set aproximado utilizando el algoritmo greedy, se puede afirmar que la cota empírica cumple que g(B)$\leq f(B) + 1$.

2.2.2. Complejidad

- En cuanto a la complejidad temporal, se tiene un bucle que itera hasta que el largo del subset B sea igual a 0. Si m es el largo de B, entonces hasta ahora se tiene una complejidad de $O(m)$. Luego, en cada iteración se recalculan los coeficientes, para lo cual se itera sobre cada subconjunto en B y para cada uno se ven sus elementos que en el peor caso serán un total de n, asi que esta parte tiene una complejidad de $O(m \cdot n)$. Posteriormente crea una lista A a partir de los jugadores que quedan y se la ordena por coeficiente, lo cual tiene la complejidad entonces es $O(n \log_2(n))$ (siendo que dicha lista tiene un máximo de n jugadores). Por último, se itera sobre B para eliminar los subconjuntos donde la intersección con el elemento elegido y el subconjunto es vacío, lo que conlleva una complejidad $O(m)$. La complejidad total obtenida es de

$$O\left(m \cdot (m \cdot n + n \log_2(n) + m)\right) = O\left(m^2 \cdot n + m \cdot n \log_2(n) + m^2\right)$$

  Acotado superiormente por $m^2 \cdot n$ y $m \cdot n \log_2(n)$ nos queda que la complejidad obtenida es: $O(m^2 \cdot n + m \cdot n \log_2(n))$.

- Se utilizaron las siguientes estructuras: diccionario que contiene los coeficientes de los jugadores y su largo maximo es n, lista a de jugadores restantes cuyo largo maximo es n y lista B de subconjuntos con largo maximo m. Como estas estructuras se crean por cada iteracion externa hasta m entonces se tiene una complejidad espacial de $O(m^2 + m \cdot n)$.

2.2.3. Código en Python

```python
# Esto es lo que se tiene que buscar minimizar en cada paso segun el K&T, 11.3
def calcular_coeficiente(subconjuntos):
    pesos = {}
    for subconjunto in subconjuntos:
        for jugador in subconjunto:
            if jugador not in pesos:
                pesos[jugador] = (0, 0)
            pesos[jugador] = (pesos[jugador][0] + 1, pesos[jugador][1] + len(subconjunto)
    )

    for jugador in pesos.keys():
        pesos[jugador] = 1 / (pesos[jugador][0] * pesos[jugador][1])

    return pesos

def hitting_set_greedy(B):
    hitting_set = []
    while B:
        pesos = calcular_coeficiente(B)

        a = [jugador for jugador in pesos.keys()]
        a.sort(key=lambda jugador: pesos.get(jugador, 0))

        hitting_set.append(a[0])

        B = [s for s in B if a[0] not in s]

        pesos.pop(a[0], None)

    return hitting_set
```

3. Ejemplos de ejecución

En cada uno de los siguientes ejemplos de ejecución se corroboró que el resultado dado por el algoritmo óptimo sea en efecto el correcto.

3.1. 5 peticiones

```
'''
Peticiones:

Barcon't,Cuti Romero,Colidio,Casco
Colo Barco,Wachoffisde Abila,Messi,Casco,Armani,Chiquito Romero
Barcon't,Wachoffisde Abila,Colidio,Casco
Messi,Cuti Romero,Casco,Pezzella
Colo Barco,Messi,Cuti Romero
'''

Algoritmo Optimo:
Cantidad de jugadores: 2
Lista de jugadores: ['Casco', 'Cuti Romero']

Algoritmo Aproximado:
Cantidad de jugadores: 2
Lista de jugadores: ['Casco', 'Cuti Romero']
```

3.2. 7 peticiones

```
'''
Peticiones:

Colo Barco,Mauro Zarate,Colidio
Colo Barco,Mauro Zarate,Dibu,Chiquito Romero,Barcon't
Chiquito Romero,Colidio,Wachoffisde Abila,Pezzella
Armani,Mauro Zarate,Barcon't
Colo Barco,Armani,Dibu,Pezzella,Barcon't
Barcon't,Mauro Zarate,Cuti Romero
Cuti Romero,Pezzella,Messi,Colidio,Barcon't,Casco
'''

Algoritmo Optimo:
Cantidad de jugadores: 2
Lista de jugadores: ['Mauro Zarate', 'Pezzella']

Algoritmo Aproximado:
Cantidad de jugadores: 2
Lista de jugadores: ["Barcon't", 'Colidio']

# Notar que puede haber mas de una solucion optima (de largo minimo)
```

3.3. 10 peticiones

```
'''
Peticiones:

Colidio,Chiquito Romero,Gallardo
Colo Barco,Palermo,Dibu,Di Maria,Cuti Romero
Casco,Gallardo,Messi
Wachoffisde Abila,Palermo,Dibu,Burrito Ortega,Colidio,Di Maria,Gallardo
Chiquito Romero,Dibu,Pezzella,Di Maria,Gallardo,Mauro Zarate
Casco,Wachoffisde Abila,Colo Barco,Riquelme,Dibu,Mauro Zarate
Casco,Wachoffisde Abila,Barcon't,Ogro Fabianni,Di Maria,Cuti Romero
Wachoffisde Abila,Colo Barco,Barcon't,Colidio,Di Maria,Gallardo,Messi
Wachoffisde Abila,Chiquito Romero,Barcon't,Palermo,Cuti Romero,Messi
Casco,Mauro Zarate,Messi
'''

Algoritmo Optimo:
Cantidad de jugadores: 3
Lista de jugadores: ['Gallardo', 'Palermo', 'Casco']

Algoritmo Aproximado:
Cantidad de jugadores: 4
Lista de jugadores: ['Wachoffisde Abila', 'Gallardo', 'Palermo', 'Casco']
```

4. Mediciones de tiempo

4.1. Cómo afecta la cantidad de peticiones n al tiempo de ejecución del algoritmo óptimo

En el siguiente benchmark se analizó el tiempo del algoritmo en función del tamaño n de la entrada. La complejidad teórica del algoritmo está explícita en la sección 2.1.2 **Complejidad**.

Para el benchmark se tomaron diferentes cantidades entre 5 y 200.

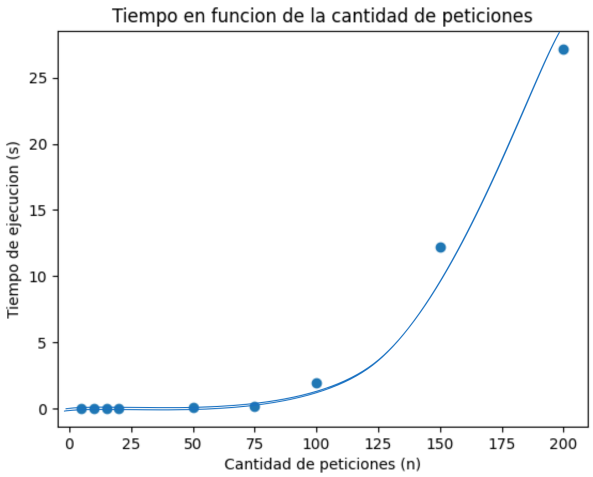


Figura 2: Se puede observar que a medida que aumenta n, el tiempo que tarda el algoritmo crece exponencialmente y se confirma la complejidad teórica.

4.2. Cómo afecta la cantidad de peticiones n al tiempo de ejecución del algoritmo aproximado

Al igual que el benchmark anterior en este caso se analizó el tiempo que tarda el algoritmo greedy en función del tamaño n de la entrada. La complejidad teórica del algoritmo está en la sección **2.2.2: Complejidad**.

Para la generación de datos se utilizaron intervalos crecientes de 0 a 2000 con un paso de 10, donde se eligen numeros al azar entre 0 y el valor del rango actual para generar un set A, y otros números al azar dentro del mismo rango para generar un set B. Para el set A se utilizan los elementos desde el 0 hasta el número elegido anteriormente con un paso de 1, mientras que para el set B se toman muestras con reposición del set A de un largo arbitrario, cumpliendo que sea menor al tamaño de dicho set.

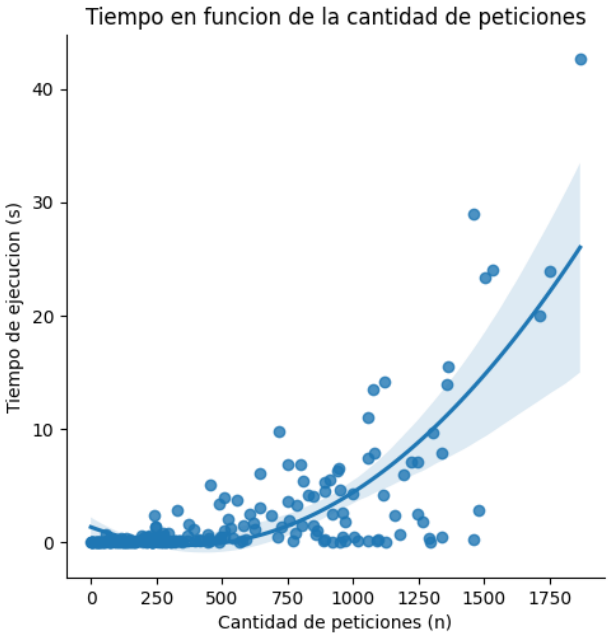


Figura 3: Se puede observar que el algorítmo lejos de ser lineal (confirmando su complejidad temporal) en su tiempo de ejecución, tiene una buena perfomance para sets de datos cuya distribución mejor se adapte a las reglas mencionadas para elegir el próximo jugador, esto es: muchos jugadores repetidos en distintos sets.

5. Conclusiones

Se definió que la obtención del hitting set no se puede hacer en tiempo polinomial, ya que hasta el día de la fecha no se demostró si P=NP. Para esto se demostró que el problema se encuentra en NP, y que es NP-Completo.

Se planteó un algoritmo que obtiene la solución óptima por backtracking y se verificó su correctitud por medio de pruebas. A su vez se confeccionó un algoritmo greedy que aproxima la solución y se ejecuta en tiempo polinomial, a diferencia del anterior mencionado que se ejecuta en tiempo exponencial, adicionalmente se encontró una cota de aproximación empírica y teórica para el susodicho.

También se realizaron mediciones de tiempos de ejecución variando la cantidad de peticiones n y se los compararon con sus complejidades teóricas previamente analizadas.