Índice general

Proyecto Final
Diseño y Análisis de Algoritmos

Belsai Arango Hernández
Daniela Rodríguez Cepero

Universidad de La Habana,
San Lázaro y L. Plaza de la Revolución, La Habana, Cuba
https://github.com/Daroce1012/DAA-problems/tree/Presupuesto-de-la-FEU
http://www.uh.cu

1. Problema

Kevin el Encargado
Kevin ha sido puesto al frente de la comisión de la facultad que elegirá las fechas de las pruebas de los k cursos que se dan en la facultad.

Cada curso tiene una cantidad de pruebas determinadas que quiere poner, y propone para esto, por ejemplo, los días $\{17, 34, 65$ y $87\}$ del curso escolar, si vemos a este como una sucesión de días en los que se imparten clases. Para mostrarse flexibles, los cursos a veces elaboran más de una propuesta incluso.

Por un problema de desorganización las propuestas se regaron y ahora no se sabe que curso propueso que propuesta, pero ya Kevin esta cansado de tanta gestión. Kevin quiere elegir k propuestas que ninguna quiera poner pruebas el mismo día que las otras, así supone que todo el mundo estará contento, ayude a Kevin.

2. Solución

Se desea elegir un subconjunto de k propuestas de las existentes, de modo tal que los días sean diferentes en todas las propuestas para que todos estén contentos.

Debido a que existe la posibilidad de que se repitan los días en las propuestas se considera un subconjunto de propuestas como una solución, si esto no ocurre entre ninguna de ellas.

Para esto se decide hacer un algoritmo que elija una propuesta y verifique si a partir de esta se puede obtener un subconjunto de propuestas válidas, en caso contrario este retirará a la propuesta del subconjunto solución.

3. Correctitud

Como se desea elegir un subconjunto de k propuestas de las existentes, de modo tal que los días sean diferentes en todas las propuestas para que todos estén contentos, se utiliza un array de marca para verificar si el día de una propuesta ya está en la solución y así se garantiza que al agregar una propuesta a la solución cumpla la condicón de que los días no se repitan por tanto para que la solución sea válida solo faltaría que se verificara si la cantidad de propuestas es igual a k.

Para la confección del array de marca se recorren los días de todas las propuestas y nos quedamos con el mayor día de todos. Luego se utiliza una lista en donde cada índice i representa un día i , inicialmente el valor de este es Falso, luego cambia a True cuando el día que representa el índice forma parte del conjunto de propuestas que son solución. Así podemos saber con solo indexar si un día es parte de la solución.

Todo esto fue elaborado en métodos independientes que llamaremos métodos auxiliares. Para la confección del método principal se utilizó la estrategia de Backtracking que va obteniendo todas las combinaciones de k propuestas, de las m que se tienen.

El nombre de este método es func proposals. En la primera línea de este se encuentra la condición de parada que sería la verificación de la existencia de k propuestas, puesto que si esto se cumple hemos llegado a la solución. Ya que al agregar una propuesta a la solución se verificaba anteriormente que esta no tuviera días en común con el resto y por tanto al ser la cantidad de propuestas que están en el posible conjunto de solución k, hemos obtenido la solución del problema.

En la próxima línea se encuentra un for que recorre todas las propuestas. Con cada propuesta i , se coge a esta y se agrega al conjunto de solución si no está agregada antes y se llama recursivo a este método con esta nueva propuesta en la solución, si a partir de la inserción de i en el conjunto solución no se llegó a la solución esta se elimina del conjunto solución y se pasa analizar la próxima propuesta. En el peor de los casos que el algoritmo no encuentre una solución este concluye al terminar el ciclo for.

Como podemos ver de forma resumida el algoritmo genera todas las posibles soluciones a través de las combinaciones sin repetición y solo retorna la que es válida.

4. Demostración que el problema Kevin el Encargado es NP-completo

Sea X un problema de decisión tal que:

1. $X \in NP$
2. Existe X' (X' es NP - completo and $X' \leq_p X$)

Entonces X es un problema NP - completo

Un problema NP es un problema que su tiempo de ejecución es superpolinomial y además cumple que el tiempo de verificación de la solución es polinomial.

Se asume que el problema 3-CNF es NP-completo.

Teorema:
El problema de hallar el clique de tamaño máximo es NP - completo.
Demostración:
Se cumple que un clique en un grafo $G =< V, E >$ es un subconjunto de V, tal que los vértices que lo integran son mutuamente adyacentes, osea un clique es un subgrafo completo de G.
El problema del clique es hallar el clique de tamaño máximo es decir hallar la cantidad máxima de vértices que son mutuamente adyacentes. El problema es NP:
Demostración
Un algoritmo para determinar que un grafo $G = (V, E)$ junto $|V|$ vértices tiene un clique de tamaño k es listar los k subconjuntos de V y verficar por cada uno si forma parte del clique. El tiempo de ejecución del algoritmo es omega $k^2\binom{V}{k}$, el cual es polinomial si k es una constante, pero k es un valor cerca de la cantidad de $|V|/2$, por tanto en ese caso es un algoritmo con un tiempo superpolinomial. Y el tiempo que toma verificar que la solución obtenida responde al problema es ponlinomial ya que sería comprobar por cada vértice que sea adyacentes a los demás, es decir n^2 donde n es el número de vértices que pertenece al clique. Por lo que queda demostrado que el problema es NP.
El problema es NP-completo:
Se conoce que el problema $3 - CNF$ es NP-completo, si se demuestra que $3 - CNF \leq_p Clique$, lo cual se cumple si se demuestra que el problema del clique es NP-hard.
Se tiene una instancia de $3 - CNF - SAT$.
Sea $\phi = C_1 \wedge C_2 \wedge ... \wedge C_k$ una fórmula booleana en $3CNF$ con k cláusulas para $r = 1, 2, ..., k$, cada cláusula C_r tiene exactamente 3 literales distintos l_1, l_2 y l_3. Podemos construir un grafo que cumpla que a es satisfacible si y solo si G tiene un clique de tamaño k.
EL grafo $G = (V, E)$ se construiría de la siguiente manera: por cada claúsula $C_r = l_1 \vee l_2 \vee l_3 \in \phi$, colocamos 3 vétices v_1, v_2 y v_3 en V. Se pondría una arista entre 2 vértices si ambos cumplen las siguientes condiciones:

1. v_i y v_j representan literales de diferentes cláusulas.

2. sus literales correspondientes son consistentes, es decir un literal no es la negación del otro.

Este grafo se puede construir en tiempo polinomial.
Para demostrar que esta transformación de a en G es una reducción: Primero vamos a suponer que ϕ tiene una asignación satisfactoria. Entonces cada cláusula C_r contiene al menos un literal l_i el cual tiene asignado 1 y cada literal le corresponde un vértice v_i. Tomando de cada cláusula un literal que es 1 se produce un conjunto V' de k vértices. Digamos que ese conjunto V' es un clique. Por cada 2 vétices que pertenencen a V' donde sus literales son de cláusulas distintas y esos literales son 1 por ser una asignación satifactoria y esos literales no son complementos por ello por las condiciones descritas anteriormente podemos decir que en la construción de G la arista entre esos 2 vértices pertenece a E. En cambio suponiendo que en G hay un clique V' de tamaño k. las aristas en G no conectan 2 vértices que representan literales que se encuentran en una misma cláusula y por tanto en V' solo se encuentra un vértice por cada cláusula. Por lo que se puede asignar 1 a cada literal l_i ya que vi pertence a V' sin temor a asignar 1 a un literal y su complemento ya que se garantiza en G que no existe aristas entre literales inconsistentes. Cada cláusula es satisfactoria y por lo tanto a es satisfactoria.

Teorema:
El problema del conjunto independiente máximo es NP-completo.
Demostración:
C - problema del conjunto independiente máximo.
Probar C pertenece a NP
Para determinar el conjunto mayor de vértices independientes en un grafo $G = (V, E)$ se forman todas las posibles combinaciones de vértices que cumplan que no tienen ninguna arista en común utilizando dinámica y teniendo en cuenta la lista de adyacencia. El tiempo de ejecución de ese algoritmo es exponencial, por tanto no forma parte de los problemas polinomiales.
El tiempo de ejecución para obtener el certificado en este problema es polinomial. El procedimiento sería verificar por cada vértice que no tiene ninguna arista en común con ninguno de los otros vértices del conjunto, este tiempo es n^2.

Lo siguiente es probar que el problema del clique máximo $\leq_p$ C, lo cual muestra que el problema del conjunto independiente máximo es NP - hard.
El algoritmo de redución empieza con una instancia del problema del clique máximo.
Sea un grafo $G = (V, E)$ con un clique $c = v_1, v_2, ..., v_k$ con k vértices.
Podemos construir un grafo que cumpla que c un clique máximo si y solo si existe otro grafo el cual tiene un conjunto independientes máximo de tamaño k.
El grafo se construiría de la siguiente manera:
Sería el grafo complemento del grafo G, es decir por cada vértice en el grafo G va a existir un vértice en G', el conjunto de aristas en G sería las aristas complemento en G'. Se puede verificar que la construcción del grafo es en tiempo polinomial,

sería por cada vértice eliminar sus aristas y formar aristas con aquellos vértices que era independiente en el grafo original.

Para mostrar que la tranformación de problema de clique máximo a problema del conjunto independiente máximo es una reducción, primero vamos a suponer que tenemos un clique c que es máximo, por tanto cada vértice que pertenece a ese clique tiene aristas con todos los demás vértices del clique y esto se cumple con todos. Si tenemos un grafo que es el complemento del grafo original, entonces todas aristas que existian entre los vértices que eran mutuamente adyacentes dejan de existir convirtiendo estos vértices sin relación alguna entre ellos, es decir convirtiendo estos vértices en un conjunto independiente.
Por que este conjunto independiente es máximo?.
Supongamos que no lo es entonces existe un vértice que es independiente con todos los vértices del conjunto en el grafo complemento. Si esto se cumple entonces en el grafo original este vértice tiene una arista con cada uno de los vértices del conjunto, por tanto es mutuamente adyacentes a ello, por lo que también forma parte de un clique, un clique donde están todos los vértices anteriores y ademas este, entonces hemos obtenido un clique de tamaño mayor al clique que teníamos al inicio. Contradición porque hemos asumido que el clique inicial era máximo.
Por lo que queda demostrado que el problema del conjunto independiente maximo es NP-completo

Teorema:
El problema kevin el encargado es NP-completo.
Demostración:
Para demostrar que el problema de kevin el encargado pertenece a NP primero debemos demostrar que su tiempo en ejecución es superpolinomial: Un algoritmo para determinar las k propuestas válidas de m propuestas que plantea los cursos es hallando las combinaciones en m de tamaño k y verificando que las propuestas seleccionadas no tienen días en común entre ellas. El tiempo de ejecución del algoritmo es $\binom{m}{k}$ donde m es la cantidad de propuestas y k la cantidad de propuestas seleccionadas, se puede decir que el algoritmo es superpolinomial.
Para que pertenezca a los problemas NP el tiempo de verificación debe ser polinomial, es decir el tiempo que debe tomar para comprobar que una solución es válida y satisface el problema debe ser polinomial. La solución al problema es un conjunto de k propuestas, la forma de comprobar que es válida es verificando por cada propuesta que los días selecionado para las pruebas son diferentes a los días de las demas propuestas, esto se haría con una lista de marcas donde cada vez que se obtenga un día de prueba se marca como ocupado esa casilla en el array. El tiempo de ejecución de este procedimiento sería la suma de las cantidad de pruebas de cada curso, es decir un tiempo polinomial.
Lo siguiente sería probar que el conjunto independiente $\leq_p$ problema de kevin, lo cual muestra que el problema de kevin es NP-hard.
Supongamos que tenemos una instancia del problema del conjunto independiente.

Sea un grafo $G = (V, E)$, podemos construir k propuestas válidas si y solo si en G existe un conjunto independiente de tamaño k.
Se construiría el problema de la siguiente forma:
Por cada vértice v_i representaremos una propuesta de un curso en el problema de Kevin. Cada arista entre 2 vértices en G representaría en el problema de Kevin 2 propuestas que tienen al menos 1 día en común. La construcción de estas propuestas sería en tiempo polinomial.

Debemos mostrar que esta tranformación del grafo en propuestas es una reducción.
Primero supongamos que existen n vértices que son el conjunto máximo independiente en el grafo. Cada vértice corresponde con una propuesta de un curso. Podemos decir entonces que la cantidad de propuestas que no tienen ningun día en común es n, ya que como se planteo anteriormente una arista entre 2 vértices representaría 2 propuestas que tienen días en común, por tanto si son independientes los vértices, las propuestas no tienen ninguna intersección entre ellas.

Utilizamos una metaheurística para nuestro problema. La meteheurística escogida fue el algoritmo génetico. Escogemos esta metaheurística debido a su procedimiento que genera nuevas soluciones a partir de un población de soluciones existentes, luego de evaluarlas, en nuestro caso el modo de evaluación fue 0 cuando la solución no era válida y 1 en caso contrario y a partir de ahi seleccionamos aletoriamente, dandonos la posibilidad de que a partir de una solucion no válida al mezclarlas con otras soluciones obtener una solución que si sea válida. Los resultados obtenidos después de aplicar la metaheurística fueron los deseados, en muchas ocaciones la población inicial han sido soluciones no válidas sin embargo luego de aplicar le mezcla se han obtenido soluciones interesantes.

5. Complejidad Temporal

El algoritmo principal selecciona una propuesta y verifique si a partir de esta se puede obtener un subconjunto de propuestas válidas, en caso contrario este retiraría la propuesta del subconjunto solución, es decir retrocede y vuelve a elegir. Al final hace una combinación de k en m. La cuál tiene un costo de $\frac{m!}{(m-k)!k!}$, pero si dividimos esa fórmula, en el numerador se tiene $m!$ y en el denominador $(m-k)!k!$, luego el numerador se cancela con $(m-k)!$ y entonces en el numerador nos queda $m*m-1*...*m-k+1$ y abajo $k!$, eso es aproximadamente $\frac{m^k}{k!}$ por tanto el costo es $O(m^k)$.

La solución del problema utiliza también otros métos auxiliares como son mark ,max matrix,not contain,mark matrix.

El método max matrix hace un recorrido for por cada propuesta que se encuentra dentro de una lista con las m propuestas iniciales de las cuales se escogeran k, por tanto hay un doble for, el primero hace m iteraciones y por cada iteración de primero se hacen k iteraciones al analizar la propuesta para determinar el mayor día de todos, entonces por ley de la multiplicacción el costo de este método es $O(mk)$.

El método mark hace un recorrido por los días de una propuesta para mantener actualizada la lista de los días de todas las propuestas que pertenecen al conjunto solución, actualizar esta implica indexar en una lista que se pasa como parámetro y cambiarle el valor a True por tanto el costo de esta operación es $O(1)$, luego por ley de la multiplicación el costo de este método es $O(k)$.

El método not contain hace un recorrido por los días de una propuesta para determinar si esta es posible agregarla a el conjunto de solución, verificar si es posible es indexar en una lista que se pasa como parámetro por tanto el costo de esta operación es $O(1)$, luego por ley de la multiplicación el costo de este método es $O(k)$.

El método mark matrix inicializa la lista de todos los días y por tanto hace un recorrido for, entonces el coste de este es $O(n)$, donde n es valor del mayor día.

Luego podemos concluir con que el costo de la solución es $O(m^k)$.