INF221 – Algoritmos y Complejidad

Clase #7
Algoritmos, valga la redundancia

Aldo Berrios Valenzuela

18 de septiembre de 2016

1. Algoritmos

Con algoritmos buscamos lo que es optimización combinatoria. Corresponde a un problema de programación de tareas (scheduling en inglés)

Ejemplo 1.1. Supongamos que usted está a cargo de programar observaciones en ALMA. Para justificar el gasto de este enorme recurso, su misión es programar el máximo número de observaciones. Las observaciones tienen instante de inicio y duración, y no pueden traslaparse.

Formalmente, el proyecto i tiene duración el intervalo abierto $[s_i, s_i + \ell_i[$. Se pide elegir el subconjunto $\Pi \subseteq P$ tales que los elementos de Π sean disjuntos y el número de elementos de Π sea máximo (Figura 1). ¿Cómo hacerlo?

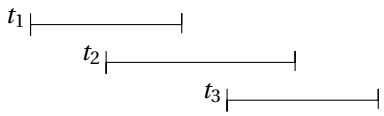

Figura 1: Cada una de las líneas que se presentan en la figura representan el intervalo de tiempo que dura cada proyecto/tarea. Para el caso, se observa que la t_1 traslapa con t_2, t_2 traslapa con t_1 y t_3 y t_3 sólo traslapa con t_2.

(básicamente, estamos suponiendo que el observatorio se arrienda por proyecto y no por ganancia). Proponemos algunas sugerencias:

- *Sugerencia 1:* Repetidamente elegir la tarea más corta que no entra en conflicto. La Figura 2 muestra un

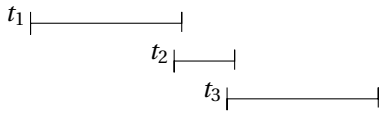

Figura 2: t_2 es la tarea más corta. Sin embargo, si empezamos con t_2, dejamos fuera a t_1 y t_3 haciendo un total de 1 tarea. Si empezamos con t_1, t_2 queda fuera, pero en su lugar realizamos t_3; un total de 2 tareas.

  contraejemplo, por lo tanto, esta sugerencia no es óptima.

- *Sugerencia 2:* Elegir la tarea con inicio más temprano que no crea conflicto. La Figura 3 muestra un contraejemplo, por lo tanto, esta sugerencia no es óptima.

- *Sugerencia 3:* Marcar cada proyecto con el número de proyectos con que entra en conflicto, programar en orden creciente de conflictos. La Figura 4 muestra un contraejemplo, por lo tanto, esta sugerencia no es óptima.

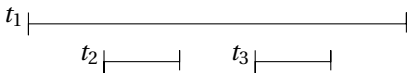

Figura 3: Si empezamos con t_1, dejamos fuera a t_2 y t_3, haciendo un total de 1 tarea. Por otro lado, si realizamos t_2 y t_3 dejamos fuera a t_1, pero hacemos 2 tareas.

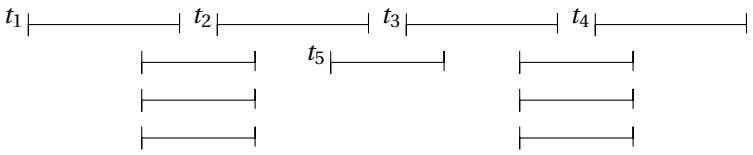

Figura 4: El algoritmo nos dice derechamente realizar las tareas t_5 junto con t_1 o cualquiera de las tareas que entran en conflicto con ella y t_4 o cualquiera de las tareas que entran en conflicto con este último. Eso nos da un total de 3 tareas, pero si hacemos t_1, t_2, t_3 y t_4 hacemos un total de 4. Me echaron a perder el día.

■

¿Estructura común?

- Eliga elementos sucesivamente según hasta que no queden opciones viables.
- Entre las opciones visibles en cada paso, elija la que minimiza (maximiza) alguna propiedad.

De eso es lo que precisamente tratan los *Greedy Algorithms* (algoritmos voraces, aunque mejor traducción sería "ávido"). Es importante destacar que estos no encuentran la solución óptima del problema, pero sí es una aproximación bastante buena. (en el fondo es la solución óptima, pero no necesariamente la *más* óptima)

1.1. Comprobar que un algoritmo es óptimo

Volvamos al ejemplo 1.1. En ese caso, el *criterio* que tenemos que usar para encontrar la solución óptima consta en elegir la tarea que *finaliza* más temprano y no entra en conflictos. Para el proyecto i, el instante de *fin* es

$$f_i = s_i + \ell_i$$

¿es óptimo?

Para comprobar que un algoritmo es óptimo debemos demostrar lo siguiente:

- *Elección voraz (greedy choice):* Para toda instancia P, hay una solución óptima que incluye el primer elemento $\hat{p}$ elegido.
- *Estructura inductiva:* Dada la elección voraz $\hat{p}$, queda un subproblema menor P' tal que si Π' es solución óptima de P', $\{\hat{p}\} \cup \Pi'$ es solución viable de P (P' no tiene "restricciones externas").
- *Subestructura óptima*: Si P' queda de P al sacar $\hat{p}$, y Π' es óptima para P', $\Pi' \cup \{\hat{p}\}$ es óptima para P.

Con estos tres podemos demostrar que la secuencia de elecciones de $\hat{p}$ da una solución óptima por inducción sobre los pasos.

1.1.1. Demostrando un algoritmo voraz

Volvamos al ejemplo 1.1. Si quisiéramos demostrar que la sugerencia 3 es óptima (a pesar de que echamos a perder el negocio con la Figura 4) solo tenemos que seguir los pasos anteriores:

- *Elección voraz:* Sea $\hat{p}$ la primera tarea elegida y Π^* la solución óptima para P. Si $\hat{p} \in \Pi^*$, estamos listos. En caso contrario, sea Π' la solución obtenida reemplazando el proyecto más temprano de Π^* por $\hat{p}$. Esto no produce nuevos conflictos, y $|\Pi^*| = |\Pi'|$, ambos son óptimos.

- *Estructura inductiva:* El elegir $\hat{p}$ nos deja un problema P' sin restricciones externas (elegir $\hat{p}$ elimina de consideración las tareas que entran en conflicto con $\hat{p}$, pero nada más).
- *Subestructura óptima:* Si P' queda después de elegir $\hat{p}$, y Π' es óptima para P', entonces $\Pi' \cup \{\hat{p}\}$ es óptima para P.

Demostración. Sea Π' como dado. Entonces $\Pi' \cup \{\hat{p}\}$ es viable para P, y $\left|\Pi' \cup \{\hat{p}\}\right| = \left|\Pi'\right| + 1$. Sea Π^* una solución óptima para P que contiene $\hat{p}$. Entonces $\Pi^* \setminus \{\hat{p}\}$ es una solución óptima para P'. Pero entonces:

$$\begin{aligned} \left|\Pi'\right| &= \left|\Pi^* \setminus \{\hat{p}\}\right| \\ &= \left|\Pi^*\right| - 1 \\ \rightsquigarrow \left|\Pi' \cup \{\hat{p}\}\right| &= \left|\Pi^*\right| \end{aligned}$$

y $\Pi' \cup \{\hat{p}\}$ es óptima. □

> No todos los problemas tienen solución tan óptima y eficiente como este.

> Sólo encontramos una de las soluciones óptimas