.UBAfiuba
FACULTAD DE INGENIERÍA

Greedy: Interval Scheduling

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

Interval Scheduling

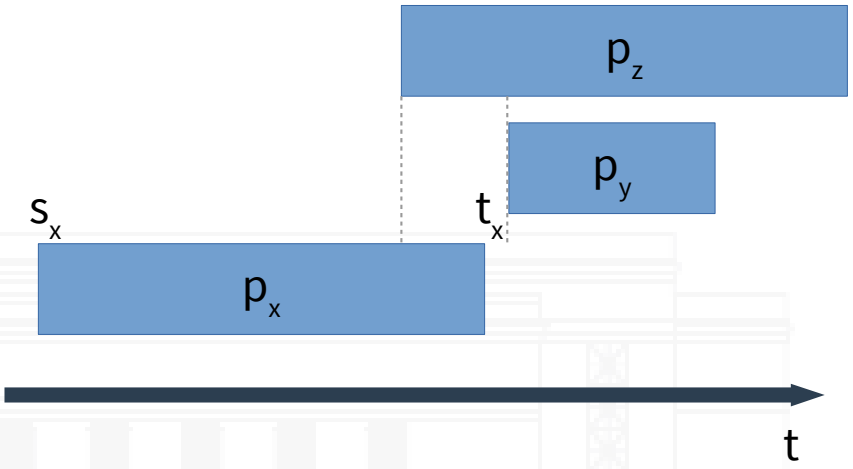
Sea

P un conjunto de n pedidos $\{p_1, p_2, \ldots, p_n\}$

Cada pedido i tiene

un tiempo si donde inicia

Un tiempo ti donde finaliza



Px y py son compatibles entre si. Sin embargo Pz no es compatible con ninguna de las anteriores

Un par de tareas $p_x, p_y \in P$

Son compatibles entre si, si - y solo si – no hay solapamiento en el tiempo entre ellas

Queremos

Seleccionar el subconjunto P de mayor tamaño posible de modo que todas las tareas seleccionadas sean compatibles entre si

Análisis del problema

Debemos determinar

Como seleccionamos los pedidos.

Al momento de seleccionar un pedido

Un subconjunto de pedidos serán incompatibles

Por lo tanto no elegibles en la solución

Iterativamente seleccionaremos pedidos

Mediante una heurística greedy

Y finalizaremos cuando no queden pedidos seleccionables

Selección greedy

Podemos construir

diferentes criterios a la hora de seleccionar los pedidos

Algunos de ellos

Aquel que comienza antes

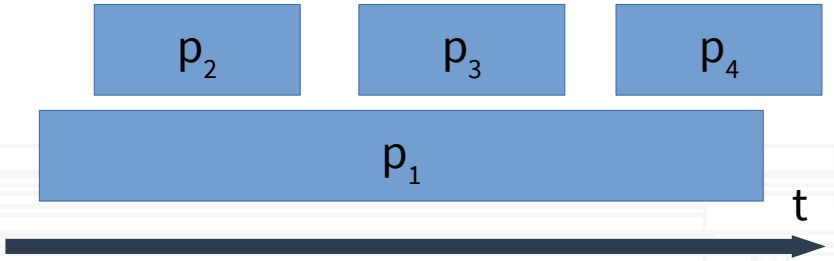
Aquel que dura menos tiempo

Aquel que tiene menos incompatibilidades

Aquel que termina antes

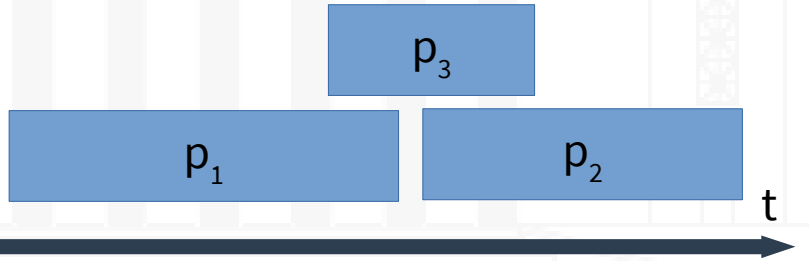
Heuristicas que no funcionan

Aquel que comienza antes



Con esta heurística seleccionaríamos P_1 y el óptimo es $\{p2, p_3, p_4\}$

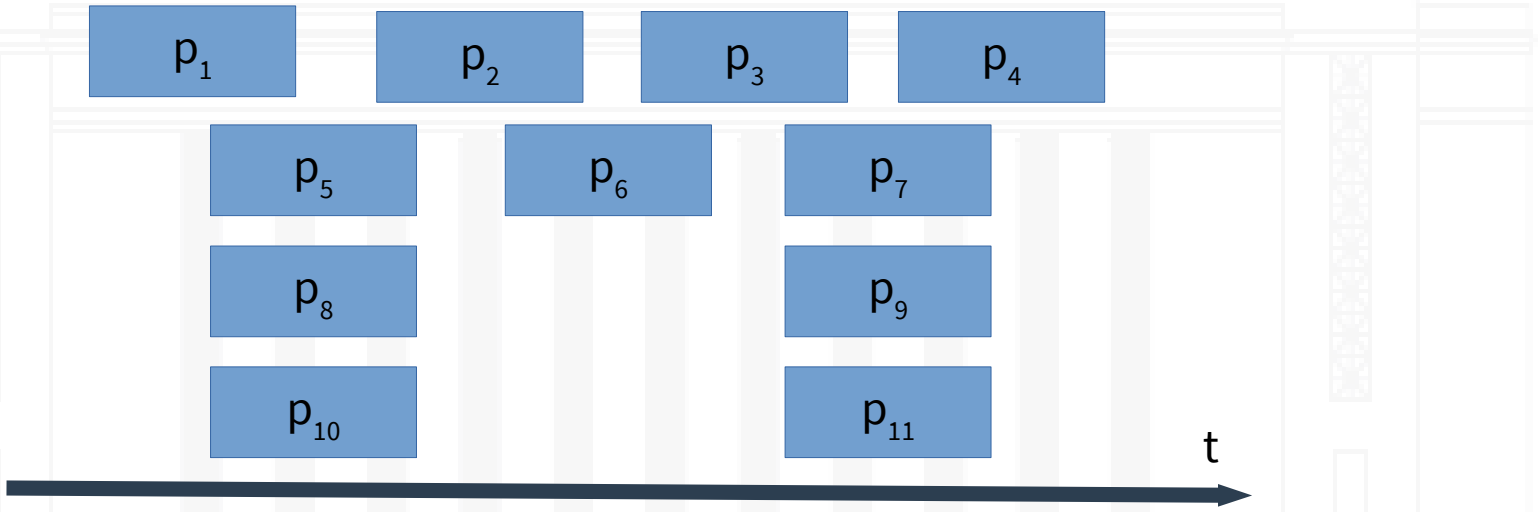
Aquel que dura menos tiempo



Con esta heurística seleccionaríamos P_3 y el óptimo es $\{p_1, p_2\}$

Heuristicas que no funcionan (cont.)

Aquel que tiene menos incompatibilidades



Con esta heurística seleccionaríamos primero P_6 y por lo tanto el tamaño máximo del conjunto final sera 3, el óptimo es $\{p_1, p_2, p_3, p_4\}$ con tamaño 4

Contraejemplos y demostración de optimalidad

En los ejemplos anteriores

Bastaba con encontrar un contraejemplo para desestimar la heuristica

A veces

puede ser difícil encontrar el contraejemplo

Si buscamos y no lo encontramos,

procedemos a intentar probar que utilizando ese criterio de selección, la misma nos lleva a obtener el optimo global

Heuristica: Aquel que termina antes

En este caso encontrar un contraejemplo

Aparenta ser imposible

Por lo tanto intentaremos

Demostrar que usando esa heurística corresponde a la eleccion greedy correcta.

Si demostramos logicamente que el resultado es optimo, entonces tendremos nuestro algoritmo greedy

Describamos como funcionará

este algoritmo

Pseudocódigo

```
Sea P set de pedidos
Sea A subconjunto maximo

Mientras P <> vacio

    Sea i el pedido en P con menor tiempo de finalización

    A = A + {i}

    Quitar de P todos los pedidos incomplatibles con i

Retornar A
```

Análisis del algoritmo: Set compatible

El set retornado por el algoritmo

Es compatible

Esto se obtiene por la naturaleza misma del algoritmo:

En cada iteración se eliminan todos los pedidos incompatibles del seleccionado.

… queda ver si ademas es óptimo

Análisis del algoritmo: Optimalidad

Llamaremos

O a un set optimo

No podemos asegurar que

A=O (podrían existir varias selecciones diferentes de tareas con la misma cantidad de pedidos)

Queremos ver que

|A|=|O|

Para demostrar esto

Utilizaremos la idea de que el algoritmo greedy “se mantiene por delante” de la solución Optima

Optimalidad - Idea

Podemos enumerar los elementos de A

Según el orden en el que fueron seleccionadas $\{i_1, i_2, \ldots i_k\}$ con $|A|=k$

Ademas, podemos enumerar los elementos de O

Ordenados por fecha de inicialización del pedido $\{j_1, j_2, \ldots, j_m\}$ con $|O|=m$

(como los pedidos son compatibles, entonces es lo mismo ordenar por fecha de inicialización o finalización)

Compararemos las soluciones parciales construidas por greedy

Con los segmentos iniciales de la solución greedy

Mostraremos que greedy

“lo está haciendo mejor” de una forma paso a paso. (tiene igual o mas pedidos seleccionados)

Optimalidad: 1er pedido

Analizamos

el primer elemento de O y A

Por como se selecciona en greedy

podemos ver que: $f(i_1) \leq f(j_1)$

Pueden ser el mismo pedido (o dos pedidos que finalizan al mismo tiempo)

En ese caso $f(i_1) = f(j_1)$

Pero nunca puede ser $f(i_1) > f(j_1)$

Seria una violación a la forma de seleccionar en el greedy

Optimalidad: Inducción

Queremos demostrar que

Para todo r ≤ k se cumple que $f(i_r) \leq f(j_k)$

Utilizaremos inducción para probarlo

Para el caso base r=1 esto es cierto

Asumiremos que es cierto para r-1 con r>1 (hipótesis inductiva)

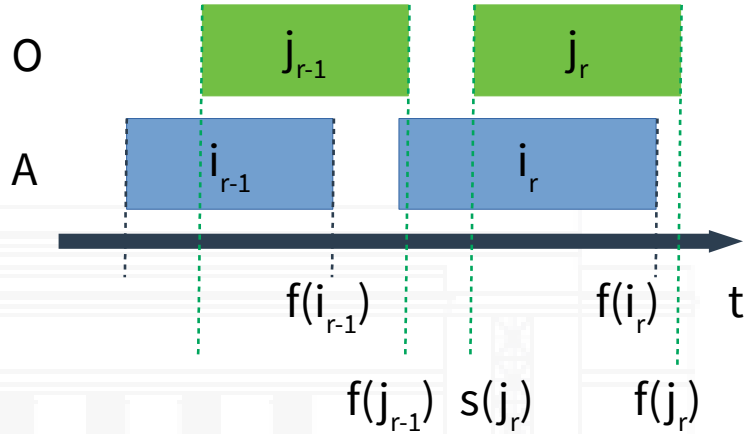
Por lo tanto asumimos $f(i_{r-1}) \leq f(j_{r-1})$

Como O esta compuesto por pedidos compatibles

$f(j_{r-1}) \leq s(j_r)$

Con estas 2 inecuaciones podemos obtener

$f(i_{r-1}) \leq s(j_r)$



Optimalidad: Inducción (cont.)

La inecuación

$f(i_{r-1}) \leq s(j_r)$

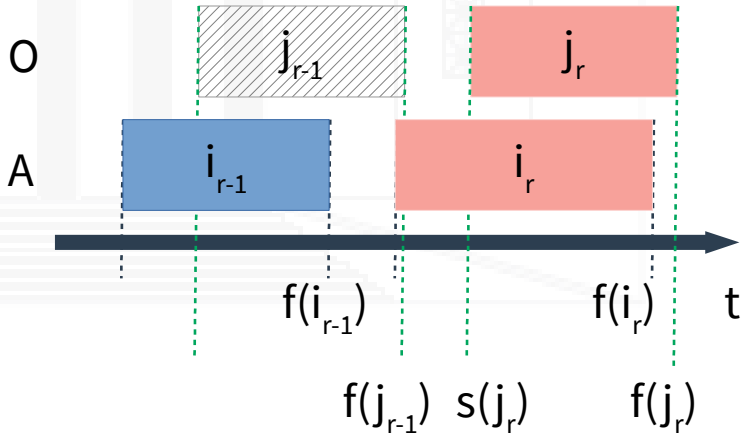
Implica que en el momento que greedy seleccionó i_r

También estaba disponible j_r

Greedy selecciona aquel disponible que termine antes

Entonces $f(i_r) \leq f(j_r)$

Lo que concluye el paso inductivo



Optimalidad: Colorario

Podemos afirmar (en base la demostración inductiva) que el algoritmo greedy retorna un set A óptimo

Si A no es optimo entonces el O debe tener mas pedidos (m > k), |O|=m y |A|=k

Dado que para todo r ≤ k se cumple que $f(i_r) \leq f(j_k)$

Si r=k entonces $f(i_k) \leq f(j_k)$

Como m > k,

Implica que existe en O un pedido j_{k+1} que comienza luego que j_k termina

Demostramos que si j_{k+1} finaliza luego que j_k

Entonces finaliza luego que i_k

Lo que indicaría que greedy, luego de seleccionar i_k

Aun tendría pedidos compatibles para seleccionar y no habría seleccionado ninguno

Lo que seria una contradicción

Implementación

Para implementar de forma eficiente el psudocodigo

Debemos encontrar la forma de recorrer los pedidos de forma conveniente según nuestra heuristica

Se puede ordenar los pedidos por tiempo de finalización

De esa forma elegimos el primero disponible

Luego iteramos

ignorando los incompatibles (comparando la fecha de finalización del último seleccionado con la fecha de inicio del analizado)

Y seleccionando el próximo disponible (pasando a ser el ultimo seleccionado)

Complejidad

El proceso de ordenamiento

Es O(n log n)

La iteración

Es O(n)

Por lo tanto la complejidad temporal total

Es O(n log n)

La complejidad espacial

Es O(n) ← si todos los pedidos son compatibles elijo los n

```
Sea P set de pedidos
Sea A subconjunto maximo

Ordenar P por fecha de finalización

A = A + {P[1]}
finilizacion = P[1].finalizacion
Desde i=2 a n

     Si P[i].inicio >= finalizacion
          A = A + {P[i]}
          finalizacion = P[i].finalizacion


Retornar A
```

Ejemplo

Sean los siguientes pedidos

i	s	f
1	4	6
2	2	5
3	1	3
4	6	8
5	3	7

Los ordenamos por tiempo f

i	s	f
3	1	3
2	2	5
1	4	6
5	3	7
4	6	8

Seleccionamos algorítmicamente

i	s	f	A	fin
3	1	3	si	3
2	2	5	X	3
1	4	6	si	6
5	3	7	X	6
4	6	8	si	8

Siendo la solución optima

{1,3,4}

.UBAfiuba

FACULTAD DE INGENIERÍA

Presentación realizada en Septiembre de 2020