Problemas de Diseño y Análisis de Algoritmos

Problema 1: Tareas y Expertos

Equipo: 3

Miembros:
- Laura Brito Guerrero
- Sheyla Cruz Castro
- Rodrigo García Gómez

Enunciado del Problema:

Imagine una cantidad n de tareas, cada una de ellas con determinada complejidad *ci*, que deben ser realizadas. Para hacer esto se cuenta con un grupo *m* de expertos, caracterizados por sus respectivas capacidades di. Se sabe que el experto i-ésimo puede ejecutar la tarea j-ésima si $di >= cj$. Cada experto consume un día en hacer una tarea y ha de recibir por su trabajo una cantidad *ki* de pesos, que siempre es la misma independientemente del número de tareas que resuelva. El objetivo en este problema es determinar el menor tiempo posible en que se pueden resolver las *n* tareas sin pagar más de *k* pesos. Se asume que las tareas se pueden hacer en cualquier orden y que distintos expertos pueden trabajar al mismo tiempo en sus respectivas tareas.

Abstract

Primeramente lo atacamos con un algoritmo $fuerza\ bruta$ exponencial que analiza todas las combinaciones posibles que resultan de distribuir los distintos expertos a tareas, y da la mejor solución posible. Este algoritmo es ineficiente, por lo cual lo intentamos por otras vías. Una solución consiste en combinar una *mochila* para obtener subconjuntos de elementos, con un algoritmo cuadrático para determinar la mejor ganancia; este resulta $n^3 * k$, también de costo exponencial ya que depende del valor de *k*. Una cuarta idea sería implementar un algoritmo para, dado un grupo de científicos fijo válido y las tareas a completar por estos, encontrar la menor cantidad de días en que pueden lograrlo, de forma polinomial; esta es costosa, pues para determinar dicho grupo tengo que generar todas las combinaciones. Por último vimos una idea *greedy* que consistía en hacer una búsqueda binaria sobre un predicado de la forma *no*,*no*...*si*,*si*... buscando el primer si que responda a la menor cantidad de días que se requieren para realizar las *n* tareas, con un costo de $n^2 \log n$.

Aclaración:

Para cada uno de los tres problemas se implementó un programa *tester generator* que se encarga de generar de forma aleatoria una inmensa cantidad de ficheros con casos de prueba y un *tester* con el cual se implementan estos casos con dos o más métodos distintos. De esta forma es posible comprobar con una altísima seguridad la correctitud de un algoritmo a partir de la comparación de sus resultados con los resultados de otro algoritmo enfocado en la *fuerza bruta* con una complejidad temporal muy superior, pero cuya correctitud es más evidente. Además, de esta forma es fácil comprobar los casos específicos en los que determinado algoritmo falla, ayudando así la búsqueda y arreglo de errores. Estas demostraciones no producen las demostraciones formales necesarias para la correcta resolución de los ejercicios, pero son muy útiles para dirigir correctamente los esfuerzos, ya sea para demostrar uno en el que se tenga una fuerte convicción de correctitud, o para comprobar los motivos por el que otro es incorrecto.

Idea Fuerza Bruta:

(brute1)

La idea consiste en generar todas las posibles asignaciones de expertos a tareas. Serían todas las formas de tomar n tareas del conjunto de tareas(como es condición necesaria que todas las tareas se realicen, se toman todas estas) donde cada una puede tomar m formas distintas,o sea pueden ser realizadas por alguno de los m expertos. De aquí nos quedamos con la combinación que consuma la menor cantidad de días entre todas las posibles distribuciones.

Inicialmente se esperan tres líneas: la primera con la cantidad de tareas a realizar,la cantidad de expertos y el presupuesto del que se dispone, n, m, y k respectivamente; luego una segunda con las capacidades de los expertos; y finalmente las complejidades de cada una de las tareas. Una vez que contamos con todos los datos creamos un array con los números en el intervalo de 0 a $n-1$,o sea de tamaño n,sobre el cual trabajaremos. Por consiguiente generamos todas las posibles distribuciones a través del método $var-rep$.

Toda tarea tendrá asignada un experto y este será único pues una tarea solo puede ser realizada una vez. La distribución admite elementos repetidos, o sea un experto puede realizar más de una tarea. Una vez que se genera una posible distribución se analiza su factibilidad,de esto se encarga el método *analize*. Para ello se comprueba que el experto i-ésimo asignado a la tarea j-ésima tenga capacidad mayor o igual que la complejidad de dicha tarea; y se comprueba además que la suma del dinero que recibe cada experto es menor o igual que el presupuesto con que se cuenta. Una distribución bajo estas re-

stricciones se considera válida. De todas ellas nos quedamos con la que requiera menor cantidad de días. La cantidad máxima de días que requiere cada disposición es igual al máximo de la cantidad de tareas que realiza cada experto, pues este requiere de todo un día para la realización de la tarea y estos pueden trabajar simultáneamente, luego el que más tareas tenga requerirá más días, y dicha cantidad será cota máxima para el tiempo total requerido. Por último nos quedamos con el mínimo de los máximos(duraciones máximas de las distintas posibilidades) pues lo que se quiere es minimizar el tiempo de realización de las n tareas.

La correctitud de esta solución se sustenta en que se analizan todas las posibilidades, garantizando así que de existir una solución se va a encontrar y se queda en todo momento con la más óptima. Este algoritmo es exponencial,genera m^n distintas posibilidades, donde m es el número de expertos y n el número de tareas a realizar. Esta solución es muy ineficiente, se demora considerablemente para pequeños valores de la entrada.

Idea 2:

(calculate-days)

Se implementó un algoritmo para, dado un grupo de científicos fijo válido y las tareas a completar por estos, encontrar la menor cantidad de días en que pueden lograrlo, de forma polinomial. Se tiene n = la cantidad de tareas y m = la cantidad de científicos. Primeramente, es necesario ordenar de forma creciente a los científicos por sus capacidades y las tareas por sus dificultades.

La idea consiste en separar a los científicos por grupos en dependencia de la cantidad de tareas que sean capaces de cumplir. Se mantiene un puntero para científicos j y otro para tareas i. Se avanza el puntero i por las tareas hasta que el científico j no sea capaz de cumplir la tarea i, entonces se agrega al nuevo grupo al científico j y a todos los científicos siguientes que tampoco sean capaces de cumplir dicha tarea, avanzando el puntero j. Los grupos llevarán constancia de la cantidad de tareas que pueden cumplir y la cantidad de científicos que poseen. La conformación de los grupos se realiza en tiempo lineal $m+n$ pues es evidente que se pasa sólo una vez por cada tarea y una por cada científicos.

Todas las tareas realizadas por un grupo se podrán realizar también con los grupos superiores, por tanto se buscará repartir las tareas de forma que si el grupo i realiza sus tareas en un tiempo superior al grupo $i+1$, este aumente sus días en uno para ayudar al grupo i. Este proceso se repetirá cada vez que el grupo i tenga mayor tiempo que el $i+1$. Al realizar esto para todos los grupos, cuando el algoritmo termine, entonces en el último

grupo tendrá la mayor cantidad de días entre todos los demás, y por tanto, como todos los demás grupos finalizarán sus tareas en un tiempo menor o igual que él, esta será la cantidad de días a retornar. Este retorno tendrá la cantidad mínima de días de trabajo, puesto que los días de los grupos superiores aumentarán un día a la vez, mientras que los de grupos inferiores disminuirán en uno o más días, por tanto, si el grupo i se completa en más días que el $i+1$, la cantidad de días que se le aumentan a este, no será mayor que la que tenía el i antes de disminuirle.

A partir de este momento, solo se trabajará con la lista de grupos de científicos recién formada. El ciclo principal iterará por dicha lista de forma descendiente, a partir de la última posición (recordemos que esta lista está ordenada). Primeramente, en cada iteración (siendo la iteración i, que se está analizando el grupo $n-i$), si la cantidad de tareas es mayor que la cantidad de científicos, se divide tareas entre científicos y se guarda en una variable q (esto representa la cantidad de días necesaria para que el grupo termine sus tareas), en caso de haber resto se incrementa q en uno y en una variable r se almacena la diferencia entre los científicos del grupo y dicho resto (esto representa el sobrante de tareas del grupo para realizar las suyas en la cantidad de días q, o sea, si el grupo tiene 2 científicos y 5 tareas, en 3 días realiza sus tareas, y tiene uno de resto para almacenar). Además, si el grupo bajo análisis actual realiza sus tareas en una cantidad de días q mayor que el grupo inmediatamente superior, le pedirá a este una ayuda, incrementando en el grupo superior una variable que almacena las ayudas a prestar al grupo inferior; luego es necesario recalcular q y r del grupo actual pero retirando una cantidad de tareas similar a la cantidad de ayudas de su grupo superior multiplicada por la cantidad de científicos de este, de lo cual se encarga el ciclo principal, que se repite siempre que ocurra una *pedida de ayuda*. Si la cantidad de tareas es menor que la cantidad de científicos, el proceso es similar, con la diferencia de que en lugar de hacer un división, en q se almacenará 1, y en r, la substracción de los científicos por las tareas (si hay 5 científicos y 3 tareas, se cumplen en un día y se guarda 2 de resto para quitarle tareas al grupo inmediatamente menor) . Al terminar el ciclo principal, la solución quedará representada en la cantidad de días q más la cantidad de ayudas prestadas por el grupo de la última posición de la lista (el de las mayores capacidades).

Idea 3:

(knapsack)

Otra solución un poco más eficiente para este problema es combinarlo con una *mochila*

y luego aplicarle un algoritmo cuadrático que garantice obtener la mejor ganancia, sobre los elementos que se encuentran escogidos en la *mochila*, para así garantizar obtener la mayor ganancia. Esta idea se lleva a cabo de la siguiente manera: inicialmente tenemos una mochila de peso k, el cual es el presupuesto destinado para la realización de las tareas. Como cada experto tiene una ganancia ki independientemente de la cantidad de proyectos que realice, esta misma representaría los pesos correspondientes a cada uno.

Como este algoritmo me garantiza que en cada momento exista una repartición válida de expertos, tenemos que ahora distribuir las tareas entre ellos tal que se cumpla que todas las tareas se tomen y de esta forma hallar la mejor combinación tal que el máximo de las tareas entre los expertos sea mínima. Esta combinación asignada resultará la ganancia de esta repartición, y en cada elección de expertos ir guardando este máximo de tareas con la intención de que la ganancia se minimice. Por lo que variaremos la salida del algoritmo *mochila* estándar, puesto que en vez de buscar la ganancia máxima (la cantidad de días máximos) se hallará la mínima.

Para hallar la combinación de las tareas por los expertos de manera eficiente se realizá el siguiente algoritmo: se tiene un arreglo de n tareas con sus respectivas complejidades, y se tienen mk expertos escogidos. Las tareas estarán ordenadas decrecientemente por complejidad, y se ordenarán por capacidad en orden decreciente a los expertos escogidos. Se recorrerán las tareas en ese orden intentando otorgárselas a los distintos expertos, de manera tal que todos los expertos tengan tareas, y siempre se tratará de otorgar la menor cantidad de tareas por experto. Esto se realiza en dos ciclos anidados siempre analizando las tareas a entregar.

Idea 4:

(greedy)

Esta idea se basa en hacer una búsqueda binaria sobre un predicado de la forma *no*,*no*...*si*,*si*... buscando el primer sí que responda a la menor cantidad de días que se requieren para realizar las n tareas. El predicado tiene esta forma pues si se pueden realizar las n tareas en una cantidad i de días, con $i <= n$, en una cantidad mayor también se podrá hacer.

Sea la función definida como sigue a continuación:

$$f(x) = \begin{cases} 0 & si \quad imposible \\ 1 & si \quad posible \end{cases} \tag{1}$$

Donde *posible* significa que se pueden hacer las tareas en x cantidad de días e *imposible* que no se pueden hacer en dicha cantidad.

Se puede ver que esta función es un predicado que aplicado sobre el intervalo $[1,n]$, siendo n la cantidad total de tareas a realizar para una entrada del problema, se obtiene una secuencia de resultados $[0,0,0,...,1,1,...,1]$. Se aprecia además que la secuencia de resultados es bimodal. Analizando el anterior razonamiento se tiene que demostrar que $f(x) = 1 => f(x+1) = 1$ o sea una vez que se puedan realizar las n tareas en x cantidad de días, se pueden realizar en una cantidad de días mayor.

Intentemos demostrarlo por vía directa:
Se tiene que se pueden realizar las n tareas en x cantidad de días, veamos si es posible realizarlas en $x+1$ días, para esto solo tenemos que asignarle $x+1$ tareas al experto de mayor capacidad entre los expertos que participan en la realización de las tareas, estas tareas deben ser las más costosas para asegurar la realización de todas, esto es posible hacerlo pues si el que más capacidad tiene entre ellos no puede hacerlas, nadie podrá, entonces no se podría haber hecho en x días, y además el presupuesto será suficiente, pues no se está variando ningún experto, se están usando los que ya participaban, luego el costo final será el mismo que para x días resolvía. Las tareas que este experto de presupuesto máximo hacía, que son una cantidad menor que $x+1$, pues sino la cantidad de días fuera esta, se le dan a los expertos de los cuales se quitaron las tareas que fueron a parar al experto de costo máximo, esto se puede hacer, pues si se le quito una tarea de mayor complejidad será entonces capaz de hacer una de menor, y además su cantidad de tareas se mantendrá igual, pues se le adicionan la misma cantidad que se le restan. El presupuesto no se ve afectado, pues los intercambios se realizan entre los expertos que ya participaban, por tanto este no varía. La cantidad de tareas máximas que realizará algún experto haciendo esta repartición será $x+1$, pues anteriormente era x, y de las que se modificaron solo una adquirió la cantidad $x+1$, el resto se mantiene igual. Por tanto se pueden realizar las n tareas en $x+1$ cantidad de días a partir de que era posible con x cantidad.

Como lo que se quiere es minimizar la cantidad de días lo que se busca es el primer *si*. Esto se puede hacer en $\log n$, el problema consiste en encontrar el experto capaz de realizar una cantidad de tareas i que coincide con la cantidad d días en cuestión, esta misma cantidad sería igual al máximo de tareas que se le puede asignar a un experto de

modo que se puedan repartir el resto de las tareas entre los expertos que quedan, este experto ya no estaría disponible pues si a él se le pudiesen asignar más tareas aumentaría la cantidad de días.

Se ordenan las tareas decrecientemente por complejidad. Se recorren en dicho orden, intentando asignarle el subconjunto de tamaño m, donde m es la cantidad de días que se determina en la búsqueda binaria, al experto que menos costo tiene(o sea al que menos cobra por realizar una tarea) entre el todos aquellos capaces de realizar todas las tareas del subconjunto. Con comprobar si el experto es capaz de realizar la tarea más costosa es suficiente, pues si puede hacer esta será capaz de realizar el resto. De nadie poder hacer esta tarea se retorna no inmediatamente pues habrá una tarea que nadie podrá realizar, lo cual no es posible. Se comprueba además que lo que cobra este experto sumado al monto total que se va acumulando en cada asignación(variable *mont*) sea menor que el presupuesto, de esto no suceder se retorna *False* inmediatamente, lo cual supone un *no* para mi predicado, pues de no poder el de menor costo de todos los que pueden realizarla, no se podrá pagar tampoco ningún otro experto de los que son capaces de hacerla(o sea no puedo acudir a ningún otro experto que cobre menos, pues este no será capaz de hacerla, sino se encontrara en mi conjunto de candidatos), pues se escogió el que menos cobraba. Una vez asignada esa cantidad de tareas de eliminan de la lista de tareas, al igual que el experto que las realizó, pues estas solo pueden ser realizadas una sola vez, y el experto en cuestión no puede recibir más tareas pues sino la cantidad de dias mínimos aumentaría. Se continúa el recorrido repitiendo el procedimiento anterior, si se consiguen asignar todas, es un *si* de mi predicado, pues todas las tareas se van a realizar, la cantidad de días va a ser m, pues ningún experto recibe más de esa cantidad, y todos reciben tareas que pueden realizar y lo que cobran está por debajo del presupuesto.

Esta idea es greedy, por lo tanto es necesario demostrar su correctitud.

Demostración

Sean las soluciones

$$Opt = \{Y_1, Y_2, ..., Y_n\},\ Y_i \textit{ vectores de la solución óptima Opt y}$$
$$G = \{X_1, X_2, ..., X_n\},\ X_i \textit{ vectores de la solución greedy G.}$$

Ambas soluciones tienen cardinalidad n ya que estos vectores representan a los expertos que realizan la tarea i. Se ha de destacar que las tareas se ordenan por complejidad de manera decreciente, por tanto en la posición i-ésima de la solución se recoge el experto que resolvió la tarea i-ésima en orden decreciente.

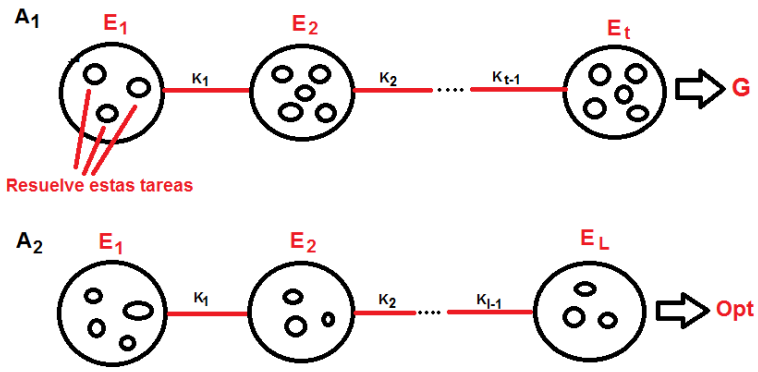

Figure 1: Árboles que representan a las soluciones G y Opt, donde los vértices representan al experto con las tareas que realiza, las aristas representa el costo de los expertos, la arista i es el costo del experto i, y el último experto se enlaza con un nodo fictisio, para poder asignarle el costo al mismo

$$Z_i \textit{ resuelve la tarea de complejidad } c_i, \forall\, Z_i \in \{X_i, Y_i\}$$

Está claro que pueden existir vectores iguales, ya que si las tareas $c_{i_1}, c_{i_2}, ..., c_{i_k}$ las realiza el mismo experto entonces $Z_{i_1}, Z_{i_2}, ..., Z_{i_k}$.

Primeramente se enunciará un lema muy útil en la demostración.

Lema 1:
Si se observa la solucíon del algoritmo greedy G como A_1(figura 1), entonces A_1 es un MST (Arbol abarcador de costo mínimo).
Demostración
La correctitud de la búsqueda del MST, en el algoritmo de Kruskall, se demostró en conferencia.
Lema 1.1: A_1 es un árbol.
Demostración: De la forma que se construye A_1 se tiene que él es un camino simple de longitud t. No existen ciclos en él puesto que a los expertos escogidos en el algoritmo se le asignan las tareas, y estos constituyen los nodos (vértices), no habrá una arista que comunique el vertice E_i con alguno de los anteriores utilizado en el camino, y esto no sucede porque significa que existe más de un experto realizando al menos una tarea en común, y esto no sucede porque a la vez que se asigne una tarea, la misma no se vuelve a analizar. Como A_1 es un camino simple entonces A_1 es un árbol, donde los nodos extremos tienen degree 1 y los interiores degree 2.

Supongamos que A_1 no es un MST. Entonces sea T un MST de G tal que $w(T) \leq k$.
Como A_1 no es un MST se cumple que $W(T) \leq w(A_1) \leq k$.
Si analizamos la idea del algoritmo greedy es fácil de percatar que al organizar las tareas por complejidades y a los expertos por capacidades de maner decreciente, se tiene un análisis de subconjuntos. Esto se explica de la siguiente forma, al escoger las x tareas que se van a asignar, se escogen un grupo de expertos capaces de ejecutarlas, pero de todos ellos se escogerá el que menos costo tenga (menos le paguen).
Recorramos el camino simple de T y A_1 por paso. Existe al menos una arista tal que $w(e_1) > w(e_2)$, donde $e_1 \in A_1$ y $e_2 \in T$. Estas aristas existen puesto que A_1 no es un MST. Esto significa que para este conjunto de tareas se escogió un experto en A_1 con presupuesto $>$ que el de T. Hasta el momento los expertos que se escogen en T son los mismos que tiene A_1. En toda iteración A_1 analiza los expertos de mayores capacidades: sea E_i con $k_i = e_1$, $E_i \in A_1$ y E_j con $k_j = e_2$, $E_j \in T$ donde $k_i > k_j$.
Se toman tres casos:

1: $d_i > d_j$
Es trivial percatarse que este subconjunto i tiene el bloque de mayores capacidades sin analizar, pero los $i-1$ subconjuntos anteriores tienen mayor capacidad que él, ya que las tareas están ordenadas decrecientemente por complejidad y separadas en intervalos de a lo sumo cardinalidad x. Por lo que se ramifucan dos subcasos:
1.1: $d_i > d_k \forall k \in N, 0 \leq k \leq i-1, d_k \in A_1$.
Si esto ocurre es porque en análisis anteriores se decidió tomar un experto con menor capacidad pero capaz de realizar las tareas del bloque k.
1.1.1: $d_k \geq d_j$
Supongamos que d_j no puede con las tareas de d_i. Significa que el máximo de las complejidades tomadas por $E_[i]$ es mayor que $d_[j]$. Si esto ocurre tratemos de colocar la tarea con complejidad máxima de E_i (llamémosle $c^{'}$) para alguno de los anteriores. Es imposible lo anterior porque los expertos escogidos tienen asignadas x taraes y no le caben más.
Como T está ordenado igualmente por capacidades y complejidades, se tiene que d_j no es válido para sus tareas asignadas, lo cual es una contradicción porque T es válido. Por lo tanto d_j puede con las tareas de d_i, nuevamente contradicción aquí, ya que A_1 hubiera escogido a E_j y no a $E_[i]$.
1.1.2: $d_k \leq d_j$
Si d_k pudo hacer tareas de mayor complejidad entonces $d_[j]$ puede hacer cualquier otro bloque posterior de tareas, lo cual es un contradicción ya que A_1 tomaría a E_j en lugar de E_i.

1.2: Si d_i es el mayor en el paso i
Si esto sucede significa que en T existe un experto con capacidad d_j que realiza las mismas tareas que d_i, donde $d_j < d_i$, como T es válido entonces se genera una contradicción, ya que A_1 tomaría a E_j, el cual es capaz de resolver ese bloque de tareas con un presupuesto menor.

2: $di < d_j$
Significa que si el subconjunto de tareas tomados de cardinalidad a lo sumo x, que tiene las mayores capacidades después del análisis de los $i-1$ subconjuntos anteriores, se puede realizar con un experto de capacidad menor o igual que otro que tenga mayor capacidad y menor presupuesto, entonces d_j también puede ejecutar ese subconjunto de tareas. Como en A_1 se toman los expertos con capacidad válida y menor presupuesto (Contradicción: A_1 tomaría E_j desde un principio).

3: $d_i = d_j$
Directamente se genera una contradicción, ya que A_1 elige el de menor presupuesto, o sea, E_j.

Por tanto, tras una cantidad finita de pasos se demuestra que A_1 es un MST.

Tomemos la primera posición i tal que $Y_i \neq X_i$. Tengamos presente que un bloque completo de G es el mismo experto(x expertos). Se tienen dos casos:
1: $X_i.k \leq Y_i.k$
Como $X_i \in V(A_1)$, entonces por *Lema*1 se cumple que $Y_i.k > X_i.k$, por lo que $Y_i.k - X_i.k$ es el excedente de presupuesto que tiene Opt con respecto a G. La idea es quitar este excedente a otros de k en adelante de manera que siga quedando factible.
Esto es válido porque como las tareas están ordenadas decrecientemente por complejidad entonces si se le asigna el vector $Y_i = X_i$ quedando el Opt factible ya que si X_i pudo realizar esta tarea con menor capacidad entonces se cambian y disminuye el presupuesto asignado al excedente de este paso. De esta manera también se garantiza que el predicado sea válido y no exista un experto con más de x tareas.
2: $X_i.k > Y_i.k$
Si esto ocurre se cumple que Y_i fue escogido necesariamente antes por G (por ser A_1 MST), de lo contrario $Y_i \in G$ por propiedad de A_1, por tanto Y_i no estaba disponible para la tarea. Para no llegar a una contradicción con respecto a $w(A_1)$ ya que es el menor

costo posible de tomar. Como en las anteriores iteraciones donde se cumplía el caso 1 y realizábamos las modificaciones con el objetivo de disminuir $w(A_2)$.
Como $w(A_1) \leq w(A_2)$ entonces es válido asignar $Y_i = x_i$, ya que el presupuesto que se quita anteriormente se va agregando de la menor manera posiblegarantizando que no ocurra: $w(A_2) < w(A_1)$ lo cual es un absurdo, por ser A_1 MST.
Por lo tanto se le agrega a $w(A_2)$ el monto de $X_i.k - Y_i.k$.

Tras una cantidad finita de pasos tenemos que $w(A_1) = w(A_2)$, equivalente a $G = Opt$ lo cual demuestra que G también es una solución óptima.

Este algoritmo tiene una complejidad polinomial, realiza $n^2 \log n$ operaciones, la búsqueda binaria en $\log n$ buscando el primer *si*, y luego para responder el predicado, hace en el peor de los casos una pasada por todas las tareas, y por cada una de estas recorre los m expertos buscando el de menor costo. Esto sería $m * n$, luego de no ser iguales, entonces uno es mayor que otro, sea el mayor m sin pérdida de generalidad, se puede decir que este es $n + k$, con $k < n$, luego esto es $n^2 + k$, quedándonos con el máximo se tiene entonces $n^2 \log n$.