Práctica 3 – 1ª parte

PROBLEMA 2.1 – CONTENEDORES EN UN BARCO

MEMORIA

Realizado por :
José Javier Alonso Ramos
Ángel Carmona García
Andrés Fernández Ortega
Paula Iglesias Ahualli
Carlos Núñez Molina

Contenido

1. Algoritmo greedy que maximiza el número de contenedores en el barco.....3
   1.1 Descripción.....3
   1.2 Pseudo-código.....3
   1.3 Otimalidad.....4
2. Algoritmo greedy para maximizar el beneficio.....5
   2.1 Descripción.....5
   2.2 Pseudo-código.....5
   2.3 Explicación.....6
   2.4 Ejemplo de no optimalidad.....6
   2.5 Soluciones óptimas del algoritmo.....7

Maximizar el número de contenedores

Descripción

Candidatos: contenedores no escogidos todavía

Seleccionados: contenedores escogidos hasta el momento

Función objetivo: maximizar el número de contenedores (maximizar cardinalidad del conjunto de seleccionados)

Función selección: escoger, de entre los candidatos, el de menor peso, p_i

Función de factibilidad: al incluir el contendor c_i de peso p_i, la suma de los pesos de los contenedores seleccionados no sea mayor que la capacidad del barco

Función solución: no se puede incluir ningún contenedor de los candidatos sin que se exceda la capacidad del barco

Pseudo-código

s [ ] : conjunto de contenedores seleccionados (dentro del barco)

c [ ] : conjunto de contenedores candidatos (aún no seleccionados)

x : elemento tipo *Contenedor → contenedor.peso ; contenedor.beneficio*

peso_actual : peso total de los elementos que se van añadiendo a *s [ ]*

k : carga máxima del barco

```
s[] = null;
sort(c);          //ordenamos candidatos según su peso
                    tal que c[i].peso <= c[i+1].peso

while( !c.empty() && peso_actual <= k){
      x=c[0]; //Función selección: escogemos el contenedor de menor peso
      c.pop_front(); //Eliminamos el contenedor que acabamos de escoger
                           de la lista de candidatos.

      if (peso_actual + x.peso <= k) //Función factibilidad
      {
            s.push_back(x); //Función solución
            peso_actual += x.peso;
      }
}
return s;
```

- **Demostración de Optimalidad**

  Sea $S = \{c_1, c_2, \ldots, c_n\}$ el conjunto formado por los contenedores de la solución greedy, ordenados en orden de pesos no decrecientes.

  Supongamos que S no es una solución óptima al problema.

  Existirán, por tanto, una o más soluciones óptimas al problema, diferentes de S.

  Llamemos S' a la solución óptima que contenga el mayor número de elementos coincidentes con S y que estén en la misma posición (en este caso hasta el elemento k).

  $S = \{c_1, c_2, \ldots c_k, c_{k+1}, \ldots, c_n\}$ $S' = \{c_1, c_2, \ldots c_k, c'_{k+1}, \ldots, c'_m\}$

  De forma que hasta k, los contenedores de S' coinciden con los de S y ambos son escogidos en el mismo orden.

  Debido a cómo el algoritmo greedy escoge los candidatos (de los que quedan, siempre el de menor peso) sabemos que entre dos contenedores hasta el número k (tanto en S como en S') no hay otro contenedor de peso intermedio, pues habría sido escogido antes que el segundo contenedor.

  Por todo lo anterior, sabemos que $c_{k+1} \neq c'_{k+1}$ y que $p_{k+1} \leq p'_{k+1}$ (por cómo escoge la solución greedy los elementos, siempre el de menor peso).

  Por tanto, podemos sustituir en S' a c'_{k+1} por c_{k+1}, en el caso de que ese contenedor no estuviera en S', y seguiría siendo S' una solución válida (no se excedería la capacidad).

  En el caso de que c_{k+1} ya perteneciera a S', en vez de sustituir los contenedores cambiamos la posición de los contenedores c_{k+1} y c'_{k+1} en S' (cambia el orden en el que se escogen).

  En ambos casos S' pasaría a ser el siguiente conjunto: $S'' = \{c_1, c_2, \ldots c_k, c_{k+1}, \ldots, c'_m\}$.

  Pero esto es una contradicción, porque dijimos que S' era la solución óptima que más elementos consecutivos y coincidentes tenía con S y ahora resulta que S'' tiene un elemento más (c_{k+1}).

  Debido a esto, la hipótesis de partida (que S no es una solución óptima al problema) es falsa y S, la solución greedy, es siempre una solución óptima al problema.

Maximizar beneficio obtenido

Descripción:

Candidatos: contenedores no escogidos todavía.

Seleccionados: contenedores escogidos hasta el momento.

Función objetivo: maximizar el beneficio obtenido.

Función selección: escoger, de entre los candidatos, el que mayor razón de beneficio por peso tiene, ci.

Función de factibilidad: al incluir el contendor c_i de peso p_i, la suma de los pesos de los contenedores seleccionados no sea mayor que k, la capacidad del barco.

Función solución: no se puede incluir ningún contenedor sin que se exceda k, la capacidad del barco.

Pseudo-código

```
Contenedor:= Registro
      Peso := integer
      Beneficio := integer
Fin Registro
CapacidadBuque := integer K
PesoCargado := integer 0
ListaCargamento := Array de Contenedor
N := integer Número total contenedores
Lleno := boolean False
//Proceso MaximizarBeneficio
Contenedores C[N] ; // Aquí se encuentran todos los Contenedores
con sus Pesos y beneficios

Ordenar(C) ; // Ordenamos los contenedores siendo el ultimo
el de mayor relación Beneficio/peso
n=N-1 //Función selección
While lleno is False do
      //Función factibilidad
      If (PesoCargado + C[n].peso) &lt;= CapacidadBuque do
            Agregar(ListaCargamento, C[n]) ;
            PesoCargado += Peso C[n];
            Eliminar(C,n);
            if PesoCargado == CapacidadBuque do //Función objetivo
                  lleno = true; // en este caso se alcanzaría el máximo
                                        beneficio
            done
      done
```

```
        n-- ; //Función selección

        if n <=0 do
              lleno = true ; // no hemos podido cargar más el Buque y queda
                                espacio libre
        done
done
```

Explicación

Tenemos un conjunto de contenedores de los que sabemos su peso y beneficio y que ordenamos de forma no decreciente según su cociente beneficio/peso.
Entonces vamos eligiendo el contenedor de la última posición, el de mayor valor de cociente beneficio/peso y comprobamos si sumando su peso al peso del resto de contenedores que ya llevamos, el peso del cargamento es menor que la capacidad máxima del buque.
si cabe lo cargamos en el buque y continuaremos con el siguiente contenedor con más valor beneficio/peso, si no cabe, probaremos con el siguiente directamente.
Este proceso se repite hasta que se consiga llenar el buque pues el peso máximo coincide con el peso del cargamento o bien, ya no quedan contenedores a seleccionar.

Ejemplo de no optimalidad

Supongamos que en el buque podemos incluir contenedores hasta llegar a un peso k=10. Y tenemos los siguientes contenedores:
C = {1, 2, 3]
Con sus respectivos pesos:
P = {6, 5, 5}
Sus respectivos beneficios:
B = {12, 7, 6}
Sus cocientes beneficio/peso:
G = {2, 1.4, 1.2}
Ordenando los contenedores según su cociente peso/beneficio de forma no creciente y representados de la forma: identificador (beneficio/peso) nos quedaría así:
C = {1(2), 2(1.4), 3(1.2)}
El algoritmo greedy escogería el primero, con cociente beneficio/peso = 2, de forma que no podría añadir más contenedores pues es de peso 6 y cualquiera de los otros dos al ser de peso 5 superarían k=10. Así el beneficio obtenido con el algoritmo greedy en este caso sería el beneficio del único contenedor que incluiríamos cuyo valor es 12.

La solución óptima al problema sería elegir los otros 2 contenedores, que igualan la capacidad máxima del buque, cuya suma de sus beneficios sería 13, superando el resultado que nos daba el algoritmo greedy.

- ¿Cuándo da soluciones óptimas este algoritmo?

Vamos a demostrar que este algoritmo da soluciones óptimas cuando la suma del peso de los contenedores que se van seleccionando coincide justo con la capacidad máxima del buque.

Lo hacemos por reducción al absurdo:

Sea S = $\{c_1, c_2,..., c_n\}$ el conjunto formado por los contenedores de la solución greedy cuya suma de pesos coincide con la máxima capacidad del buque.

Supongamos que S no es una solución óptima al problema.

Existirán, por tanto, una o más soluciones óptimas al problema, diferentes de S.

Llamemos S' a la solución óptima que contenga los contenedores cuya suma de beneficios es la máxima posible, la solución óptima.

Esta solución S' contiene elementos coincidentes con S y que estén en la misma posición (en este caso hasta el elemento k) ya que están ordenados en orden no decreciente según su cociente beneficio/peso.

S = $\{c_1, c_2,... c_k, c_{k+1},..., c_n\}$ S' = $\{c_1, c_2,... c_k, c'_{k+1},..., c'_m\}$

De forma que hasta k, los contenedores de S' coinciden con los de S.

El algoritmo greedy escoge el candidato con mayor cociente beneficio/peso y sabemos que entre dos contenedores hasta el número k (tanto en S como en S') no hay otro contenedor con valor de este cociente intermedio, pues habría sido escogido antes.

Por todo lo anterior, sabemos que $c_{k+1} \neq c'_{k+1}$ y que $b_{k+1}/p_{k+1} \geq b'_{k+1}/p'_{k+1}$.

Pero no solo sabemos esto, debido a que a partir de la posición k se escogen en la solución greedy los contenedores con mayor densidad beneficio/peso, a partir de esa posición todos los contenedores de la solución greedy, S, tienen mayor densidad que los escogidos en la solución S' excepto que la solución greedy en algún momento no haya podido introducir un contenedor de mayor densidad debido a que no cabía en el buque.

Aunque haya introducido contenedores de menor densidad, a partir del valor k, la media ponderada de la densidad beneficio/peso por peso será mayor en la solución greedy que en cualquier otra solución pues los contenedores que ha tenido que meter de menor densidad serán de menor peso e influirán menos al cálculo de la media ponderada.

Entonces:

$$media_{ponderada} = \sum_{k}^{j} (densidad * peso) / peso_{total}$$

Media_ponderada_greedy >= Media_ponderada_S'

A su vez sabemos que desde el contenedor k, debido a que esta solución greedy llena el barco, el peso total del resto de contenedores de S' tendrá menor peso que el de los contenedores de la solución greedy.

$$Peso_{total\ greedy} \geq Peso_{total\ s}$$

Podemos calcular el beneficio total como:

$$beneficio_{total} = media_{ponderada} * peso_{total}$$

Al ser en la solución greedy mayor o igual tanto la media ponderada como el peso_total:

$$beneficio_{total\ greedy} \geq beneficio_{total\ S'}$$

De forma que, en el mejor caso, si la solución greedy llena el barco, cualquier otra solución S' será como mucho tan buena como la solución greedy pues con cualquier transformación de n bloques de la solución greedy por otros m bloques, estos m bloques supondrán, como mucho, tanto beneficio como suponían los n bloques a sustituir. Llegamos a la contradicción de la existencia de una S' distinta de S con más beneficio quedando demostrada que la solución greedy es óptima siempre que llena el barco.

Podríamos generalizar más esta conclusión:

$$\frac{beneficio\,total_{greedy}}{beneficio\,total_{s'}} = \frac{\frac{media\ ponderada_{greedy}}{media\ ponderada_{s'}} * peso\ total_{greedy}}{peso\,total_{s'}}$$

Si el beneficio total de la solución greedy es mayor, el cociente de beneficios será mayor que 1, esto ocurre cuando:

$$\frac{\frac{media\ ponderada_{greedy}}{media\ ponderada_{s'}} * peso\,total_{greedy}}{peso\,total_{s'}} \geq 1$$

$$\frac{media\,ponderada_{greedy}}{media\,ponderada_{s'}} \geq \frac{peso\,total_{s'}}{peso\,total_{greedy}}$$

De forma general, podemos decir que siempre que se cumpla esta condición, la solución que da el algoritmo greedy es óptima aunque no llene el buque.