UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Algorítmica (11593)

Voraces y RP–23 de enero de 2018

etsinf
Escuela Técnica Superior de Ingeniería Informática

NOMBRE: _____ **NÚM.:** _____

1 **4 puntos**

Se quiere planificar la compra de piensos compuestos para la alimentación de una ganadería. Las necesidades alimenticias requieren una cantidad mínima MIN de calorías. Se dispone de una oferta de N lotes de pienso. Para cada lote i se indica el número de calorías c_i y su precio p_i. Diseña un algoritmo de Ramificación y Poda que seleccione el subconjunto de lotes de pienso que satisfaga nuestras necesidades mínimas de calorías y que minimice el coste total. Contesta a los siguientes apartados:

a) Expresa el problema en términos de optimización: expresa formalmente el conjunto de soluciones factibles, la función objetivo a maximizar y la solución óptima buscada.
b) Describe los siguientes conceptos sobre los estados que serán necesarios para el algoritmo:
   1) Representación de un estado (no terminal) y su coste espacial.
   2) Condición para que un estado sea solución.
   3) Identifica el estado inicial que representa todo el conjunto de soluciones factibles.
c) Define una función de ramificación. Contesta a las siguientes cuestiones:
   1) Explica la función.
   2) Define la función (en python o en lenguaje matemático).
   3) Calcula el coste temporal.
d) Diseña una cota optimista no trivial. Contesta a las siguientes cuestiones:
   1) Explica la cota (caso general, de un estado intermedio).
   2) Explica el cálculo de la cota del estado inicial.
   3) Define la cota (en python o en lenguaje matemático). Calcula el coste temporal.
   4) Estudia si se puede mejorar el cálculo de la cota, haciéndolo incremental. Define la cota así definida (en python o en lenguaje matemático). Calcula el coste temporal.

Solución: La representación de los estados es similar al problema de la mochila discreta visto en clase. El conjunto de soluciones factibles serán las colecciones de lotes que sumen un mínimo de MIN calorías:

$$X = \{(x_1, \ldots, x_N) \in \{0,1\}^N \mid \sum_{i=1}^{N} x_i c_i \geq MIN\}$$

es decir, $x_1 = 1$ indica que se ha seleccionado el lote de piensos i y $x_i = 0$ indica que no se ha seleccionado. La restricción impuesta es que la suma de las calorías de los lotes seleccionados debe alcanzar o superar la cantidad MIN de calorías.

La función objetivo a minimizar es el precio de los lotes seleccionados:

$$f((x_1, \ldots, x_N)) = \sum_{i=1}^{N} x_i p_i$$

Y la solución óptima buscada es la selección de lotes que cumple la restricción de igual o superar las calorías exigidas y que minimiza el precio:

$$\hat{x} = \operatorname*{argmin}_{x \in X} \sum_{i=1}^{N} x_i p_i$$

Un estado $(x_1, \ldots, x_k)$ es solución cuando $k = N$ y se cumplan la restricción $\sum_{i=1}^{N} x_i c_i \geq MIN$. La secuencia de longitud 0 representada como $x = (?)$ será el estado inicial. Un estado $(x_1, \ldots, x_k)$ con $k < N$ será un estado no terminal, y representa todas las soluciones alcanzables que lo contienen con ese prefijo, y tendrá asociado un coste espacial $O(N)$.

En la ramificación siempre obtendremos 2 estados: no comprar el siguiente lote y comprarlo. En realidad, se puede ajsutar más y:

- sólo obtener el hijo con $x_{k+1} = 0$ si con los lotes aún no considerados se pueden alcanzar las calorías mínimas
- sólo obtener el hijo con $x_{k+1} = 1$ si con la parte conocida aún no se han alcanzado las calorías mínimas

En el caso más general, ramificamos un estado (x_1, x_2, x_k), con $k < N$ de la forma siguiente:

$$branch((x_1, x_2, \ldots, x_k, ?)) = \{(x_1, x_2, \ldots, x_k, 0, ?), (x_1, x_2, \ldots, x_k, 1, ?)\}$$

Posibles cotas optimistas:

- Una cota trivial. Suponer que no hay que llegar a las calorias mínimas. Entonces la mejor solución alcanzable es no comprar nada más, y el la cota es el precio de los lotes que llevamos hasta el momento.

$$cota\_inferior(x_1, x_2, \ldots, x_k, ?) = \sum_{i=1}^{k} x_i p_i$$

  - En el estado inicial: $cota\_inferior(?) = 0$
  - En un estado intermedio se puede calcular con coste constante de forma incremental:

$$cota\_inferior(x_1, x_2, \ldots, x_k, 1, ?) = cota\_inferior(x_1, x_2, \ldots, x_k, ?) + p_{k+1}$$

$$cota\_inferior(x_1, x_2, \ldots, x_k, 0, ?) = cota\_inferior(x_1, x_2, \ldots, x_k, ?)$$

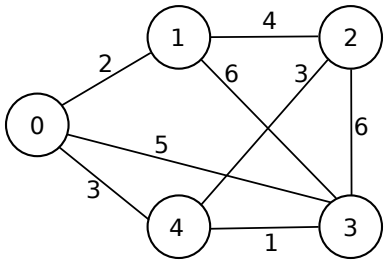
- Una cota inferior más ajustada sería asumir que las calorías que quedan por alcanzar hasta la cantidad MIN se pueden conseguir resolviendo el "problema de la mochila con fraccionamiento" con los lotes que quedan por considerar:

  $$cota\_inferior(x_1, x_2, \ldots, x_k, ?) = \sum_{1 \leq i \leq k} x_i p_i + MochilaConFraccionamiento(MIN - \sum_{1 \leq i \leq k} x_i c_i, \{c_{k+1}, \ldots, c_N\})$$

  donde $MochilaEnteraConFraccionamientoo(MIN - \sum_{1 \leq i \leq k} x_i c_i, \{c_{k+1}, \ldots, c_N\})$ va añadiendo lotes desde la posición $k+1$ hasta que se pueda, asumiendo que están ordenados por menor precio por caloría. (Esta ordenación se puede hacer en un preproceso con un coste $O(N \log N)$.) Cuando el siguiente lote sobrepase las calorías, se compra la parte proporcional. El cálculo de esta cota es $O(N)$.

2 **3 puntos**

Aplicar la técnica de ramificación y poda con poda explícita al problema del viajante de comercio: Dado un grafo no dirigido y ponderado, encontrar un ciclo simple de coste mínimo que pase por todos los vértices una sola vez. Muestra la ejecución sobre el siguiente grafo.



Ten en cuenta:

a) Como lo que buscamos es el ciclo simple de menor coste, es indiferente comenzar en un vértice u otro, por lo que comienza en el vértice 0.
b) Utiliza como cota inferior el peso del camino recorrido hasta el momento más, para la parte desconocida, el peso de la arista de menor coste que incide en cada vértice no visitado, incluyendo al último vértice del camino recorrido hasta el momento.

Responde a las siguientes cuestiones:

a) Explica brevemente cómo vas a representar: el estado inicial, un estado incompleto y un estado solución. Pon un ejemplo sobre la instancia.
b) Explica cómo calculas la cota inferior para el estado inicial. ¿Qué coste tiene?
c) Explica cómo calculas la cota inferior para un estado incompleto y cómo puedes calcularla de forma incremental en un estado hijo. ¿Qué coste tiene?
d) Para la traza sobre la instancia presentada, ten en cuenta lo siguiente: La traza se debe mostrar como el *conjunto de estados activos* de cada iteración del algoritmo indicando la cota inferior de cada estado (ej: como superíndice) y subrayando el estado que se selecciona para la siguiente iteración. Hay que indicar también si se actualiza la variable mejor solución y la poda explícita u otras podas.

Solución: El estado inicial será $(0, ?)$, un estado incompleto representará un camino incompleto y un estado solución será un camino que parte del vértice 0, recorre todos sin repetir ninguno y vuelve al 0.

En la ramificación, asumiremos que cuando el camino recorrido tiene talla $|V|-1$, se obtiene directamente, si existe, el ciclo.

No se puede encontrar una solución inicial con un coste razonable, por lo que la variable mejor solución queda: $x = None$ y $fx = inf$

$A_0 = \{(0?, 9)*\}$

$A_1 = \{(01?, 9)*, (03?, 12), (04?, 10)\}$

$A_2 = \{(012?, 11), (013?, 13), (03?, 12), (04?, 10)*\}$

$A_3 = \{(012?, 11), (013?, 13), (03?, 12), (042?, 12), (043?, 10)*\}$

$A_4 = \{(012?, 11)*, (013?, 13), (03?, 12), (042?, 12), (0431?, 15), (0432?, 15)\}$

$A_5 = \{(0123?, 14), (0124?, 11)*, (013?, 13), (03?, 12), (042?, 12), (0431?, 15), (0432?, 15)\}$

Al ramificar el estado seleccionado se obtiene la primera solución, (012430), con un valor de 15, se actualiza la variable mejor solución con $fx = 15$, por lo que se entra en el proceso de poda explícita y se elimina aquellos estados con una cota mayor o igual a 15, quedando:

$A_5 = \{(0123?, 14), (013?, 13), (03?, 12), (042?, 12)*\}$

$A_6 = \{(0123?, 14), (013?, 13), (03?, 12)*, (0421?, 13)\}$. El estado (0423?) no se guarda pues tiene una cota igual a 15.

$A_7 = \{(0123?, 14), (013?, 13), (034?, 12)*, (0421?, 13)\}$. Los estados (031?) y (032?) tienen cota superior a 15.

$A_8 = \{(0123?, 14), (013?, 13), (0342?, 14), (0421?, 13)*\}$

Al ramificar el estado seleccionado se obtiene la solución, (042130), con un valor de 21, que se desecha.

$A_9 = \{(0123?, 14), (013?, 13)*, (0342?, 14)\}$

$A_{10} = \{(0123?, 14), (0134?, 13)*, (0342?, 14)\}$. El estado (0132?) se desecha.

Al ramificar el estado seleccionado no se obtiene ninguna solución y queda:

$A_{10} = \{(0123?, 14)*, (0342?, 14)\}$

Se obtiene la solución (012340) de coste 16, que se desecha y queda:

$A_{11} = \{(0342?, 14)*\}$

Se obtiene la solcuión (034210) de valor 15, que no mejora la actual. La lista de estados activos se vacía y el algoritmo termina.

3 **3 puntos**

Todos los años se procede a leer el Quijote en el Día del Libro en el Círculo de Bellas Artes de Madrid. Se han presentado N candidaturas de personas que desean leer una parte, pero son muy exigentes: cada candidato i está dispuesto a leer únicamente si le asignan los párrafos desde p_i hasta f_i. Debes realizar una selección intentando contentar al mayor número de candidatos, sabiendo que las partes del libro no seleccionadas por estos podrán ser leídas sin problema por otros voluntarios.

Se pide realizar una función Python que reciba la lista de las N candidaturas:

$$C = \{(p_1, f_1), (p_2, f_2), \ldots, (p_N, f_N)\}$$

y que devuelva una lista de las candidaturas seleccionadas. Sigue una estrategia voraz e indica el coste del algoritmo desarrollado y si resuelve o no este problema de manera óptima.

```
C = [(23,40), (12,50), (4,8), (10,12), (20,25)]
print("Los candidatos seleccionados leerán los siguientes pasajes:")
print(seleccionar(C))
```

Solución: Claramente este problema se corresponde al problema visto en voraces llamado "selección de actividades". Para este problema se conoce una estrategia voraz óptima consistente en elegir las actividades que no solapen ordenadas *por instante de finalización* (elegir primero las que terminen antes).

```
def actselection(C):
  x  = set()
  if len(C)>0:
    t2 = min(s for (s,t) in C)
    for (s,t) in sorted(C,key=lambda (s,t): t):
      if t2 <= s:
        x.add( (s,t) )
        t2 = t
  return x
```