EI1022/MT1022 - PROBLEMAS SESIÓN 6 – RAM. Y ACOTACIÓN MOCHILA Y *BIN PACKING*

13-Nov-2018

Ramificación y acotación – David Llorens

El problema de la mochila (*knapsack*)

El problema de la mochila (*knapsack problem*)

- El problema:
  - Disponemos de N productos donde cada producto tiene un peso w_i y un valor v_i. Además disponemos de una mochila con capacidad de carga W.
  - Queremos cargar la mochila, sin sobrepasar su capacidad de carga, de forma que el valor de lo que contenga sea máximo.
  - NO podemos fraccionar los productos.

- El **conjunto de soluciones factibles:**

$$X = \left\{ (x_0, x_1, \dots, x_{N-1}) \in \{0,1\}^N \mid \sum_{0 \le i < N} x_i w_i \le W \right\}$$

Versión de partida (bab_knapsack.py)

- En el aula virtual dispones de la versión del programa vista en la clase de teoría.

- Modifica esta versión para que incluya las siguientes cotas mejoradas:
  - Cota pesimista: Utiliza un algoritmo voraz para encontrar una solución.
  - Cota optimista: Utiliza un algoritmo voraz para encontrar la solución óptima al problema de la mochila continua.

- Para facilitar la implementación y no tener que trabajar con un nivel adicional de índices, asumiremos que los objetos están ordenados de mayor a menor ratio valor/peso.

El problema de empaquetado (*binpacking*)

El problema del empaquetado (*bin packing*)

- Tenemos N objetos de peso w_i y queremos cargarlos en el menor número posible de contenedores, todos con una capacidad de carga C. Asumimos que ningún objeto pesará más de C.

- Expresaremos una solución como una tupla $(x_0, x_1, \ldots, x_{N-1})$ donde x_i indica el contenedor donde se cargará el objeto i-ésimo.

Formalización

- Asumiremos que los contenedores se numeran del 0 en adelante y que no hay ninguno vacío.

- El conjunto de soluciones factibles, X, es el conjunto de tuplas de N enteros no negativos, tales que la suma de los pesos cargados en cada contenedor sea menor o igual a C.

- Dada una solución (tupla de N enteros), el número de contenedores será el entero máximo de dicha tupla más uno (pues el primer contenedor es el 0). Por lo tanto, la función objetivo será:

$$f\big((x_0, x_1, \ldots, x_{N-1})\big) = 1 + \max_{0 \le i < N} x_i$$

Ejemplo

- Disponemos de contenedores de capacidad C =10 y una lista de objetos cuyo pesos son la tupla (1, 2, 8, 7, 8, 3).

- Recordemos que, tanto los contenedores como los objetos, los contamos desde 0.

- La solución (2, 0, 0, 1, 2, 1) indica:
  - Contenedor 0: objetos 1 y 2.
  - Contenedor 1: objetos 3 y 5.
  - Contenedor 2: objetos 0 y 4.

Cota optimista

- Trata de rellenar los huecos en los contenedores ya usados como si el problema fuera continuo (trata los objetos restantes como líquido) pero con dos restricciones:
  - No consideres los objetos mayores que el mayor hueco de los contenedores.
  - No rellenes aquellos contenedores donde no quepa ni siquiera el objeto más pequeño.

- Divide el peso que no has utilizado entre la capacidad para estimar el número de contenedores adicionales.

Cota optimista: Algoritmo

- Obtén el peso de los objetos sobre los que todavía no has decidido y que quepan en alguno de los contenedores.
- Utiliza esos objetos como si fueran líquido para llenar los huecos de aquellos contenedores que ya estás utilizando pero cuyo espacio libre sea mayor o igual que el menor de los objetos pendientes.
- Calcula el número de contenedores necesarios para lo que sobre dividiéndolo por la capacidad.

Cota pesimista

- Podemos utilizar un algoritmo voraz para encontrar una cota pesimista.

- En el tema de voraces vimos dos algoritmos de aproximación:

  - "En el primero en el que quepa". Garantiza que el número de contenedores que utiliza es menor o igual que $17/10 \cdot M_{OPT}$

  - "De mayor a menor, en el primero en el que quepa". Garantiza que el número de contenedores que utiliza es menor o igual que $6/9+(11/9) \cdot M_{OPT}$

Versión de partida (bab_binpacking.py)

- En el aula virtual dispones de un versión inicial del programa en la que debes implementar las cotas mejoradas.

- Nota:
  - RyA encontrará la solución mucho más rápidamente si los objetos están ordenados de mayor a menor peso.
  - Para facilitar la implementación y no tener que trabajar con un nivel adicional de índices, asumiremos que la lista de objetos cumple con esta ordenación.