Naomi Lahera Champagne
Examen Final

Diseño y análisis de algoritmos
Universidad de La Habana

Bin Packing

1 Definición del problema

Dado un conjunto de n elementos, cada uno con un peso w_i, y un conjunto de contenedores B, cada uno con una capacidad fija c, encuentra el número mínimo de contenedores necesarios para empacar todos los elementos.

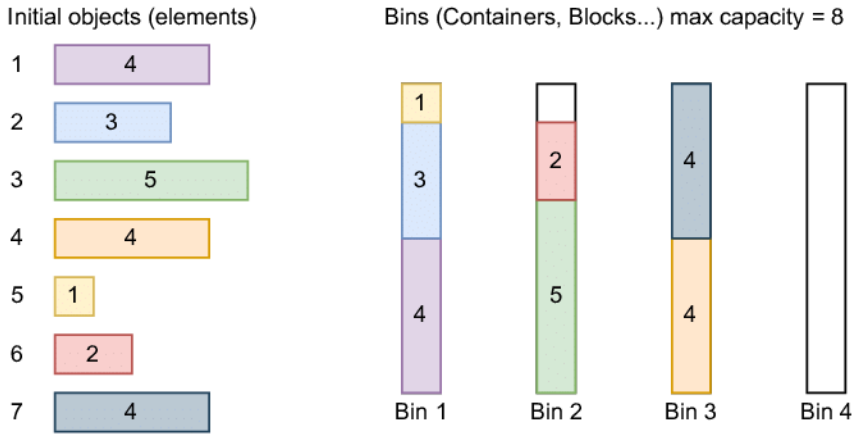

Figure 1: Posible solución a instancia de Bin Packing con $c = 8$ y $W = [4, 3, 5, 4, 1, 2, 4]$. Fuente: *An efficient cloudlet scheduling via bin packing in cloud computing. Amine Chraibi, Said Ben Alla, Abdellah Ezzati. 2022*

2 Análisis

El problema en cuestión tiene como objetivo encontrar el mínimo valor que cumpla con una condición dada, lo que nos lleva a pensar que estamos tratando un problema de optimización. Cuando minimizamos en un espacio de búsqueda finito una vía efectiva es evaluar cada posible solución y quedarnos con la 'mejor', así estaríamos entonces frente a un problema de optimización combinatoria pues el número de posibles asignaciones de objetos a contenedores crece exponencialmente a medida que aumentan el número de objetos. Esto nos que nos lleva a preguntarnos si es de la familia NP.

3 Bin PAcking es NP-Hard

A continuación demostraremos que Bin Packing es tan duro con Partition. Partiiton es un problema de decisión, entonces definamos Partition y reformulemos Bin Packing como un problema de decisión.

$$\text{Partition} = \left\{ A \;\middle|\; \exists\ B,\ C \subseteq A,\ B \cap C = \emptyset,\ B \cup C = A,\ \sum_{b \in B} b = \sum_{c \in C} c \right\}$$

$$\text{Bin-Packing} = \left\{ \langle W, c, b \rangle \;\middle|\; \exists\ S_1, ..., S_b\ \subseteq W,\ S_i \cap S_j = \emptyset,\ \bigcup_{i=1}^{b} S_i = W,\ \sum_{s \in S_i} s \leq c \right\}$$

3.1 Partition $\leq_p$ Bin Packing

Construyamos una instancia de Bin-Packing a partir de una instancia de Partition.

- **W** = A.
- Sea $n = \frac{\sum_{a_i \in A} a_i}{2}$ definamos **c** = $\lfloor \frac{n}{2} \rfloor$
- $b = 2$.

Esta construcción se puede hacer en tiempo polinomial (O(n)).

3.1.1 $A \in$ Partition $\Rightarrow \langle W, c, b \rangle \in$ Bin-Packing

Si $A \in$ partition entonces existe una forma de particionar A en dos conjuntos disjuntos B y C que cumplen $\sum_{b \in B} b = \sum_{c \in C} c$. Sean $S_1 = A$ y $S_2 = B$ como $A \in Partition$ se cumple que:

1. $S_1 \cap S_2 = \emptyset$.
2. $S_1 \cup S_2 = W$.
3. $\sum_{x \in S_1} x = \sum_{y \in S_2} y = \frac{\sum_{a_i \in A} a_i}{2} = \frac{n}{2} \leq c$.

Se cumple entonces que $W \in$ Bin-Packing. Como la suma de los elementos de cada partición S_1 y S_2 es exactamente la capacidad de los contenedores (c = $\frac{n}{2}$) sabemos que no es posible empacar los elementos de W en un único contenedor. Luego $b = 2$ es el mínimo. Hemos demostrado que si una instancia satisface las restricciones de Partition entonce la instancia de Bin-Packing que construimos a partir de ella satisafce las restricciones de Bin-Packing y da el óptimo 2.

3.1.2 $A \in$ Bin-Packing $\Rightarrow \langle W, c, b \rangle \in$ Partition

Demostraremos por contrarecíproco. Si $A \notin$ Partition entonces no existe forma de particionar A en dos conjuntos B y C que cumplan $\sum_{b \in B} b = \sum_{cinC} c = \frac{n}{2}$. Luego $\forall X \subseteq A$ se cumple que $\sum_{x \in X} x < \frac{n}{2}$

ó $\sum_{x \in X} x > \frac{n}{2}$. Entonces para empacar los elementos de W tendremos que seleccionar siempre una partición formada por dos conjuntos donde uno ellos cumple que la suma de sus elementos es estrictricamente menor que $\frac{n}{2}$ y por tanto menor estrictamente que c. La suma de los elementos del conjunto restante es por consiguiente estrictamente mayor que $\frac{n}{2}$ y por tanto estrictamente mayor que c siendo imposible empacar los elementos de W en un contenedor de tamaño c, serían necesarios más de 2 contenedores. Podemos concluir que $\langle W, b, c \rangle$ no pertenece a Bin-Packing.

Podemos afirmar entonces que Bin-Packing es tan duro como Partition (Partition $\leq_p$ Bin Packing)

4 Algoritmo de aproximación

4.1 Definición del problema

- **W:** Array de elementos que representan los tamaños de los objetos.
- **c:** Capacidad máxima de cada contenedor.

4.2 First Fit

Para cada elemento de 'W', verificamos si puede colocarse en algún contenedor existente. Esto lo hacemos recorriendo de izquierda a derecha los contenedores. Colocamos el nuevo elemento en el primer contenedor que sea lo suficientemente grande para contenerlo. Si aun no podemos colocar el elemento actual luego de revisar todos los contenedores no vacíos entonces añadimos un nuevo contenedor a la colección y colocamos al elemento en este contenedor.

4.2.1 Complejidad

- **Temporal:** Para cada uno de los n elementos, iteramos sobre los m contenedores, lo que resulta en una complejidad de $O(n * m)$), donde 'm es el número de contenedores utilizados (en el peor caso, puede ser$O(n^2)$.
- **Espacial :** Requiere $O(n)$ espacio para almacenar los contenedores y los elementos, ya que en el peor de los casos podría haber un contenedor por cada elemento.

4.2.2 Radio de aproximación

Sea $\hat{b}$ el número mínimo de contenedores requeridos para empacar una lista W de elementos se cumple que First Fit utilizará a lo sumo $b \leq 2\hat{b}$ contenedores.

Sea $tw = \sum_{w \in W} w$ se cumple $\frac{tw}{c} \leq \hat{b}$ porque todos los elementos deben ubicarse en un contenedor y cada contenedor tiene capacidad c, luego necesitan como mínimo una cantidad de congenedores

Algorithm 1 First Fit Bin Packing

```
Input: W, c
bins ← [ ]
for each item in items do
    placed ← False
    for each b in bins do
        if b + item ≤ c then
            b ← b + item
            placed ← True
            break
        end if
    end for
    if not placed then
        bins.append(item)
    end if
end for
return len(bins)
```

igual a la suma de los elementos de W. Veamos ahora que $b \leq 2tw + 1$. Notemos que al finalizar First Fit a lo sumo habrá un contenedor con $\frac{c}{2}$ de capacidad libre, pues en caso de haber dos S_1 y S_2 (S_2 fue el último que se creó) los el algoritmo hubiese situado todos los elementos de S_2 en S_1, pues en esto consiste su funcionamiento: empaca los elementos en el primer conenedor que encuentre con capacidad suficiente. Tenemos entonces $b - 1$ contenedores tienen más de la mitad de su capacidad ocupada, como el contenedor restante tiene al menos un elemento se cumple que $\frac{c(\hat{b}-1)}{1} < tw$. Tenemos entonces $b < \frac{2tw}{c} + 1 \leq 2\hat{b} + 1$. Podemos concluir entonces que $b < 2\hat{b}$ por tando nuestro Fist Fit es una 2-aproximación para Bin-Packing.

4.3 First Fit Decreasing

El algoritmo First Fit Decreasing comienza ordenando al array de la entrada en orden no creciente. Luego hace exactamente lo mismo que el algoritmo First Fit. Tiene una complejidad temporal $O(n^2)$.

4.3.1 Radio de Aproximación

Esta algoritmo es una $\frac{3}{2}$-aproximación para Bin Packnig.

4.4 Cota para el radio de aproximación

En las secciones anteriores vimos algoritmos de aproximación con radios de aproximación 2 y $\frac{3}{2}$. Existirá o podrá existir un algoritmo con un radio de aproximación menor? Sabemos que podemos reducir Partition a Bin-Packing (sección 3). La instancia de partición tiene una solución si y solo si

existe una solución para el problema de Bin-Packing utilizando solo dos contenedores. Supongamos que existe una aproximación con un factor ρ tal que $\rho < \frac{3}{2}$. Entonces, esta aproximación debe encontrar una solución usando menos de $2 \cdot \frac{3}{2} = 3$ contenedores. Por lo tanto, esta aproximación debe encontrar la solución óptima y por ende debe enconctrar en tiempo polinomial una solución al problema Partition cosa que no es posible a menos que P = NP.

Implementación disponible en: GitHub