CMSC 641 Design and Analysis of Algorithms **Spring 2023**

Homework 9: Bin Packing

Instructor: Adam Bargteil *Student: Hemanth Reddy Samidi*

9.1 NP-Hardness Proof: Minimum Bin Packing Problem

To prove that the Bin Packing problem is NP-hard, we use reduction. We will reduce the Subset Sum problem to Bin Packing problem. The Subset Sum problem is known to be NP-complete. This reduction will show that solving the Bin Packing problem is at least as hard as solving the Subset Sum problem.

Subset Sum Problem: Given a set of integers $a_1, a_2, ..., a_n$ and a target sum T, find out whether there is a subset of the integers that sums to T.

Reduction:

Let's say we have an instance of the subset-sum problem: a set $S = s_1, s_2, \ldots, s_n$ and a target sum T. Now, we will convert this instance into an instance of the bin packing problem as follows:

1. Normalizing set S by dividing each element by T. As a result, we will have a new set $S' = \frac{s_1}{T}, \frac{s_2}{T}, \ldots, \frac{s_n}{T}$. Here all elements of S' are in the range $(0, 1]$.
2. Setting bin size to 1.

The subset-sum problem is now transformed into a bin packing problem with the set of objects S' and bins with unit size. This reduction is done in polynomial time.

Now, let's show how the solution to the bin packing problem can help us solve the subset-sum problem:

- If there exists a subset of S that sums up to T, then there is a subset of S' that sums up to 1 (since each element of S' is just the corresponding element of S divided by T). This implies that one can pack that subset of S' into a single bin with size 1. By finding such a subset, one can say that the subset-sum problem has a solution.
- Conversely, if there exists a subset of S' that sums up to 1, then there is a subset of S that sums up to T (since each element of S' is the corresponding element of S divided by T). By finding such a subset, one can say that the subset-sum problem has a solution.

Therefore, we can determine whether there is a solution to the subset-sum problem in polynomial time if we can solve the bin packing problem. Since the subset-sum problem is NP-hard, this implies that the bin packing problem is also NP-hard.

9.2 Argue that the optimal number of bins required is at least $\lceil S \rceil$

To argue that the optimal number of bins required is at least $\lceil S \rceil$, we have to consider the total size of objects (S) and the capacity of the bins. Let's denote the size of object i as s_i, the total size of all objects as S, and the capacity of each bin as C. We know that:

$$S = \sum_{i=1}^{n} s_i \tag{9.1}$$

Now, let's look at the minimum number of bins required to store these objects. If we fit all objects completely into bins with no space, then the minimum number of bins required will be:

$$min\_bins = \frac{S}{C} \tag{9.2}$$

It's unlikely that the items will fit into the bins exactly. It will be tough to put all items in simply S/C containers because there will likely be some space left in each bin. We need at least one extra bin in this situation to hold the remaining items. Considering this, the minimum number of bins required can be represented by the smallest integer greater than or equal to S/C. Mathematically, this is given by the ceiling function, $\lceil S/C \rceil$. We know that each bin has a capacity of 1, i.e., $C = 1$. Therefore, the optimal number of bins required is at least $\lceil S \rceil$.

$$b_{opt} \geq \lceil S \rceil \tag{9.3}$$

9.3 Argue that the first-fit heuristic leaves at most one bin at most half full

To argue that the first-fit heuristic leaves at most one bin at most half full, we have to consider the way the algorithm places objects into bins. We know that first-fit heuristic takes each object and places each object into the lowest-numbered bin that still has space for it. If none of the available bins can fit the item, a new bin is opened and the item is put inside of it.

9.3.1 Proof by Contradiction

Let's say there are more than one bin that is at most half full. Let's take two such bins b_i and b_j, where $i < j$. This implies that both b_i and b_j have a free space of more than $\frac{1}{2}$. Let's now look at the moment the algorithm began to take the first object for bin b_j into account. Bin b_i must have been less than or equal to 50% filled at that point. Otherwise, the first-fit heuristic, which always selects the lowest-numbered bin that can still fit the object, would have put the object in bin b_i rather than creating a new bin b_j.

This contradicts our initial assumption that both b_i and b_j are at most half full, as the first-fit heuristic algorithm would have filled b_i further before opening b_j. Therefore, there can be at most one bin that is at most half full when the first-fit heuristic is applied.

9.4 Prove that the number of bins used by the first-fit heuristic never exceeds $\lceil 2S \rceil$

To prove that the number of bins used by the first-fit heuristic never exceeds $\lceil 2S \rceil$, let's analyze the algorithm and the total space in the bins. We know that first-fit heuristic takes each object in turn and places each object into the lowest-numbered bin that still has room for it. If no bin can accommodate the object, a new bin is opened, and the object is placed into it.

Let's denote the number of bins used by the first-fit heuristic as b_{ff}, the size of object i as s_i, and the total size of all objects as S. We have:

Each object can only be up to 1/2 the size of the bin, having a capacity of 1. Since there can be at most one bin that is at most half full, all other bins used by the algorithm are more than half full, which means they have at least 1/2 of their capacity filled. Let's now take a look at the total amount of space that the first-fit heuristic consumed across all bins except the final one (which can be at most half full):

$$total\_space\_used = (b_{ff} - 1) * \frac{1}{2} \tag{9.4}$$

As the total space used by the first-fit heuristic in all bins except last one has to be less than or equal to the total size of all objects (S), we will get:

$$(b_{ff} - 1) * \frac{1}{2}) \leq S \tag{9.5}$$

Multiplying by 2 on both sides, we get:

$$b_{ff} - 1 \leq 2S \tag{9.6}$$

Now add 1 to both sides, we get:

$$b_{ff} \leq 2S + 1 \tag{9.7}$$

As the number of bins must be an integer, we transform to:

$$b_{ff} \leq \lceil 2S \rceil \tag{9.8}$$

Hence proved.

9.5 Prove an approximation ratio of 2 for the first-fit heuristic

To prove that the first-fit heuristic has an approximation ratio of 2, which is nothing but the number of bins used by the first-fit heuristic (b_{ff}) is at most twice the number of bins in the optimal solution (b_{opt}).

We now know that: The optimal number of bins required is at least $\lceil S \rceil$. The number of bins used by the first-fit heuristic never exceeds $\lceil 2S \rceil$.

we need to show that $b_{ff} \leq 2b_{opt}$. From equation 9.6, we have:

$$b_{ff} \leq 2S + 1$$

From equation 9.3, we have:

$$b_{opt} \geq \lceil S \rceil$$

Multiplying by 2 on both sides, we get:

$$2 * b_{opt} \geq 2\lceil S \rceil \geq 2S$$

As $b_{ff} \leq 2S + 1$ and $2b_{opt} \geq 2S$, we can conclude that:

$$b_{ff} \leq 2 * b_{opt} \tag{9.9}$$

Hence proved.

9.6 Implementation and Time Complexity Analysis of the First-Fit Heuristic Algorithm

Implementation of the First-Fit heuristic using a balanced binary search tree:

Algorithm 1 First Fit Algorithm

```
1: procedure FirstFit(objects)
2:     bins ← SortedDict()
3:     bin_count ← 0
4:     for each obj in objects do
5:         bin_idx ← bins.bisect_left(1 − obj)
6:         if bin_idx == len(bins) then
7:             bins[1 − obj] ← 1
8:             bin_count ← bin_count + 1
9:         else
10:            remaining_capacity ← bins.popitem(bin_idx)[0]
11:            bins[remaining_capacity − obj] ← 1
12:        end if
13:    end for
14:    return bin_count
15: end procedure
```

Running time analysis:

- Iterating through the objects takes $O(n)$ time.
- For each object, we perform a binary search (using bisect_left) to find a bin. As this method performs a binary search on the SortedDict, it takes $O(\log n)$ time.
- Updating the remaining capacity of a bin involves a pop ($O(\log n)$) and an insertion ($O(\log n)$).

Thus, the overall running time complexity of the First-Fit heuristic using a balanced binary search tree is $O(\log n)$, where n is the number of objects.

References

[1] Thomas H. Cormen, Charles E.Leiserson, Ronald L. Rivest and Clifford Stein, Introduction to Algorithms, Third Edition (3rd. ed.), (2009), *Approximation Algorithms* pp. 1106–1133.

[2] https://grantjenks.com/docs/sortedcontainers/sorteddict.html