Study and implementation of an approximate algorithm for L-2D knapsack

Autumn 2019

Sumeet Shirgure - 15CS30035

1 Introduction

We study the approach given in [1] to construct approximate solutions for 2D Knapsack problem (2DK) and implement a pseudo polynomial time algorithm for a subproblem that the paper introduces named 'L-2D packing' (L2DK).

2DK is a geometric version of the classical knapsack problem, where you are given a set of n rectangles, each rectangle r_i having dimensions $w(r_i), h(r_i)$ and an associated profit $c(r_i)$, from which we have to select a subset that fits into a knapsack square of size $N*N$ which maximizes the total profit, defined as the sum of profits of rectangles selected. W.L.O.G, all said dimensions can be assumed integral. Also, the rectangles' orientations are considered to be fixed, so they can't be rotated, and must be embedded inside the knapsack in an axis parallel manner.

L2DK attempts to solve the same problem, with additional constraints. Firstly, the shape of the knapsack is an 'L' in Euclidean space, defined as $[0, N]^2 \backslash ([X, N] \times [Y, N])$ for some $X, Y \leqslant N$. Secondly, all of the rectangles have the longer side at least N/2 units long (called *long* rectangles). These constraints are posed only because the algorithms given in [1] for constructing the approximate solutions for 2DK use L2DK as a subroutine on *long* rectangles in its instances.

One additional assumption made by [1] (and in general) is that for a given error margin ε, the algorithms provided (for both 2DK and L2DK) can safely discard *large* rectangles having both sides longer than εN. This is because there are only at most $O(1/\varepsilon^2)$ $(O_\varepsilon(1))$ such rectangles, and if an optimal solution contains some of them, one can just use brute force without breaking polynomial complexity.

And to make the analysis of L2DK simple, $\varepsilon < 1/2$ is assumed from now on. This means that the smaller sides of rectangles in L2DK instances are always smaller than N/2, allowing us to use terms like *wide* or *horizontal* $(w > N/2, h < N/2)$ or *tall* or *vertical* $(h > N/2, w < N/2)$.

A $\frac{3}{4} - O(\varepsilon)$ approximate solution for 2DK with long rectangles uses the following observation – we can shift *long* rectangles such that they form four stacks at the sides of the knapsack in a ring-shaped region. This is possible since any *tall* rectangle can't have *wide* rectangles on both of its sides laterally in any feasible solution, because of the N/2 constraint. So if we delete the least profitable of these four stacks, and rearrange the remaining long items into an L shaped packing, we get at least 3/4 of the maximum profit from these long rectangles.

2 A pseudo polynomial time algorithm for L2DK

First some notations : let H and V be the set of all *wide* and *tall* rectangles respectively. We will index each of the $n_h := |H|$ elements in H by decreasing order of their widths : $H \equiv (p_1, ..., p_{n_h})$, $w(r_i) \geqslant w(r_{i+1})$. And similarly define n_v for $V \equiv (q_1, ..., q_{n_v})$ and index it by decreasing order of their heights. Also, define $L(x, y) := [x, N] \times [y, N]$

We can always find an optimal L shaped packing where the *tall* rectangles are pushed to the left and the top when sorted in this manner. Likewise to the bottom and right for the *wide* ones. This can be seen as a consequence of an exchange argument where we take any optimal L shaped packing and swap any two adjacent elements violating height (width) order, and obtain a new solution without losing optimality.

Now if we consider the longest *vertical* and *horizontal* rectangles, (i.e p_1 and q_1 respectively), and try placing them so that they touch the bottom right and top left corners (of $L(0,0)$) respectively, then there are only these possibilities :

- H and V are empty, and we are done.
- p_1 (or q_1) isn't a part of any optimal solution. In which case, we can recursively solve the problem for $L(0,0)$ and the lists $((p_2, ..., p_{n_h}), (q_1, ..., q_{n_v}))$ (or $((p_1, ..., p_{n_h}), (q_2, ..., q_{n_v}))$)
- p_1 is present in some optimal solution, which means that we can combine a solution for $L(w(p_1), 0), (p_2, ..., p_{n_h}), (q_1, ..., q_{n_v})$ and p_1's bottem left corner placed at $(0, N - h(p_1))$ adding a profit $c(p_1)$
- similarly q_1 could be placed at $(N - w(q_1), 0)$ and the remaining rectangles in $L(0, h(q_1))$

The dynamic program is now straightforward, table$[i, j, x, y]$ stores the optimal solution for the optimal L packing of rectangles $(p_i, ..., p_{n_h}), (q_j, ..., q_{n_v})$ in $L(x, y)$. The table is now populated in a decreasing lexicographic order of tuples $(i, j, x, y), 0 \leqslant i \leqslant n_h, 0 \leqslant j \leqslant n_v, 0 \leqslant x, y \leqslant N$. The base cases are table$[n_h + 1, n_v + 1, x, y] = 0 \forall x, y$. We can compute each of the $O(N^2 n^2)$ states in time $O(1)$ using the strictest of these lower bounds :

$\text{table}[i, j, x, y] \geqslant \max(\text{table}[i + 1, j, x, y], \text{table}[i, j + 1, x, y)$

$\text{table}[i, j, x, y] \geqslant c(p_i) + \text{table}[i + 1, j, x + w(p_i), y]$ if $i \leqslant n_h \wedge x + w(p_i) \leqslant N \wedge h(p_i) \leqslant N - y$

$\text{table}[i, j, x, y] \geqslant c(p_j) + \text{table}[i, j + 1, x, y + h(p_j)]$ if $i \leqslant n_v \wedge y + h(p_j) \leqslant N \wedge w(p_j) \leqslant N - x$

Finally, table$[1, 1, 0, 0]$ contains the required solution. Using the standard back pointers trick, we can also reconstruct one of the optimal solutions, both the chosen subset and its embedding.

Of course, the $O(N^2)$ dependency means that the algorithm isn't still truly polynomial. The reason we need to keep the (x, y) coordinates in our dynamic program is because the end points of the rectangles can be any integer within $[0, N]$. [1] addresses this issue, and actually provides a polynomial time approximation scheme (PTAS) by restricting the (x, y) coordinates to come only from a polynomial ($n^{O(1/\varepsilon^{1/\varepsilon})}$ to be precise) sized subset of $[0, N]^2$, incurring a small loss in profit.

3 Implementation details

The attached python codes implement the algorithm given in section 2. Refer to the attached README explaining the function of each module.

Some examples showing the strictness of 0.75 factor approximation are also provided.

4 References

[1] Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and

Andreas Wiese, Approximating Geometric Knapsack via L-packings.

[2]https://en.wikipedia.org/wiki/Pseudo-polynomial_time

[3]https://en.wikipedia.org/wiki/Polynomial-time_approximation_scheme