A 2D Bin Packing Problem

Hanoi University of
Science and Technology

Our Team

20214953

Bùi Hải Dương

20214886

Nguyễn Bá Dương

20214960

Phùng Đình Gia Huy

Introduction

A quick brief to understand
what we're dealing with

Packages

A large number of items (rectangles) varying in size, and we need to deliver them all to our customers.

Trucks

To deliver those packages, vehicles are needed. Each truck carries a container, which also varies in size. In addition, an operation cost is applied at random to every truck.

$300 - 4x4

Truck 1

$500 - 3x5

Truck 2

$850 - 1x2

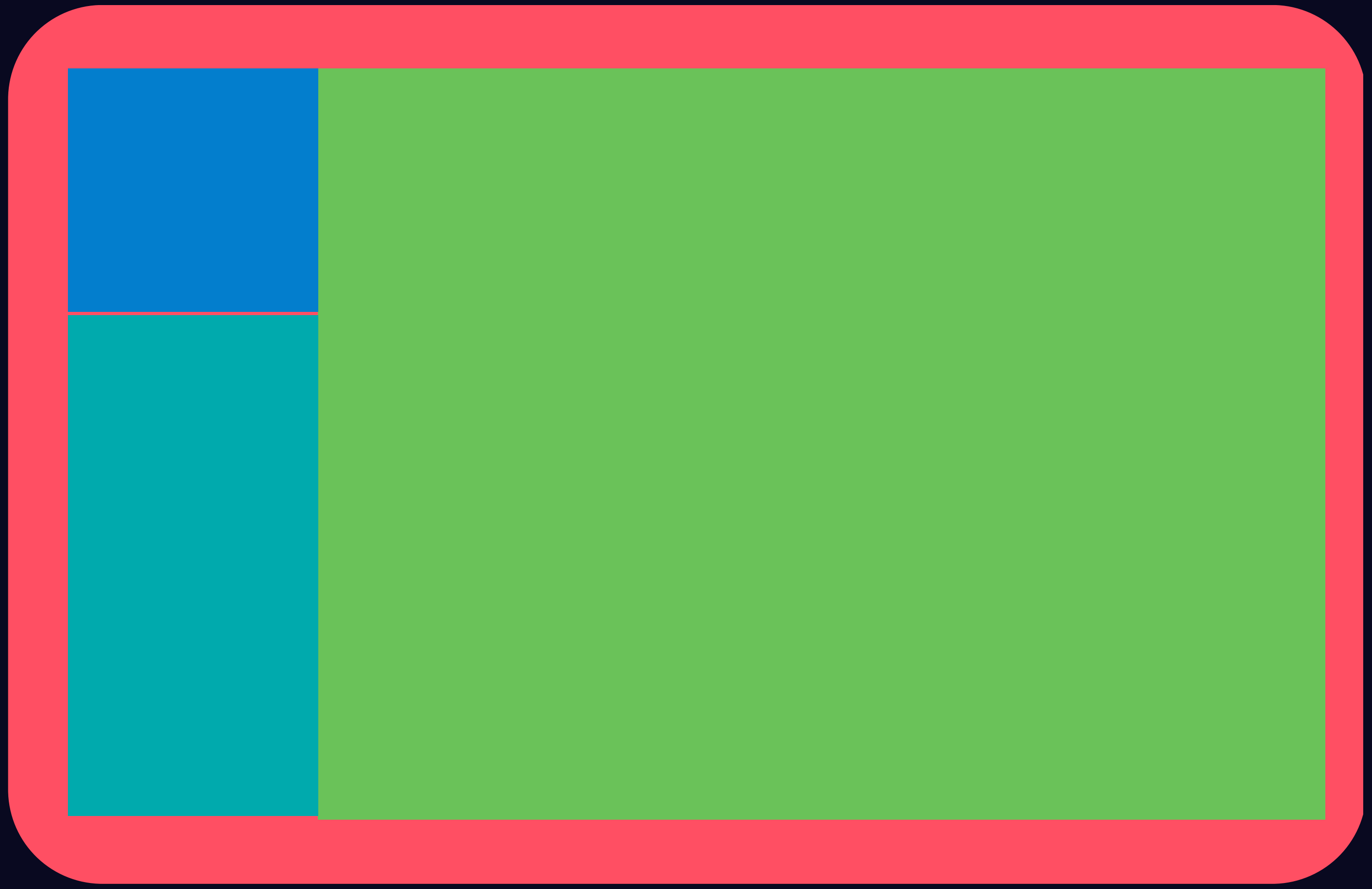
Truck 3

Unused because of
ridiculus cost

Objective

The goal here is to minimize the cost of operation while having all packages delivered.

Total cost: $800



Some Remarks

All packages can be rotated at an 90° angle so that all items in a truck could fit in it orthogonally

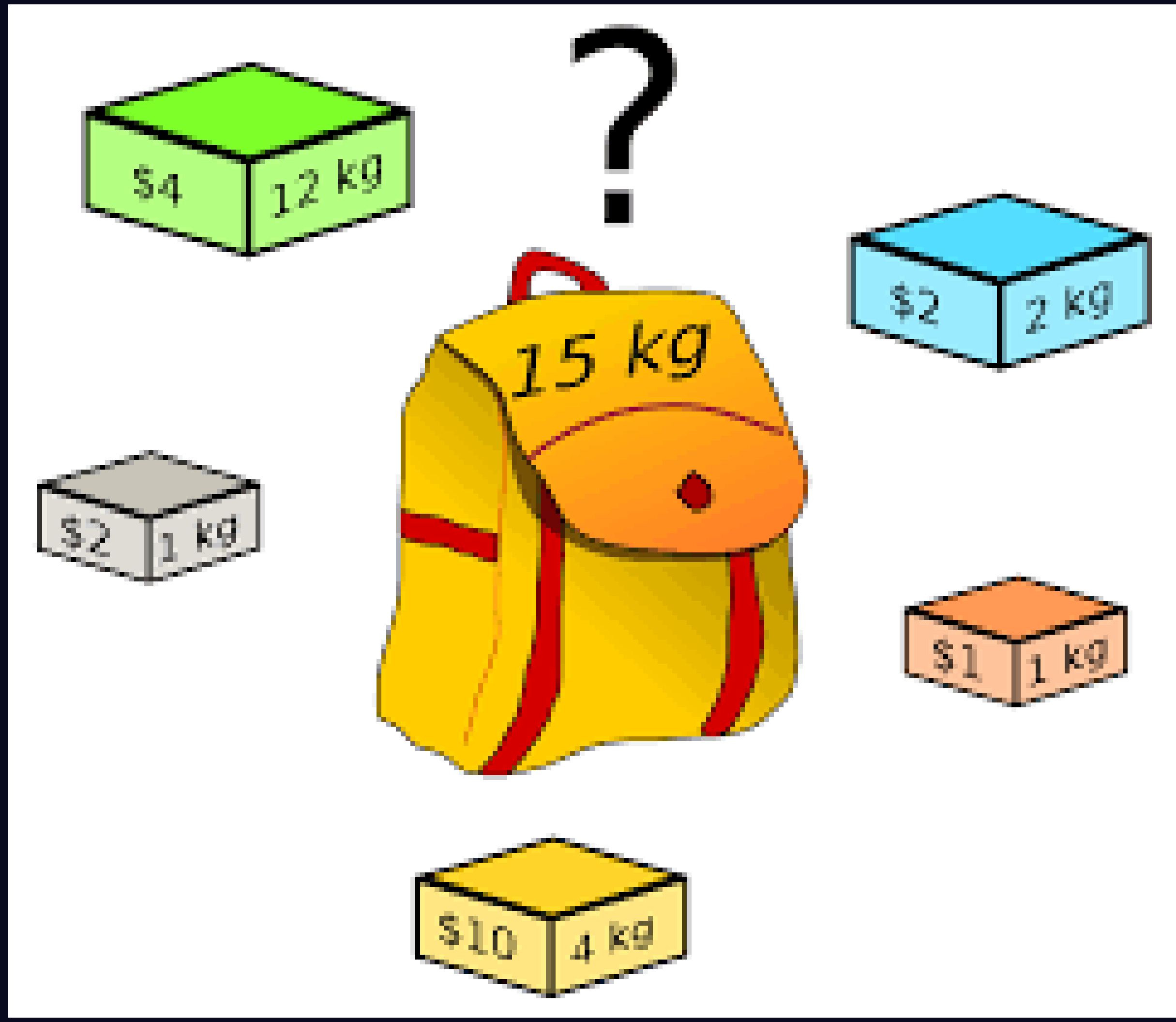
Each package has a size and must be in one of the trucks

Each truck has a storage capacity and operation cost

Minimize the total cost of trucks used in delivery

Fun Fact!

This problem is a variant of the 1-dimensional "Knapsack Problem", where items has a "weight/size" and "value".



Data Generation

Set a package count for each file

74 files are generated with increasing number of packages

- 50 basic and intermediate files: 5-54 packages with step 1
  i.e. 5, 6, 7,...
- 10 hard files: 60-330 packages with step 30
  i.e. 60, 90, 120,...
- 14 very hard files: 350-1000 packages with step 50
  i.e. 350, 400, 450,...

- Along with 1 sample file, creating a data set of 75 files in total.

Data Generating Algorithm

Step 1

Randomize packages

Step 2

Randomly load packages in trucks

Step 3

Shrink trucks' sizes

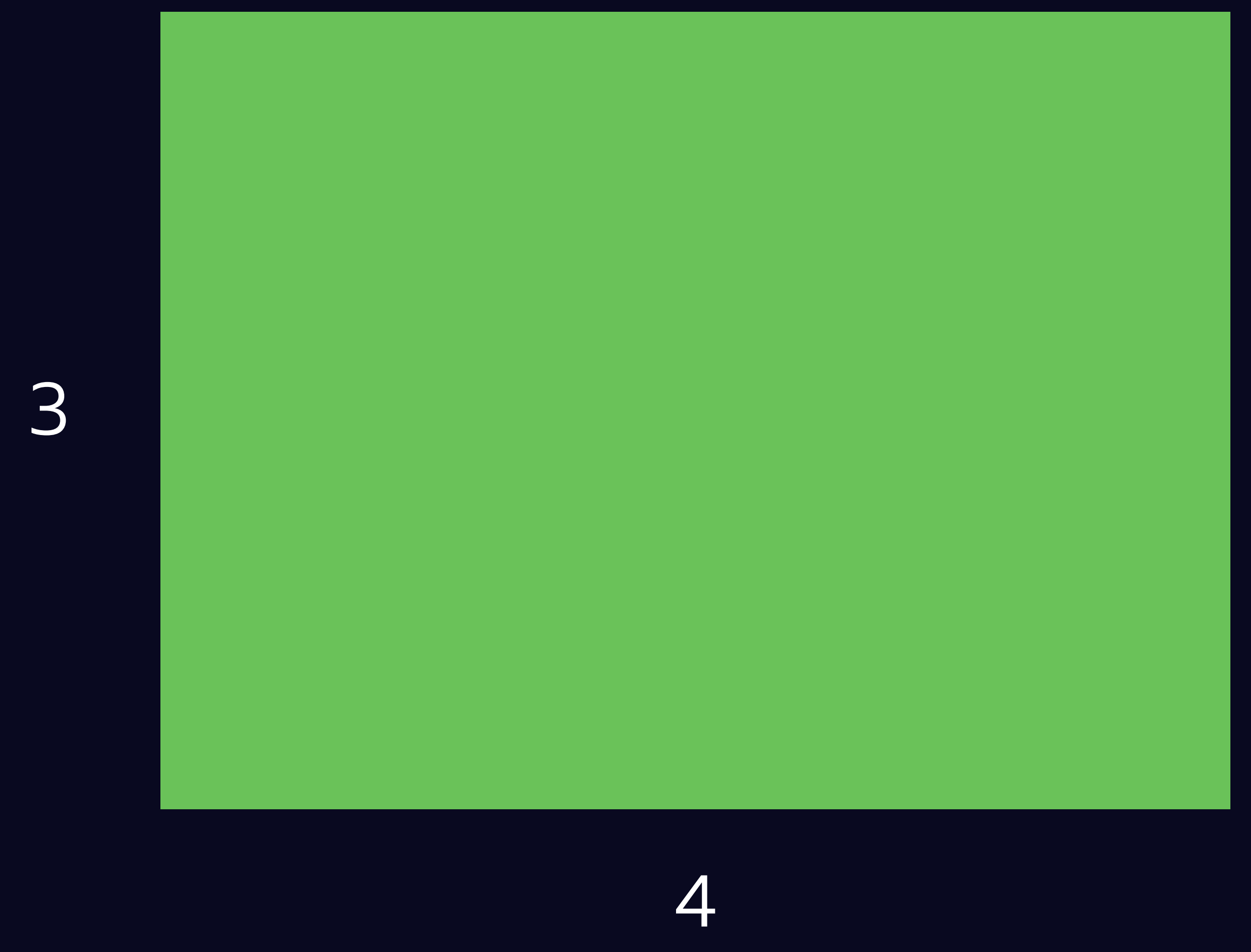
Step 4

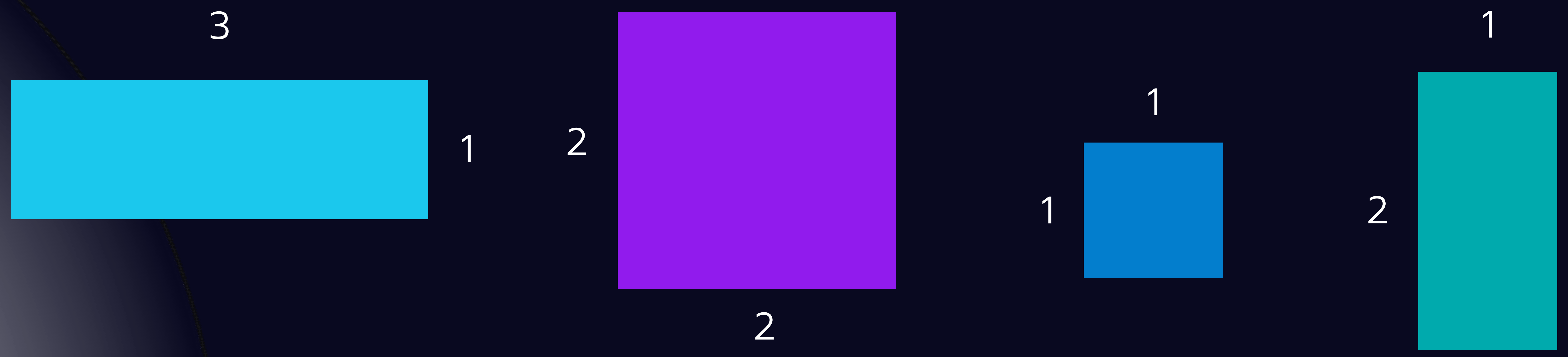
Randomly generate a few more trucks and then assign cost for all trucks

Randomize Packages

Each package has sizes ***a*** x ***b***, in which ***a*** and ***b*** can be any integer from 1 to 10.

n iterations are needed to obtain ***n*** packages.





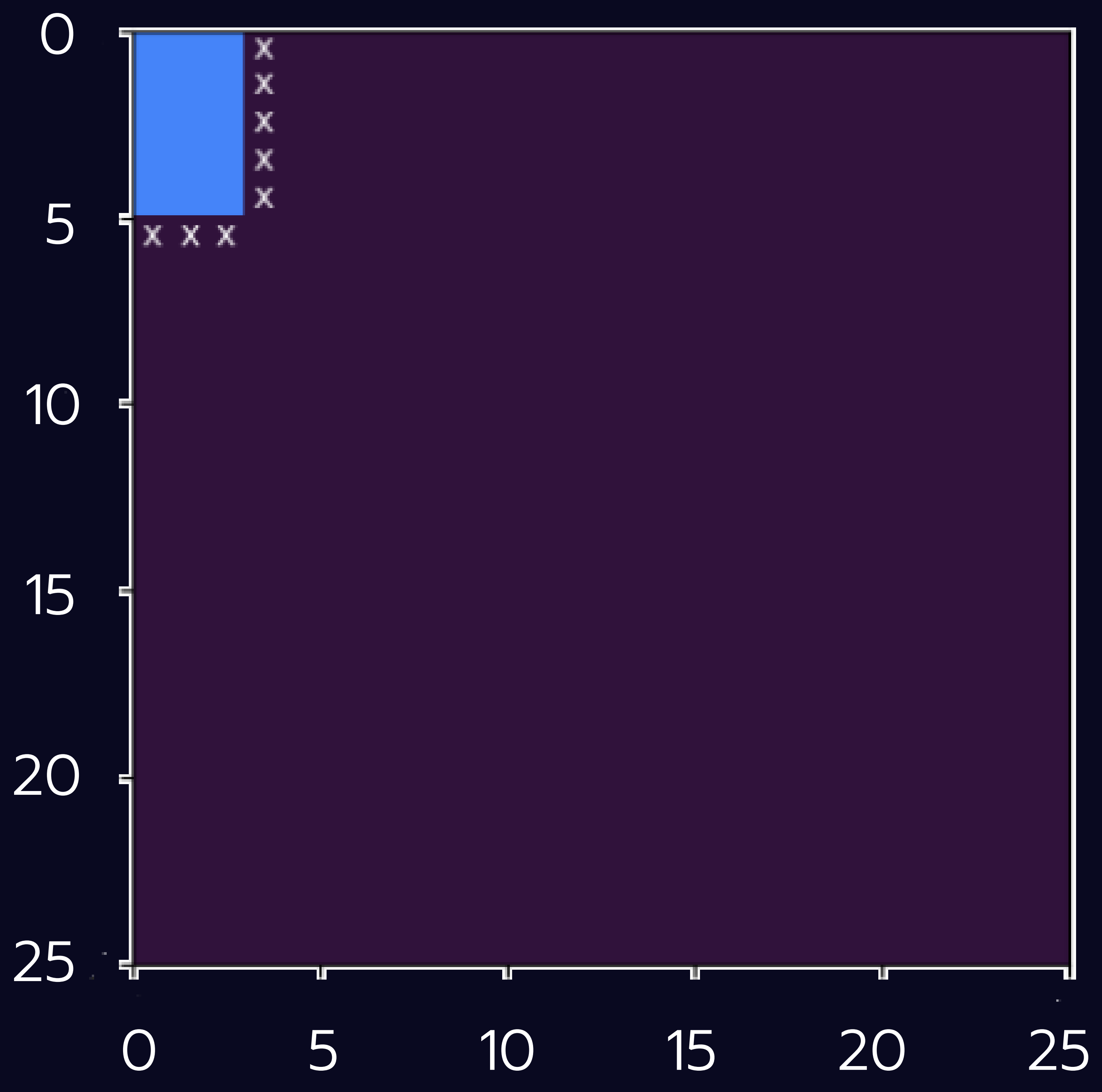
Randomly put packages in trucks

Out of *n* packages, pick from 2 to 5 at random.

Initialize a 30x30 truck.

Put the first package in the upper left corner.

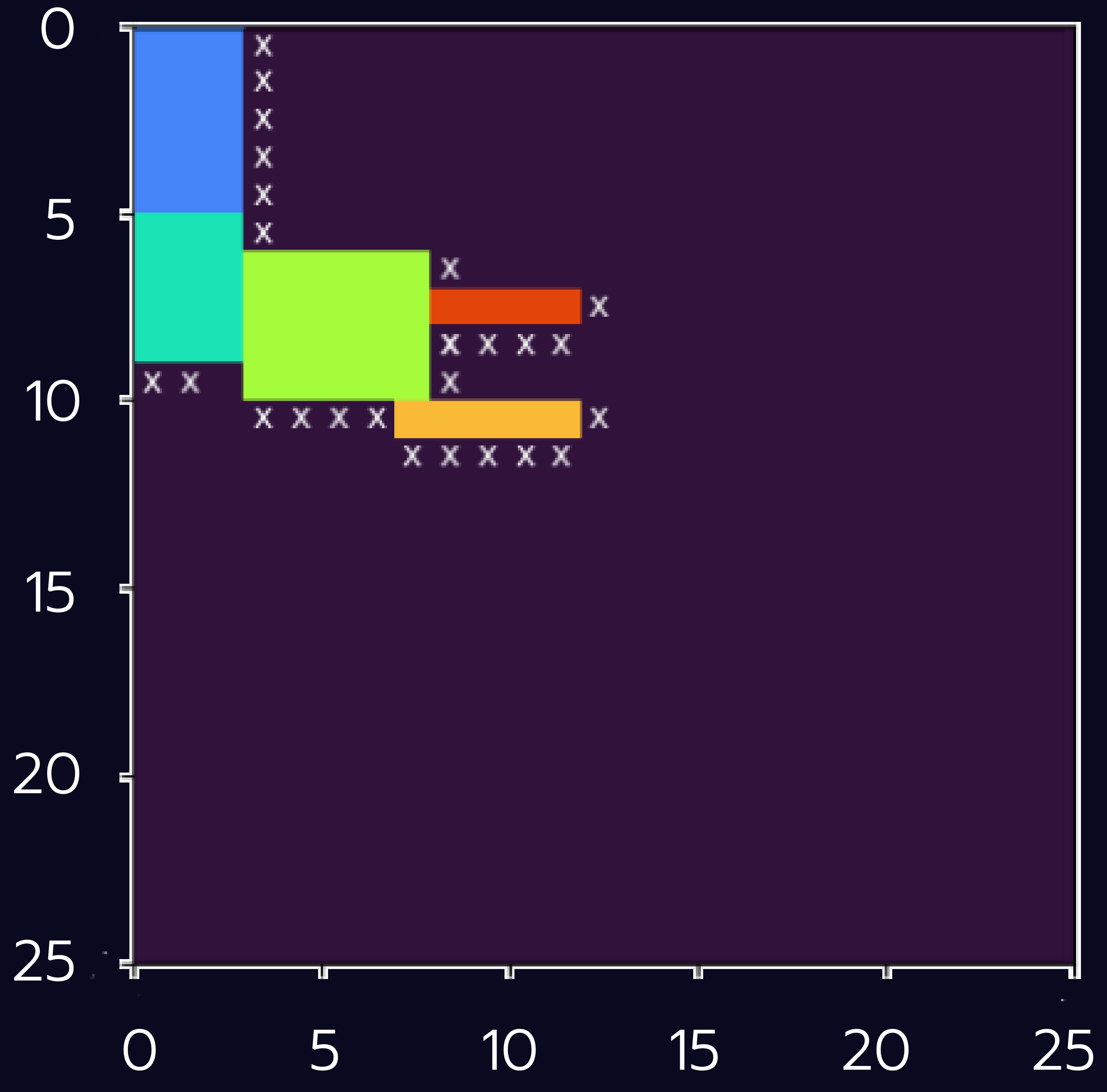
Mark all the available spaces on the right or bottom of the package.



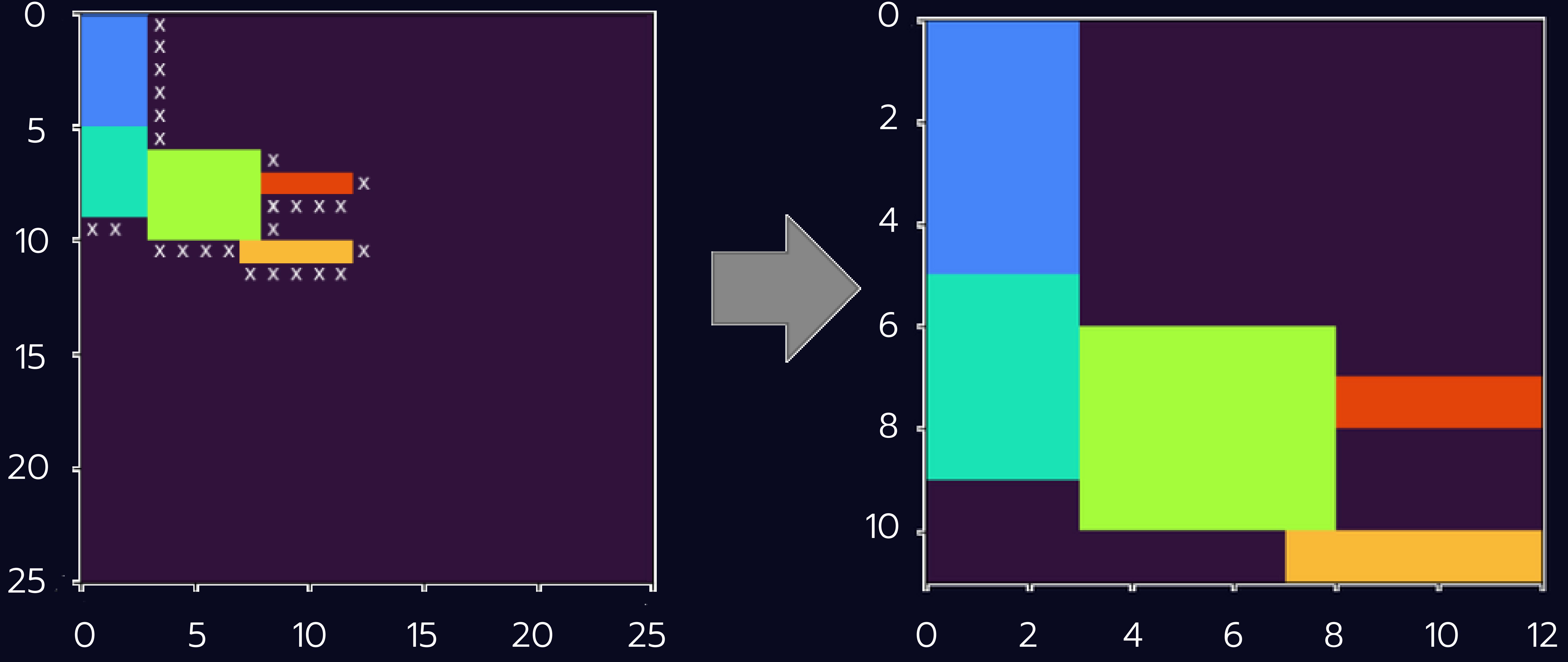
Randomly put packages in trucks

Choose one out of the remaining package(s) and randomly put it in the truck.

Repeat the above step until no packages remain.



Shrink trucks' sizes



Randomly generate a few more trucks and then assign cost for all trucks

Generate a few more trucks with their dimensions varying from 1 to 25
(i.e. 1x18, 21x12, 20x14, 25x25, etc.)

Assign cost from 100 to 1000 with step 50
(i.e. 100, 150, 200, ..., 1000)

Building CP and MIP models

Denotation

R = {1, 2, ..., n} is the set of given packages.
Item i has width w_i and height h_i.

B = {1, 2, ..., m} is the set of available trucks.
Container k has width w_k, height h_k and cost c_k.

Variables

$o_i \in \{0, 1\}$ represents the orientation of package i.

l_i, r_i, t_i, b_i are left, right, top and bottom coordinates of package i.

Binary variable u_k is 1 if truck k is used.

Binary variable p_{ik} is 1 if package i is placed in truck k.

$y_i = k \in \{1, \dots, m\}$, i.e. package i is placed in truck k.

CP Model - Constraint

$o_i = 0 \Rightarrow r_i = l_i + w_i \land t_i = b_i + h_i$	$\forall\, i \in \boldsymbol{R}$	(1)
$o_i = 1 \Rightarrow r_i = l_i + h_i \land t_i = b_i + w_i$	$\forall\, i \in \boldsymbol{R}$	(2)
$y_i = y_j \Rightarrow r_i \leq l_j \lor r_j \leq l_i \lor t_i \leq b_j \lor t_j \leq b_i$	$\forall\, i, j \in \boldsymbol{R}, i < j$	(3)
$y_i = k \Rightarrow r_i \leq W_i \land t_i \leq H_i$	$\forall\, i, j \in \boldsymbol{R}, k \in \boldsymbol{B}$	(4)
$p_{ik} = 1 \Rightarrow y_i = k$	$\forall\, i \in \boldsymbol{R}, k \in \boldsymbol{B}$	(5)
$\sum_{i=1}^{n} p_{ik} \geq 1 \Rightarrow u_k = 1$	$\forall\, k \in \boldsymbol{B}$	(6)

CP Model - Constraint

(1) If an item doesn't rotate, its right = its left + its width and its top = its bottom + its height

(2) If the item rotates then its right = its left + its height and its top = its bottom + its width

(3) If two items are placed in the same bin, they can't overlap each other

(4) If one item is place in a bin then its right and top coordinates can't exceed the bin

(5) Item **i** is placed in bin **k**

(6) Bin **k** is used when at least one item is placed in it

MIP Model - Constraint

$o_i = 0 \Rightarrow r_i = l_i + w_i \wedge t_i = b_i + h_i$ (1); $\quad o_i = 1 \Rightarrow r_i = l_i + h_i \wedge t_i = b_i + w_i$ (2)

To MIP: $r_i = l_i + w_i * (1 - o_i) + h_i * o_i$; $\quad t_i = b_i + h_i * (1 - o_i) + w_i * o_i$

$y_i = y_j \Rightarrow r_i \leq l_j \vee r_j \leq l_i \vee t_i \leq b_j \vee t_j \leq b_i$ (3)

To MIP: $r_i \leq M * (1 - x_1) + l_j$; $r_j \leq M * (1 - x_2) + l_i$;

$t_i \leq M * (1 - x_3) + b_j$; $t_j \leq M * (1 - x_4) + b_i$;

$x_1 + x_2 + x_3 + x_4 + (1 - x_0) * M \geq 1$;

$x_1 + x_2 + x_3 + x_4 \leq x_0 * M$; $b = 1 - x_{00}$;

$y_j + x_{00} \leq y_i + M * x_{01}$; $y_i + x_{00} \leq y_j + M * x_{02}$; $x_{01} + x_{02} \leq x_{00}$;

MIP Model - Constraint

$y_i = \text{k} \Rightarrow r_i \leq W_i \wedge t_i \leq H_i$ (4)

To MIP: $k + x_{00} \leq y_i + M * x_{01};\ y_i + x_{00} \leq k + M * x_{02}; x_{01} + x_{02} = x_{00};$

$l_i \leq w_k + x_{00} * M; r_i \leq w_k + x_{00} * M; t_i \leq h_k + x_{00} * M; b_i \leq h_k + x_{00} * M;$

$p_{ik} = 1 \Rightarrow y_i = \text{k}$; To MIP: $y_i = k * p_{ik}$ (5)

$\sum_{i=1}^{n} p_{ik} \geq 1 \Rightarrow u_k = 1$ (6)

To MIP: $S = \sum_{i=1}^{n} p_{ik}$

$S \leq (1 - e) * M; S + e * M \geq 1; u_k = 1 - e;$

Heuristics

The Brute-force Algorithm

First step: Sort

Trucks are sorted based on their cost/area ratio, in ascending order.

Packages are sorted based on their size, in descending order.

This is done in order to maximize efficiency by prioritizing cost-efficient trucks and largest packages.

The Brute-force Approach

Second step: Best-fit

From the sorted list, initialize the trucks, each containing no packages.

Iterate through the truck list for each package. If the package fits in the container, break the truck loop and check the next item.

The Brute-force Approach

Second step: Best-fit

The larger packages will fit in some of the best trucks, and when the algorithm iterate through the list, some smaller packages can also fit in the best trucks.

Therefore we can make use of the best trucks as much as possible.

The Brute-force Approach

But why sort the packages?

Top trucks on the list are usually the most spacious ones, and thus leave room for items that take up considerable space.

This also makes the trucks have fewer packages, reducing the backtracking algorithm's running time.

Another Approach: Guillotine

A Thousand Ways to Pack the Bin - A Practical Approach to Two-Dimensional Rectangle Bin Packing

Jukka Jylänki

February 27, 2010

Abstract

We review several algorithms that can be used to solve the problem of packing rectangles into two-dimensional finite bins. Most of the presented algorithms have well been studied in literature, but some of the variants are less known and some are apparently regarded as "folklore" and no previous reference is known. Different variants are presented and compared. The main contribution of this survey is an original classification of these variants from the viewpoint of solving the finite bin packing problem. This work focuses on empirical studies on the problem variant where rectangles are placed orthogonally and may be rotated by 90 degrees. Synthetic tests are used as the main benchmark and solving a practical problem of generating texture atlases is used to test the real-world performance of each method. As a related contribution, an original proof concerning the number of maximal orthogonal rectangles inside a rectilinear polygon is presented.

Keywords: Two-dimensional bin packing, optimization, heuristic algorithm, on-line algorithm, NP-hard

Another Approach: Guillotine

This algorithm is based on a operation callled "guillotine split placement". This procedure involves placing a package in the corner of a container, after which the remaining space is split into two disjoint smaller space.

The process of packing several items is then modelled as an iterative application of the above operation.

Another Approach: Guillotine

How it works

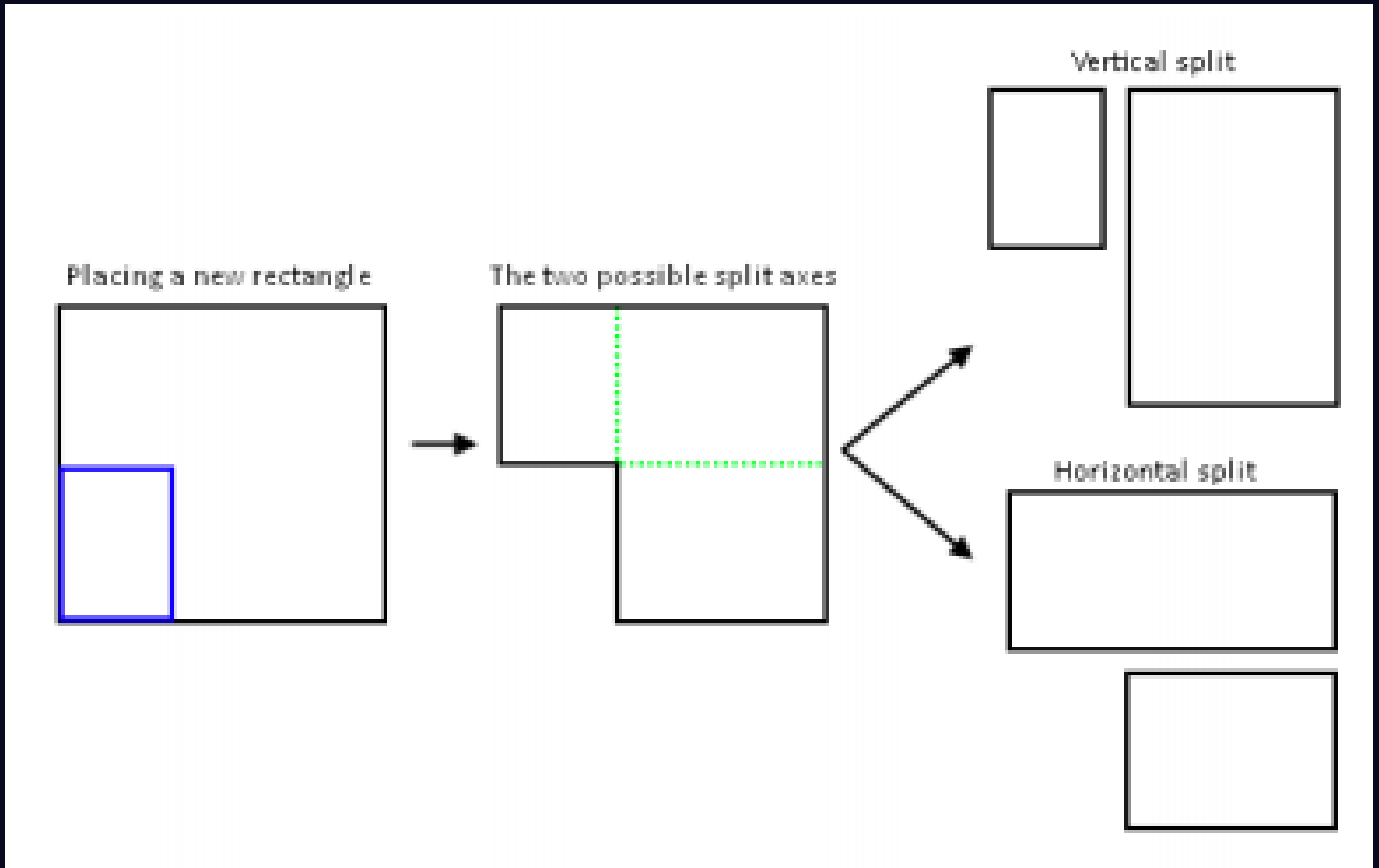
A list of smaller containers $F = \{F_1, \dots , F_n\}$ (which are pairwise disjoint) representing the free space of the truck is created.

The algorithm starts with a single container $F = \{F_1\}$. At each packing step, a free container F_i is picked at first to place the next package into.

Another Approach: Guillotine

How it works

The item is place at the bottom left corner of F_i, which is then split using the guillotine rule to make two smaller containers F' and F'', which replace F_i in the list of free containers.



Another Approach: Guillotine

This algorithm is appealing as it monitor the free sections of the truck and never "forgets" any free space.

The downside is that the algorithm only trys to fit items in a single free space F_i.

Algorithm Analyzation

REFERENCES

(1): Jukka Jylänki's A Thousand Ways to Pack the Bin - a Practical Approach to Two-Dimensional Rectangle Bin Packing