Branch and Price with PySCIPOpt

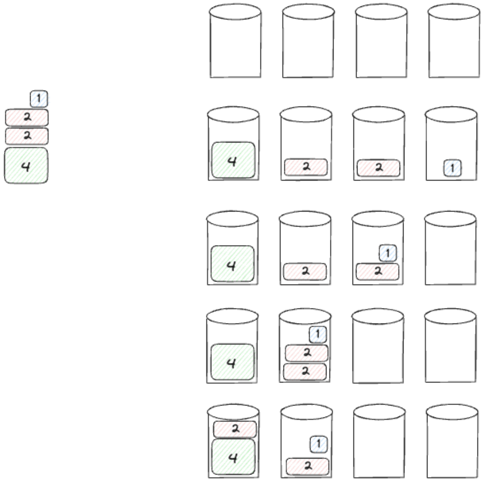
CO@Work 2024

Mohammed Ghannam & João Dionísio

Zuse Institute Berlin

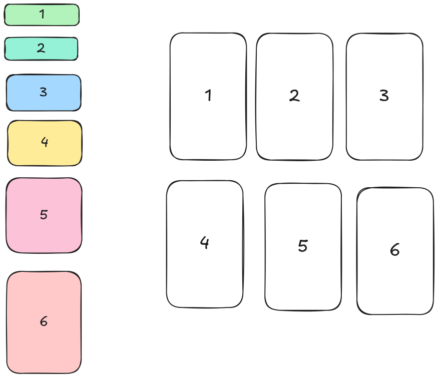
Bin Packing

- Need to store items in bins
- Items have weight and bins have capacity
- Use minimum bins with items not exceeding their capacity



Compact Formulation

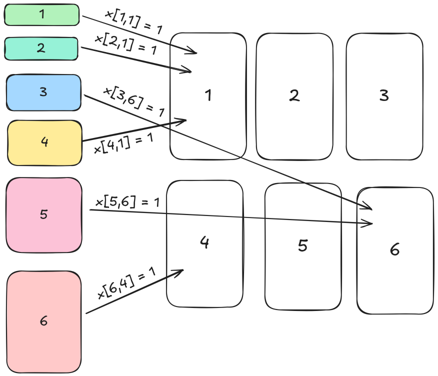
How can we formulate this?



Compact Formulation

How can we formulate this?

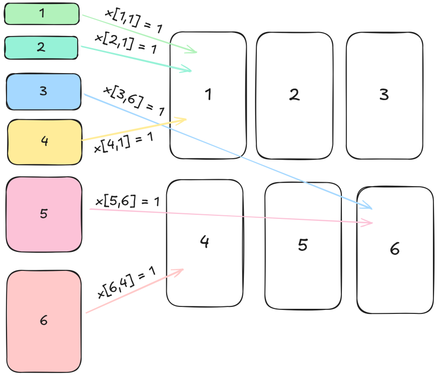
- Variable saying where each item is packed



Compact Formulation

How can we formulate this?

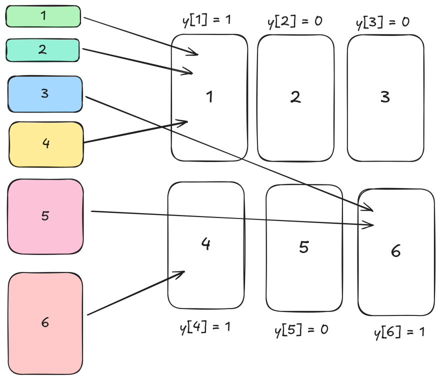
- Variable saying where each item is packed
- Enforce all items are packed



Compact Formulation

How can we formulate this?

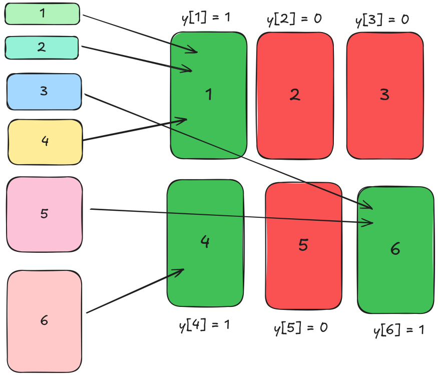
- Variable saying where each item is packed
- Enforce all items are packed
- Variable saying whether a bin is being used



Compact Formulation

How can we formulate this?

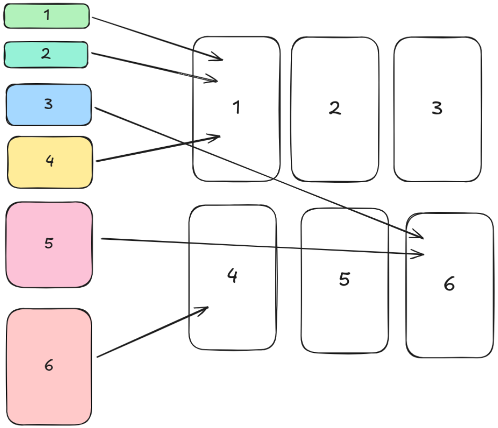
- Variable saying where each item is packed
- Enforce all items are packed
- Variable saying whether a bin is being used
- Minimize the number of used bins



Compact formulation and its poor scaling

DEMO

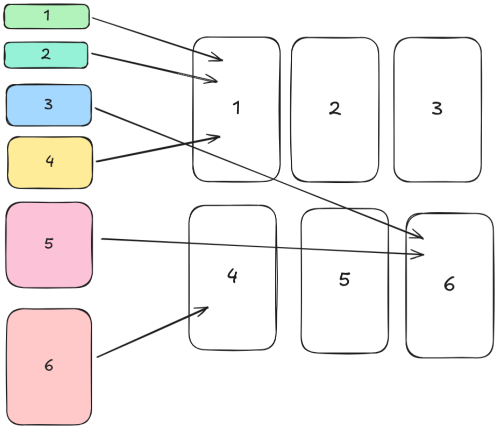
Why doesn’t this work?



x[1,1] = 1
x[2,1] = 1
x[3,6] = 1
x[4,1] = 1
x[5,6] = 1
x[6,4] = 1

It doesn’t seem complicated…

Why doesn't this work?



x[1,1] = 1
x[2,1] = 1
x[3,6] = 1
x[4,1] = 1
x[5,6] = 1
x[6,4] = 1

x[1,2] = 0
x[1,3] = 0
x[5,3] = 0
x[6,6] = 0
...

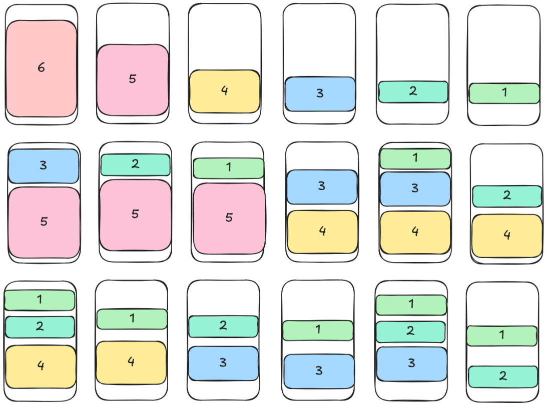
Extended Formulation: Modeling with Packings

We need a new formulation. Let’s change our perspective.

Extended Formulation: Modeling with Packings

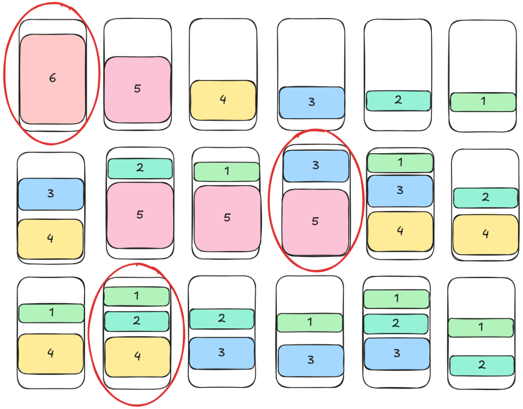
We need a new formulation. Let’s change our perspective.

Let’s look at all the ways of doing this (packings) and choose the best combination.

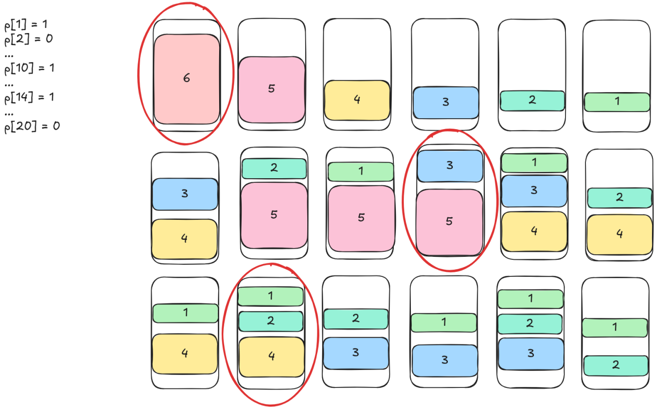


What does a solution look like?

p[1] = 1
p[2] = 0
...
p[10] = 1
...
p[14] = 1
...
p[20] = 0

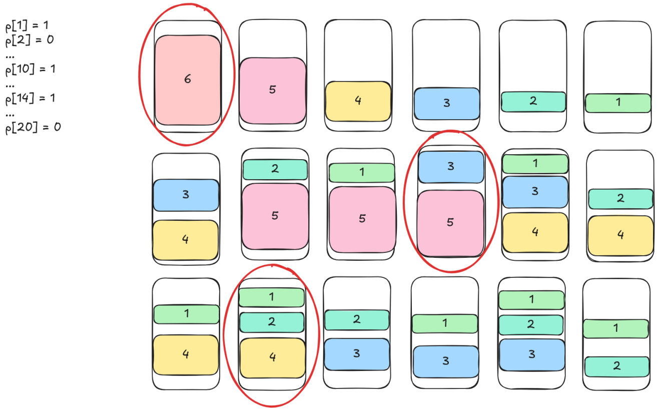


What does a solution look like?



Problem: There is an exponential number of packings.

What does a solution look like?



Problem: There is an exponential number of packings.

Integer Master Problem

For a list of all feasible packings $\mathcal{P}$, $a_i^p = 1$ if item $i \in \mathcal{I}$ is in packing p.

$$\begin{aligned}
\min \quad & \sum_{p \in \mathcal{P}} z_p \\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}} a_i^p z_p = 1, \forall i \in \mathcal{I} \quad (\pi_i) \\
& z_p \in \{0, 1\}, \forall p \in \mathcal{P}
\end{aligned}$$

Integer Master Problem

For a list of all feasible packings $\mathcal{P}$, $a_i^p = 1$ if item $i \in \mathcal{I}$ is in packing p.

$$\begin{aligned}
\min \quad & \sum_{p \in \mathcal{P}} z_p \\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}} a_i^p z_p = 1, \forall i \in \mathcal{I} \quad (\pi_i) \\
& z_p \in \{0, 1\}, \forall p \in \mathcal{P}
\end{aligned}$$

- Problem: There is an exponential number of packings.

Integer Master Problem

For a list of all feasible packings $\mathcal{P}$, $a_i^p = 1$ if item $i \in \mathcal{I}$ is in packing p.

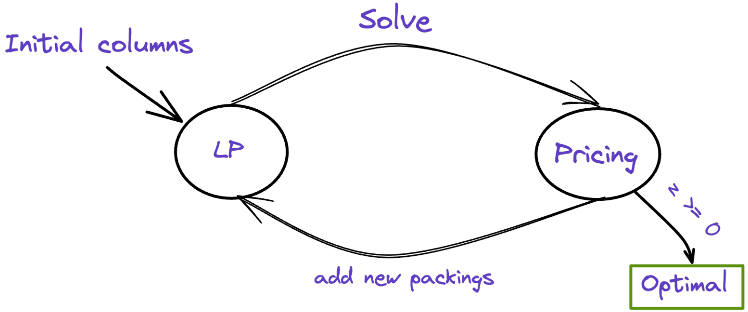
$$\begin{aligned} \min \quad & \sum_{p \in \mathcal{P}} z_p \\ \text{s.t.} \quad & \sum_{p \in \mathcal{P}} a_i^p z_p = 1, \forall i \in \mathcal{I} \quad (\pi_i) \\ & z_p \in \{0, 1\}, \forall p \in \mathcal{P} \end{aligned}$$

- Problem: There is an exponential number of packings.
- Solution: **Branch and Price!**

Game Plan

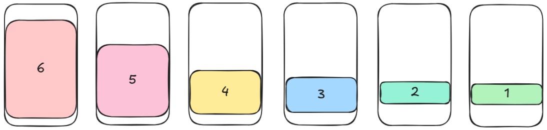
1. Solve the LP relaxation with Column Generation.
2. Embed in a branch-and-bound scheme to get optimal integer solution.
3. Improve our solver!

First Step: Solving the LP relaxation



Initial Columns

We need to initialize the RMP with some packings.



Let's go with the simplest way of assigning one item per packing.

Generating new columns

$$\begin{aligned}
\min \quad & \sum_{p \in \mathcal{P}'} z_p \\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}'} a_i^p z_p = 1, \forall i \in \mathcal{I} \quad (\pi_i) \\
& 0 \leq z_p \leq 1, \forall p \in \mathcal{P}'
\end{aligned}$$

How can we know which columns to add?

Generating new columns

$$\begin{aligned} \min \quad & \sum_{p \in \mathcal{P}'} z_p \\ \text{s.t.} \quad & \sum_{p \in \mathcal{P}'} a_i^p z_p = 1, \forall i \in \mathcal{I} \quad (\pi_i) \\ & 0 \leq z_p \leq 1, \forall p \in \mathcal{P}' \end{aligned}$$

How can we know which columns to add? **Reduced Cost < 0**

$$\text{minimize} \quad \underbrace{1}_{\text{obj. fn. coefficient}} \quad - \quad \overbrace{\sum_{i \in \mathcal{I}} a_i^p \pi_i}^{\text{column coefs. * dual vector}}$$

Pricing for Bin Packing

$$\begin{aligned} \text{minimize} \quad & 1 - \sum_{i \in \mathcal{I}} a_i \pi_i \\ \text{subject to} \quad & \sum_{i \in \mathcal{I}} s_i a_i \leq C \\ & a_i \in \{0, 1\}, \quad \forall i \in \mathcal{I} \end{aligned}$$

Pricing for Bin Packing

$$\begin{aligned} \text{minimize} \quad & 1 - \sum_{i \in \mathcal{I}} a_i \pi_i \\ \text{subject to} \quad & \sum_{i \in \mathcal{I}} s_i a_i \leq C \\ & a_i \in \{0, 1\}, \quad \forall i \in \mathcal{I} \end{aligned}$$

Let's focus on the objective function.

Pricing for Bin Packing

$$\begin{aligned} \text{minimize} \quad & 1 - \sum_{i \in \mathcal{I}} a_i \pi_i \\ \text{subject to} \quad & \sum_{i \in \mathcal{I}} s_i a_i \leq C \\ & a_i \in \{0, 1\}, \quad \forall i \in \mathcal{I} \end{aligned}$$

Let's focus on the objective function.

$$\min 1 - \sum_{i \in \mathcal{I}} a_i \pi_i$$

Pricing for Bin Packing

$$\begin{aligned} \text{minimize} \quad & 1 - \sum_{i \in \mathcal{I}} a_i \pi_i \\ \text{subject to} \quad & \sum_{i \in \mathcal{I}} s_i a_i \leq C \\ & a_i \in \{0, 1\}, \quad \forall i \in \mathcal{I} \end{aligned}$$

Let's focus on the objective function.

$$\min 1 - \sum_{i \in \mathcal{I}} a_i \pi_i = 1 + \min - \sum_{i \in \mathcal{I}} a_i \pi_i$$

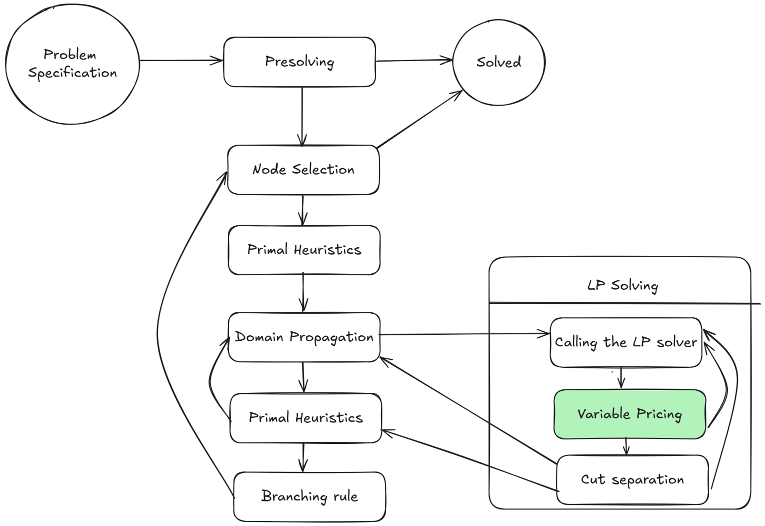
Pricing for Bin Packing

$$\begin{aligned} \text{minimize} \quad & 1 - \sum_{i \in \mathcal{I}} a_i \pi_i \\ \text{subject to} \quad & \sum_{i \in \mathcal{I}} s_i a_i \leq C \\ & a_i \in \{0, 1\}, \quad \forall i \in \mathcal{I} \end{aligned}$$

Let's focus on the objective function.

$$\min 1 - \sum_{i \in \mathcal{I}} a_i \pi_i = 1 + \min - \sum_{i \in \mathcal{I}} a_i \pi_i = 1 - \max \sum_{i \in \mathcal{I}} a_i \pi_i$$ = 1 - **Knapsack!**

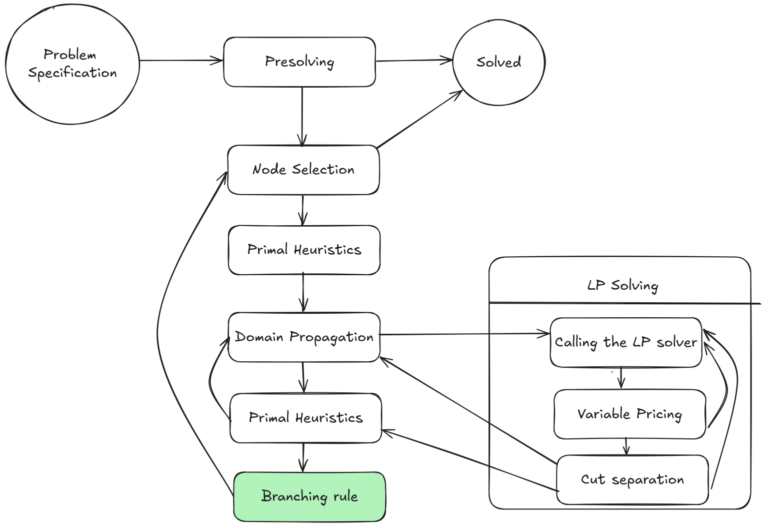
How to implement this in SCIP?



Exercise 1

Implement the `solve_knapsack` function (exercise details in Day3/README.md)

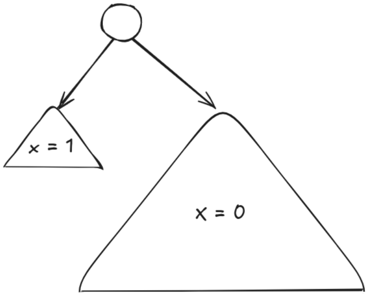
Getting Integer Solutions



Branching on master variables

Branching on master variable x has 2 options:

1. $x = 1$: we force the packing. Very restrictive
2. $x = 0$: we forbid the packing. Not restrictive at all.

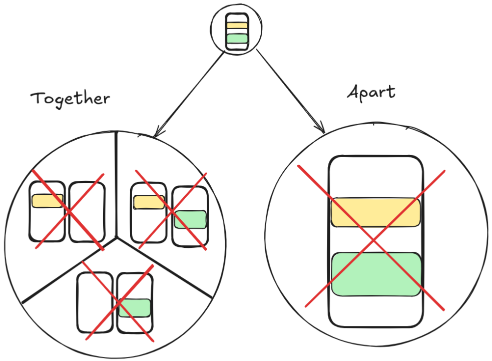


Leads to very unbalanced trees...

Ryan Foster Branching

Here there are two options:

1. Forbid two items from appearing in a new packing
2. Ensure that they appear in the same packing



Finding fractional pairs

Exercise 2

Find the fractional pairs to branch on.
Refer to the README for additional information.

Branching

Exercise 3

Complete the branching rule in `ryan_foster.py`

Pricing problem with branching decisions

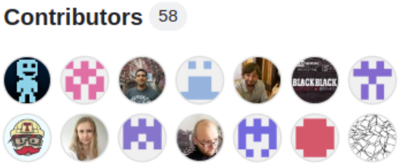
Exercise 4

Enforce the branching decisions in the pricing problem

PySCIPOpt break

As an open-source solver, PySCIPOpt appreciates the help of its users!

It’s your chance to be our 800th start :)



Many contributors started as users!

Improving our solver!

The next part is **self-paced**. We included some ideas in varying levels of difficulty to the README as bonus exercises.

Try to implement as many as you can!

Using integrality

Given the integrality of the objective function, we can get better bounds.

Hint: use PySCIPOPt’s setObjIntegral().

Better initial solution

One item per bin gives you the worst feasible solution.

Implement heuristics that find a better solution (closer to the optimal) and give them to the pricer.

Handling numerics

Trying to run the code for many items ($= 200$) results in a loop.

Find out why and fix the problem.
Hint 1: Read the slide title.
Hint 2: Look at the reduced cost of the columns you are generating.

Speeding up pricing

Especially if the pricing problem is difficult, it’s a good idea to:

1. Solve it heuristically (only need to solve to optimality at final iteration)
2. Return multiple negative reduced cost columns

This way, the burden of the pricing rounds is reduced.

In the case of knapsack, there are also more efficient algorithms.

Different-sized bins

If bins have different sizes **s**, we need different pricing problems

$$
\begin{aligned}
1 - \text{maximize} \quad & \sum_{i \in \mathcal{I}} a_i \pi_i \\
\text{subject to} \quad & \sum_{i \in \mathcal{I}} s_i a_i \leq C_{\mathbf{s}} \\
& a_i \in \{0, 1\}, \quad \forall i \in \mathcal{I}
\end{aligned}
$$

Implement this in your pricer (you also need to adapt the data generation in ‘generator.py‘).

Lagrangian bound

Column generation might offer tighter bounds!

$$\overbrace{v^*_{RMP}}^{\text{RMP optimal solval}} + \overbrace{|\mathcal{B}|}^{\text{N bins}} \times \underbrace{v^*_{PP}}_{\text{PP optimal solval}} \leq \overbrace{v^*_{MP}}^{\text{MP optimal solval}}$$

Why is this true? Implement it in your pricer (hint: add a ‘lowerbound’ key to the dictionary return in the pricer).

How does this bound look with multiple pricing problems?