Optimization. Exact algorithms

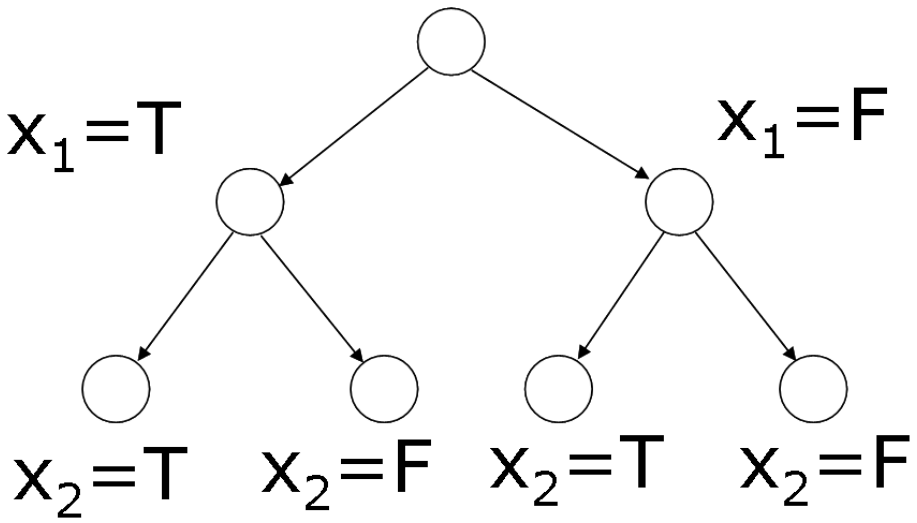
Maciej Komosinski

Institute of Computing Science
Poznan University of Technology
www.cs.put.poznan.pl/mkomosinski

Organize the set of solutions in the SAT problem

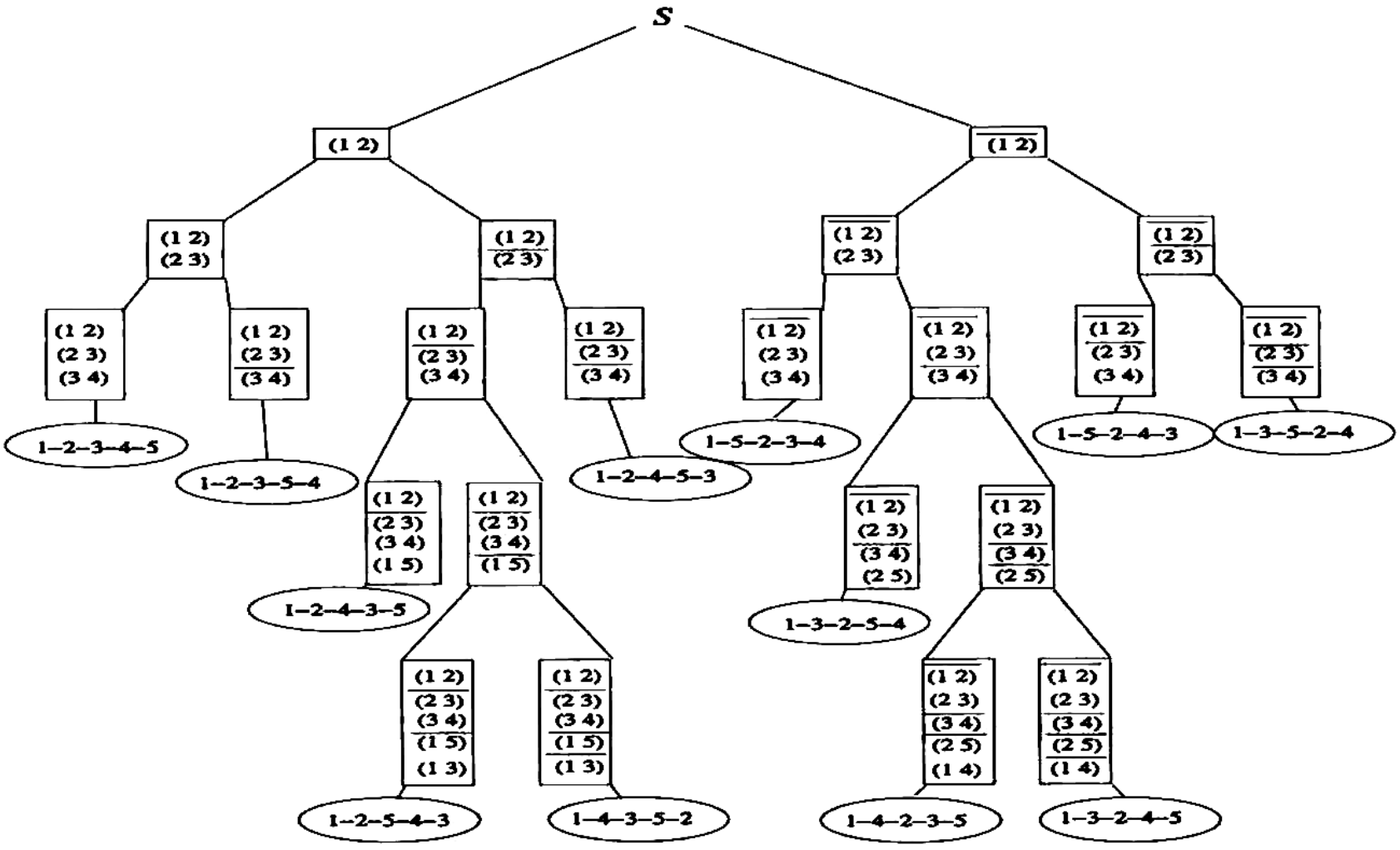
Organize the set of solutions in the SAT problem

Multiple division into two subsets: $x_1 = T, x_1 = F, ...$



Organize the set of solutions in the TSP problem

Organize the set of solutions in the TSP problem



If we include incomplete (partial) solutions in the tree...

Is it possible to somehow limit the search in such a tree?

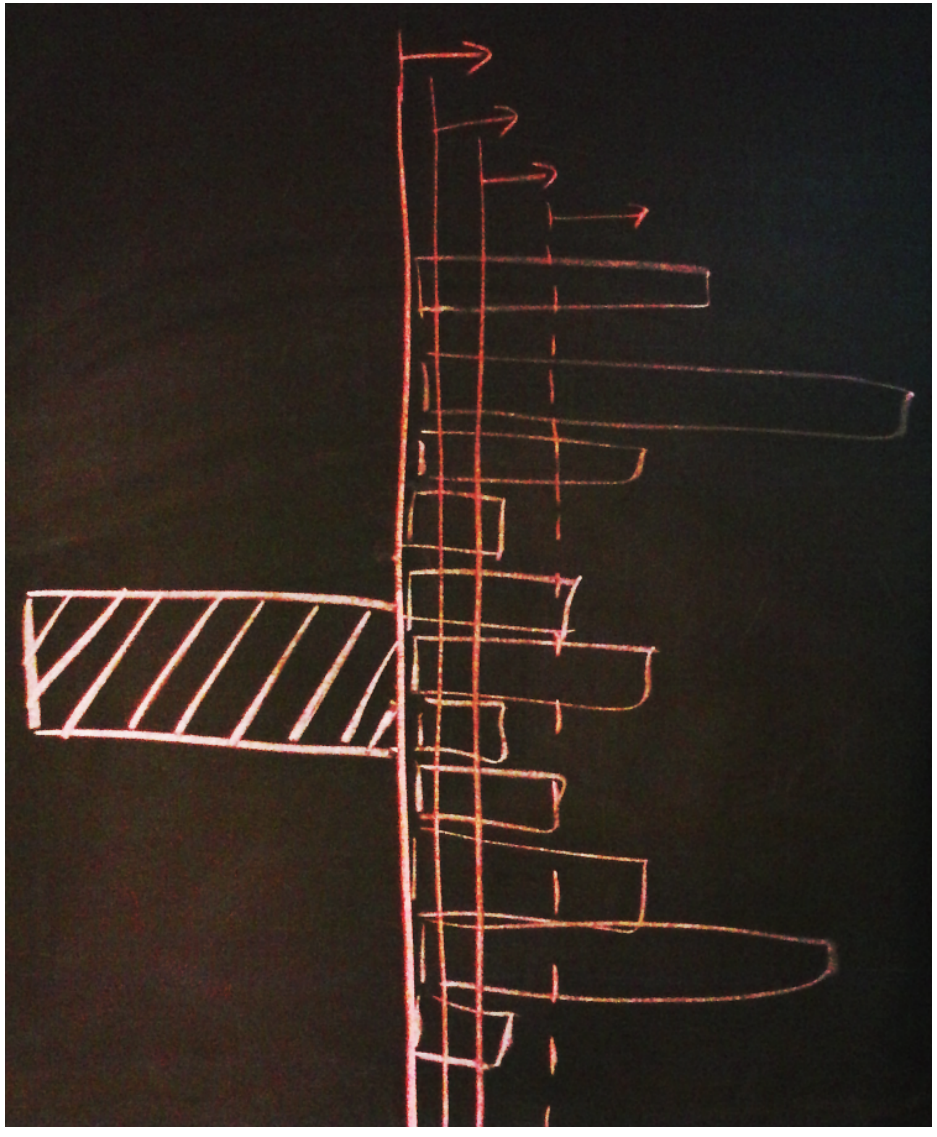
If we include incomplete (partial) solutions in the tree...

Is it possible to somehow limit the search in such a tree?

- Yes – we can skip branches that lead to only unacceptable solutions
- Yes – we can skip branches where we will certainly not find a better solution than the best one already found

Illustration from the lecture – increasingly better bounds

The red dotted line shows an incorrect bound – routes can be shorter



Rucksack problem – mathematical formulation

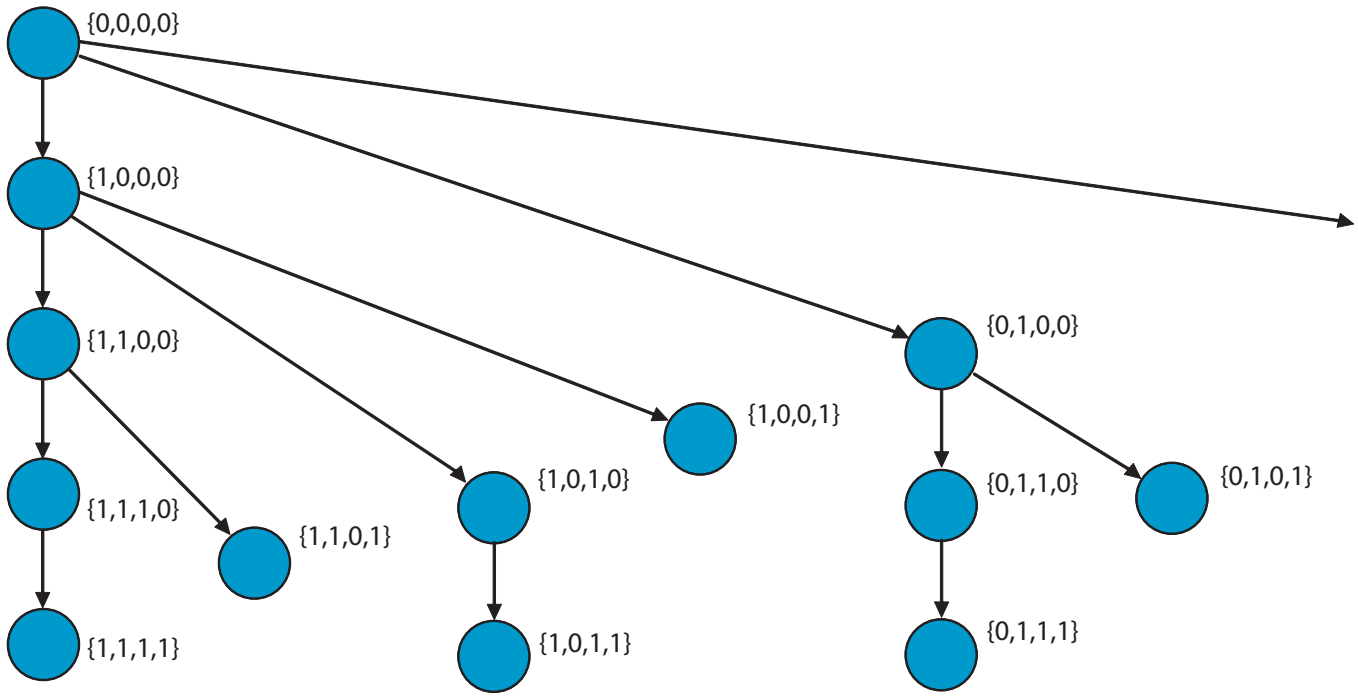
Rucksack problem – mathematical formulation

- Variables
  - $x_i \in \{0, 1\} - 1$ if the i-th item is put into the rucksack
- Maximize
  - $\sum_i x_i c_i$
- Constraint
  - $\sum_i x_i w_i \leq W$

- The rucksack problem is NP-hard

The Branch & Bound method

Constructing a solution occurs while traversing the tree.
E.g. for the rucksack problem:



Upper (lower) bound

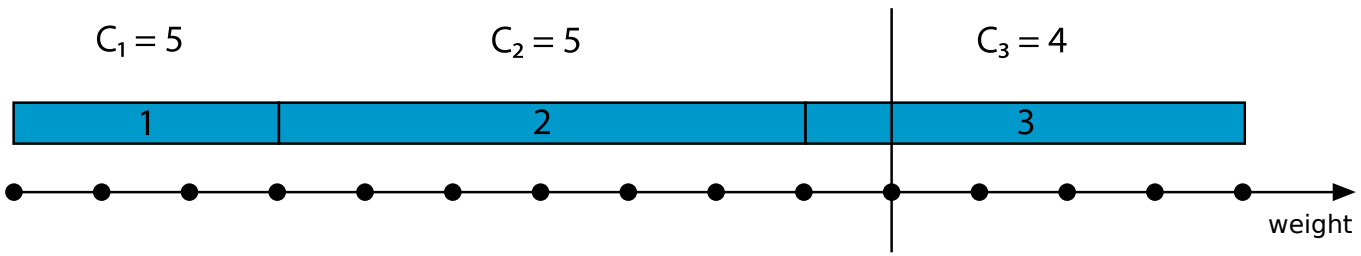
- Upper bound – a value that will certainly not be exceeded by the objective function for any of the solutions in which certain elements (e.g. variables) have been determined
- For example, what is the upper bound for solutions in which $x_1 = 1$ and $x_5 = 1$?
- The upper bound does not have to be (and usually it is not) achieved by any solution
- Defining the upper bound depends on the problem

Defining the upper bound for the rucksack problem

- Suppose we can put items into the rucksack partially.
- Sort items according to the c_i/w_i ratio.
- Insert items into the rucksack starting with the item with the highest ratio c_i/w_i, until the rucksack is full.
- The last item can be put into the rucksack partially.

Example. Rucksack capacity = 10

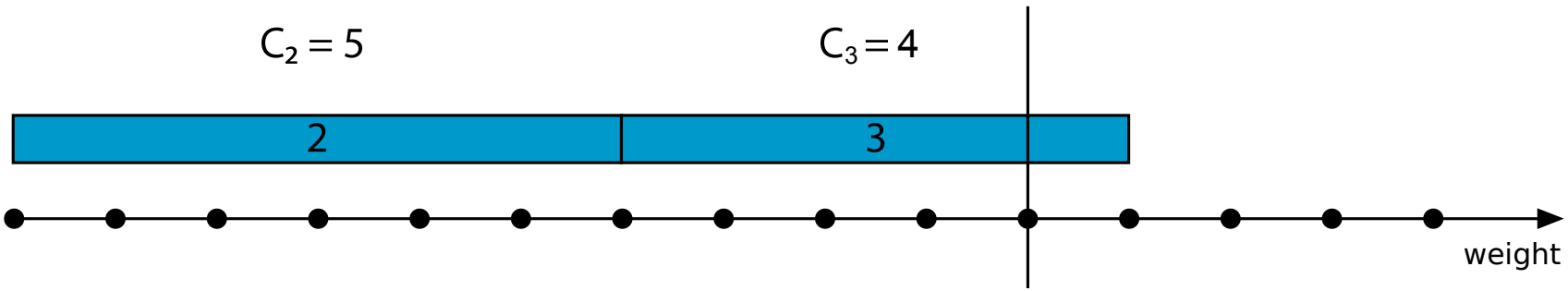
Item i	w_i	c_i	Ratio c_i/w_i
1	3	5	1.666667
2	6	5	0.833333
3	5	4	0.8
4	2	1.5	0.75
	3	1	0.333333
	4	1	0.25



$$\text{Upper bound} = 5 + 5 + \tfrac{1}{5} \cdot 4 = 10.8$$

Determining the upper bound for the rucksack problem

- If some parts of the solution have been already determined (items are selected – 1, or not – 0), then we consider only the remaining items
- E.g. if we have decided that we do not select item #1, item #2 is already in the rucksack, and we still have 4 units of space free:

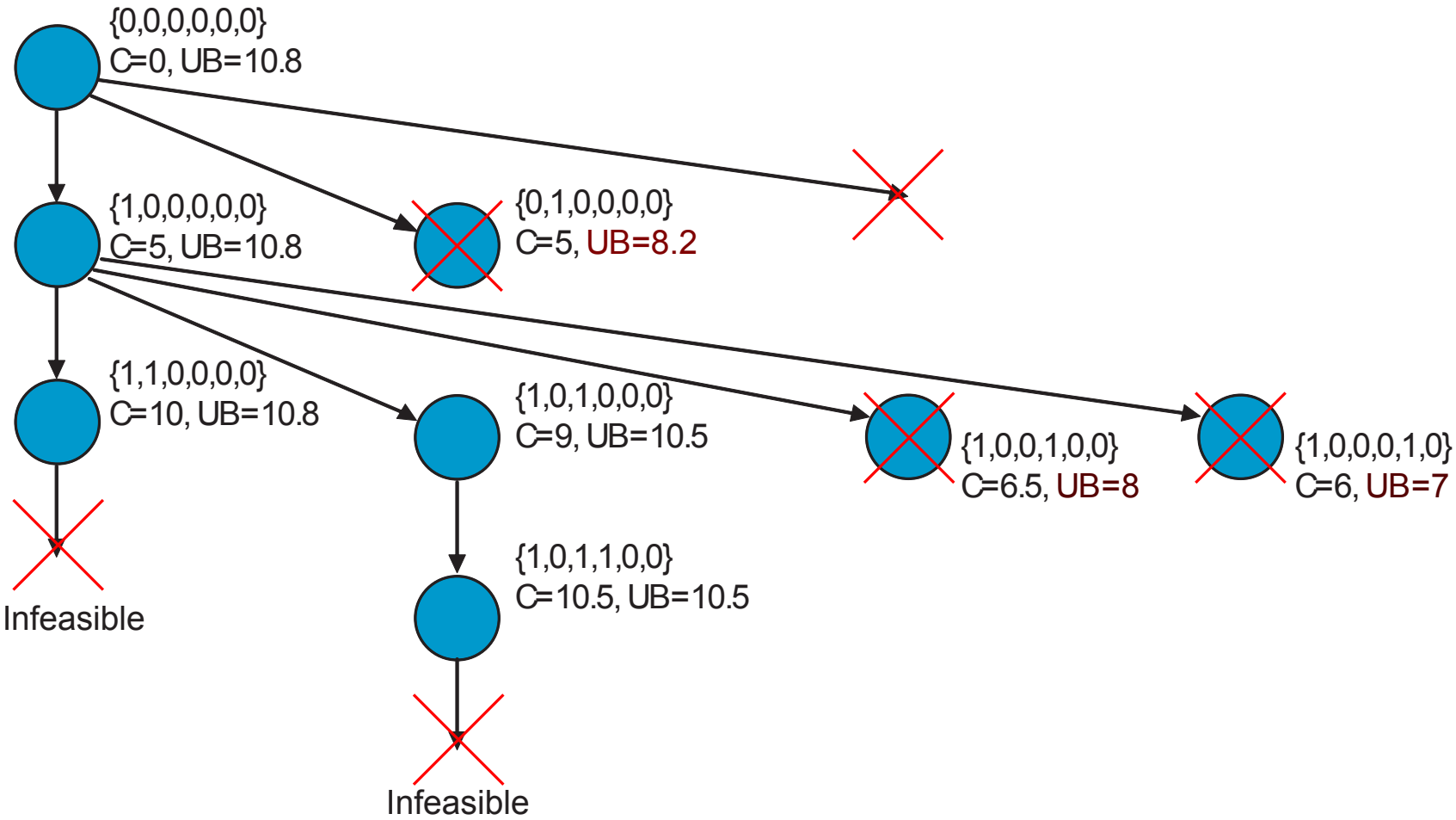


$$\text{Upper bound} = 5 + \frac{4}{5} \cdot 4 = 8.2$$

Branch and bound in maximization

If, for a given tree branch, the upper bound ("in this branch there will be at most as good as") is lower than the already known (best) solution, then this branch can be skipped.

B&B – example (knapsack problem)



B&B – solving an integer MP

- If we remove integer-value constraints from the MP problem, we may improve the optimal solution or we may be unable to improve it
- Therefore, the optimal solution of the continuous MP is not worse than the optimal solution of the discrete MP
- Therefore, the optimal solution of the continuous MP is a bound ("will not be better than") on the optimal solution of the discrete MP
- In the root we solve the continuous MP, and as we go deeper in the tree we successively add constraints on successive variables in such a way that the interval containing non-integer values of the variable is excluded
- In each node we solve the continuous MP in order to determine the bound, and possibly skip the branch if the bound is worse than the best integer solution already found
- Example: https://www.youtube.com/watch?v=WNRRmXZkRi0

Branch & Bound – manufacturing

Find the allocation that maximizes total efficiency.

Efficiency c_{ij}:

Device \ Product	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

1. Formulate a general problem as MP
2. Propose an algorithm to solve this problem using B&B
3. Apply this algorithm to the above instance.

B.O.D.I., ex. 2.21