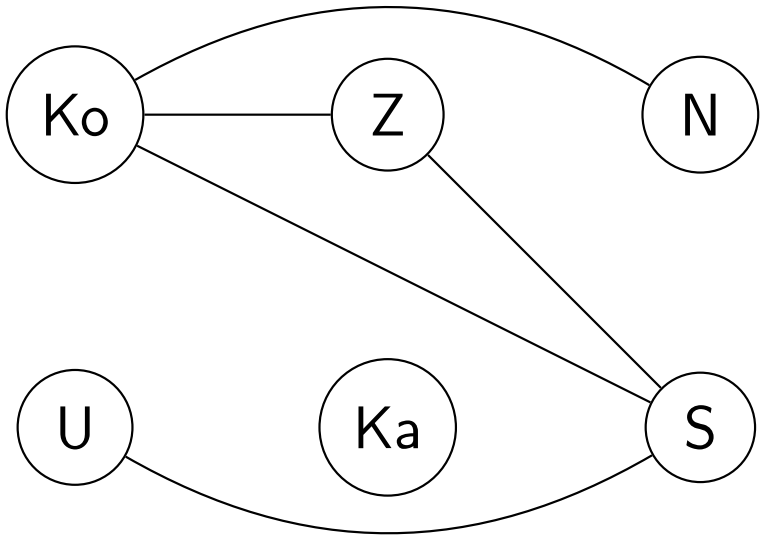
Chapter 4 : Branch and bound

ENSIIE - Operations Research Module

Dimitri Watel (dimitri.watel@ensiie.fr)

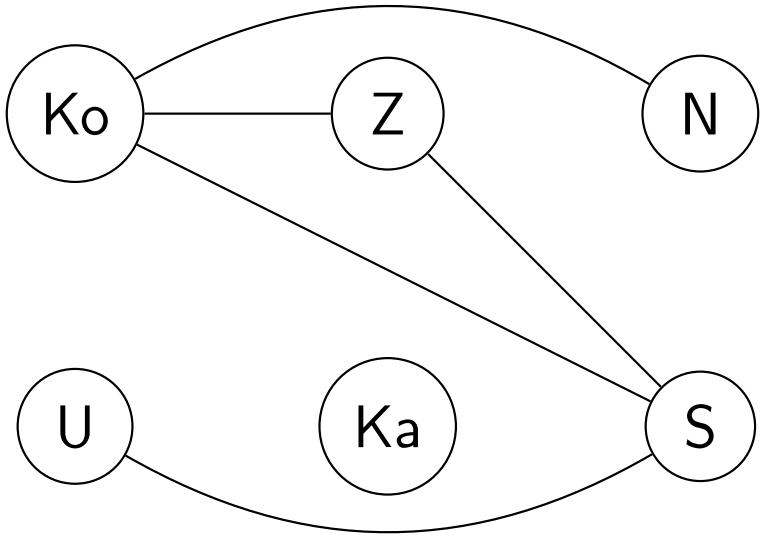
2025

The Maximum independant set problem



How many unconflicted people can we put in the crew?

The Maximum independant set problem



How many unconflicted people can we put in the crew? 64 possibilities

Ko	*Z*	*N*	*U*	*Ka*	*S*	Valid ?	Weight
×	×	×	×	×	×	N	–
×	×	×	×	×		N	–
×	×	×	×		×	N	–
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

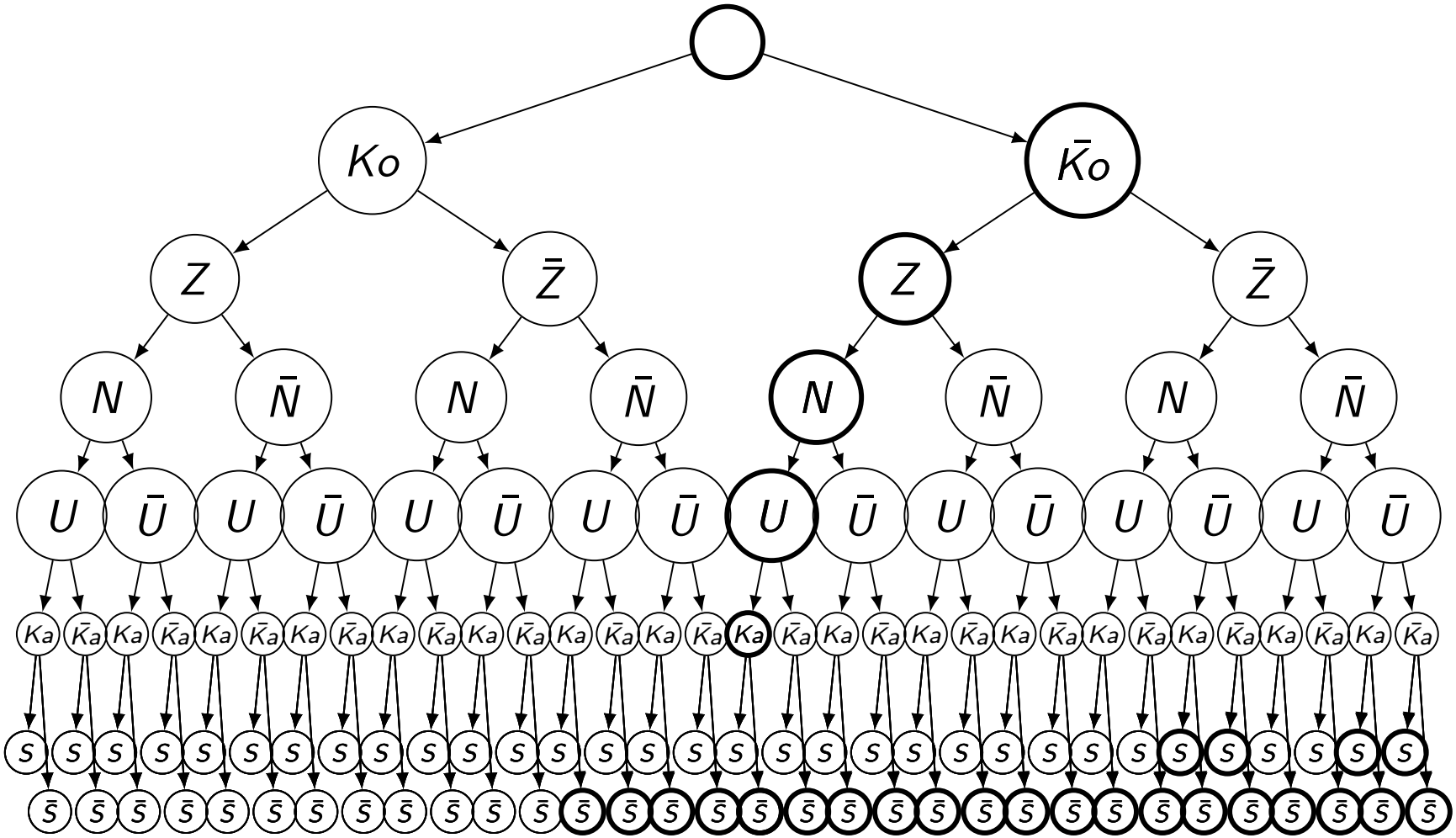
The Maximum independant set problem

64 possibilities in decreasing size order

Ko	Z	N	U	Ka	S	Valid ?	Weight
×	×	×	×	×	×	N	–
×	×	×	×	×		N	–
×	×	×	×		×	N	–
×	×	×		×	×	N	–
×	×		×	×	×	N	–
×		×	×	×	×	N	–
	×	×	×	×	×	N	–
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	×	×	×	×		V	4

Optimal solution : Z, N, U, Ka

Tree representation : branching



Tree representation : bound

If a solution contains Ko, what maximum value can we obtain?
3, because Ko removes Z, N and S.
It would be useless to explore that case if we already know a solution with value 3 or more.

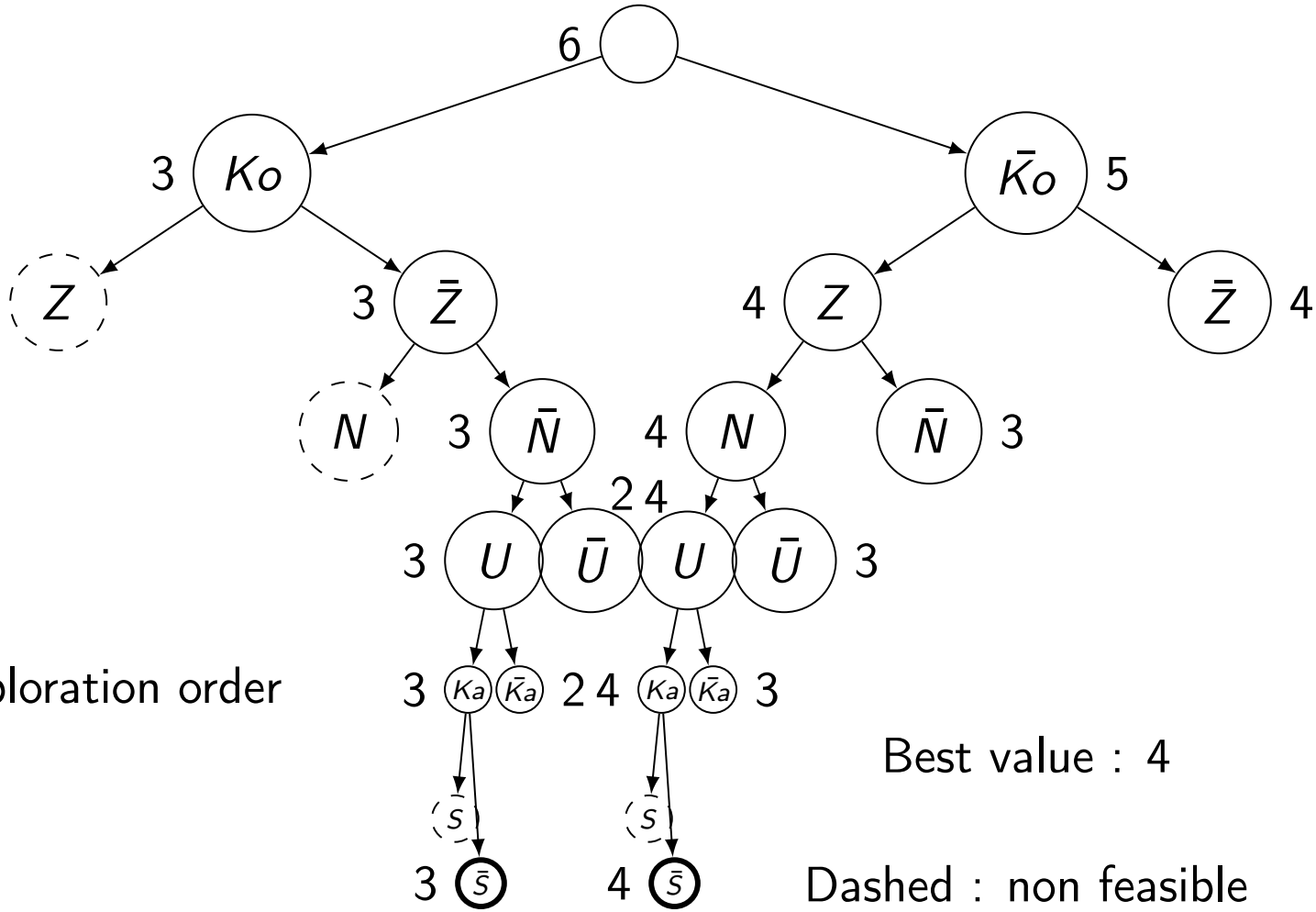
Tree representation : bound

If a solution contains $\bar{K}o$ and Z, what maximum value can we obtain?

4, because Z removes S, and $\bar{K}o$ removes Ko.

It would be useless to explore that case if we already know a solution with value 4 or more.

Left depth first search

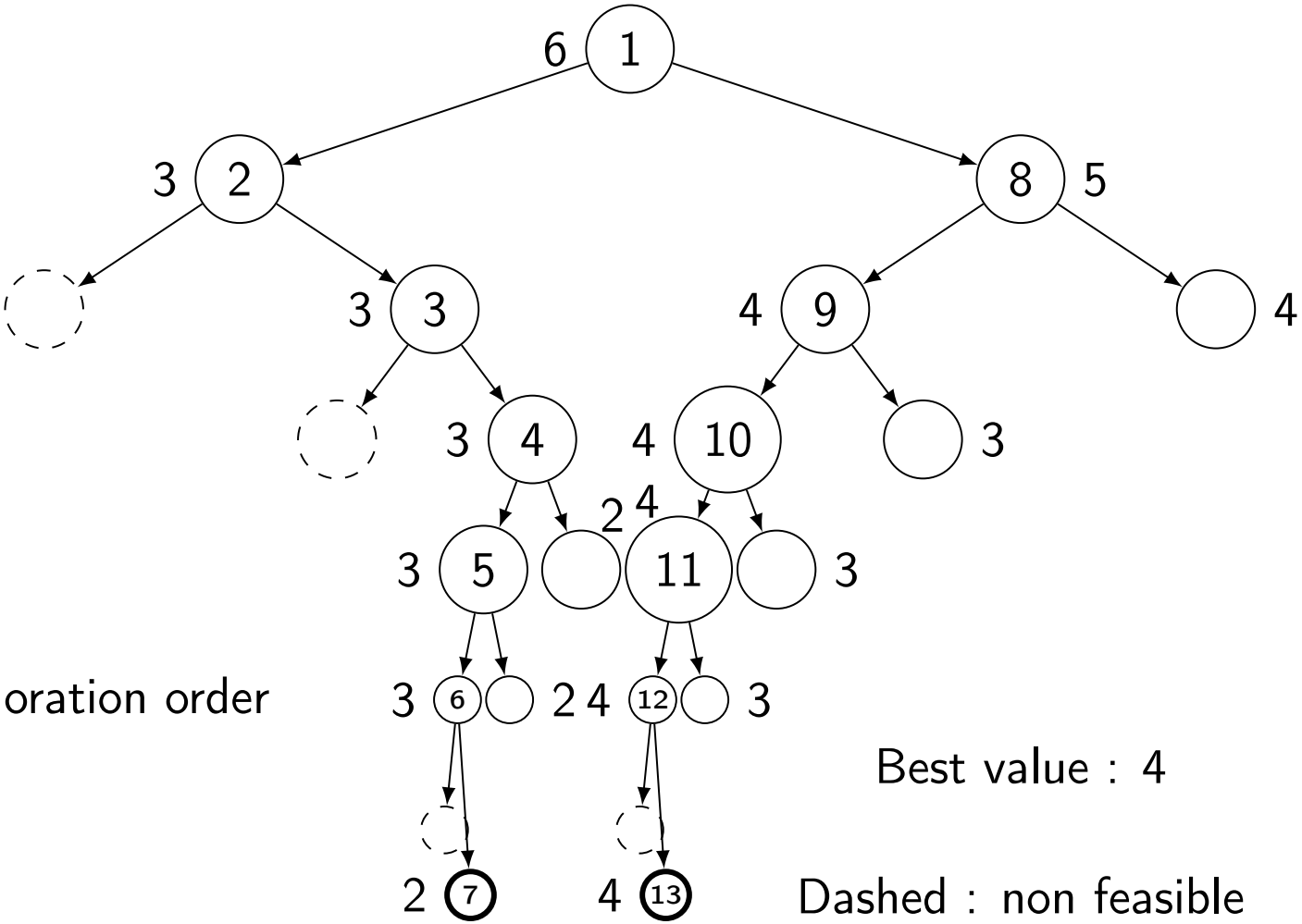


Exploration order

Best value : 4

Dashed : non feasible

Left depth first search

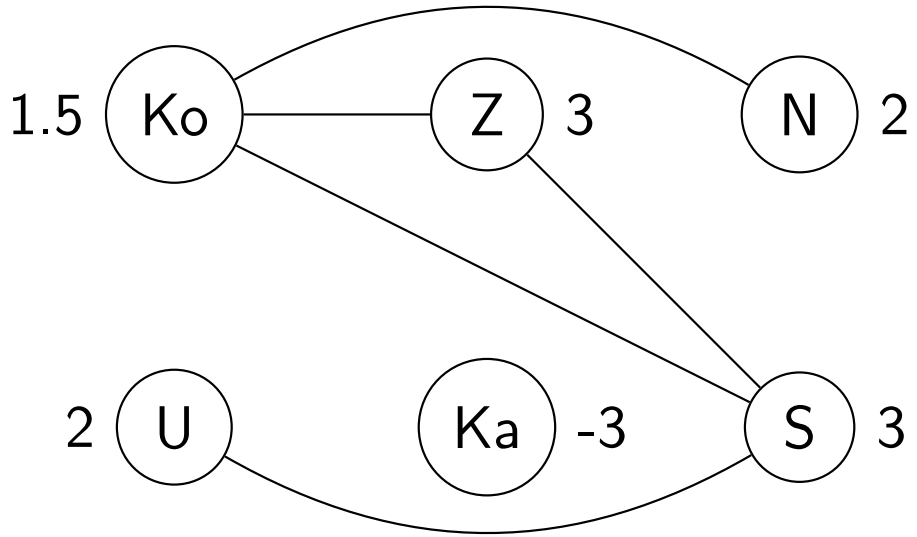


Exploration order

Best value : 4

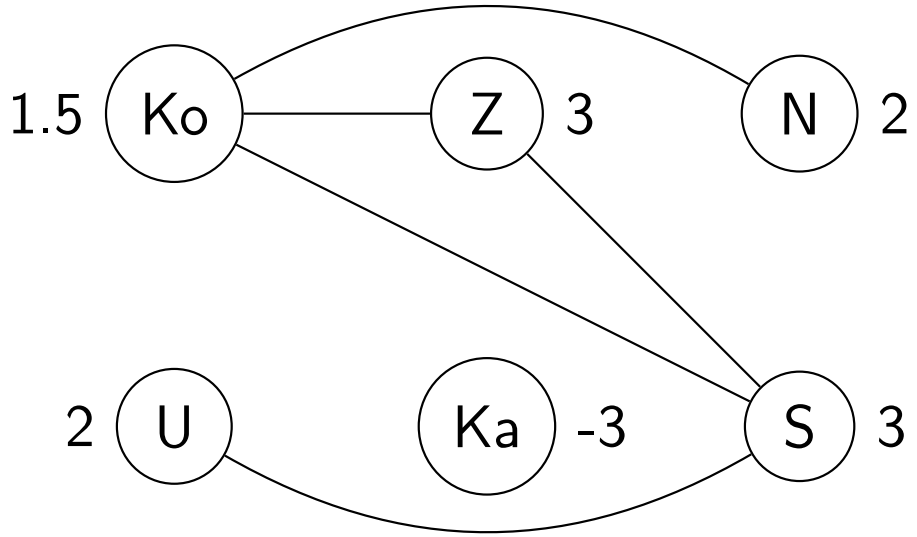
Dashed : non feasible

Maximum weighted selection problem



Each edge costs -1, each node is rewarded with its value, which nodes maximize the total value?

Maximum weighted selection problem



Each edge costs -1, each node is rewarded with its value, which nodes maximize the total value? 64 possibilities

Ko	*Z*	*N*	*U*	*Ka*	*S*	Weight
×	×	×	×	×	×	3.5
×	×	×	×	×		3.5
×	×	×	×		×	6.5
⋮	⋮	⋮	⋮	⋮	⋮	⋮

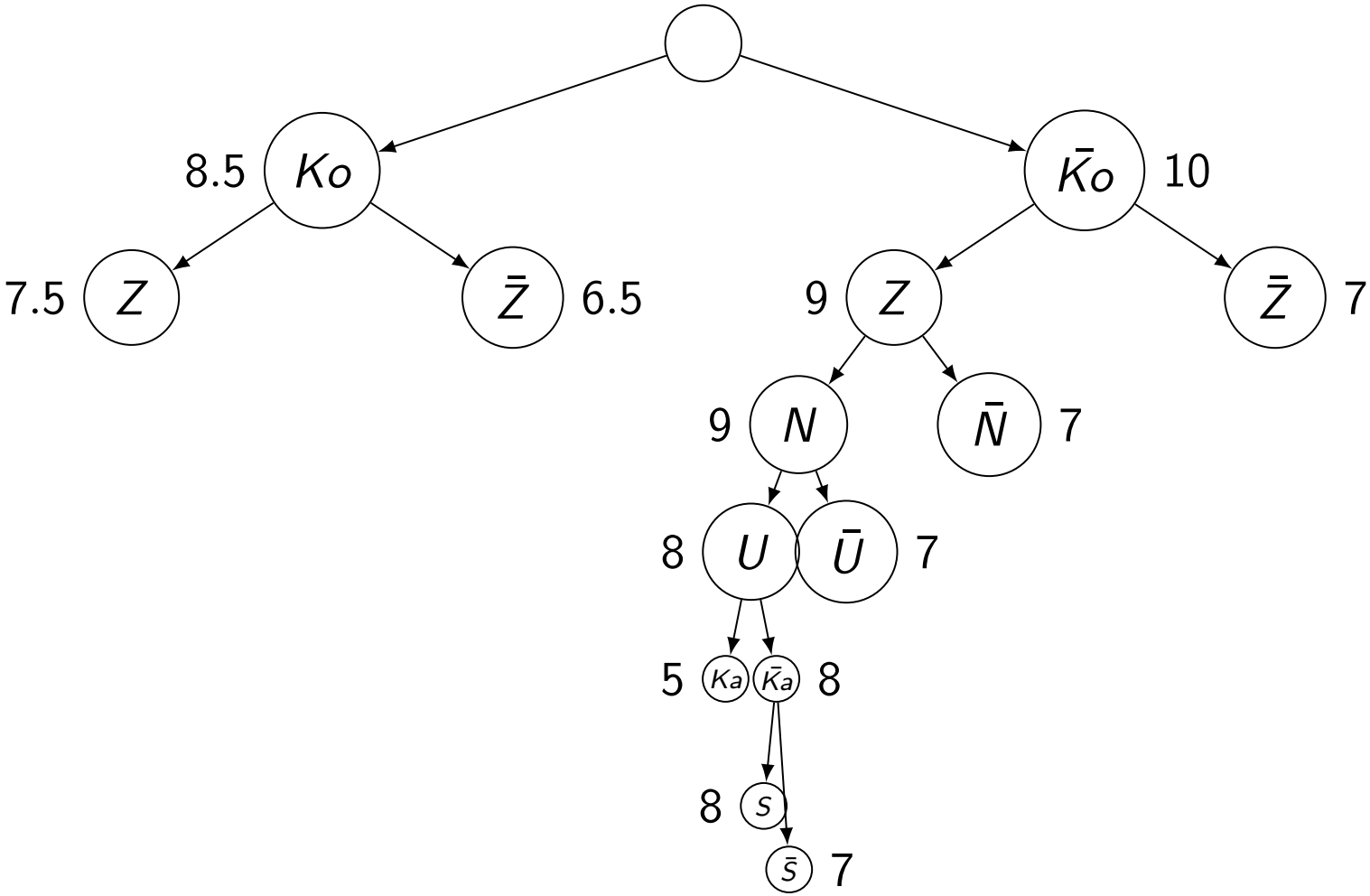
Tree representation : bound

If a solution contains *Ko*, what maximum value can we obtain?
8.5, indeed:

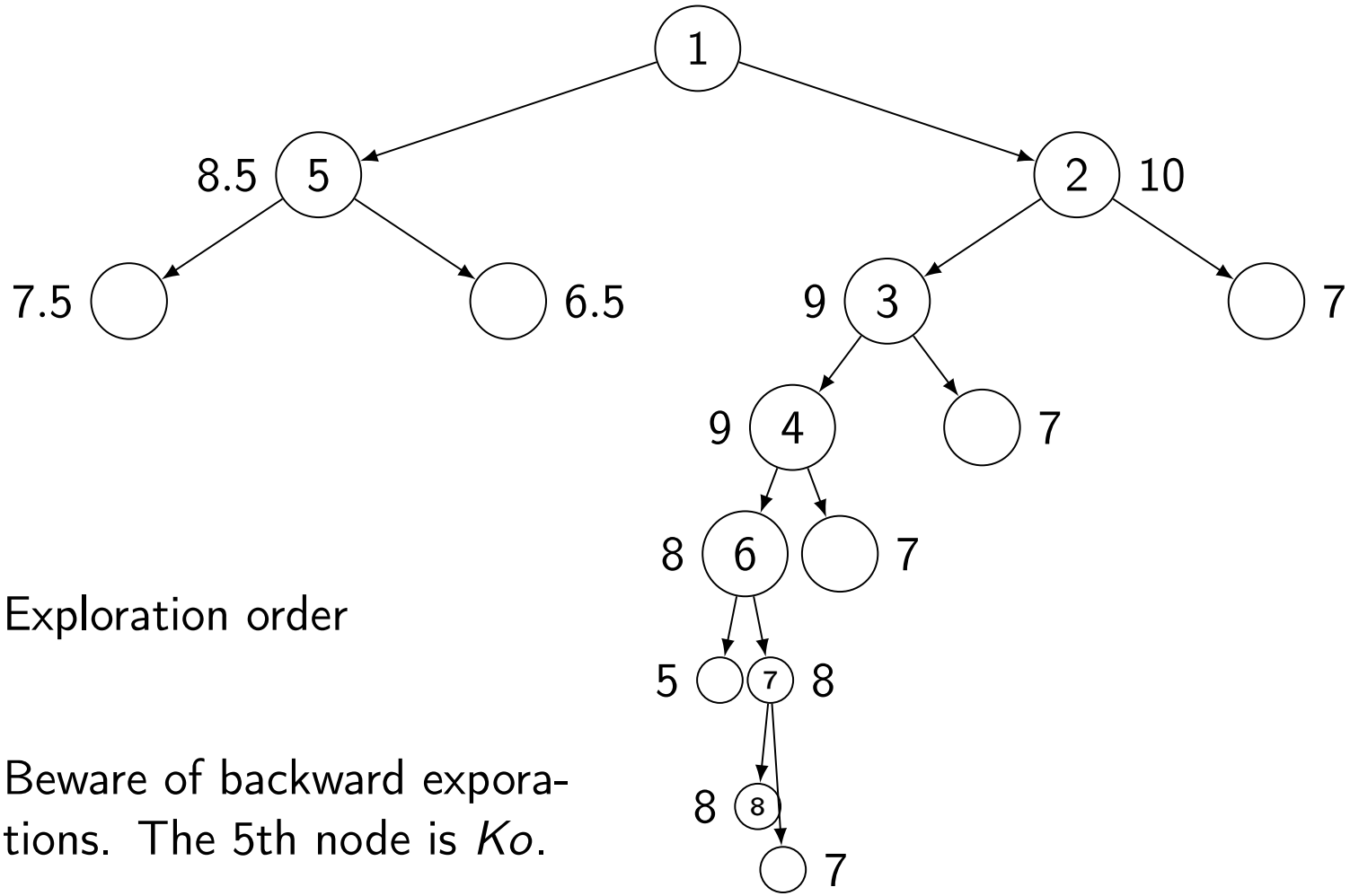
- *Ko* is rewarded with 1.5
- *Z* is rewarded with at most 2 instead of 3
- *N* is rewarded with at most 1 instead of 2
- *U* is rewarded with at most 2
- *Ka* is rewarded with at most 0
- *S* is rewarded with at most 2 instead of 3

It would be useless to explore that case if we already know a solution with value 8.5 or more.

Best first search



Best first search



Exploration order

Beware of backward exporations. The 5th node is *Ko*.

Other examples

On board

Principle

Definition

Branch and bound is a method for solving problems, it is not an algorithm but a way to build an algorithm.

In order to solve a problem with branch and bound, we need

- divide the problem into subproblems (usually we fix a variable);
- decide how to explore the subproblems tree;
- a way to bound the optimal solution of each subproblem.

Dividing into subproblems

Dividing into subproblems

Let Π be an optimization problem for which the set of feasible solutions is $\xi(\Pi)$. We say the function $S(\Pi)$ divide Π into subproblems if it returns a set of optimization problems $(\Pi_1, \Pi_2, \ldots \Pi_p)$ such that

$$\bigcup_{j=1}^{p} \xi(\Pi_j) = \xi(\Pi)$$

(Example on board)

Bound a subproblem optimal solution

Bound a subproblem optimal solution

Let Π be a **minimization** problem. Let $\omega(s)$ be the value of a feasible solution s of $\xi(\Pi)$. Finally, let s^* be an optimal solution of Π (minimizing $\omega(s^*)$).
A bound $B(\Pi)$ is a number that **can be computed fast** satisfying

$$B(\Pi) \leq \omega(s^*)$$

Note : if Π is a maximization problem,

$$B(\Pi) \geq \omega(s^*)$$

(Example on board)

Explore the subproblem trees

Explore a node

Exploring a node Π of the tree means compute $S(\Pi)$ and $B(\pi)$ for all $\pi \in S(\Pi)$.

Explore the subproblem trees

Depth first search exploration

We iterate through the tree with a depth first search algorithm: we always explore the deepest leftmost unexplored node. This method quickly find feasible solutions.

Best first exploration

We always explore the unexplored node with the best bound (the lowest for a minimization problem, the highest otherwise). This method firslty find a close-optimal solution with great probability.

Other method exists, of course.

Link with dynamic programming

Depth first search exploration algorithm

Algorithm 1 $BB(\Pi, uB)$, Default $uB = \infty$

Require: a **minimization** problem Π, and a real $uB \geq \omega(s^*)$

Ensure: The weight $\omega(s^*)$ of an optimal solution s^* of Π

1: **if** Π cannot be separated **then return** The weight $\omega(s^*)$ of an optimal solution s^* of Π
2: Explore Π
3: **for** $\Pi_j \in S(\Pi)$ such that $\xi(\Pi_j) \neq \emptyset$ **do**
4: $\quad lb \leftarrow B(\Pi_j)$
5: $\quad$ **if** $lb < ub$ **then**
6: $\quad\quad \omega \leftarrow BB(\Pi_j, ub)$
7: $\quad\quad ub \leftarrow \min(ub, \omega)$
8: **return** uB

Best first exploration algorithm

Algorithm 2 $BB(\Pi)$

Require: a **minimization** problem Π

Ensure: The weight $\omega(s^*)$ of an optimal solution s^* of Π

1: $L \leftarrow [\Pi]$; $uB \leftarrow +\infty$
2: **while** $L \neq \emptyset$ **do**
3: $\quad \pi \leftarrow \arg\min_{\pi \in L}(B(\pi))$; Remove π from L
4: $\quad$ **if** $B(\pi) \geq ub$ **then**
5: $\quad\quad$ **Break**
6: $\quad$ **if** π cannot be separated **then**
7: $\quad\quad ub = \min(ub,$ Weight $\omega(s^*)$ of an opt sol s^* of $\pi)$
8: $\quad$ **else**
9: $\quad\quad$ Explore Π
10: $\quad\quad$ Add all elements of $S(\Pi)$ to L
11: **return** uB