Node Selection Strategies

- We want to divide the current problem into two or more subproblems that are easier than the original. A commonly used branching method:

$$x_i \le \lfloor x_i^* \rfloor, \; x_i \ge \lceil x_i^* \rceil,$$

  where x_i^* is a fractional variable.
- Which variable to branch? A commonly used branching rule: Branch the most fractional variable.

Node Selection Strategies

Floor and Ceiling Functions, and Fractional Components

- The floor and ceiling functions map a real number to the largest previous or the smallest following integer, respectively.
- More precisely, floor(x) $= \lfloor x \rfloor$ is the largest integer not greater than x and ceiling(x) $= \lceil x \rceil$ is the smallest integer not less than x.
- The **fractional part**, is denoted by $\{x\}$ for real x and is defined by the formula

$$\{x\} = x - \lfloor x \rfloor.$$

- For all x, $0 \leq \{x\} < 1$.

x	Floor $\lfloor x \rfloor$	Ceiling $\lceil x \rceil$	Fractional part $\{x\}$
2	2	2	0
2.4	2	3	0.4
2.9	2	3	0.9

The *most fractional variable* is indicated by the number with the highest fractional part.

Node Selection Strategies

- We would like to choose the branching that minimizes the sum of the solution times of all the created subproblems.
- How do we know how long it will take to solve each subproblem?
  Answer: We don't.
  Idea: Try to predict the difficulty of a subproblem.
- A good branching rule: The value of the linear programming relaxation changes a lot!

Which Node to Select?

- An important choice in branch and bound is the strategy for selecting the next subproblem to be processed.
- Goals: (1) Minimizing overall solution time. (2) Finding a good feasible solution quickly.
- Some commonly used search strategies:
  - Best First
  - Depth-First
  - Hybrid Strategies
  - Best Estimate

The Best First Approach

- One way to minimize overall solution time is to try to minimize the size of the search tree. We can achieve this by choosing the subproblem with the best bound (lowest lower bound if we are minimizing).
- Drawbacks of Best First
  - Doesn't necessarily find feasible solutions quickly since feasible solutions are "more likely" to be found deep in the tree
  - Node setup costs are high. The linear program being solved may change quite a bit from one node evaluation to the next
  - Memory usage is high. It can require a lot of memory to store the candidate list, since the tree can grow "broad"

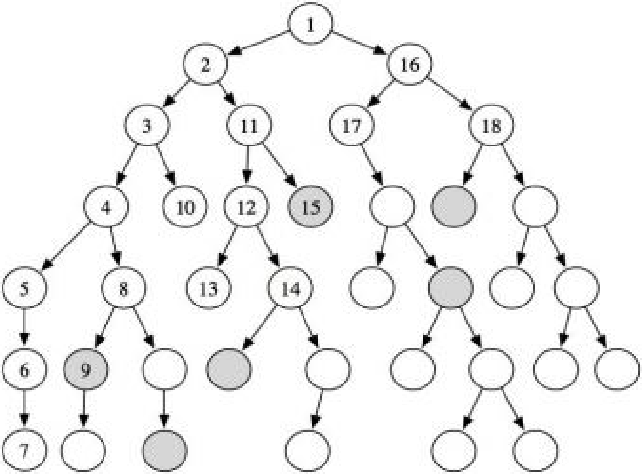
Node Selection Strategies

The Depth First Approach

- The depth first approach is to always choose the deepest node to process next. Just dive until you prune, then back up and go the other way
- This avoids most of the problems with best first: The number of candidate nodes is minimized (saving memory). The node set-up costs are minimized
- LPs change very little from one iteration to the next. Feasible solutions are usually found quickly
- Drawback: If the initial lower bound is not very good, then we may end up processing lots of non-critical nodes.
- Hybrid Strategies: Go depth-first until you find a feasible solution, then do best-first search

Node Selection Strategies

The nodes expanded in depth-first branch-and-bound search:



Review

- Be able to compare and contrast node selection strategies.