Branch & Bound Algorithms

- standard technique: very simple, but basis of all comercial codes (enhanced by other techniques + tricks)
- Idea: divide-and-conquer approach to explore the set of feas. integer solutions using bounds on opt. cost to avoid full exploration

$$\boxed{\begin{array}{l} \max\ c^T x \\ \text{s.t.}\ \ x \in F \end{array}} \longleftarrow F: \text{feasible region}$$

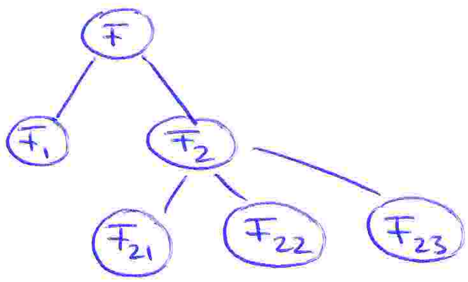
Branching:

If we cannot solve origin. problem directly ⇒ optimize over subproblems

Partition F into subsets $F_1 \cup \dots \cup F_k$,

$$\text{then } \max_{x \in F} c^T x = \max_{1 \le i \le k} \left\{ \max_{x \in F_i} c^T x \right\}.$$

F_i's might be still hard to solve ... ⇒ recurse.



... in extreme case: full enumeration! (avoid by bounding)

Bounding:

- Let z be some lower bound on the optimal cost: $z \le \max_{x \in F} c^T x$
- Note: any feas. solution for a subproblem i gives a lower bound on the opt. cost ⇒ update z if $z_i \ge z$.
- For each subproblem F_i:
  - → either, solve it optimally and update z if $z_i \ge z$ (might be difficult)
  - → or, compute upper bound $b(F_i) \ge \max_{x \in F_i} c^T x$
    - if $b(F_i) < z$ then ignore subproblem i
    - othw. branch or invest more into computing opt. z_i

Very generic framework. Questions (flexibility in):

1) How to break into subproblems
2) Which subproblem to choose?
3) How to obtain upper bounds?

1) How to break into subproblems? $\rightsquigarrow$ several ways, problem dependent

example A: general ILP
example B: TSP in directed graphs

} see later

2) Which subproblems to choose?

some typical ways: DFS, BFS, Best-First-Search (best bound), and various others or combinations

3) How to obtain upper bounds?

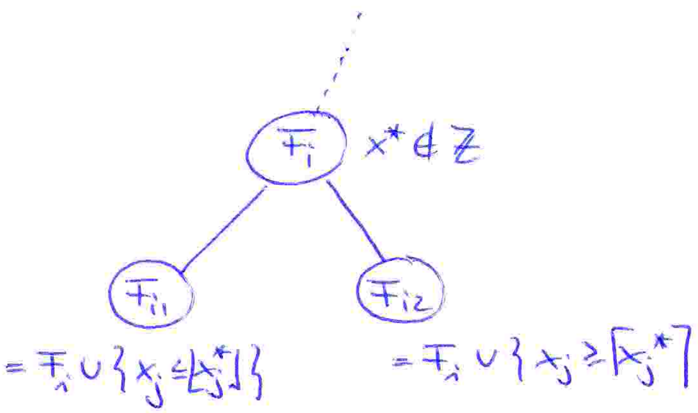
e.g.:
- LP relaxation
- other relaxations (Lagrangean relax, next)
- utilize cutting planes

Ex.: general ILP

- bounding: LP relaxation
- if solution x^* is integral $\Rightarrow$ no further branching, update $\underline{z}$

otherw. pick some non-integral component x_i^* and create two subproblems by adding one of the two constraints:

$$x_i \leq \lfloor x_i^* \rfloor \qquad \text{or} \qquad x_i \geq \lceil x_i^* \rceil$$



Ex. TSP in directed graphs

$$\min \sum_{i,j} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_j x_{ij} = 1 \quad , \; i = 1, \dots, n$$

$$\sum_i x_{ij} = 1 \quad , \; j = 1, \dots, n$$

$$\left.\sum_{i,j \in S} x_{ij} \le |S| - 1 \; , \; S \subseteq \{1, \dots, n\} \quad \right] \text{ subtour elimin. const.}$$

$$x_{ij} \in \{0,1\}$$

⟶ Relaxation w/out subtour elimin. const.

= assignment problem (earlier lecture)

⟹ can be solved efficiently to optimality.

Branch & bound for TSP:

- bounding strategy: solve assignment problem relaxation ⇝ x^*
- branching: if x^* ~~satisfies~~ is a tour ⟹ optimal for subproblem

  otherw. branch as follows:

  Choose shortest cycle (resp. # edges) $e_1, \dots, e_k$ and create subproblems by setting one

  $x_{e_i} = 0$, for $i = 1, \dots, k$.

Comment on finding feas. solutions for TSP (heuristics)

- Nearest neighbor (Greedy)
  - start at any node; visit the nearest node not yet visited (repeat); return to start node
  - fast, easy to implement, empirically quite good "1.26 times opt cost" on TSPLIB '97. (not a guarantee)
- Tour improvement: 2-OPT (k-OPT, local search)
  - given a tour, consider any two non-adjacent edges: remove them and reconnect (unique way) to new tour. If cost improved, then keep, otherw. discard and continue.

  [illegible] TSPLIB '97 [illegible]