Lecture 3 - Branch and Bound

Lan Peng, Ph.D.

School of Management, Shanghai University, Shanghai, China

"The time to relax is when you don't have time for it."

1 Preliminary

1.1 Why Rounding Can be Bad?

IP example Rounding can be bad because the optimal of IP can be far away from optimal of LP. For example,

$$\begin{aligned}
\max \quad & z = x_1 + 0.64x_2 \\
\text{s.t.} \quad & 50x_1 + 31x_2 \le 250 \\
& 3x_1 - 2x_2 \ge -4 \\
& x_1, x_2 \ge 0 \quad \text{(for LP)} \\
& x_1, x_2 \in Z^+ \quad \text{(for IP)}
\end{aligned}$$

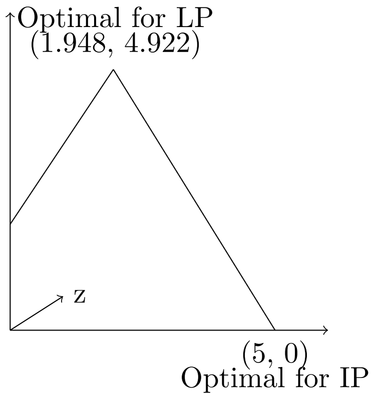

Figure 1: Optimal solution for LP / IP

QAP example Rounding can make the LP useless. For example, for QAP problem, the IP model is

$$\begin{aligned}
\min \quad & z = \sum_{i\in D}\sum_{s\in O} c_{is}x_{is} + \sum_{i\in D}\sum_{j\in D}\sum_{s\in O}\sum_{t\in O} w_{ij}^{st}y_{ij}^{st} \\
\text{s.t.} \quad & \sum_{i\in D} x_{is} = 1, \quad s \in O \\
& \sum_{s\in O} x_{is} = 1, \quad i \in D \\
& x_{is} \in \{0,1\}, \quad i \in D, s \in O
\end{aligned}$$

$$\begin{aligned}
&y_{ij}^{st} \geq x_{is} + x_{jt} - 1, \quad i \in D, j \in D, s \in O, t \in O \\
&y_{ij}^{st} \geq 0, \quad i \in D, j \in D, s \in O, t \in O \\
&y_{ij}^{st} \leq x_{is}, \quad i \in D, j \in D, s \in O, t \in O \\
&y_{ij}^{st} \leq x_{jt}, \quad i \in D, j \in D, s \in O, t \in O
\end{aligned}$$

We can get the optimal solution for LP supposing $\forall i, s \quad x_{is} \in [0,1]$

$$\begin{aligned}
&x_{is} = \frac{1}{|D|}, \quad i \in D, s \in O \\
&y_{ij}^{st} = 0, \quad i \in D, j \in D, s \in O, t \in O
\end{aligned}$$

1.2 Relaxation

Local Optimal v.s. Global Optimal Let

$$\begin{aligned}
Z_s &= \min\{f(x) : x \in S\} \\
Z_t &= \min\{f(x) : x \in T\} \\
S &\subset T
\end{aligned}$$

then

$$Z_t \leq Z_s$$

Notice that if $x_T^* \in S$ then $x_S^* = x_T^*$, to generalized it, We have

$$\begin{aligned}
&\begin{cases} x_T^* \in \arg\min\{f(x) : x \in T\} \\ x_T^* \in S \end{cases} \\
&\Rightarrow x_T^* \in \arg\min\{f(x) : x \in S\}
\end{aligned}$$

Especially for IP, we can take the LP relaxation as T and the original feasible region of IP as S, therefore, if we find an optimal solution from LP relaxation T which is also a feasible solution of S, then it is the optimal solution for IP (S)

LP Relaxation To perform the LP relaxation, we expand the feasible region

$$\begin{aligned}
x \in \{0,1\} &\to x \in [0,1] \\
y \in Z^+ &\to y \geq 0
\end{aligned}$$

If we have $Z_{LP}(s) = conv(s)$ then

$$LP(s) : x \in R_+^n : Ax \leq b$$

The closer $LP(s)$ is to $conv(s)$ the better. Interestingly, we only need to know the convex in the direction of c.

For IP problem

$$\begin{aligned} Z_{IP} \quad \max \quad & z = cx \\ \text{s.t.} & Ax \le b \\ & x \in Z^n \end{aligned}$$

In feasible region $S = \{x \in Z^n, Ax \le b\}$, the optimal solution $Z_{IP} = \max\{cx : x \in S\}$. Denote $conv(S)$ as the convex hull of S then

$$Z_{IP}(S) = Z_{IP}(conv(S))$$

Relation Between LP Relaxation and IP Let

$$\begin{aligned} Z_{IP} &= \max_{x \in S} cx, \quad \text{where } s \text{ is a set of integer solutions} \\ Z_{LP} &= \max cx, \quad \text{the LP relaxation of IP} \end{aligned}$$

then

$$\begin{aligned} Z_{IP} &= \max_{1 \le i \le k} \{\max_{x \in S_i} cx\} \\ \text{iff} \quad S &= \bigcup_{1 \le i \le k} S_i \end{aligned}$$

Notice that S_i don't need to be disjointed.

LP feasibility and IP(or MIP) feasibility Solve the LP relaxation, one of the following things can happen

- LP relaxation is infeasible $\rightarrow$ MIP is infeasible
- LP relaxation is unbounded $\rightarrow$ MIP is infeasible or unbounded
- LP relaxation has optimal solution $\hat{x}$ and $\hat{x}$ are integer ($\hat{x} \in S$), $\rightarrow Z_{IP} = Z_{LP}$
- LP relaxation has optimal solution $\hat{x}$ and $\hat{x}$ are not integer ($\hat{x} \notin S$), now defines a new upper bound, $Z_{LP} \ge Z_{IP}$

If the first three happens, stop, if the fourth happens, we branch and recursively solve the sub-problems.

2 Branch and Bound

2.1 Algorithm overview

Algorithm 1 Branch and Bound (For maximization problem)

```
 1: find a feasible solution as the initial Lower bound $L$
 2: put the original LP relaxation in candidate list $S$
 3: while $S \neq \emptyset$ do
 4:     select a problem $\hat{S}$ from $S$
 5:     solve the LP relaxation of $\hat{S}$ to obtain $u(\hat{S})$
 6:     if LP is infeasible then
 7:         $\hat{S}$ pruned by infeasibility
 8:     else if LP is unbounded then
 9:         return Unbounded
10:     else if LP $u(\hat{S}) \leq L$ then
11:         $\hat{S}$ pruned by bound
12:     else if LP $u(\hat{S}) > L$ then
13:         if $\hat{x} \in S$ then
14:             $u(\hat{S})$ becomes new $L$, $L = u(\hat{S})$
15:         else if $\hat{x} \notin S$ then
16:             branch and add the new sub-problems to $S$
17:             if LP $u(\hat{S})$ is at current best upper bound then
18:                 set $U = u(\hat{S})$
19: if Lower bound exists then
20:     find the optimal at $L$
21: else
22:     Infeasible
```

2.2 Idea of Divide and Conquer

For each iteration, divide the feasible region of LP into two parts (and an infeasible part), solve the LP in those parts.

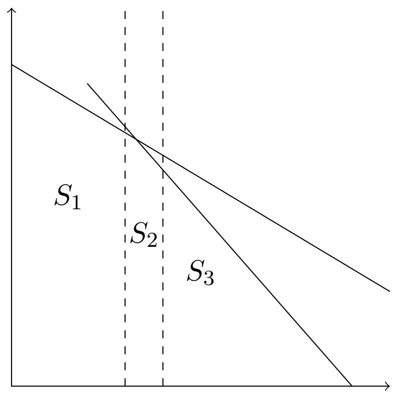

Figure 2: Divide and Conquer

In this iteration, the original feasible region have been partition into three parts, where S_2 is infeasible for IP because there is not integer point in it. We continue the iteration for S_1 and S_2. Each partition is suppose to give a new upper bound / lower bound and reduce the infeasible space. If the temp optimal integer in S_1 is larger than the LP relaxation in S_3, we can cut S_3.

3 Searching, Branching, and Pruning

3.1 Strategies in B&B

$$\begin{tikzcd}
 & & \text{Branch-and-Bound} \arrow[dll] \arrow[d] \arrow[drr] & & \\
\text{Search Strategy} \arrow[dr] & & \text{Branching Strategy} \arrow[dl] \arrow[dr] & & \text{Pruning Strategy} \arrow[dl] \\
 & \begin{matrix}\text{Phase 1:}\\ \text{Search}\end{matrix} \arrow[rr] \arrow[d] & & \begin{matrix}\text{Phase 2:}\\ \text{Verification}\end{matrix} \arrow[d] & \\
 & \begin{matrix}\text{Heuristic}\\ \text{Solution}\end{matrix} & & \begin{matrix}\text{Certification of}\\ \text{Optimality}\end{matrix} &
\end{tikzcd}$$

3.2 Search Strategy: Choose Node to Branch

Depth-first search It can be implemented by maintaining the list of unexplored subproblems L as a stack. The algorithm removes the top item from the stack to choose the next subproblem to explore, and when children are generated as a result of branching, they are inserted on the top of L. Thus, the next subproblem that is explored is the most recently generated subproblem.

- DFS does not need to store the entire list of unexplored subproblems (only stores the path from the root of T to the current subproblem)
- If no unexplored children remain, the algorithm backtracks to the closest ancestor node with unexplored children.
- Use the LP relaxations as lower bounds. The LP solver can often reuse information from the parent LP solution as a starting point for the child LP solution
- Naive implementations of DFS do not use any information about problem structure or bounds
- Search tree is extremely unbalanced

Breadth-first search BFS explores all subproblems that are at a fixed distance from the root before exploring any deeper subproblems.

- Finding an optimal solution that is closest to the root of the tree, thus operating well on unbalanced search trees
- Generally unable to exploit pruning rules that compare against the current incumbent solution $\Rightarrow$ high memory
- Best bound search $\Rightarrow$ improve dual bound
- Best estimate search $\Rightarrow$ improve primal bound

Best-first search, A* search makes use of a heuristic measure-of-best function to compute every subproblem by a admissible index μ.

3.3 Branching Strategy: Choose Branching Variable

The Most Violated Integrality constraint Pick the j of which $x_j - \lfloor \hat{x_j} \rfloor$ is the closest to 0.5

Strong Branching Select a few candidates (K), create sub-problems for each of these variables, run a few dual simplex iterations to see the improved bounds, select the variable with the best bounds, and then selects for branching the variable that induces the most change in the objective.

- Might fix variable, when one side is infeasible
- Detect infeasibility, when both side are infeasible
- Can be speed up by limit the number of iterations, or stop when improvement is found.
- Domain propagation

Pseudo-cost Branching Pseudo-cost attempts to predict the per-unit change in the objective function for each candidate branching variable, based on past experience in the tree. The basic idea of this strategy is to keep track for each variable x_i the change in the objective function when this variable was previously chosen as the variable to branch on. By branching on a variable that is expected to produce a significant change in the objective function, it is more likely that the generated subproblems can be pruned. One difficulty with pseudocost branching is that no information about past branching behavior is available at the beginning of the algorithm, so the pseudocosts for each variable must be initialized in some way.
For variable $x_j \in K$,

$$\begin{cases} P_j^+, & \text{bound reduction if rounded up} \\ P_j^-, & \text{bound reduction if rounded down} \end{cases}$$

define $f_j = x_j - \lfloor x_j \rfloor$

$$\begin{cases} D_j^+ = P_j^+(1 - f_j) \\ D_j^- = P_j^- f_j \end{cases}$$

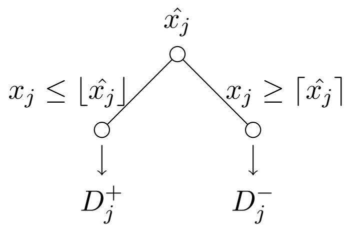
$\hat{x_j}$
$x_j \leq \lfloor \hat{x_j} \rfloor$
$x_j \geq \lceil \hat{x_j} \rceil$
D_j^+
D_j^-

Figure 3: Strong Branching

For those variables in K find the

- $\max\{\min\{D_j^+, D_j^-\}\}$, or

- $\max\{\max\{D_j^+, D_j^-\}\}$, or
- $\max\{\frac{D_j^+ + D_j^-}{2}\}$, or
- $\max\{\alpha_1 \min\{D_j^+, D_j^-\} + \alpha_2 \max\{D_j^+, D_j^-\}\}$

to branch.

3.4 Branching Strategy: Create Offspring

Traditional Branching For $\hat{x} \notin S$, $\exists j \in N$ such that

$$\hat{x_j} - \lfloor \hat{x_j} \rfloor > 0$$

Create two sub-problems

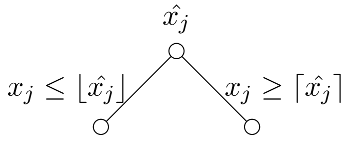
$\hat{x_j}$
$x_j \leq \lfloor \hat{x_j} \rfloor$
$x_j \geq \lceil \hat{x_j} \rceil$

Figure 4: Traditional Branching

Constraint Branching Use parallel constraints to branch, e.g.

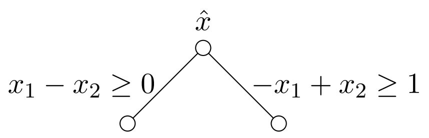

Figure 5: Constraint Branching

Special Ordered Sets of type 1 (SOS1 or S1) are a set of variables, at most one of which can take a non-zero value, all others being at 0.

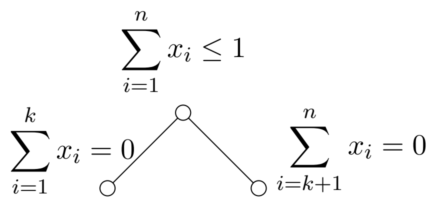

Figure 6: SOS1 Branching

Special Ordered Sets of type 2 (SOS2 or S2) : an ordered set of non-negative variables, of which at most two can be non-zero, and if two are non-zero these must be consecutive in their ordering.

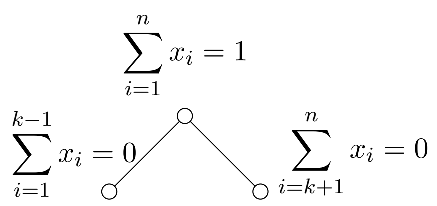

Figure 7: SOS2 Branching

Ryan-Foster Ryan-Foster is for Set covering problem. The typical model is

$$\begin{aligned} \min \quad & \sum_{i \in C} x_i \\ \text{s.t.} \quad & \sum_{i \in C} a_{ij} x_i \geq 1, \quad \forall j \in U \\ & x_i \in \{0,1\}, \quad \forall i \in C \end{aligned}$$

For any fractional solution, there are at least two elements (i, j) so that i and j are both partially covered by the same set S, but there is another set T that only covers i

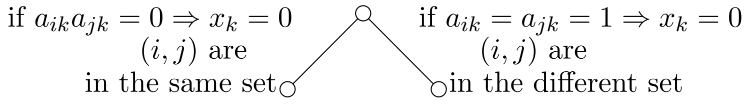
if $a_{ik}a_{jk} = 0 \Rightarrow x_k = 0$
(i, j) are
in the same set
if $a_{ik} = a_{jk} = 1 \Rightarrow x_k = 0$
(i, j) are
in the different set

Figure 8: Ryan Foster Branching

3.5 Pruning Rules

Lower bound The most common way to prune is to produce a lower bound on the objective function value at each subproblem, and use this to prune subproblems whose lower bound is no better than the incumbent's solution value. Lower bounds are computed by relaxing various aspects of the problem.

- Many different lower bound can be computed
- Attempt to prune using the easy lower bounds first, then move on to more complex
- One of the "easy lower bound" can be LP relaxation
- Another "easy lower bound" can be Lagrangian relaxation for IP.

Dominance relations In contrast to lower bounding rules, dominance relations allow subproblems to be pruned if they can be shown to be dominated by some other subproblem. In other words, if subproblem S_1 dominates subproblem S_2 , this means that for any solution that is a descendant of S_2 , there exists a complete solution descending from S_1 that is at least as good.

4 *Use Branch-and-Bound to Solve Other Problems

Close-enough Traveling Salesman Problem Given a set of vertices $V = \{0, 1, \cdots, n\}$ in a 2-dimensional space with coordinates $(\bar{x}, \bar{y}), i = 0, 1, \cdots, n$. Each vertex i is in the center of a convex region bounded by a circle D_i with radius r_i. Assume that $(\bar{x}_i, \bar{y}_i) \neq (\bar{x}_j, \bar{y}_j), \forall i, j \in V, i \neq j$. The problem lies in determining the value of the coordinates of the hitting points $(\bar{x}_i, \bar{y}_i) \in \mathbb{R}^2$ and a sequence $L = (k_0, k_1, \cdots, k_n), k_i \in V$ representing the order in which the vertices are covered, such that the tour over the hitting points forms a Hamiltonian cycle of minimum length and $(\bar{x}_i, \bar{y}_i) \in D_i, i \in V$.

Coutinho et al. 2016 Each branch-and-bound node is associated with an optimal partial tour that needs to visit only a given subset of vertices in a particular order. At the root node, the algorithm chooses three vertices to generate an initial sequence of nodes that need to be visited. Because there are only three vertices involved in this sequence and costs are symmetric, their order will not affect the solution. Therefore, the problem of finding an optimal tour that visits these three vertices in the given order is a valid relaxation of the main problem regardless of the choice of the initial sequence. If the associated solution is feasible, i.e., all customers are covered, then this solution is optimal and the problem is solved. Otherwise, for this root node, the algorithm branches into three subproblems; in each of them, a vertex that does not belong to the tour is inserted in a different position. A node is discarded if its cost is greater than or equal to the best known upper bound or if its associated solution is feasible. Otherwise, a branching is performed over this node using the same rationale applied in the root node. Note that the number of child nodes (subproblems) for every node in the tree is equal to the number of vertices in the partial sequence of the parent node. (Coutinho et al. 2016)

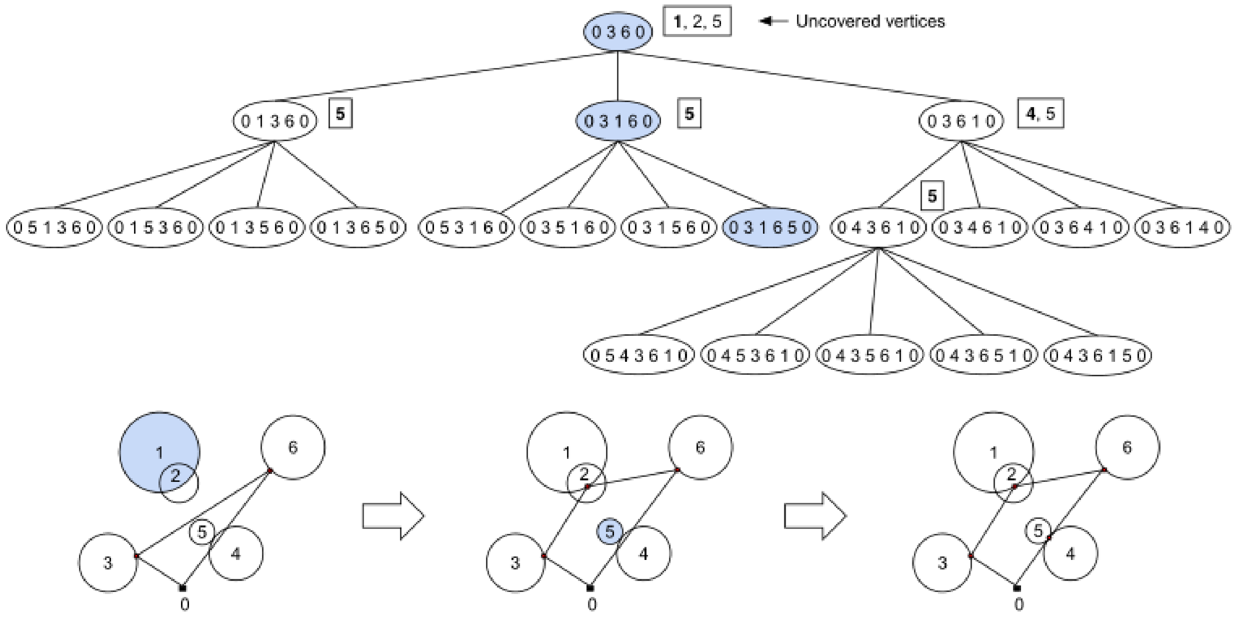

Figure 9: An example of the B&B algorithm for an instance with 7 vertices

References

W. Coutinho, R. Nascimento, A. Pessoa, and A. Subramanian. A branch-and-bound algorithm for the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 28(4):752–765, 2016.