Branch and Bound

Milind G. Sohoni[1]

July 23, 2014

[1] Indian School of Business, Hyderabad, India, e-mail: milind_sohoni@isb.edu

Outline

Branch and Bound

An Example

Branch and Bound Algorithm

- *Branch and bound* is the most commonly-used algorithm for solving MILPs
- Basically, it is a "divide and conquer" approach
- Suppose F is the feasible region for some MILP and we wish to solve $\min_{x \in F} c^T x$
- Consider a "partition" of F into subsets $F_1, \ldots, F_k$. Then,

$$\min_{x \in F} c^T x = \min_{1 \leq i \leq k} \left\{ \min_{x \in F_i} c^T x \right\}.$$

- In other words, we can optimize over each set separately
- *Idea*: If we can't solve the original problem directly, we might be able to solve the smaller subproblems recursively
- Dividing the original problem into subproblems is called *branching*
- In the extreme case, this algorithm is equivalent to complete enumeration

Branch and Bound Algorithm

- Next, let us discuss the role of *bounding*
- For simplicity let us assume that all "variables" have finite upper and lower bounds
- Any feasible solution to the problem provides and upper bound $u(F)$ on the optimal objective value
- We can use heuristics to obtain an upper bound
- *Idea*: After *branching* we try to obtain a lower bound $b(F_i)$ on the optimal solution for each of the subproblems
- If $b(F_i) \geq u(F)$, then we don't need to consider the subproblem i
- One easy way to obtain an lower bound is to solve the "LP relaxation" obtained by dropping the integrality constraints

LP-based Branch and Bound

- In "LP-based branch and bound", we first solve the "LP relaxation" of the original problem. The result is one of the following:
  - The LP is infeasible $\Rightarrow$ MILP is infeasible
  - The LP is feasible with an integer solution $\Rightarrow$ Optimal solution to the MILP
  - LP is feasible but has a fraction solution $\Rightarrow$ Lower bound for the MILP
- In the first two cases, we are done
- In the third case, we must *branch* and recursively solve the resulting subproblems

Branching in LP-based Branch and Bound

- The most common way to *branch* is as follows:
  - Select a variable i whose value $\hat{x}_i$ is fractional in the LP solution
  - Create two subproblems:
    - In one subproblem, impose the constraint $x_i \geq \lceil \hat{x}_i \rceil$
    - In the other subproblem, impose the constraint $x_i \leq \lfloor \hat{x}_i \rfloor$
  - This is called a *branching rule*
  - Why is this a valid rule?
- We will look at an example for a 0-1 integer program

Continuing the Algorithm after Branching

- After *branching* we solve the subproblems recursively
- Now we have to consider something else
- If the optimal objective value of the LP relaxation is greater than the current upper bound, we need not consider the current subproblem further (*pruning*)
  - If $Z_i^{LP} > Z^{IP}$ then prune subproblem i
- This is the key to the potential efficiency of the problem
- Terminology
  - If we picture the subproblems graphically, they form a *search tree*
  - Each subproblem is linked to its parent and eventually to its *children*
  - Eliminating a problem from further consideration is called *pruning*
  - The act of bounding and then branching is called *processing*
  - A subproblem that has not yet been considered is called a *candidate* for processing
  - The set of candidates for processing is called the *candidate list*

LP-based Branch and Bound Algorithm

1. To begin, we find and upper bound U using a pre-processing/heuristic routine
2. We start with the original problem on the candidate list
3. Select problem S from the candidate list and solve the *LP relaxation* to obtain the lower bound $b(S)$
   - If LP is infeasible $\Rightarrow$ node is pruned
   - Otherwise, if $b(S) \geq U \Rightarrow$ node is pruned
   - Otherwise, if $b(S) < U$ and the solution is feasible for the MILP $\Rightarrow$set $U \leftarrow b(S)$
   - Otherwise, *branch* and add the new subproblem to the candidate list
4. If the candidate list is non-empty, go to STEP 2. Otherwise, the algorithm is done

Selecting the Candidate to Solve

- Selecting the next candidate to process
  - "Best-first" always chooses the candidate with the lowest lower bound
  - This rule minimizes the size of the "tree" (Why?)
  - But there are many practical reasons to deviate from such a rule
- "Depth-first" or "Breadth-first" are other possibilities; the former being more common to find an initial upperbound quickly
- Choosing a branching rule
  - Branching "wisely" is important; else the algorithm can take very long to converge, if at all
- LP relaxation is the most common method used to find lower bounds

An Example

- Consider the following 0-1 knapsack problem:

$$\begin{aligned} \max \quad & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & x \in \{0, 1\}^4 . \end{aligned}$$

- The linear relaxation solution is $x = \{1, 1, 0.5, 0\}$ with the objective value of 22. The solution is not integral
- We choose to branch on x_3. Essentially, the next two subproblems will have $x_3 = 0$ and $x_3 = 1$ as constraints respectively
- Here is the B & B solution tree

The Branch and Bound Tree

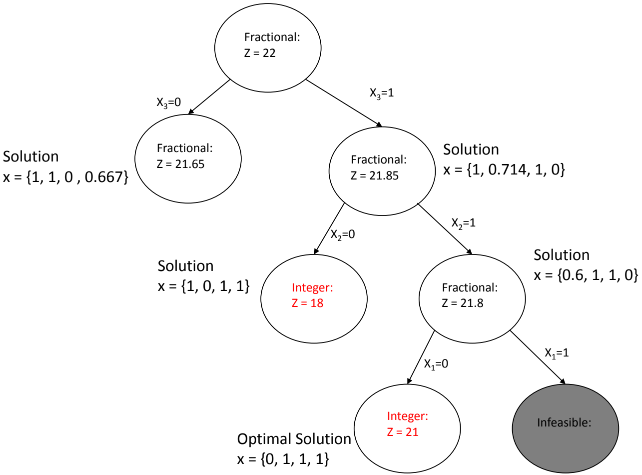

Figure: The Branch and Bound Solution Tree.

The DFS Tree

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

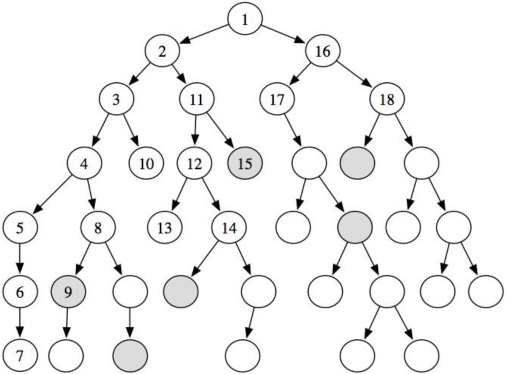

Figure: The Depth First Search Tree.

The DFS 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