Branch and Bound

Josh Wilcox (jw14g24)

May 6, 2025

Table of Contents

Overview

Overview of Branch and Bound

- Branch and bound is a general algorithm design paradigm that can be used to solve optimization problems
- It is a **generalisation** of **backtracking** algorithms
  - Recall backtracking explores the **entire** search space to find a solution
- Branch and bound can be more efficient than simple backtracking as it **prunes** the search space if it knows all branches underneath a certain points are not possible
  - This is done by using the **upper** and **lower** bounds of the problem to discard certain branches of the search tree

Branch and Bound for Subset Sum

Problem Definition

- Given: Set of integers $S = \{x_1, x_2, ..., x_n\}$ and target sum T
- Goal: Find subset $S' \subseteq S$ such that $\sum_{x \in S'} x = T$

Branch and Bound Strategy

- **Branching:**
  - At each node, make two choices for current element x_i:
    - Left branch: Include x_i in subset
    - Right branch: Exclude x_i from subset
- **Bounding:**
  - Let `current_sum` = sum of selected elements
  - Let remaining = sum of elements still to consider
  - Prune branch if:
    - $currentSum > T$
    - $currentSum + remaining < T$

Example Search Space

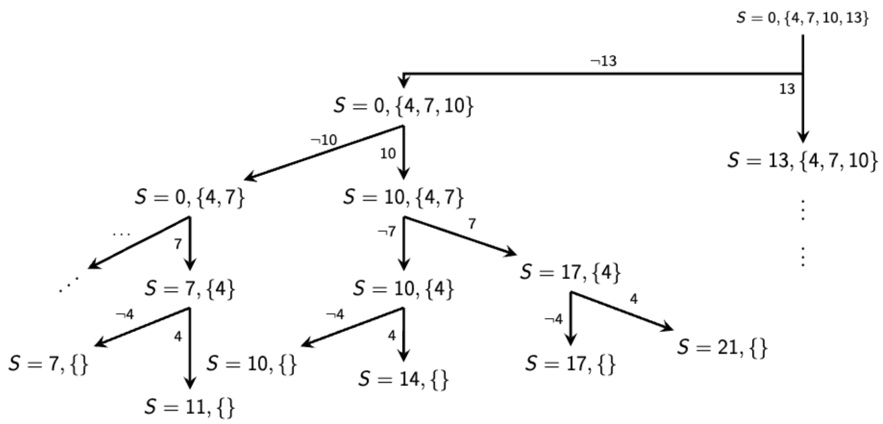
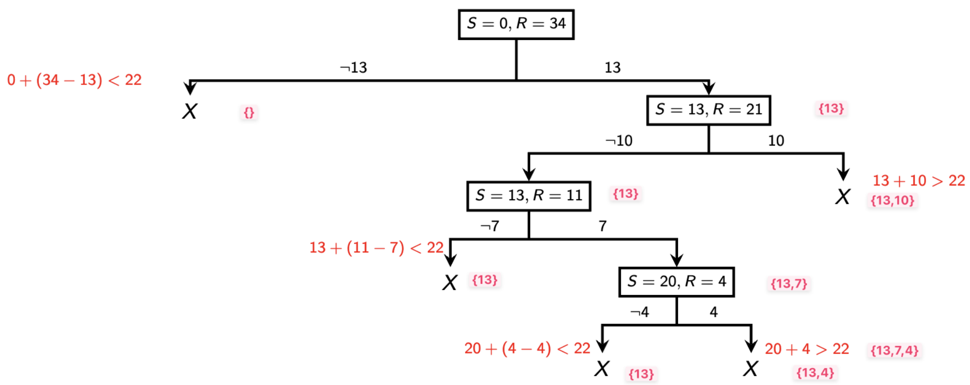


Figure: Search space tree for subset sum.

Bounded Tree

- Consider the instance $\{4, 7, 10, 13\}$ with target $t = 22$



Efficiency

- Without bounding: $O(2^n)$ nodes to explore
- With bounding: Significantly fewer nodes, as impossible branches are pruned early

Branch and Bound: Knapsack

Recall Greedy Strategy

- For fractional knapsack, greedy strategy sorts items by value/weight ratio
- Take items with highest ratio first
- If can't fit whole item, take fraction to fill remaining capacity
- This gives optimal solution for fractional case
- Does NOT work for 0/1 knapsack!

Greedy Maths

$$S = v_1 + v_2 + \cdots + v_k + \frac{C - w_1 - \cdots - w_k}{w_{k+1}} v_{k+1}$$

$$\text{Since } \frac{v_1}{w_1} > \frac{v_2}{w_2} > \cdots > \frac{v_k}{w_k} > \frac{v_{k+1}}{w_{k+1}}$$

$$S \leq \frac{w_1}{w_1} v_1 + \frac{w_2}{w_1} v_1 + \cdots + \frac{w_k}{w_1} v_1 + \frac{C - w_1 - \cdots - w_k}{w_1} v_1$$

$$S \leq \frac{C \cdot v_1}{w_1}$$

Explanation

- S represents the total value from taking items 1 to k fully and a fraction of item k+1
- Since items are sorted by value/weight ratio, we can replace each term with the ratio of the first item
- This gives us an upper bound on the solution: $\frac{C \cdot v_1}{w_1}$
  - This makes sense, as **best value density** $\times$ **capacity** will always give the best result
- This bound is useful for the branch and bound algorithm as it helps prune branches

Branching Process

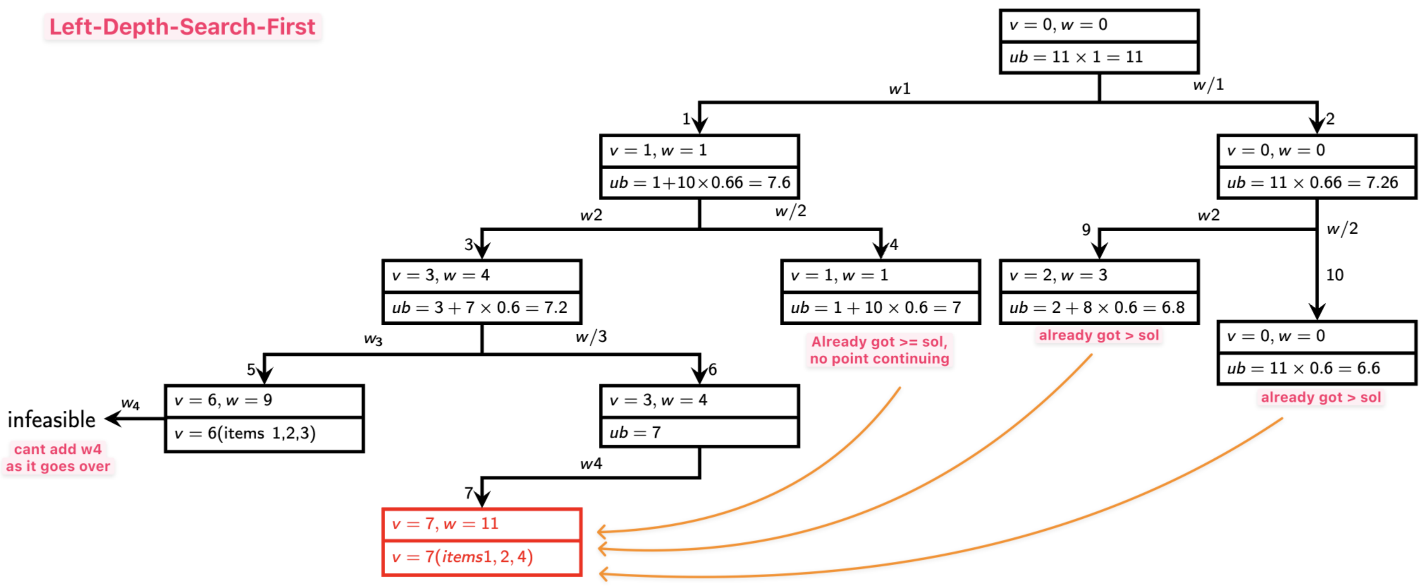
- At each node in the search tree:
  - Left branch: Include current item (if it fits)
  - Right branch: Exclude current item
- Each level represents decision for one item

Bounding Process

- For each node, calculate:
  - Current value: sum of values of selected items
  - Current weight: sum of weights of selected items
  - Upper bound: current value + (remaining capacity $\times$ best value density)
- Prune branch if:
  - Current weight > capacity (infeasible)
  - Upper bound < best solution found so far

Example

item	Value	Weight	Density
1	1	1	1
2	2	3	0.66
3	3	5	0.6
4	4	7	0.57



Interger Linear Programming

How it works

- Develop a general method to solve ILP by using LP relaxation **and branch and bound**
- For simplicity we will also consider another 0-1 knapsack instance
- Iterations of the BB algorithm necessitates repeated change in the constraints of the variables
  - This requires LP solvers else thats a lot of simplex!

Branch and Bound Technique

- Let z^* be the minimum value of ILP and $\overline{z^*}$ the LP relaxation. We have $\overline{z^*} \leq z^*$
- Solve the relaxed solution and consider the objective solution z
  - If $z \geq \overline{z^*}$ or the problem is infeasible, we can't get a better solution so backtrack
  - If $z < z^*$, we have two cases
    - Otherwise, for each variable x_i having value $f \notin \mathbb{N}$ we explore the two branches by including additional constraints: $x_i \leq \lfloor f \rfloor$ in one branch and $x_i \geq \lceil f \rceil$ in the other branch