Branch and Bound

IE 496 Lecture 18

Reading for This Lecture

- Primary
  - Horowitz and Sahni, Chapter 8

Integer Knapsack Problem

- We are given n objects.
- Each object has a weight w_i and a profit p_i.
- We also have a knapsack with capacity M.
- Objective: Fill the knapsack as profitably as possible.
- We do not allow fractional objects.
- This is an *NP*-hard problem.

Branch and Bound Methods

- *Branch and Bound* is a general method that can be used to solve many NP-complete problems.
- Components of Branch and Bound Algorithms
  - Definition of the state space.
  - Branching operation.
  - Feasibility checking operation.
  - Bounding operation.
  - Search order.

Definition of the State Space

- To apply branch and bound, the solution is typically expressible as an *n-tuple* $(x_1, x_2, ..., x_n)$ where x_i is chosen from a finite set S_i.
- A set of all possible *n-tuples* is the state space S.
- Knapsack Problem
- Bin Packing Problem

Decisions, Feasibility, Optimization

- Feasibility problems
  - A defined subset of the state space contains the "feasible" elements.
  - There are various ways to define "feasibility".
  - The goal is to find one feasible element of the state space.
- Optimization problems
  - We are also given an *objective function* f which assigns a cost to each element of the state space.
  - We would like to find a feasible state with the lowest cost.
- Decision problems

Branching Operation

- Operation by which the original state space is partitioned into at least two non-empty *subproblems*.
- Typical branching operation
  - Pick an element i of the n-tuple.
  - Generate $|S_i|$ subproblems by setting x_i to each of its possible values in succession.
- Knapsack
- Bin Packing

Feasibility Checking Operation

- Given a subproblem, we need to check whether it contains any feasible solutions.
- This may or may not be possible for partially defined states.
- It must be possible if the state is fully defined.
- Knapsack Problem
- Bin Packing Problem

Bounding Operation

- If applicable, we want upper and lower bounds on the optimal value of the current subproblem.
- This may not be possible.
- *Upper bounds* generally come from finding a feasible solution.
- Upper bounds are global
- *Lower bounds* can come from a number of sources.
- Knapsack
- Bin Packing

Basic Branch and Bound Algorithm

BBound (S, U)

S = {s: s is a feasible state}, U = current upper bound

```
if (FEASIBLE(S) == FALSE) return(∞);
if (LBOUND(S) >= U)  return(∞);
if (UBOUND(S) < U) U = UBOUND(S);
if (LBOUND(S) < U)
      BRANCH(S) -> S_1, . . . , S_k;
        for (i = 0; i < k; i++)
           if (BB(S_i,U) < U) U = BB(S_i);
return(U);
```

More Generally

- Associate branch and bound with a search tree.
- Maintain a priority queue of candidate subproblems.
- Iterate
  - Pick a subproblem from the queue and process it.
    - Check feasibility.
    - Perform upper and lower bound.
  - Prune if infeasible or lower bound greater than or equal to upper bound.
  - Branch.
  - Add new subproblems to the queue.

Search Strategies

- Depth First
- Breadth First
- Highest Lower Bound
- Lowest Lower Bound

The Traveling Salesman Problem