Laboratório de Programação Avançada 2018/19

Week 7 – Branch-and-Bound

· U C ·

UNIVERSIDADE DE COIMBRA

Outline

1. Introduction to branch-and-bound
2. Examples

Reading about problem solving with branch-and-bound algorithms

- S. Dasgupta et al., Algorithms, Chapter 9.1.2
- J. Clausen. Branch and Bound Algorithms - Principles and Examples

Backtracking revisited

- Backtracking starts with an empty solution and attempts to complete it step by step, in a recursive fashion.

- Recursion stops once a partial solution is found to be invalid, i.e., it cannot lead to a valid complete solution.

- Each partial solution represents a set of complete solutions: those that can be generated from it.

Backtracking revisited

- At each recursion step, the current set of possible complete solutions is partitioned into several subsets: *branching*.

- Representing the current set of possible complete solutions by a node, each subset in the partition may be represented by a child of the original node.

- Backtracking relies on a rejection function to avoid searching the entire tree: *pruning*

Branch-and-bound

A branch-and-bound strategy includes two main ingredients:

- *Branching*: a way of partitioning a set of solutions into two or more disjoint subsets
- *Bounding*: a way of calculating a bound on the objective function of any solution in a given subset

By branching, we reduce the problem into a smaller subproblem with less variables.

By bounding, we might be able to avoid branching further.

Optimization problem

Assume a maximization problem whose goal is to maximize an *objective function* f over a set S of *feasible solutions*:

$$\max_{x \in S} f(x)$$

Set S is defined by the contraints of the optimization problem.

Optimization problem

Assume a maximization problem whose goal is to maximize an *objective function* f over a set S of *feasible solutions*:

$$\max_{x \in S} f(x)$$

Set S is defined by the contraints of the optimization problem.

Bounding function (maximization version)

A *bounding function* g is such that

$$\max_{x \in S} f(x) \leq \max_{x' \in P} g(x')$$

where $P \supseteq S$ can be obtained by relaxing some constraints.

Knapsack problem: Given n objects, each of which with a value v_i and weight w_i, $i = 1, \ldots, n$, and a constraint W, find a subset that maximizes the total value such that the total weight does not exceed W.

- S is set of all solutions that do not exceed a total weight W

$$w(x) = \sum_{i \in x} w_i \leq W \quad \text{for all } x \in S$$

- The objective function $f(x)$ of a feasible solution $x \in S$ is

$$f(x) = v(x) = \sum_{i \in x} v_i$$

- The goal is to find a feasible solution $x^* \in S$ such that

$$\max_{x^* \in S} f(x^*)$$

First example of bounding function

Let x' be a partial solution for the subproblem with objects $\{1, \ldots, k\}$, $k \leq n$, with value $v(x')$. Let $g(x')$ be defined as follows:

$$g(x') = v(x') + \sum_{j=k+1}^{n} v_j$$

Then, $g(x') \geq f(x)$, for all $x \in S$ such that $x \supseteq x'$.

First example of bounding function

Let x' be a partial solution for the subproblem with objects $\{1, \ldots, k\}$, $k \leq n$, with value $v(x')$. Let $g(x')$ be defined as follows:

$$g(x') = v(x') + \sum_{j=k+1}^{n} v_j$$

Then, $g(x') \geq f(x)$, for all $x \in S$ such that $x \supseteq x'$.

Second example of bounding function

Let z be the optimal value of the fractional knapsack problem for the objects $\{k+1, \ldots, n\}$ and constraint $W - w(x')$. Let $g(x')$ be defined as follows:

$$g(x') = v(x') + z$$

Then, $g(x') \geq f(x)$, for all $x \in S$ such that $x \supseteq x'$.

Optimization problem (minimization version)

Assume a minimization problem whose goal is to minimize an *objective function* f over a set S of *feasible solutions*:

$$\min_{x \in S} f(x)$$

Bounding function

A *bounding function* g is such that

$$\min_{x' \in P} g(x') \leq \min_{x \in S} f(x)$$

Branch-and-bound template (maximization version)

Function $BB(x)$

if $g(x) \leq f(x^*)$ **then** {bounding test}
  return
if $accept(x) = \texttt{true}$ **then** {base case}
  if $f(x^*) < f(x)$ **then** {update best}
    $x^* = x$
  return
while $condition(x) = \texttt{true}$ **do**
  $x' = update(x)$ {new candidate solution}
  $BB(x')$ {recursive step}
return

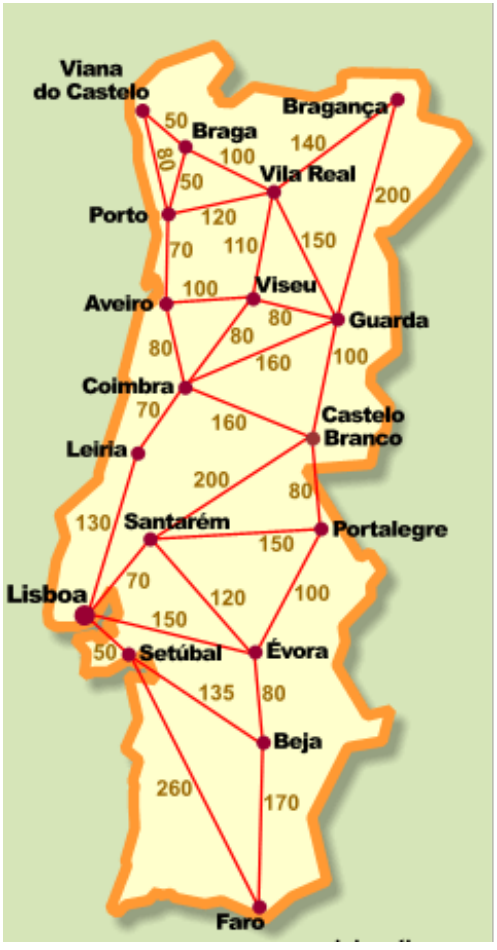
- The bounding function $g(x)$ gives an upper bound on the best value that can be obtained from (partial) solution x.
- In minimization, $g(x)$ gives a lower bound.

Knapsack problem

Function $knapsack(x, i)$
  if $w(x) > W$ **then** {rejection test}
    return
  if $g(x) \leq f(x^*)$ **then** {bounding test}
    return
  if $i = n$ **then** {base case}
    if $f(x^*) < f(x)$ **then**
      $x^* = x$
    return
  $knapsack(x, i + 1)$ {don't take object i}
  $knapsack(x \cup \{i\}, i + 1)$ {take object i}
  return

- The bounding function $g(x)$ gives an upper bound on the best value that can be obtained from (partial) solution x.

Quick and Cheap!



João is a truck driver. He uses his truck to transport goods between several cities in Portugal. Since he has several clients spread over the country, he has to go through toll highways in order to be fast enough. However, he has to spend a large amount of money at the end of the month. Therefore, he decided to fix the maximum amount of money that he has to spend in toll roads and be as fast as possible. However, finding such path is no longer an easy task. Could you help him?

Constrained shortest path

Let G be a network, $G = (V, A)$, where each arc $(i, j) \in A$ has a length $\ell(i, j)$ and cost $c(i, j)$, and W is the maximum amount of money.

The goal is to find a path x from node s to node t that minimizes the total length

$$\ell(x) = \sum_{(i,j) \in x} \ell(i, j)$$

such that

$$c(x) = \sum_{(i,j) \in x} c(i, j) \leq W$$

Note: If we ignore the constraint, we have the shortest path problem from node s to node t. That path is shorter than or equal to the optimal path of the constrained shortest path.

Constrained shortest path

Function $ShortPath(x, i)$

```
if ℓ(x) ≥ ℓ(x*) or c(x) > W then                    {rejection test}
    return
if i = t then                                        {base case}
    if ℓ(x) < ℓ(x*) then
        x* = x
    return i
for each (i, j) ∈ A and visit[j] = false do
    visit[j] = true                                  {mark as visited}
    ShortPath(x ∪ {(i, j)}, j)                       {recursive step}
    visit[j] = false                                 {mark as unvisited}
```

- Backtracking for the constraint shortest path problem, where i denotes the last inserted node in the path x.

Constrained shortest path

Function *ShortPath*(x, i)
  if $\ell(x) \geq \ell(x^*)$ or $c(x) > W$ **then** {rejection test}
    return
  if $g(x) \geq \ell(x^*)$ **then** {bounding test}
    return
  if $i = t$ **then** {base case}
    if $\ell(x) < \ell(x^*)$ **then**
      $x^* = x$
    return
  for each $(i, j) \in A$ **and** *visit*$[j] =$ `false` **do**
    visit$[j] =$ `true` {mark as visited}
    ShortPath$(x \cup \{(i, j)\}, j)$ {recursive step}
    visit$[j] =$ `false` {mark as unvisited}

- Branch-and-bound where $g(x)$ is $\ell(x)$ plus the length of the (unconstrained) shortest path from node i to node t.
- Then, $g(x)$ gives a lower bound on the length of the (constrained) shortest path that can be constructed from x.