Branch and Bound

(Exact Methods II)

Joshua Knowles

School of Computer Science

The University of Manchester

COMP60342 - Week 2 2.15, March 20th 2015

Branch-and-Bound is Intelligent Enumeration

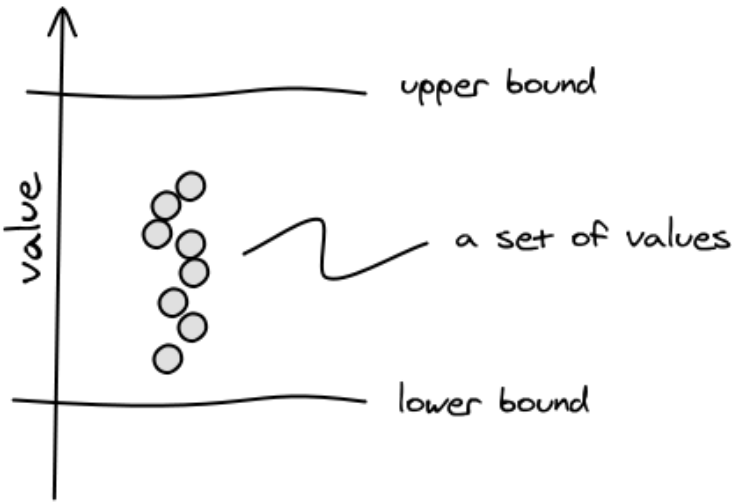
Previous lecture: ***enumeration*** (exhaustive search)
$\longrightarrow$ a very general but *unintelligent* method.

Branch-and-bound is turbo-charged enumeration.

It still lists and "ticks off" all solutions. But it employs "***lookahead***" to be more intelligent.

A mouse takes a more global view of the problem!

What is a bound?



Informally,

An **upper bound** is a value larger than or equal to the largest value in a set.

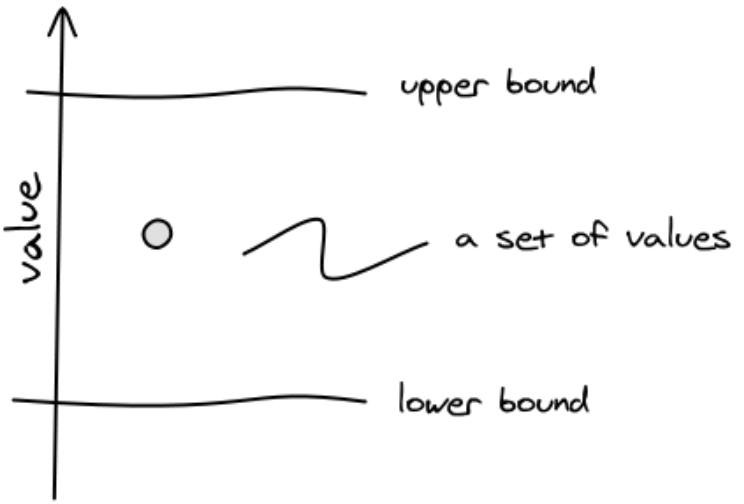
A **lower bound** is a value smaller than or equal to the smallest value in a set.

Bounds can be used to express some certainty about uncertain events...

...e.g. you roll a standard die 5 times without showing me.

I am certain that you won't have rolled any number greater than 6 or less than 1 (even though I don't know what you have rolled).

What is a bound?



The set of interest may be of cardinality 1 (i.e., may have only **one** member).

E.g. you roll a die 5 times, and your score is the **sum** of your rolls. An upper bound on the score (a set of cardinality 1) is 30. A lower bound is 5.

What is a bound?

So bounds are estimates (not guesses). To qualify as a bound, it must be certain that the number is greater than or equal (upper bound) or lower than or equal (lower bound) than the largest, or respectively smallest value in the set.

For you to do: What is an upper bound and a lower bound on the annual salary of the highest-paid professional footballer?

Give reasons.

What is a bound?

Definition:
An upper bound on a subset S of a partially ordered set $(P, \leq)$ is an element of P which is greater than or equal to every element of S.

A lower bound is defined analogously.

Example: $S = \{3, 5, 7\}$, P is the set of natural numbers.

Then an upper bound of S is 22.

Another upper bound of S is 8. We say that 8 is a ***tighter*** upper bound than 22.

The least upper bound (also known as the supremum) of S is 7.

What is a bound?

Give the least upper bound of this set $S = \{-1.0, 45.2, 17.5\}$, when P is the set of real numbers.

What is the tightest *lower* bound of the set?

Why are bounds useful in optimization?

A. If we have a bound on the value of the best possible solution, and we achieve that bound then we can stop searching.

Or, if we can tolerate an *approximate solution*, then we can stop searching when we are merely *close* to the bound.

Q. Why are tight bounds more useful than loose bounds?

An example

PARTITION: Given a set X of positive integers, divide the set into two subsets such that the difference between the sums of their elements is minimized.

Instance: $X = \{11, 7, 6, 4, 9, 14\}$

How could a lower bound be calculated?

Is the following solution optimal?

$\{11, 14\},\ \{7, 6, 4, 9\}$

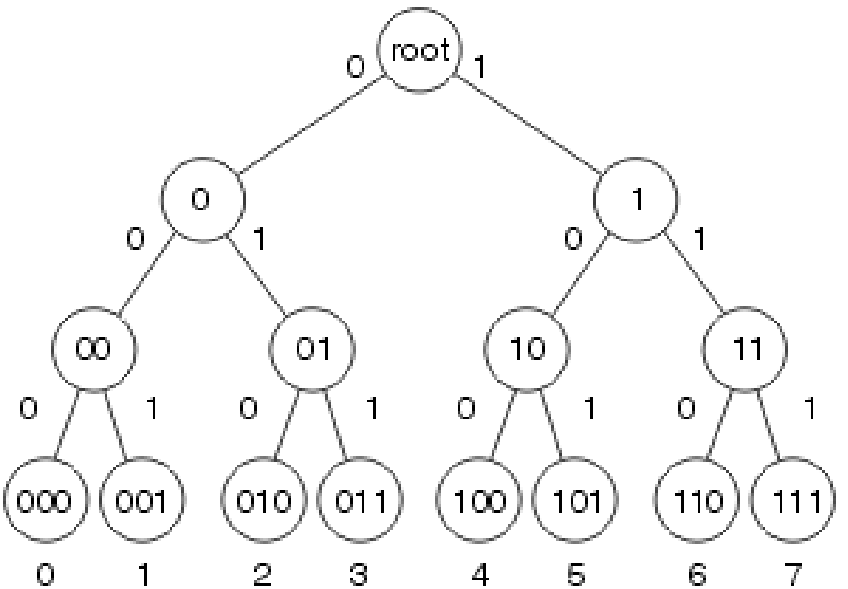
The Branching Part

Branch-and-Bound uses a ***partition*** of the solution space into subsets

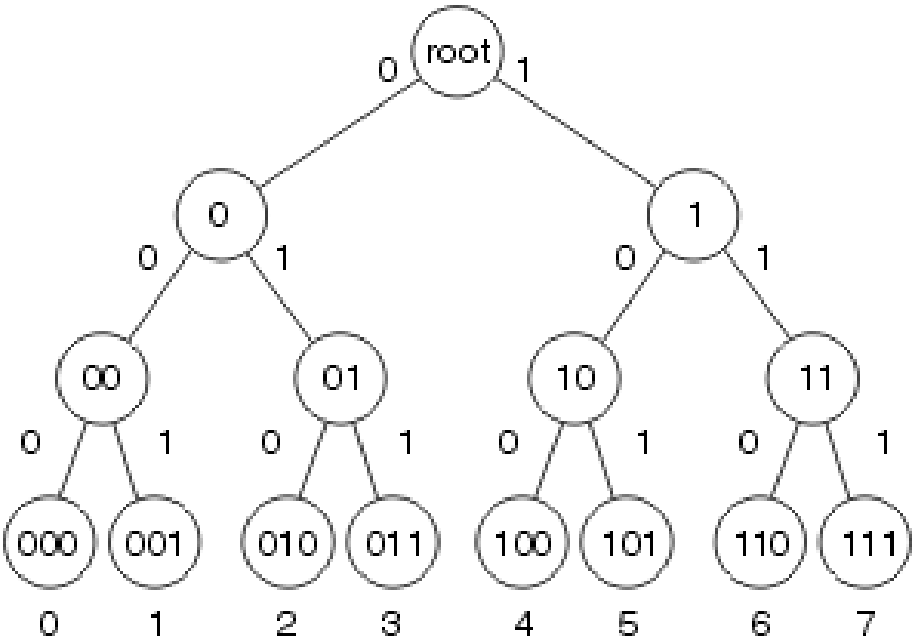
Usually the subsets are arranged in a ***tree structure***

Leaves in the tree are solutions. Internal nodes are ***partial solutions***

The partial solutions allow reasoning about large subspaces of the search space.



Since a partial solution, and solutions at the leaves of its subtree, are related, so are their objective function values.
This can be used to great advantage by branch-and-bound.



How Branch-and-Bound Works (Broad View)

Branch-and-bound organizes the search so that nodes in the search tree are expanded one-by-one.

Branch&Bound: 'Shall I expand you?'

Internal Node: 'Yes, you can make a good solution from me; my bound is good'

It expands a node **if and only if** that node has the ***potential*** to yield a better solution. This ***potential*** can be determined by calculating a **bound** on the value of solutions in the subtree rooted at the node.

Branch-and-Bound is Intelligent Enumeration

The intelligence in Branch-and-Bound comes from

(1) recognising when to stop, i.e. when an optimum has been found

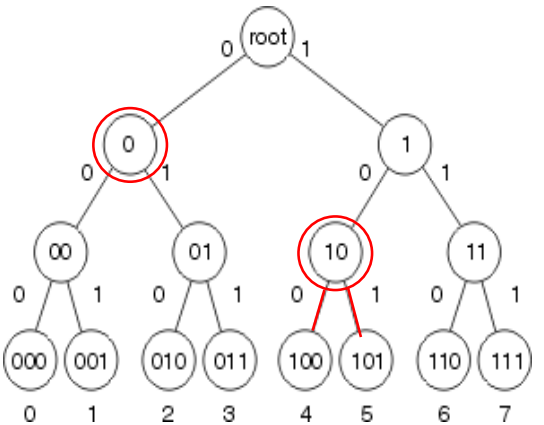
(2) recognising that large parts of the space **cannot contain an optimum**

(3) recognising that large parts of the space are not even feasible

(4) searching in promising regions of the search space before searching less promising regions.

NB: For a particular problem, one or more of these features may not apply and/or one may be much more important than the other.

How Branch-and-Bound Works (Broad View)



Some nodes in the tree are ***active***.
These are the nodes waiting to be explored.

Branching refers to exploring the subtree of an active node.

Bounding refers to estimating a bound on the solutions in the subtree rooted at the active node.

The Branch-and-Bound Algorithm

begin
    activeset :=$\{\emptyset\}$;
    bestval:=NULL;
    currentbest:=NULL;
    while activeset is not empty **do**
        choose a branching node, node k $\in$ activeset;
        remove node k from activeset;
        generate the children of node k, child i, i=1,. . . ,$\mathrm{n_k}$,
         and corresponding optimistic bounds $\mathrm{ob_i}$;
        for i=1 to $\mathrm{n_k}$ **do**
            if $\mathrm{ob_i}$ worse than bestval **then** kill child i;
            else if child is a complete solution **then**
                bestval:=$\mathrm{ob_i}$, currentbest:=child i;
            else add child i to activeset
        end for
    end while
end

The above is a very simple basic branch and bound pseudocode.

Here, currentbest stores the best complete solution found so far, also known as the *incumbent*.

The value of currentbest is the bestval. This value is used to see if it is worth expanding (creating children of) nodes.

The ***bound*** b is the optimistic estimate of how good a partial solution (or node) may be once completed; if it is not better than currentbest, there is no need to evaluate the children of that node (so we don't add it to activeset).

This is a general scheme; there are many variants.

Upper and Lower Bounds in Maximization

In a maximization problem, a ***lower bound*** on the optimal solution is provided by any feasible solution, since if y is the value (or profit) of a feasible solution, the optimal solution, $y^* \geq y$.

The (current) ***tightest*** lower bound is provided by the value of the ***best*** solution we have found. This is also referred to as the **incumbent**.

An ***upper bound*** ob on the value (or profit) obtainable within a *given set of solutions* $X' \subseteq X$ can sometimes be calculated. If this upper bound is lower (less good) than the lower bound we already have, then it is not worth enumerating these solutions. This upper bound is the **optimistic bound**.

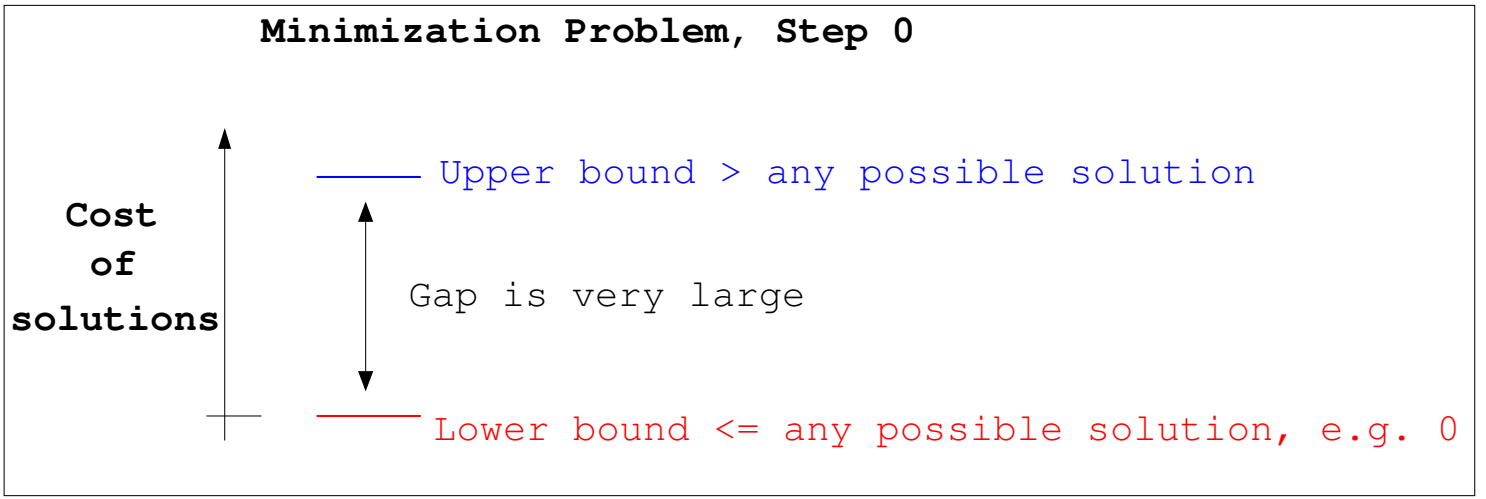
Upper and Lower Bounds in Minimization

In a minimization problem, an ***upper bound*** on the optimal solution is provided by any feasible solution, since if y is the cost of a feasible solution, then the optimal solution $y^* \geq y$.

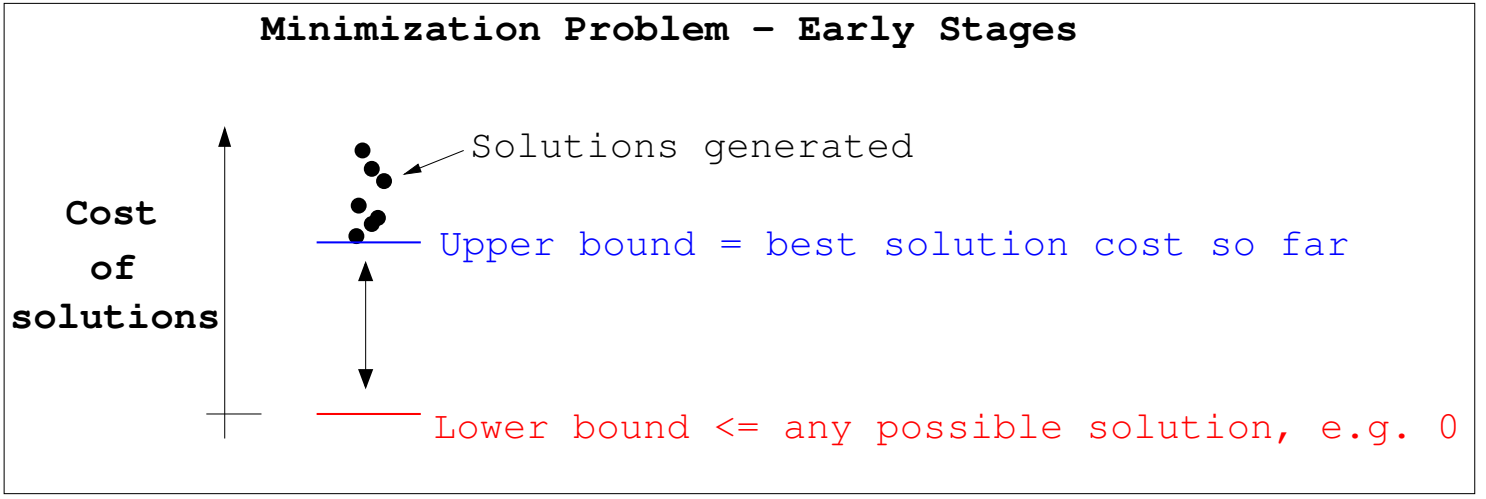
The (current) ***tightest*** upper bound is provided by the cost of the ***best*** solution we have found. This is also referred to as the **incumbent**.

A ***lower bound*** ob on the cost obtainable within a *given set of solutions* $X' \subseteq X$ can sometimes be calculated. If this lower bound is higher (less good) than the upper bound we already have, then it is not worth enumerating these solutions. This lower bound is the **optimistic bound**.

Seeing How the Bounds Operate

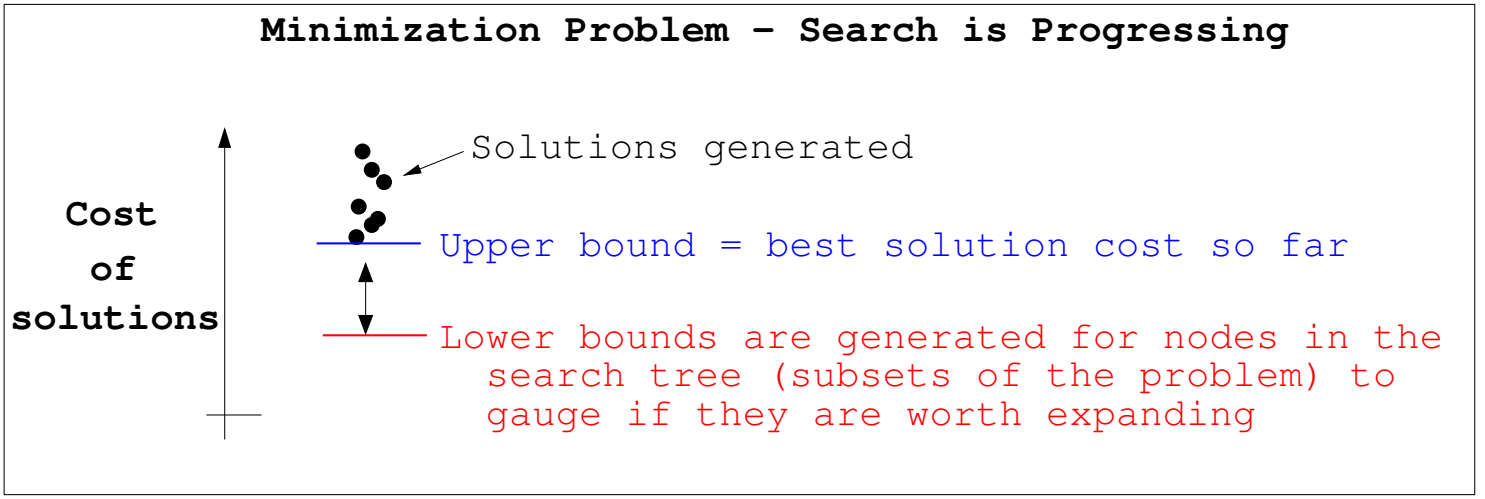


Seeing How the Bounds Operate



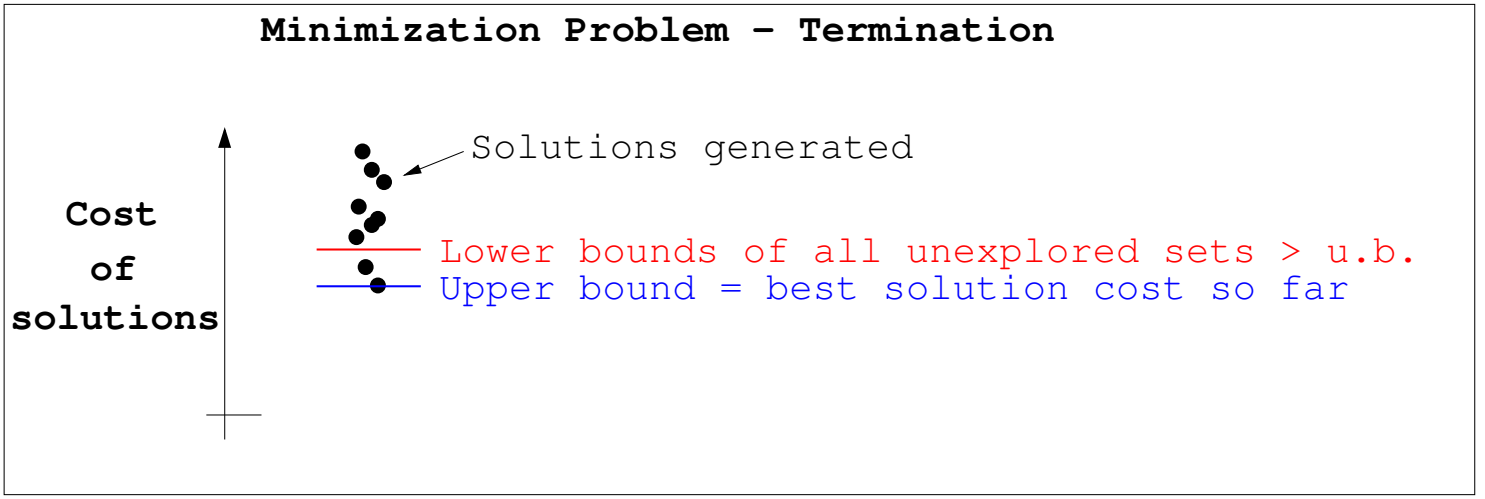
Any complete solutions we have found serve as upper bounds on the best cost.

Seeing How the Bounds Operate



Lower bounds are computed using relaxations or heuristics. They must generate an underestimate or exact estimate of the best cost that some candidate set can possibly achieve.

Seeing How the Bounds Operate



When the lower bounds of all unexplored subsets of the problem are greater than the upper bound (the best solution found so far) the algorithm can terminate. The best solution found is optimal.

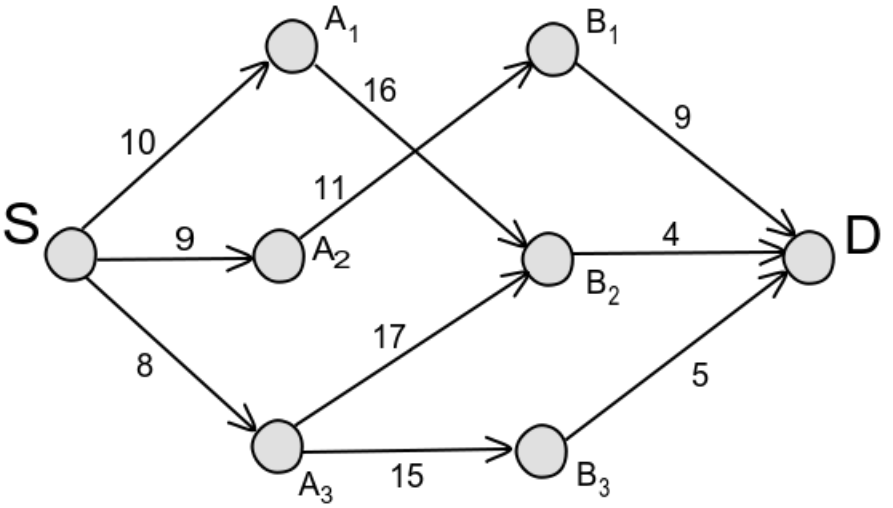
Where do Optimistic Bounds Come From?

What do you think?

How can we guarantee the best cost of some set of (unseen) solutions is not lower than some value, without enumerating all these solutions and checking?

What is a Relaxation?

Consider the following **constrained shortest path problem**.

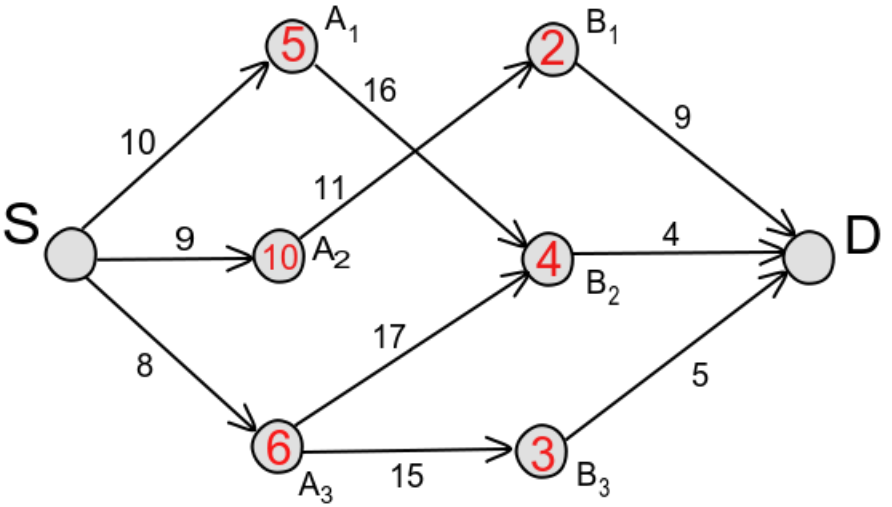


You must find the shortest route from S to D that also has at least 10 as the sum of node values.

How can we efficiently obtain a lower bound on this?

What is a Relaxation?

Consider the following **constrained shortest path problem**.



You must find the shortest route from S to D that also has at least 10 as the sum of node values.

How can we efficiently obtain a lower bound on this?

Dijkstra's algorithm will give us a lower bound.

Why?

The unconstrained version of a constrained problem is known as a ***relaxation*** (since we relax the constraints).

The solution to a relaxation is always an *optimistic bound* on the solution to the corresponding constrained problem.

Often, it is a good (i.e., tight) bound.

The next trick is to compute bounds for *subsets* of the solution space, to get even tighter bounds. . .

Computing Optimistic Bounds from a Partial Solution

Assume we are solving a **minimization** problem.

- Assume X' is defined as the set of (valid) solutions that complete a partial solution $x_p = \{x_p^1, x_p^2, \ldots, x_p^{m<n}\}$.
- Assume cost is calculated as some simple sum over the defined elements in x_p
- Then a simple lower bound for X' is just this cost. (Why ?)
- A tighter bound is this cost plus some extra costs from adding minimal elements in each position without worrying about constraints.
- The latter can be calculated using greedy methods.
- Tighter bounds take more computation time to obtain, but may ***prune*** the search tree much more effectively. There is a **trade-off**.

Varieties of Branch-and-Bound

The search tree can be explored using

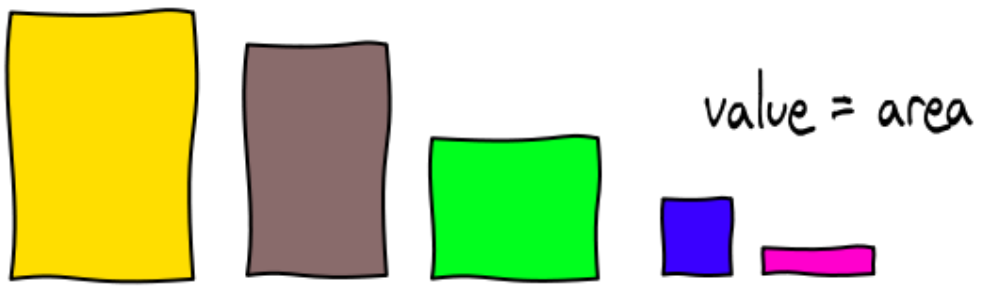
- **Depth-first search**; this will construct feasible solutions sooner
- **Breadth-first-search**; this may prune more of the search tree sooner
- **Best-first-search**; sets with better bounds are explored first
- Other **heuristics**.

NB: the most suitable/efficient one may depend on problem, bounding function, or problem instance.

0/1 Knapsack — Today's Lab

To compute bounds, we ***relax*** the constraint about taking 0 or 1 of an item. We allow taking a fraction.

Here's how we do that.

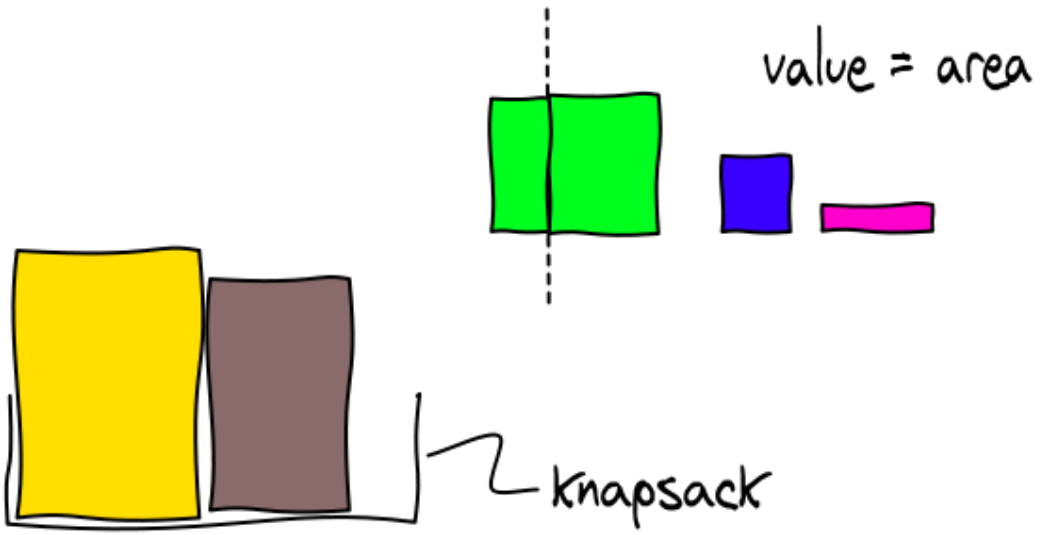


Now let's see how that is used in solving knapsack with branch and bound.

0/1 Knapsack — Today's Lab

To compute bounds, we ***relax*** the constraint about taking 0 or 1 of an item. We allow taking a fraction.

Here's how we do that.

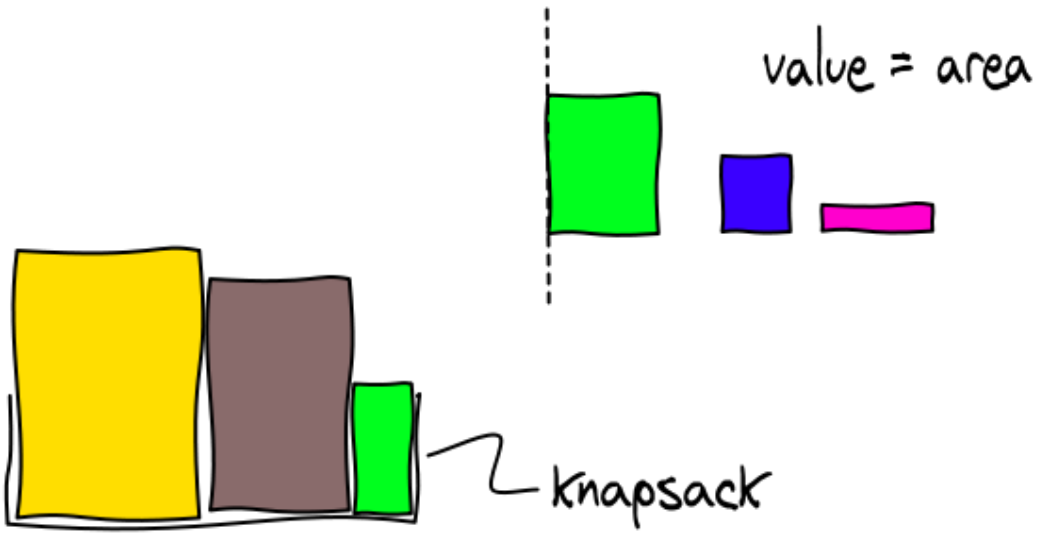


Now let's see how that is used in solving knapsack with branch and bound.

0/1 Knapsack — Today's Lab

To compute bounds, we ***relax*** the constraint about taking 0 or 1 of an item. We allow taking a fraction.

Here's how we do that.



Now let's see how that is used in solving knapsack with branch and bound.

0/1 Knapsack — Today's Lab

0/1 Knapsack — Today's Lab

i	3	1	2	
v	12	10	7	C=10
w	6	5	4	

items sorted in descending v/w

0/1 Knapsack — Today's Lab

active set
step 0: { }

ob = 20
v = 0

* * *

i	3	1	2	
v	12	10	7	C=10
w	6	5	4	

items sorted in descending v/w

0/1 Knapsack — Today's Lab

active set

step 0: { }

1: {***}

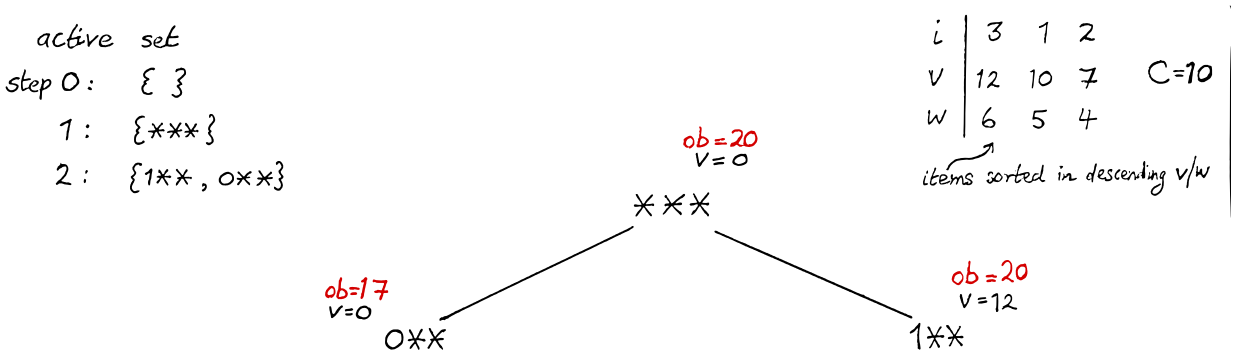
ob = 20
v = 0

***

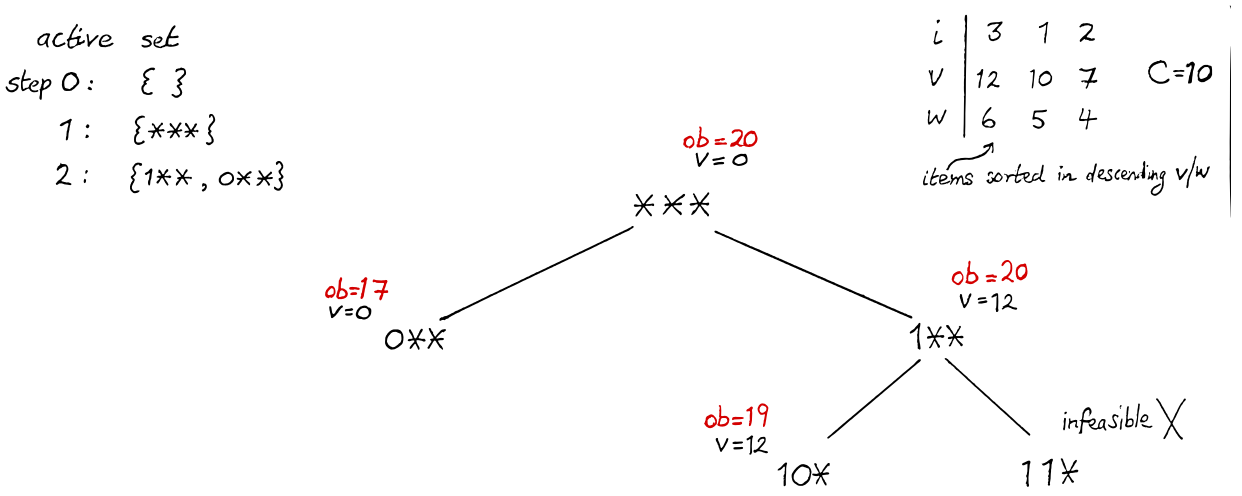
i	3	1	2	
v	12	10	7	C=10
w	6	5	4	

items sorted in descending v/w

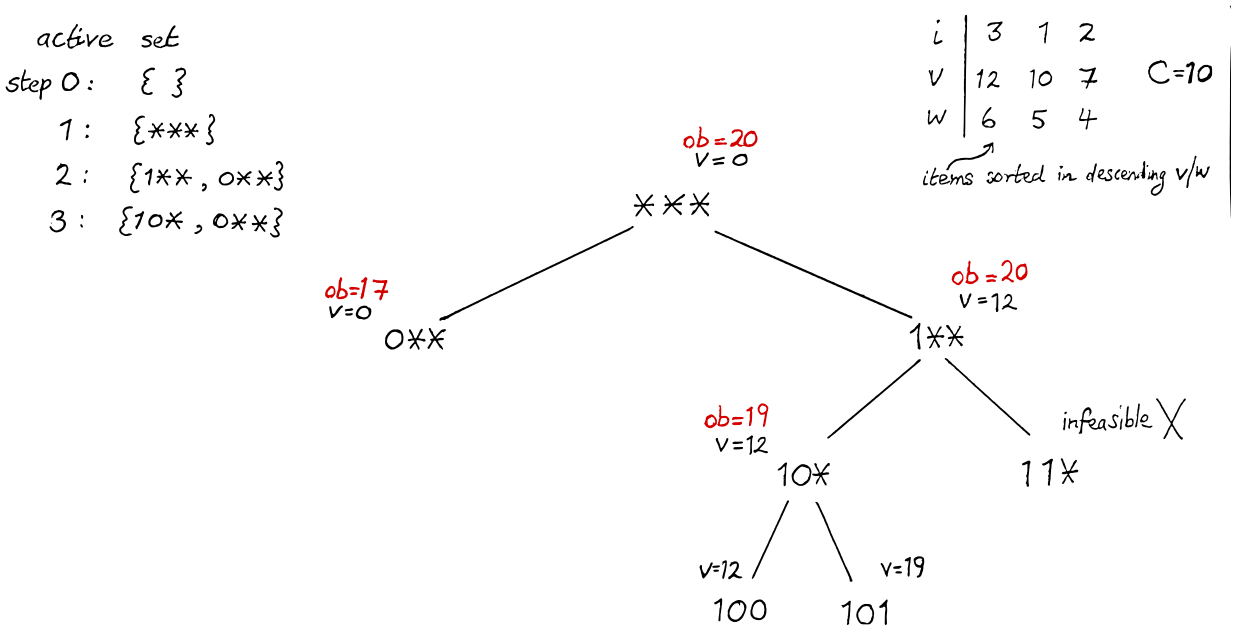
0/1 Knapsack — Today's Lab



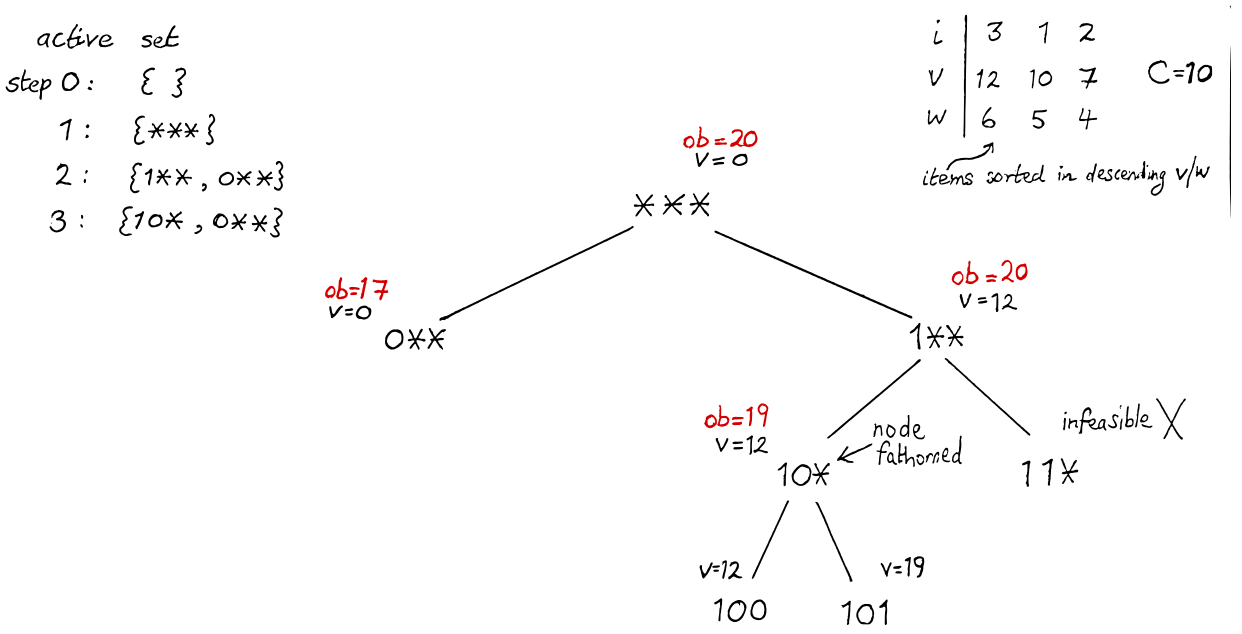
0/1 Knapsack — Today's Lab



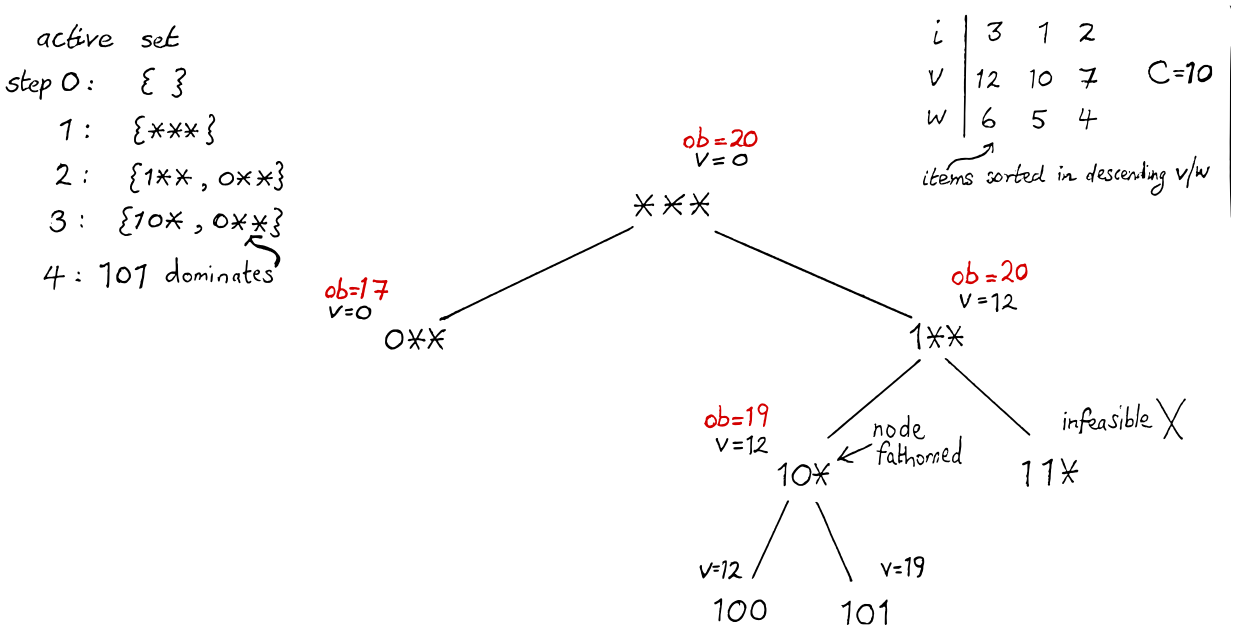
0/1 Knapsack — Today's Lab



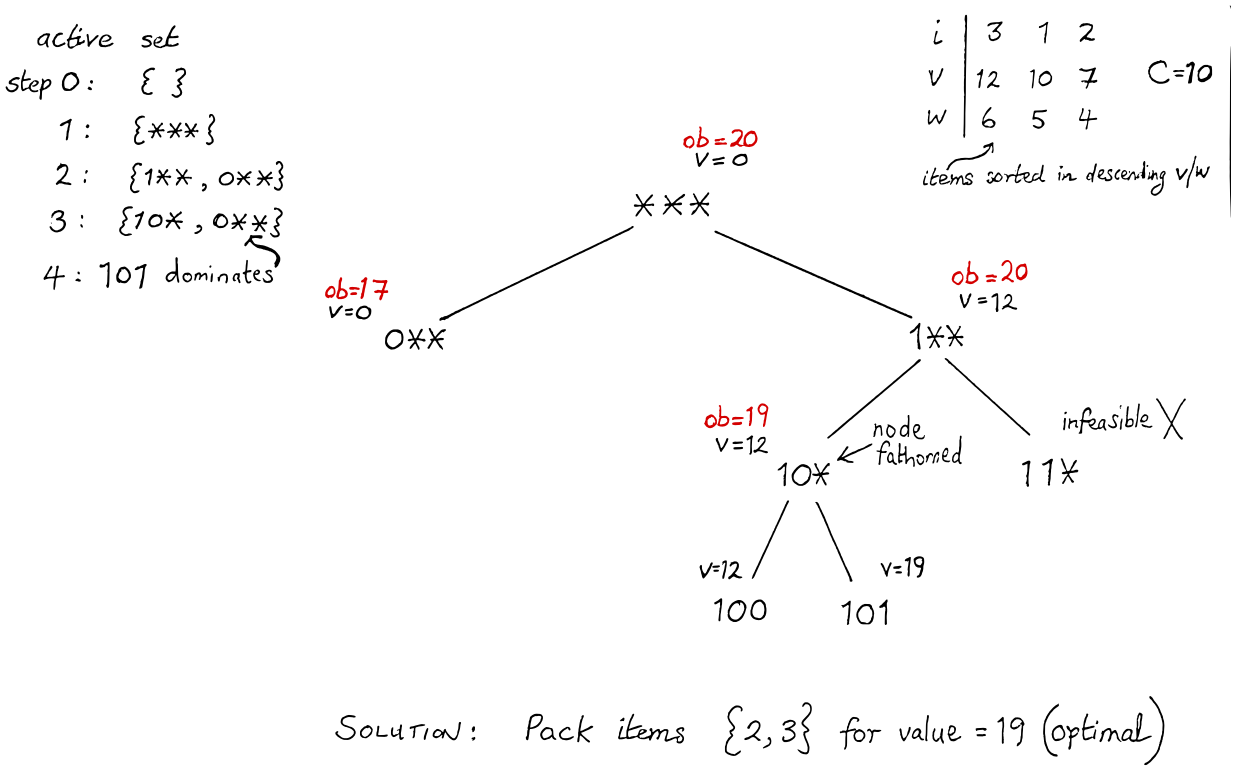
0/1 Knapsack — Today's Lab



0/1 Knapsack — Today's Lab



0/1 Knapsack — Today's Lab



SOLUTION: Pack items $\{2, 3\}$ for value = 19 (optimal)

Other Types of Pruning: Fathoming a Node

Sometimes, in computing an optimistic bound, we notice that we have constructed a feasible complete solution.

Since it is also an optimistic bound (by definition), this complete solution must be the **best possible feasible expansion of the current node**.

This means we have ***'fathomed'*** the node. Really, there is no need to continue expanding the node's children one at a time. We already have the best possible feasible solution from this node's children, and can move on.

We have successfully pruned off more of the search tree.

Other Types of Pruning 2: Dominance Relations

> If we can show at any point that the best descendant of a node y is at least as good as the best descendant of node x, then we say y *dominates* x and y can kill x

Papadimitriou and Steiglitz (Section 18.3).

An example using a Shortest Path Problem is given.

Other Heuristics in Branch-and-Bound

An idea: Use additional computations to estimate which branches of the active set to explore first. (These need not be computations of bounds).

May allow earlier computation of a good **pessimistic bound** and so more of the search tree can be pruned before it is explored.

Example: In a knapsack problem, we may use a heuristic like preferring to pack larger items first (leaving smaller items until later).

Heuristic rules may improve performance and they may not, depending on characteristics of the problem instances. One has to try them out.

Branch and Bound: Sometimes a Heuristic

Branch and bound is an enumeration algorithm, and hence an ***exact technique*** (i.e., gives guaranteed exact optima).

However, the search tree can become very large and can take a long time to search, even when pruning is working correctly.

In this case, branch and bound could be *stopped early*, in which case it becomes only a ***heuristic***.

One way to terminate the search early, but to get at least a guaranteed level of approximation, is to stop it **iff** the incumbent solution has $value \leq (1 + \alpha)$.current_lower_bound (for a minimization problem), where α sets the desired level of approximation.

Summary

- Branch and bound enumerates intelligently
- Branch and bound is an exact method, meaning it terminates with a globally optimal solution
- It prunes infeasible and unpromising regions of the search space. Bounds are used to prove certain regions need not be explored
- The best solution found so far — the incumbent — acts as one type of bound
- The other type of bound, the optimistic bound, provides an estimate of whether to expand a node
- Optimistic bounds are usually found by ***relaxation*** of the problem constraints
- Tighter bounds take more computation but can prune more of the search tree.

Glossary

upper bound: value greater than or equal to largest value in a set

optimistic bound: value better than or equal to best possible solution

incumbent: value of current best solution (lower bound in a maximization problem)

relaxation: a problem with one or more constraints removed

partial solution: a candidate solution only partially specified

pruning: removing part of the search tree because it contains no optimal solution

fathoming: when a solution's value equals the bound of a node, the node needs no further exploration

dominating: when a solution found by expanding one node is better than the bound of another node, the second node is dominated

heuristic: a method for finding good but not necessarily optimal solutions