Branch-and-Bound

Leo Liberti

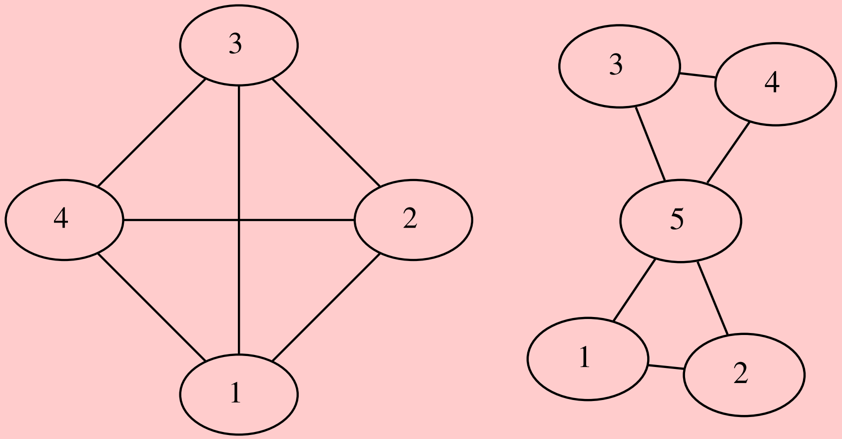
LIX, École Polytechnique, France

Reminders

Problems

- **Decision problem**: a question admitting a YES/NO answer
  Example

HAMILTONIAN CYCLE. Given an undirected graph $G = (V, E)$, does it have a simple spanning cycle?



YES instance *NO instance*

Spanning subgraph: a subgraph whose vertex set is V

Cycle: even degree subgraph

Simple cycle: connected with all vertices having degree 2

- **Optimization problem**: finding a mathematical structure of optimum cost
  Example

TRAVELLING SALESMAN PROBLEM (TSP). Given a set V and a positive square matrix (d_{ij}) (where $i, j \in V$) find a tour (=order on V) of minimum total cost

Solutions

- The mathematical structures sought in decision/optimization problems are also called **solutions** or **certificates**
- Solutions can be **feasible** if they satisfy the constraints given in the problem statement or **infeasible** otherwise
- Feasible solutions of optimization problems can be **optimal** if their cost is best amongst all solutions, or **locally optimal** if their cost is best in a neighbourhood

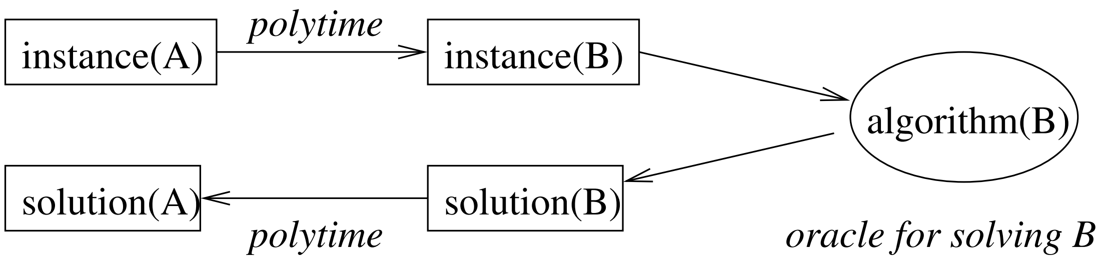
Two faces of the same coin

- **Reduction optimization$\to$decision**: if we know how to find an optimal solution, then we trivially know how to answer the question "does the problem have a feasible solution?"
- **Reduction decision$\to$optimization**: restate "find minimum cost solution" by "is there a solution with cost at most K?", then run a bisection search on a sequence of decision problems with different values of K

Easy & hard problems

- Characterization of easy and hard problems championed by Jack Edmonds
- A problem is considered *easy* if there exists an algorithm which solves it in worst-case polynomial time in function of the instance length
- A problem is considered *difficult* if it is **NP**-hard

Problem A is **NP**-hard if there is another **NP**-hard problem B whose solutions can be used to obtain solutions of A in polynomial time



- **N.B.**: "*difficult* $=\neg$*easy*" is an open question ($\Rightarrow$**P**$\neq$**NP**)

Solution space

- A difficult problem must have more than polynomially many solutions

  otherwise complete enumeration would be a polyomial-time algorithm

- Typically, it has exponentially or factorially many solutions

  Example

  The number of solutions of a TSP instance on the set $V = \{1, \ldots, n\}$ is $(n-1)!$: fix 1 as the first tour element, then there are $n-1$ choices for the second, $n-2$ for the third, and so on

- Under the hypothesis that *difficult* $=\neg$*easy*, we cannot do much better than complete enumeration in the worst case

- Look for **practically efficient** methods

The branch…

The TSP again

- Consider the following graph-based TSP formulation

Given a complete digraph $G = (V, A)$ with arc weight function $d : A \to \mathbb{R}_+$, find a tour of minimum cost

- In this formulation, a tour is a set of arcs which defines a strongly connected spanning subgraph H of G where each vertex has indegree=outdegree=1
- Spanning $\Rightarrow V(H) = V$
- indegree=outdegree=1 $\Rightarrow |A(H)| = |V| = n$
- Hence, we can consider every set of arcs to be a solution; feasible solutions correspond to tours

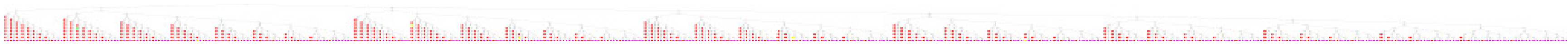
Growing a good arc set

- For a set S of arcs, let $d(S) = \sum\limits_{(i,j)\in S} d_{ij}$ with $d(\varnothing) = \infty$
- Let $\texttt{isTour}(S)$ be TRUE if S is a tour
- **Disjunction**: For $a \in A$, either $a \in S$ or $a \notin S$
- *Exhaustive exploration*: for a given arc a, either use it or not, then recurse the search
- Let $Y, N, S \subseteq A$: $Y =$candidate solution, $N =$forbidden arcs, $S =$best tour so far (*incumbent*)
  1. if $\texttt{isTour}(Y)$ and $d(Y) < d(S)$, replace S with Y
  2. if Y is not a tour, choose an arc a not in $Y \cup N$
  3. recurse with Y replaced by $Y \cup \{a\}$ and then with N replaced by $N \cup \{a\}$

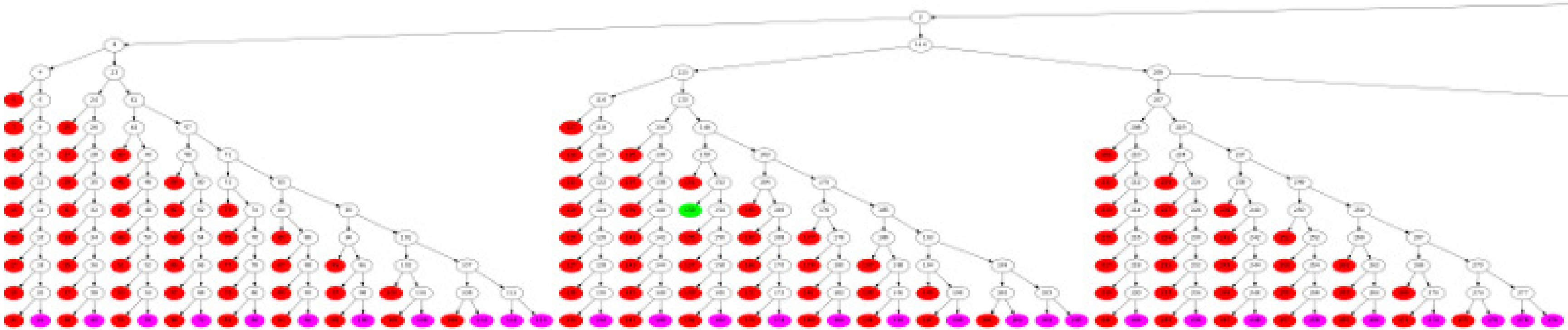
Tree-based enumeration algorithm

```
treeSearch(Y, N, S):
if isTour(Y) then
  if d(Y) < d(S) then
    S = Y
  end if
else
  if Y ∪ N ≠ A ∧ Y ∩ N = ∅ then
    choose a ∈ A ∖ (Y ∪ N)
    treeSearch(Y ∪ {a}, N, S)
    treeSearch(Y, N ∪ {a}, S)
  end if
end if
```

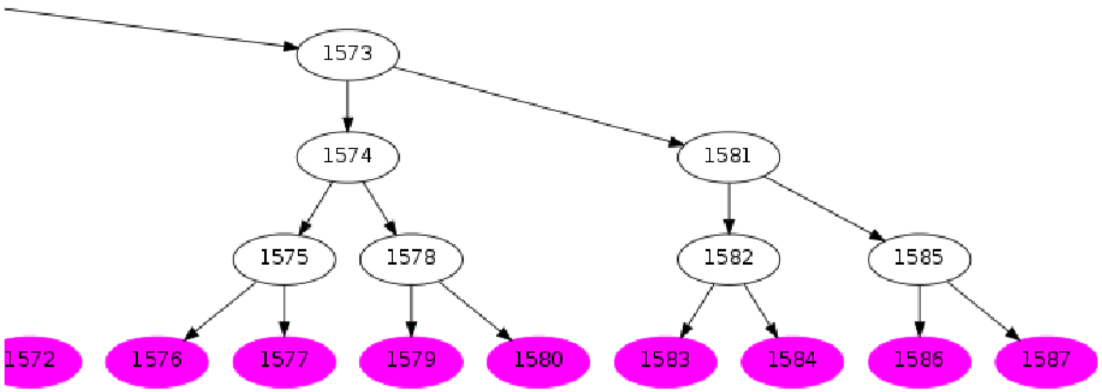
Instance with $|V| = 4$



the whole tree



zooming on the left



zooming on the right

. . . and the bound

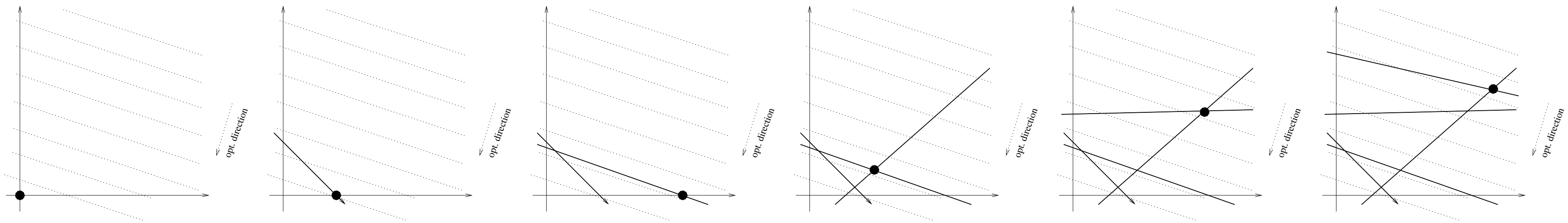
Improving the “branch” idea

- What if we could determine that a branch growing from a given node α does not lead to any improving tour?
- For example, suppose we can determine that a given branch will lead to tours whose cost is **at best** $\bar{d}$
- Suppose also that our incumbent has cost $d(S) < \bar{d}$
- Then we can immediately determine that the branch should be discarded

α is said to be **pruned by bound**

Relaxations

- How can we compute the bound to the cost in the optimization direction (lower if minimum, upper if maximum)?
- Consider the case of linear programming (optimizing along a linear direction on a polyhedron):



As the number of constraints increases, the optimum gets worse

- In general, if $C \supseteq D$ are two sets whose elements are weighted, $\min C \leq \min D$ and $\max C \geq \max D$
- If the problem requires optimizing over D, then optimizing over C gives a **guaranteed bound** for the problem on D
- Optimization on C is a **relaxation** of the optimization on D

A relaxation is useful only if optimizing on C is easier than on D

A bound for the TSP

$\mathcal{T}$ = set of all tours on V	$\mathcal{O}$ = set of all orders on V
$\mathcal{S}$ = set of all permutations of V	$\mathcal{M}$ = set of all maps $V \to V$

- We have $\mathcal{T} = \mathcal{O} \subsetneq \mathcal{S} \subsetneq \mathcal{M}$, so $\mathcal{M}$ is a relaxation of $\mathcal{T}$
- Finding the minimum cost map $f : V \to V$ is easy:

$$\boxed{\forall v \in V \quad f(v) = \text{argmin}\{d_{vw} \mid w \in \delta^+(v)\},} \qquad (*)$$

(where $\delta^+(v) = \{w \in V \mid (u, v) \in A\}$ is the *outstar* of v)

- Consider the TSP instance:

	1	2	3	4
1	∞	1.2	0.6	2
2	1.2	∞	3	1.1
3	0.6	3	∞	0.9
4	2	1.1	0.9	∞

- The map $1 \to 3, 2 \to 4, 3 \to 1, 4 \to 3$ has minimum cost

0.6+1.1+0.6+0.9= $\boxed{\textbf{3.2}}$

No tour can ever have lower cost

Adapting the bound to a node

- A node α in the search tree is a quadruplet (Y, N, S, c') where $Y, N, S \subset A$ and $c' \in \mathbb{R}$ (discussed later)
- Every tour in the subtree rooted at α contains the arcs in Y and does not contain the arcs in N
- $\Rightarrow$ we need a bound on tours extending Y and not containing N
- Again use maps $V \to V$ as target relaxations

  Consider mapping $u \to v$ for all $(u, v) \in Y$
  and **not** mapping $w \to z$ for all $(w, z) \in N$
- We update $(*)$ as follows

$$\forall u \in V \quad f(u) = \begin{cases} v & \text{if } (u,v) \in Y \\ \text{argmin}\{d_{vw} \mid w \in \delta^+(v) \smallsetminus N\} & \text{otherwise} \end{cases} \qquad (**)$$

- Define `lowerBound(`α`)` to return the set of arcs defined by f

Implicit exhaustive exploration

- BB generates exploration trees (also called **BB trees**)
- These trees are not explicitly exhaustive (as for `treeSearch`) because some nodes are **pruned by bound**
- Since pruning by bound is guaranteed not to lose any optimal solution, the search is said to be "implicitly exhaustive"
- To reduce CPU time, aim to reduce the tree size by adjusting some parameters

Choosing the next node

- `treeSearch` works using a depth-first exploration
- This imposes a depth-first order to the BB tree nodes
- Is this the best possible order as regards tree size?

- Intuitively, we wish to find the best tour S^* as early as possible in the search:
  - since $d(S^*)$ is small, the chance that other nodes have lower bounds $\bar{d}$ with $\bar{d} \geq d(S^*)$ increases
  - more nodes could be pruned by bound
- Choose order which maximizes chances to find S^* early:

Process node with lowest lower bound

Infeasible nodes

- Suppose that, at node α, Y contains the arcs $(1,2),(2,1)$ and that $n > 2$
- Since $(1,2),(2,1)$ is already a tour (of length 2),

  no extension of $(1,2),(2,1)$ can ever be Hamiltonian

- No need to explore the subtree rooted at α: the node is **infeasible** and can be pruned
- In general, if Y contains a tour shorter than n, it can be pruned (**pruned by infeasibility**)
- Also, in $(**)$ it might happen that, given u, $(u,v) \in N$ for each v, so $f(u) = \varnothing \Rightarrow$ node is infeasible
- Nodes are also infeasible if $Y \cap N \neq \varnothing$
- Let $\texttt{isFeasible}(\alpha)$ return TRUE iff α is a feasible node

The algorithm

Require: set Q of nodes (Y, N, S, c') ordered by increasing $c' \in \mathbb{R}$

  Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
  while $Q \neq \varnothing$ **do**
    Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$; let $Q = Q \smallsetminus \{\alpha\}$
    if `isFeasible`(α) **then**
      Let $L =$ `lowerBound`(α)
      if $d(L) < d(S)$ **then**
        if $\neg$ `isTour`(L) **then**
          if $Y \cup N \neq A$ **then**
            choose $a \in A \smallsetminus (Y \cup N)$
            Let $\beta = (Y \cup \{a\}, N, S, d(L))$; let $\gamma = (Y, N \cup \{a\}, S, d(L))$
            Let $Q = Q \cup \{\beta, \gamma\}$
          end if
        else
          Let $S = L$
        end if
      end if
    end if
  end while
  return S

In action on the root node

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \smallsetminus \{\alpha\}$
  if $\underline{\texttt{isFeasible}(\alpha)}$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\neg\, \texttt{isTour}(L)$ **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \smallsetminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

root node

$Y = N = S = \varnothing$

$\texttt{isFeasible}(\alpha) =$ TRUE

because $\varnothing$ can be extended to a tour

In action on the root node

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \smallsetminus \{\alpha\}$
  if $\texttt{isFeasible}(\alpha)$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\neg\ \texttt{isTour}(L)$ **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \smallsetminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

root node

Instance:

	1	2	3	4
1	∞	1.2	$\boxed{0.6}$	2
2	1.2	∞	3	$\boxed{1.1}$
3	$\boxed{0.6}$	3	∞	0.9
4	2	1.1	$\boxed{0.9}$	∞

$L = \{(1,3), (2,4), (3,1), (4,3)\}$ by greedy choice of cheapest next vertex

In action on the root node

```
Let Q = {(∅, ∅, ∅, −∞)}
while Q ≠ ∅ do
  Let α = (Y, N, S, c′) = argmin_{c′} Q
  Let Q = Q ∖ {α}
  if isFeasible(α) then
    Let L = lowerBound(α)
    if d(L) < d(S) then
      if ¬ isTour(L) then
        if Y ∪ N ≠ A then
          choose a ∈ A ∖ (Y ∪ N)
          Let β = (Y ∪ {a}, N, S, d(L))
          Let γ = (Y, N ∪ {a}, S, d(L))
          Let Q = Q ∪ {β, γ}
        end if
      else
        Let S = L
      end if
    end if
  end if
end while
```

root node

$L = \{(1,3),(2,4),(3,1),(4,3)\}$

$d(L) = 3.2 < \infty = d(\varnothing)$

In action on the root node

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \smallsetminus \{\alpha\}$
  if $\texttt{isFeasible}(\alpha)$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\neg\, \texttt{isTour}(L)$ **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \smallsetminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

root node

$L = \{(1,3), (2,4), (3,1), (4,3)\}$

is evidently not a tour

In action on the root node

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \smallsetminus \{\alpha\}$
  if $\texttt{isFeasible}(\alpha)$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\neg\ \texttt{isTour}(L)$ **then**
        if $\underline{Y \cup N \neq A}$ **then**
          choose $a \in A \smallsetminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

root node

$$Y = N = \varnothing \Rightarrow Y \cup N \neq A$$

In action on the root node

```
Let Q = {(∅, ∅, ∅, −∞)}
while Q ≠ ∅ do
  Let α = (Y, N, S, c′) = argmin_{c′} Q
  Let Q = Q ∖ {α}
  if isFeasible(α) then
    Let L = lowerBound(α)
    if d(L) < d(S) then
      if ¬ isTour(L) then
        if Y ∪ N ≠ A then
          choose a ∈ A ∖ (Y ∪ N)
          Let β = (Y ∪ {a}, N, S, d(L))
          Let γ = (Y, N ∪ {a}, S, d(L))
          Let Q = Q ∪ {β, γ}
        end if
      else
        Let S = L
      end if
    end if
  end if
end while
```

root node

for example, take a as the cheapest arc in $A \setminus \varnothing = A$, i.e. $a = (1, 3)$

In action on the root node

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \smallsetminus \{\alpha\}$
  if $\texttt{isFeasible}(\alpha)$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\neg\ \texttt{isTour}(L)$ **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \smallsetminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

root node

create a new node with $Y = \{(1,3)\}$, $N = \varnothing$, $S = \varnothing$, $d(L) = 3.2$

In action on the root node

```
Let Q = {(∅, ∅, ∅, −∞)}
while Q ≠ ∅ do
  Let α = (Y, N, S, c′) = argmin_{c′} Q
  Let Q = Q ∖ {α}
  if isFeasible(α) then
    Let L = lowerBound(α)
    if d(L) < d(S) then
      if ¬ isTour(L) then
        if Y ∪ N ≠ A then
          choose a ∈ A ∖ (Y ∪ N)
          Let β = (Y ∪ {a}, N, S, d(L))
          Let γ = (Y, N ∪ {a}, S, d(L))
          Let Q = Q ∪ {β, γ}
        end if
      else
        Let S = L
      end if
    end if
  end if
end while
```

root node

create a new node with $Y = \varnothing$, $N = \{(1,3)\}$, $S = \varnothing$, $d(L) = 3.2$

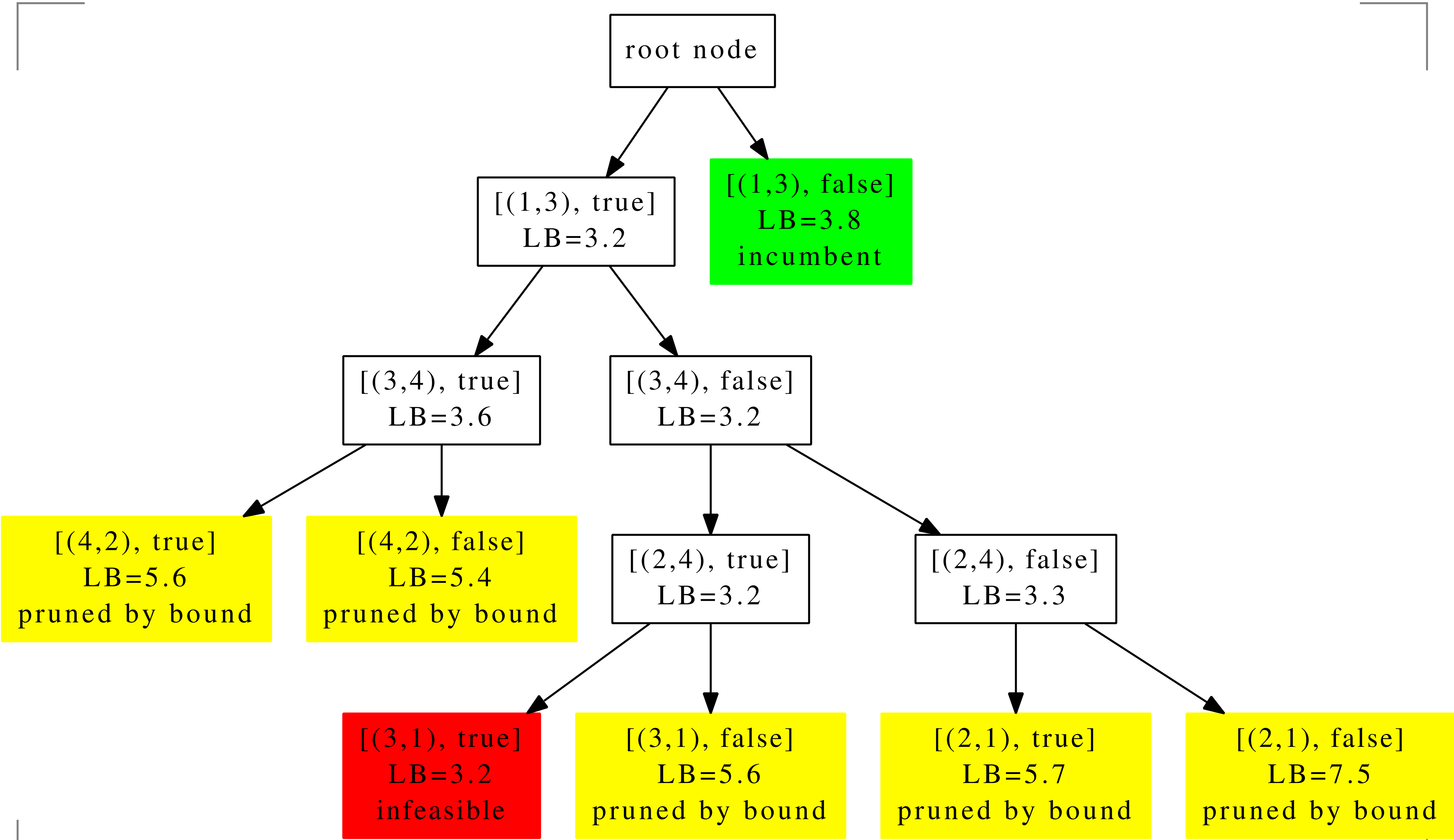
In action on the root node

```
Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ do
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \setminus \{\alpha\}$
  if isFeasible($\alpha$) then
    Let $L =$ lowerBound($\alpha$)
    if $d(L) < d(S)$ then
      if $\neg$ isTour($L$) then
        if $Y \cup N \neq A$ then
          choose $a \in A \setminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while
```

root node

add these new nodes to the queue Q

The BB tree


root node
[(1,3), true]
LB=3.2
[(1,3), false]
LB=3.8
incumbent
[(3,4), true]
LB=3.6
[(3,4), false]
LB=3.2
[(4,2), true]
LB=5.6
pruned by bound
[(4,2), false]
LB=5.4
pruned by bound
[(2,4), true]
LB=3.2
[(2,4), false]
LB=3.3
[(3,1), true]
LB=3.2
infeasible
[(3,1), false]
LB=5.6
pruned by bound
[(2,1), true]
LB=5.7
pruned by bound
[(2,1), false]
LB=7.5
pruned by bound

In action on the incumbent

```
Let Q = {(∅, ∅, ∅, −∞)}
while Q ≠ ∅ do
  Let α = (Y, N, S, c′) = argmin_{c′} Q
  Let Q = Q ∖ {α}
  if isFeasible(α) then
    Let L = lowerBound(α)
    if d(L) < d(S) then
      if ¬ isTour(L) then
        if Y ∪ N ≠ A then
          choose a ∈ A ∖ (Y ∪ N)
          Let β = (Y ∪ {a}, N, S, d(L))
          Let γ = (Y, N ∪ {a}, S, d(L))
          Let Q = Q ∪ {β, γ}
        end if
      else
        Let S = L
      end if
    end if
  end if
end while
```

"incumbent" node

the queue Q is not empty

In action on the incumbent

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  $\underline{\text{Let } \alpha = (Y, N, S, c') = \text{argmin}_{c'} Q}$
  Let $Q = Q \setminus \{\alpha\}$
  if $\texttt{isFeasible}(\alpha)$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\neg\, \texttt{isTour}(L)$ **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \setminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

"incumbent" node

$Q = \{(\{(1,3)\}, \varnothing, \varnothing, 3.2)$, $(\varnothing, \{(1,3)\}, \varnothing, 3.2)\}$

since both nodes have associated bound $c' = 3.2$*, both can be chosen*

for example, choose the node $\alpha = (\varnothing, \{(1,3)\}, \varnothing, 3.2)$

In action on the incumbent

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \smallsetminus \{\alpha\}$
  if $\texttt{isFeasible}(\alpha)$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\neg\ \texttt{isTour}(L)$ **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \smallsetminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

"incumbent" node

$$Q = \{(\{(1,3)\}, \varnothing, \varnothing, 3.2)\}$$

In action on the incumbent

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \setminus \{\alpha\}$
  if $\underline{\texttt{isFeasible}(\alpha)}$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\neg\ \texttt{isTour}(L)$ **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \setminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

“incumbent” node

Since $Y = \varnothing$, there are no circuits of length < 4

the node is feasible

In action on the incumbent

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \smallsetminus \{\alpha\}$
  if `isFeasible`(α) **then**
    Let $L =$ `lowerBound`(α)
    if $d(L) < d(S)$ **then**
      if $\neg$ `isTour`(L) **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \smallsetminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

"incumbent" node

$(N = \{(1, 3)\})$

The greedy choice $\{(1, 3), (2, 4), (3, 1), (4, 3)\}$ *does not work because* $(1, 3) \in N$

	1	2	3	4
1	∞	$\boxed{1.2}$	~~0.6~~	2
2	1.2	∞	3	$\boxed{1.1}$
3	$\boxed{0.6}$	3	∞	0.9
4	2	1.1	$\boxed{0.9}$	∞

"next best" choice is to replace $(1, 3)$ with $(1, 2)$, obtaining $L = \{(1, 2), (2, 4), (3, 1), (4, 3)\}$ with cost $d(L) = 3.8$

In action on the incumbent

```
Let Q = {(∅, ∅, ∅, −∞)}
while Q ≠ ∅ do
  Let α = (Y, N, S, c′) = argmin_{c′} Q
  Let Q = Q ∖ {α}
  if isFeasible(α) then
    Let L = lowerBound(α)
    if d(L) < d(S) then
      if ¬ isTour(L) then
        if Y ∪ N ≠ A then
          choose a ∈ A ∖ (Y ∪ N)
          Let β = (Y ∪ {a}, N, S, d(L))
          Let γ = (Y, N ∪ {a}, S, d(L))
          Let Q = Q ∪ {β, γ}
        end if
      else
        Let S = L
      end if
    end if
  end if
end while
```

“incumbent” node

$(S = \varnothing)$

$3.8 = d(L) < \infty = d(\varnothing)$

In action on the incumbent

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \smallsetminus \{\alpha\}$
  if $\texttt{isFeasible}(\alpha)$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\underline{\neg\ \texttt{isTour}(L)}$ **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \smallsetminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        Let $S = L$
      end if
    end if
  end if
end while

“incumbent” node

L yields a function

$1 \to 2, 2 \to 4, 4 \to 3, 3 \to 1$

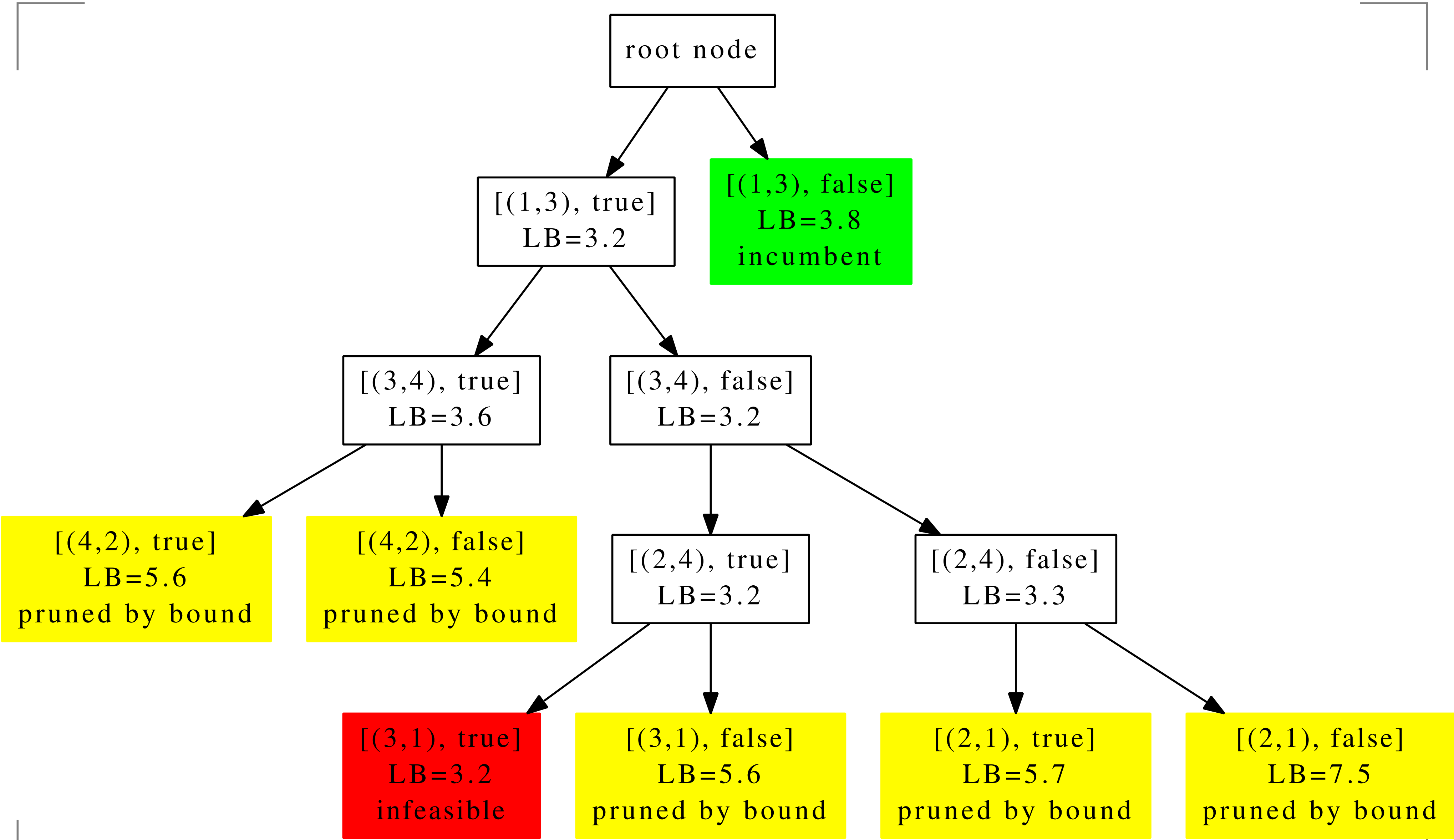
which is a Hamiltonian tour

In action on the incumbent

Let $Q = \{(\varnothing, \varnothing, \varnothing, -\infty)\}$
while $Q \neq \varnothing$ **do**
  Let $\alpha = (Y, N, S, c') = \text{argmin}_{c'} Q$
  Let $Q = Q \smallsetminus \{\alpha\}$
  if $\texttt{isFeasible}(\alpha)$ **then**
    Let $L = \texttt{lowerBound}(\alpha)$
    if $d(L) < d(S)$ **then**
      if $\neg\ \texttt{isTour}(L)$ **then**
        if $Y \cup N \neq A$ **then**
          choose $a \in A \smallsetminus (Y \cup N)$
          Let $\beta = (Y \cup \{a\}, N, S, d(L))$
          Let $\gamma = (Y, N \cup \{a\}, S, d(L))$
          Let $Q = Q \cup \{\beta, \gamma\}$
        end if
      else
        <u>Let $S = L$</u>
      end if
    end if
  end if
end while

“incumbent” node
we update S with
$L = \{(1,2),(2,4),(3,1),(4,3)\}$
and $d(S) = 3.8$

The BB tree again



It works

Thm.

The Branch-and-Bound algorithm finds a tour of minimum cost

Proof

(Sketch)

(*termination*) Once Y, N and the node set order have been given, there are unique possible values for S, c'. Since $Y, N \subset A$, the number of nodes is finite. Notice no node is ever considered more than once: hence the algorithm terminates.

(*optimality*) Suppose, to get a contradiction, that the algorithm returns a tour S but that the optimum S^* has $d(S^*) < d(S)$. Then there exist BB tree nodes with $Y \subset S^*$ and $N \subset A \setminus S^*$ which are feasible (for otherwise S^* would not be Hamiltonian), and such that $d(Y) < d(S)$ (because $Y \subseteq S^* \Rightarrow d(Y) \leq d(S^*) < d(S)$). In particular, there will be a branch leading to a node with $L = S^*$: since $d(L) = d(S^*) < d(S)$, the algorithm will set $S = L = S^*$, against the assumption.

In general

- A general Branch-and-Bound algorithm rests on the following fundamental primitives:
  - a (not necessarily binary) disjunction over which to **branch**, such that a recursive search over the disjunction lists all possible solutions
  - the ability to compute a **bound** value (in the optimization direction) which improves as the number of constraints imposed on the solution increases
- Speed-up heuristics include:
  - a good strategy for choosing the next node to process
  - computation of a good incumbent
  - good choice of branching disjunction

Application to Mixed-Integer Linear Programming

MILP formulation

- A MILP is like a Linear Program (LP) where some of the variables are constrained to be integer
- Formulation: given known vectors $c \in \mathbb{R}^n, b \in \mathbb{R}^m$, a known matrix $A \in \mathbb{R}^{nm}$, a known set $Z \subseteq \{1, \ldots, n\}$ and a vector $x \in \mathbb{R}^n$ of *decision variables*,

$$\begin{array}{rrcl} & \min c^{\mathsf{T}} x & & \\ & Ax & \leq & b \\ \forall i \in Z & x_i & \in & \mathbb{Z} \end{array}$$

- This expresses the minimum value of the *objective function* $c^{\mathsf{T}} x$ subject to the *linear constraints* $Ax \leq b$ and the *integrality constraints* $\forall i \in Z \ (x_i \in \mathbb{Z})$
- If $Z = \varnothing$, get LP — can solve it by the simplex method or by some interior point method

MILP BB

- **Solution**: assignment of values $\bar{x}$ to decision variables x
- **Feasible solution**: $\bar{x}$ satisfies all constraints
- **Relaxation**: all solutions $\bar{x}$ satisfying linear but not necessarily integrality constraints (*solve using simplex*)
- **Lower bound**: minimum objective value of the relaxed LP
- **Branching**: pick a variable x_i with $i \in Z$ s.t. $\bar{x} \notin \mathbb{Z}$:

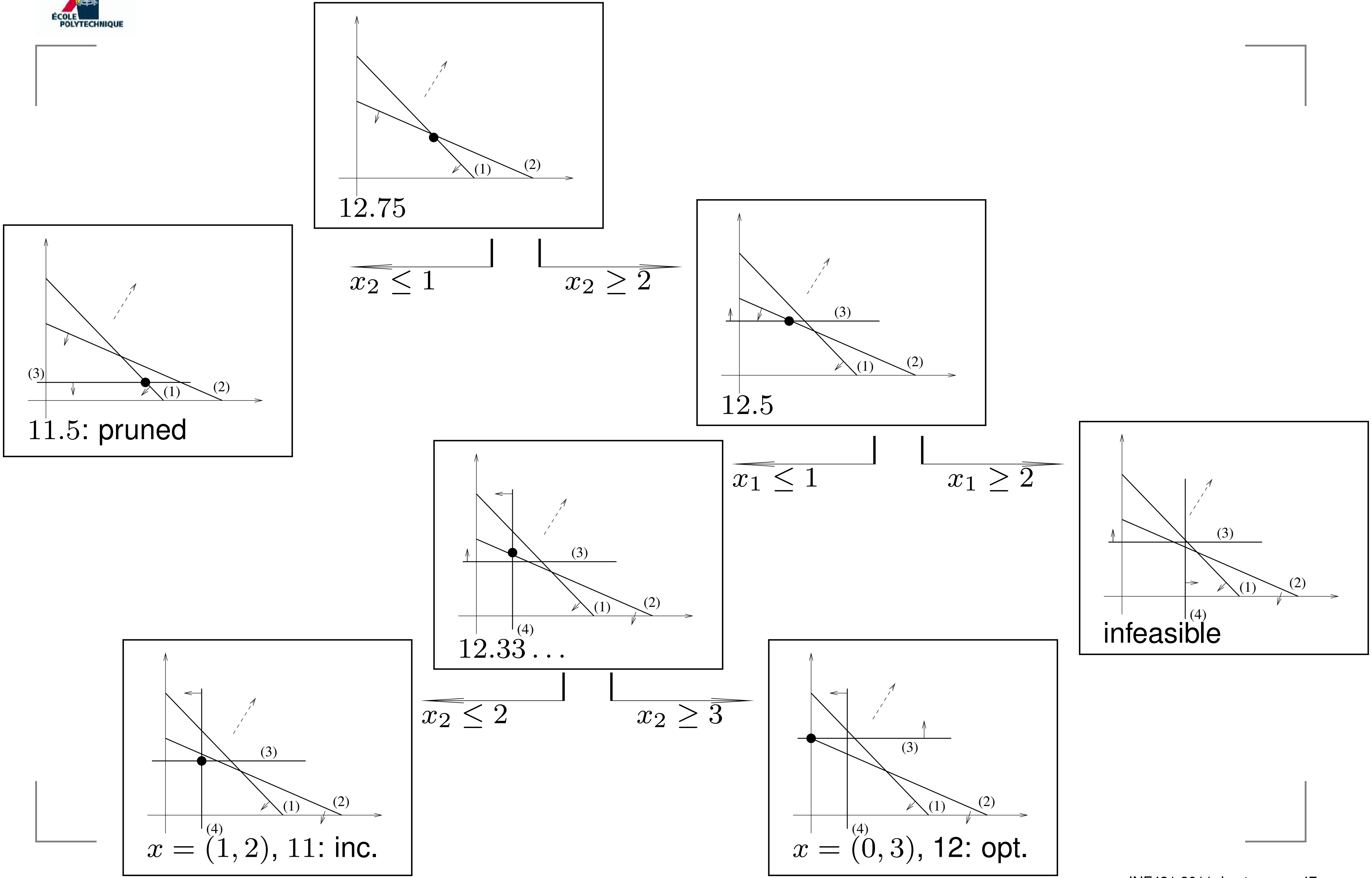
  $$x \leq \lfloor \bar{x} \rfloor \quad \vee \quad x \geq \lceil \bar{x} \rceil$$

  is a valid disjunction
- **Applications**: too many to mention! (*scheduling, energy production, network design, vehicle routing, logistics, stock management, combinatorial optimization problems...*)
- **Implementations**: *IBM-ILOG CPLEX (commercial), FICO XPress (commercial), CBC (open source), GLPK (open source)*

MILP example

$$
\begin{aligned}
\max 3x_1 + 4x_2 & & & \\
2x_1 + x_2 & \leq & 6 & \quad (1) \\
2x_1 + 3x_2 & \leq & 9 & \quad (2) \\
x_1, x_2 & \geq & 0 & \\
x_1, x_2 & \in & \mathbb{Z} &
\end{aligned}
$$

MILP BB Tree


12.75
(1)
(2)
$x_2 \leq 1$
$x_2 \geq 2$
11.5: pruned
(3)
12.5
$x_1 \leq 1$
$x_1 \geq 2$
12.33...
(4)
infeasible
$x_2 \leq 2$
$x_2 \geq 3$
$x = (1, 2)$, 11: inc.
$x = (0, 3)$, 12: opt.

Historical notes

First reference to BB: 1960

ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

BY A. H. LAND AND A. G. DOIG

In the classical linear programming problem the behaviour of continuous, nonnegative variables subject to a system of linear inequalities is investigated. One possible generalization of this problem is to relax the continuity condition on the variables. This paper presents a simple numerical algorithm for the solution of programming problems in which some or all of the variables can take only discrete values. The algorithm requires no special techniques beyond those used in ordinary linear programming, and lends itself to automatic computing. Its use is illustrated on two numerical examples.

1. INTRODUCTION

THERE IS A growing literature [**1**, **3**, **5**, **6**] about optimization problems which could be formulated as linear programming problems with additional constraints that some or all of the variables may take only integral values. This form of linear programming arises whenever there are indivisibilities. It is not meaningful, for instance, to schedule 3 – 7/10 flights between two cities, or to undertake only 1/4 of the necessary setting up operation for running a job through a machine shop. Yet it is basic to linear programming that the variables are free to take on any positive value,[1] and this sort of answer is very likely to turn up.

In some cases, notably those which can be expressed as transport problems, the linear programming solution will itself yield discrete values of the variables. In other cases the percentage change in the maximand[2] from common sense rounding of the variables is sufficiently small to be neglected. But there remain many problems where the discrete variable constraints are significant and costly.

Until recently there was no general automatic routine for solving such problems, as opposed to procedures for proving the optimality of conjectured solutions, and the work reported here is intended to fill the gap. About the time of its completion an alternative method was proposed by Gomory [**5**] and subsequently extended by Beale [**1**]. Gomory's method

[1] Or more generally, any value within a bounded interval.

[2] We shall speak throughout of maximisation, but of course an exactly analogous argument applies to minimisation.

497

First reference to TSP: 1759

SOLUTION

D'UNE

QUESTION CURIEUSE QUI NE PAROIT SOUMISE À AUCUNE ANALYSE,

PAR M. EULER.

1.

Je me trouvai un jour dans une compagnie, où, à l'occasion du jeu d'echecs quelqu'un proposa cette question: *de parcourir avec un cavalier toutes les cases d'un échiquier, sans parvenir jamais deux fois à la même, & en commençant par une case donnée.* On mettoit pour cette fin des jettons sur toutes les 64 cases de l'échiquier, à l'exception de celle où le Cavalier devoit commencer sa route; & de chaque case où le Cavalier passoit conformément à sa marche, on ôtoit le jetton, de sorte qu'il s'agissoit d'enlever de cette façon successivement tous les jettons. Il faloit donc éviter d'un côté, que le cavalier ne revint jamais à une case vuide, & d'un autre côté il faloit diriger en sorte sa course, qu'il parcourut enfin toutes les cases.

2. Ceux qui croyoient cette question assez aisée firent plusieurs essais inutiles sans pouvoir atteindre au but; après quoi celui qui avoit proposé la question, ayant commencé par une case donnée, a sçu si bien diriger la route, qu'il a heureusement enlevé tous les jettons. Cependant la multitude des cases ne permettoit pas qu'on ait pû imprimer à la mémoire la route qu'il avoit suivie; & ce n'étoit qu'après plusieurs essais, que j'ai enfin rencontré une telle route, qui satisfit à la question; encore ne valoit-elle que pour une certaine case initiale. Je ne me souviens plus, si on lui a laissé la liberté de la choisir lui-même; mais il a très positivement assuré qu'il étoit en état de l'éxécuter, quelle que soit la case où l'on voulut qu'il commençat.

3.

toute particuliere, que Mr. Bertrand de Geneve m'a fournie; car, quoiqu'elle soit legere en elle-même, & tout à fait étrangere à la Géométrie, elle doit être regardée comme très remarquable, dès qu'on aura trouvé moyen d'y appliquer l'Analyse. Or je ferai voir qu'elle est susceptible d'une analyse tout particuliere, qui doit mériter d'autant plus d'attention, que cette analyse demande des raisonnemens peu usités ailleurs. On convient aisément de l'excellence de l'Analyse, mais on la croit communément bornée à de certaines recherches, qu'on rapporte aux Mathématiques; & partant il sera toujours fort important d'en faire usage dans des matieres qui lui semblent refuser tout accès: puisqu'il est certain qu'elle renferme l'art de raisonner dans le plus haut degré. On ne sauroit donc étendre les bornes de l'Analyse, sans qu'on ait raison de s'en promettre de très grands avantages.

6. Or d'abord je remarque, qu'on pourroit satisfaire à la question, si l'on trouvoit une telle route, où la derniere case marquée par 64 seroit éloignée de la premiere 1 d'un saut de cavalier, de sorte qu'il pourroit sauter de la derniere sur la premiere. Car, ayant trouvé une telle route rentrante en elle-même, on pourra commencer par quelque case que ce soit, & de là continuer la course suivant l'ordre des nombres jusqu'à la case marquée par 64, d'où, en sautant à celle qui est marquée par 1, il acheveroit la course jusqu'à retourner à celle d'où il étoit parti. Or voilà une telle route rentrante en elle-même,

42	57	44	9	40	21	46	7
55	10	41	58	45	8	39	20
12	43	56	61	22	59	6	47
63	54	11	30	25	28	19	38
32	13	62	27	60	23	48	5
53	64	31	24	29	26	37	18
14	33	2	51	16	35	4	49
1	52	15	34	3	50	17	36

7.

First reference to BB for TSP: 1963

AN ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM

John D. C. Little

Massachusetts Institute of Technology

Katta G. Murty*

Indian Statistical Institute

Dura W. Sweeney†

International Business Machines Corporation

Caroline Karel

Case Institute of Technology

(Received March 6, 1963)

A 'branch and bound' algorithm is presented for solving the traveling salesman problem. The set of all tours (feasible solutions) is broken up into increasingly small subsets by a procedure called branching. For each subset a lower bound on the length of the tours therein is calculated. Eventually, a subset is found that contains a single tour whose length is less than or equal to some lower bound for every tour. The motivation of the branching and the calculation of the lower bounds are based on ideas frequently used in solving assignment problems. Computationally, the algorithm extends the size of problem that can reasonably be solved without using methods special to the particular problem.

THE TRAVELING salesman problem is easy to state: A salesman, starting in one city, wishes to visit each of $n-1$ other cities once and only once and return to the start. In what order should he visit the cities to minimize the total distance traveled? For 'distance' we can substitute time, cost, or other measure of effectiveness as desired. Distance or costs between all city pairs are presumed known.

The problem has become famous because it combines ease of statement with difficulty of solution. The difficulty is entirely computational, since a solution obviously exists. There are $(n-1)!$ possible tours, one or more of which must give minimum cost. (The minimum cost could conceivably be infinite—it is conventional to assign an infinite cost to travel between city pairs that have no direct connection.)

The traveling salesman problem recently achieved national prominence

The end