BACKTRACKING

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Introduction

Backtracking and branch-and-bound

- Solving some larger instances of intractable problems
- Best than exhaustive search
- In the worst case, still intractable

Based on (explicit or implicit): state-space trees

- Constructs a partial solution, if this one is not promising, it goes back and try a different one

Branch-and-bound: applicable to optimisation problems

State-space exploration

- Backtracking: usually depth-first
- Branch-and-bound: usually best-first

Centro de Informática UFPE

Universidade Federal de Pernambuco

Agenda

n-Queens problem

Place n queens on an $n \times n$ chessboard such that no two queens attack each other

- $n = 1$: trivial solution
- $n = 2 \vee n = 3$: there is no solution

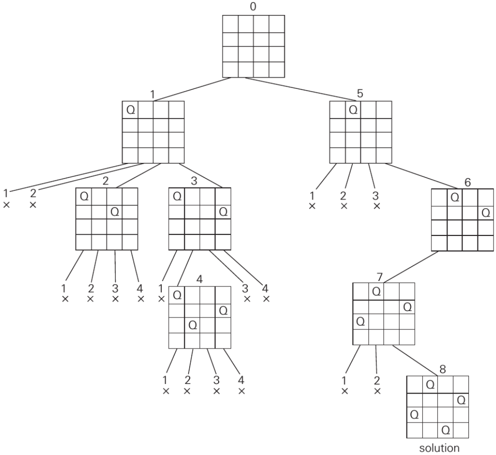
Simplification: assign a column (or row) for each queen

- Start with the empty board
- Try to place the first queen in the first possible position
- If possible, tries to place the next queen
- If not possible, tries a different position for the previous one
- If all queens are placed, a solution has been found (may stop)

Centro de Informática UFPE

Universidade Federal de Pernambuco

n-Queens problem[1]

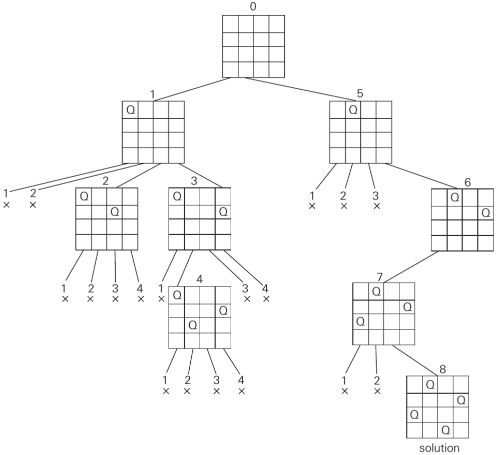


Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

```
if l = n then return true;
else
    for i ← 0 to n − 1 do
        if valid(M, l, i) then
            M[l][i] ← 1;
            if qns(l + 1, M) then
                return true;
            else M[l][i] ← 0 ;
    return false;
```

[1] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[2]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

```
if l = n then return true;
else
    for i ← 0 to n − 1 do
        if valid(M, l, i) then
            M[l][i] ← 1;
            if qns(l + 1, M) then
                return true;
            else M[l][i] ← 0 ;
    return false;
```

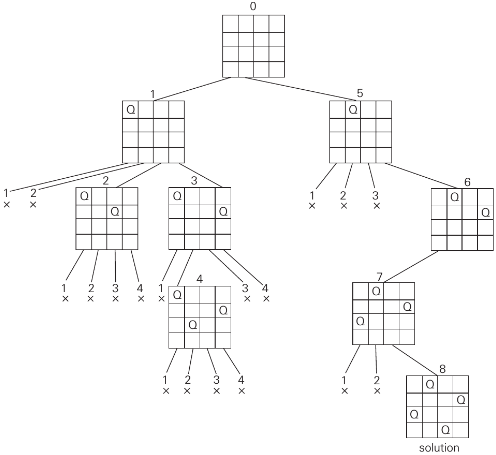
qns(0,M)

Centro de Informática UFPE

Universidade Federal de Pernambuco

[2] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[3]



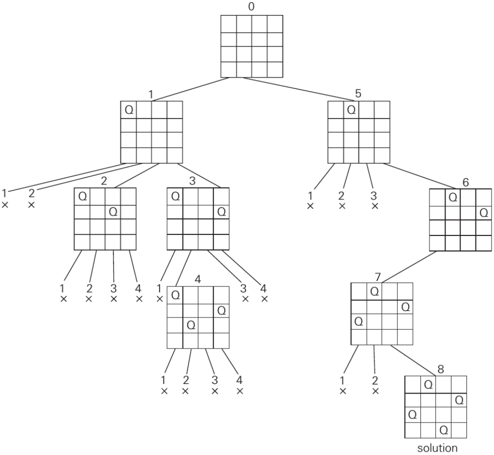
Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

```
if l = n then return true;
else
    for i ← 0 to n − 1 do
        if valid(M, l, i) then
            M[l][i] ← 1;
            if qns(l + 1, M) then
                return true;
            else M[l][i] ← 0 ;
    return false;
```

qns(0,M) > qns(1,M)

[3] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[4]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

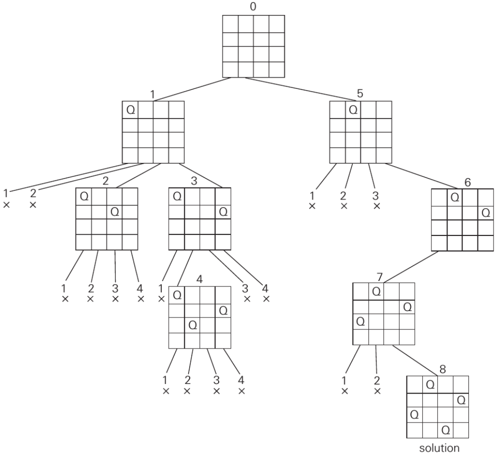
```
if l = n then return true;
else
    for i ← 0 to n − 1 do
        if valid(M, l, i) then
            M[l][i] ← 1;
            if qns(l + 1, M) then
                return true;
            else M[l][i] ← 0 ;
    return false;
```

qns(0,M) > qns(1,M) > qns(2,M)

backtracking!

[4] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[5]



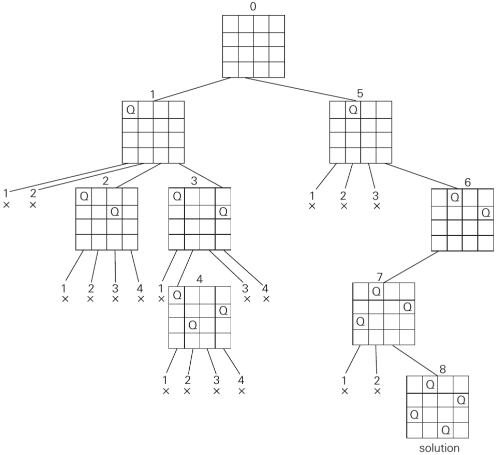
Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

if $l = n$ **then return** *true*;
else
    for $i \leftarrow 0$ **to** $n - 1$ **do**
        if $valid(M, l, i)$ **then**
            $M[l][i] \leftarrow 1$;
            if $qns(l + 1, M)$ **then**
                return *true*;
            else $M[l][i] \leftarrow 0$;
    return *false*;

qns(0,M) > qns(1,M)

[5] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[6]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

```
if l = n then return true;
else
    for i ← 0 to n − 1 do
        if valid(M, l, i) then
            M[l][i] ← 1;
            if qns(l + 1, M) then
                return true;
            else M[l][i] ← 0 ;
    return false;
```

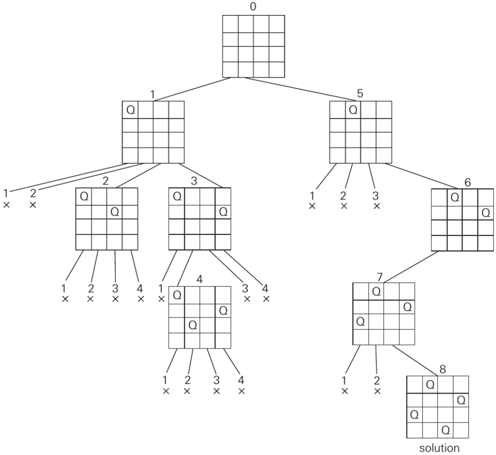
qns(0,M) > qns(1,M) > qns(2,M)

Centro de Informática UFPE

Universidade Federal de Pernambuco

[6] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[7]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

```
if l = n then return true;
else
    for i ← 0 to n − 1 do
        if valid(M, l, i) then
            M[l][i] ← 1;
            if qns(l + 1, M) then
                return true;
            else  M[l][i] ← 0 ;
    return false;
```

qns(0,M) > qns(1,M) > qns(2,M) > qns(3,M)

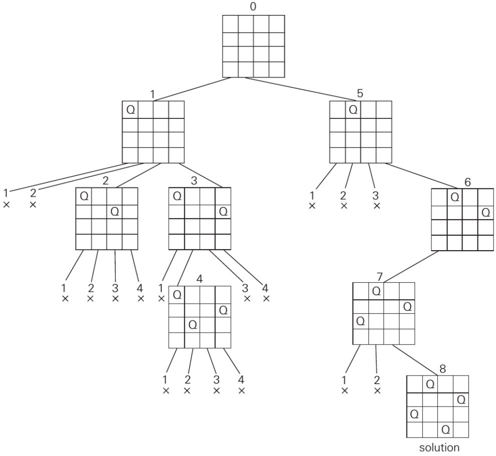
backtracking!

Centro de Informática UFPE

Universidade Federal de Pernambuco

[7] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[8]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

if $l = n$ **then return** *true*;
else
  for $i \leftarrow 0$ **to** $n - 1$ **do**
    if $valid(M, l, i)$ **then**
      $M[l][i] \leftarrow 1$;
      if $qns(l + 1, M)$ **then**
        return *true*;
      else $M[l][i] \leftarrow 0$;
  return *false*;

qns(0,M) > qns(1,M) > qns(2,M)

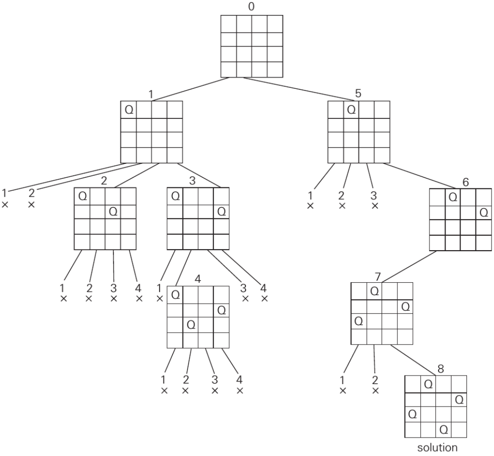
backtracking!

Centro de Informática UFPE

Universidade Federal de Pernambuco

[8] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[9]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

if $l = n$ **then return** *true*;
else
  for $i \leftarrow 0$ **to** $n - 1$ **do**
    if $valid(M, l, i)$ **then**
      $M[l][i] \leftarrow 1$;
      if $qns(l + 1, M)$ **then**
        return *true*;
      else $M[l][i] \leftarrow 0$;
  return *false*;

qns(0,M) > qns(1,M)

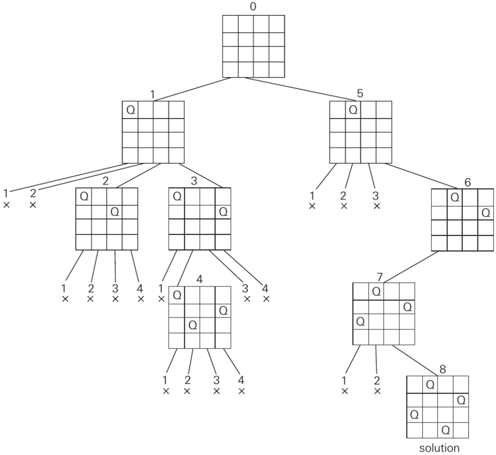
backtracking!

Centro de Informática UFPE

Universidade Federal de Pernambuco

[9] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[10]



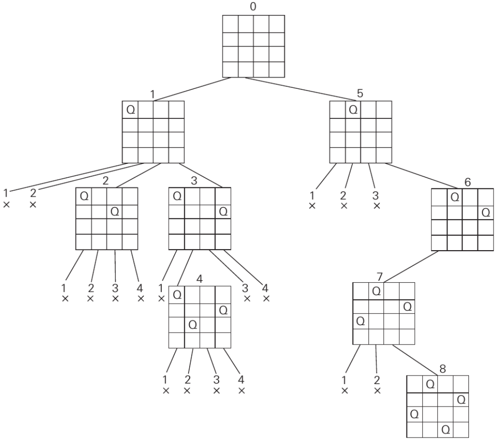
Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

if $l = n$ **then return** *true*;
else
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if $valid(M, l, i)$ **then**
      $M[l][i] \leftarrow 1$;
      if $qns(l+1, M)$ **then**
        return *true*;
      else $M[l][i] \leftarrow 0$;
  return *false*;

qns(0,M)

[10] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[11]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

```
if l = n then return true;
else
    for i ← 0 to n − 1 do
        if valid(M, l, i) then
            M[l][i] ← 1;
            if qns(l + 1, M) then
                return true;
            else M[l][i] ← 0 ;
    return false;
```

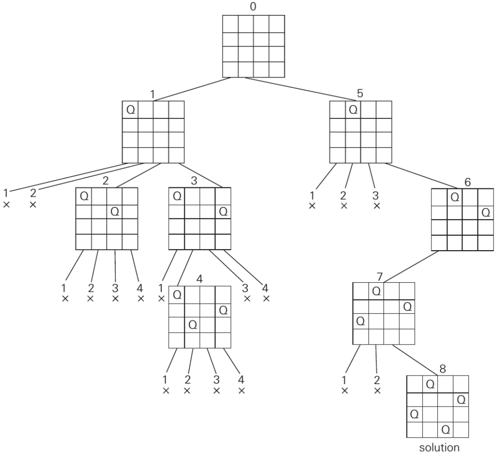
qns(0,M) > qns(1,M)

Centro de Informática UFPE

Universidade Federal de Pernambuco

[11] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[12]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

if $l = n$ **then return** *true*;
else
  for $i \leftarrow 0$ **to** $n - 1$ **do**
    if $valid(M, l, i)$ **then**
      $M[l][i] \leftarrow 1$;
      if $qns(l + 1, M)$ **then**
        return *true*;
      else $M[l][i] \leftarrow 0$;
  return *false*;

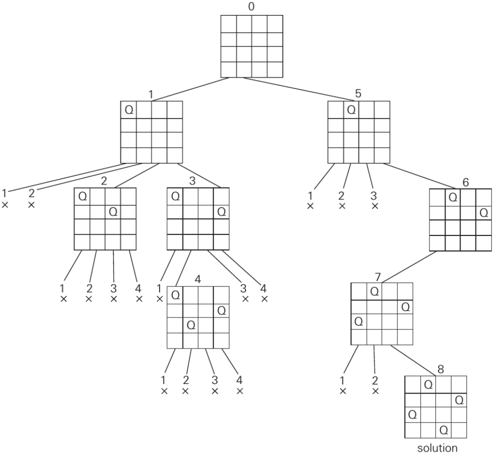
qns(0,M) > qns(1,M) > qns(2,M)

Centro de Informática UFPE

Universidade Federal de Pernambuco

12 Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[13]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

if $l = n$ **then return** *true*;
else
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if $valid(M, l, i)$ **then**
      $M[l][i] \leftarrow 1$;
      if $qns(l+1, M)$ **then**
        return *true*;
      else $M[l][i] \leftarrow 0$;
  return *false*;

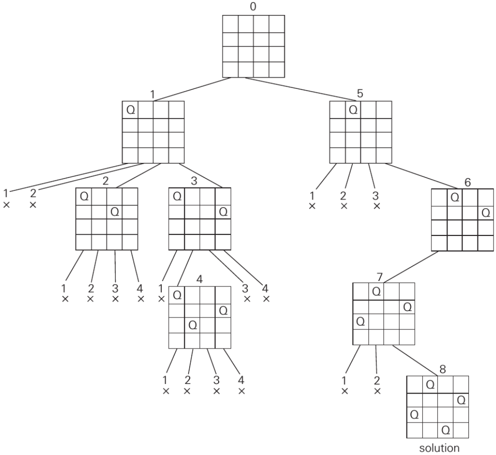
qns(0,M) > qns(1,M) > qns(2,M) > qns(3,M)

Centro de Informática UFPE

Universidade Federal de Pernambuco

[13] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

n-Queens problem[14]



Algorithm: bool qns (int l, int M[0..n-1,0..n-1])

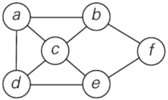
if $l = n$ **then return** *true*;
else
    for $i \leftarrow 0$ **to** $n-1$ **do**
        if *valid*(M, l, i) **then**
            $M[l][i] \leftarrow 1$;
            if *qns*$(l+1, M)$ **then**
                return *true*;
            else $M[l][i] \leftarrow 0$;
    return *false*;

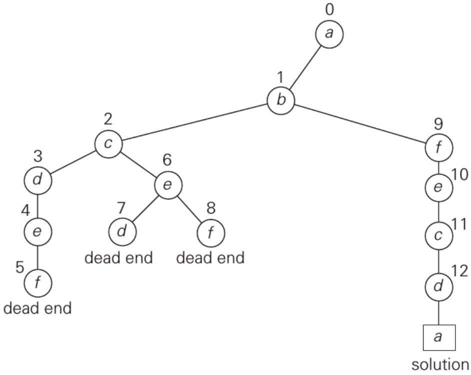
qns(0,M) > qns(1,M) > qns(2,M) > qns(3,M) > qns(4,M) solution found!

[14] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Hamiltonian circuit problem[15]

Start from an arbitrary vertex, move to the next ones until returning to the first one. If not possible, go back and try the next vertex.





Centro de Informática UFPE

Universidade Federal de Pernambuco

[15] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Subset-sum problem

Let $A = \{a_1, \cdots, a_n\}$ be a set of positive integers, find $A' \subseteq A$ such that the sum of the elements in A' is equal to $d \in \mathbb{N}$.

- It is convenient to sort the elements in increasing order

Go back if:

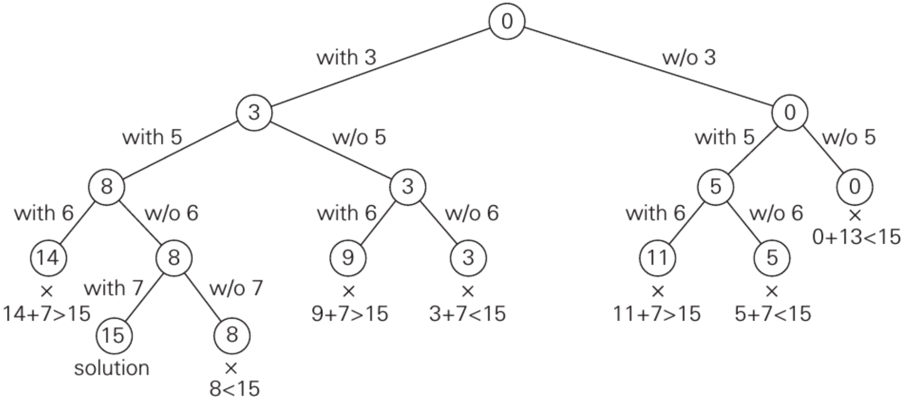
- $s + a_{i+1} > d$ (i.e., the sum s is too large)
- $(s + \sum_{j=i+1}^{n} a_j) < d$ (i.e., the sum s is too small)

Example: $A = \{3, 5, 6, 7\}$ and $d = 15$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Problema da soma dos subconjuntos[16]



[16] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Backtracking

In summary:

- Problems do not become tractable (only some larger instances)
- Time efficiency depends on the problem and the instance

Optimisations:

- Explore symmetry of combinatorial problems
- Rearrange data of a given instance

Other applications:

- Graph colouring
- Knight's tour

Agenda

Centro de Informática UFPE

Universidade Federal de Pernambuco

Branch-and-bound

Optimisation problem: minimise or maximise some objective function

- Feasible solution: satisfies all constraints
- Optimal one: feasible and the best value of the objective function

Compared to backtracking, two additional items:

- For each node of the state-space tree, a bound on the best value of the objective function
- The value of the best solution so far

Typically, expands based on the best-first

- Nodes with bound lower than best solution so far is pruned

Centro de Informática UFPE

Universidade Federal de Pernambuco

Assignment problem[17]

Assign *n* people to *n* jobs so that
the total cost of assignment is as small as possible

$$
\begin{array}{cc}
 & \begin{array}{cccc} \text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} \end{array} \\
C = & \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}
\begin{array}{l} \text{person } a \\ \text{person } b \\ \text{person } c \\ \text{person } d \end{array}
\end{array}
$$

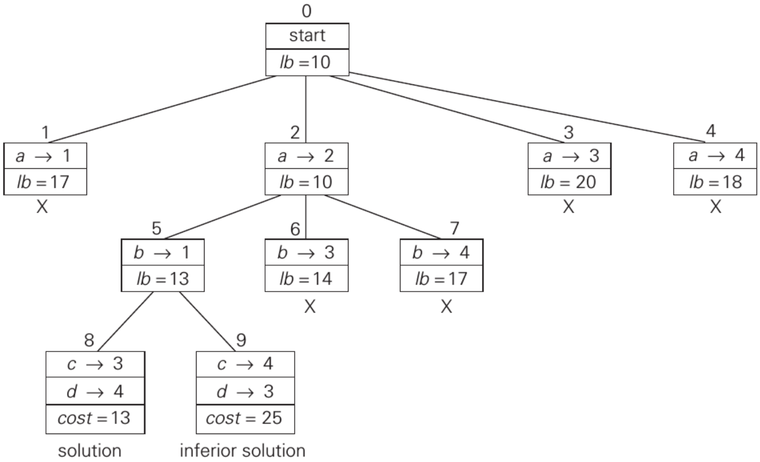
Lower bound: sum of the smallest elements in each row
(not necessarily a feasible solution)

Centro de Informática UFPE

Universidade Federal de Pernambuco

[17] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Assignment problem[18]



There is a polynomial (and deterministic) algorithm: Hungarian method

[18] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Travelling salesman person (TSP-OPT)

Find the shortest Hamiltonian circuit for a graph G

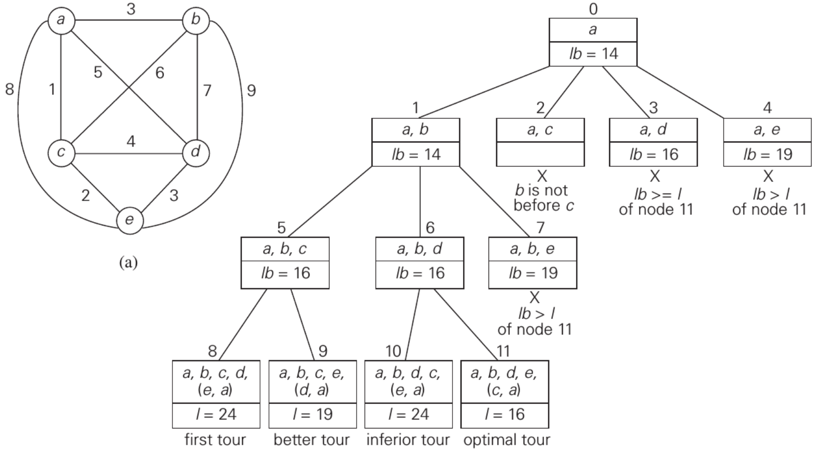
Lower bound: for each city $1 \leq i \leq n$, find the sum s_i of the distances to the two nearest cities, compute $s = s_1 + \cdots + s_n$, then compute the lower bound as $\lceil s/2 \rceil$

- Adjust the lower bound considering the already selected edges

Simplifications (without loss of generality)

- Consider only tours that start at an arbitrary node
- After visiting $n - 1$ cities, it needs to visit the remaining one and go back to the starting point
- If the graph is undirected, ignore symmetric tours

Travelling salesman person (TSP-OPT)[19]



19 Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Branch-and-bound

Best-first is not always the best option

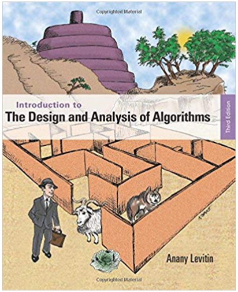
- AI: other heuristics

Challenge of defining a good lower/higher bound

- It needs to be easy to compute
- It cannot be too simple (effect on pruning)

Agenda

Bibliography



Chapter 12 (pp. 423–440)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3rd edition. Pearson. 2011.

Chapter 17 (pp. 552–555)
Clifford Shaffer.
Data Structures and Algorithm Analysis. Dover, 2013.

Centro de Informática UFPE

Universidade Federal de Pernambuco

BACKTRACKING

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Centro de Informática UFPE

Universidade Federal de Pernambuco