Laboratório de Programação Avançada 2018/19
Week 7 – Branch-and-Bound

· U C ·

Universidade de Coimbra

$d = 3$

Find a minimum spanning tree where each node has at most degree d.

$d = 3$

Find a minimum spanning tree where each node has at most degree d.

The length of the minimum spanning tree is less than or equal to the length of the degree-constrained minimum spanning tree.

Let G be a network, $G = (V, E)$, where each edge $\{i, j\} \in E$ has a length $\ell(i, j)$ and let d be the maximum degree.

Let $deg(i)$ denote the degree of node i. The goal is to find a spanning tree x in G that minimizes the total length

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

such that $deg(i) \leq d$, for all $i \in V$

Note: If we ignore the degree constraint, we have the minimum spanning tree problem in G. That tree is shorter than or equal to the optimal tree of the degree-constrained problem.

Function $MST(x, p)$
  if $\ell(x) \geq \ell(x^*)$ **then** {rejection test}
    return
  if $\ell(x) < \ell(x^*)$ **and** $p = n$ **then** {base case}
    $x^* = x$
    return
  for each $\{i, j\} \in E$ **do**
    if $visit[i] = \texttt{false}$ **and** $visit[j] = \texttt{true}$ **then** {rejection test}
      $visit[i] = \texttt{true}$ {mark as visited}
      $MST(x \cup \{i, j\}, p + 1)$ {recursive step}
      $visit[i] = \texttt{false}$ {mark as unvisited}

- Find the minimum spanning tree of G.
- At each step, test the insertion of edge $\{i, j\}$ to the tree.

There is a faster way of finding the minimum spanning tree, which will be discussed in the week about graphs.

```
Function dMST(x, p)
  if ℓ(x) ≥ ℓ(x*) then                                    {rejection test}
    return
  if ℓ(x) < ℓ(x*) and p = n then                          {base case}
    x* = x
    return
  for each {i, j} ∈ E do
    if visit[i] = false and visit[j] = true then          {rejection test}
      if deg[j] < d then
        visit[i] = true                                   {mark as visited}
        deg[j]++, deg[i]++
        dMST(x ∪ {i, j}, p + 1)                           {recursive step}
        visit[i] = false                                  {mark as unvisited}
        deg[j]- -, deg[i]- -
```

- Find the degree-constraint minimum spanning tree of G.
- Array deg counts the degree of each node in G.

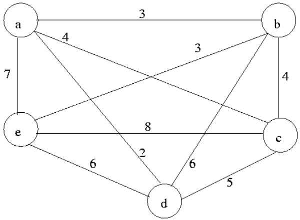
Function $dMST(x, p)$
  if $\ell(x) \geq \ell(x^*)$ **then** {rejection test}
    return
  if $g(x) \geq \ell(x^*)$ **then** {bounding test}
    return
  if $\ell(x) < \ell(x^*)$ **and** $p = n$ **then** {base case}
    $x^* = x$
    return
  for each $\{i, j\} \in E$ **do**
    if $visit[i] = \texttt{false}$ **and** $visit[j] = \texttt{true}$ **then** {rejection test}
      if $deg[j] < d$ **then**
        $visit[i] = \texttt{true}$ {mark as visited}
        $deg[j]{+}{+}$, $deg[i]{+}{+}$
        $dMST(x \cup \{i, j\}, p + 1)$ {recursive step}
        $visit[i] = \texttt{false}$ {mark as unvisited}
        $deg[j]$- -, $deg[i]$- -

- Function $g(x)$ is $\ell(x)$ plus the minimum spanning tree for the remaining edges (p.e., using Kruskal algorithm – see lecture about graphs in the following weeks).

- Then, $g(x)$ is less or equal to the minimum that can be achieved with the current partial tree x.

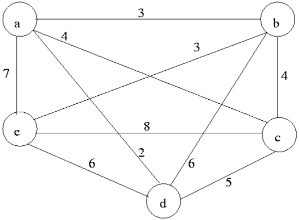
Let G be a network, $G = (V, E)$, where each arc $(i, j) \in A$ has a distance $d(i, j)$. The goal is to find a shortest Hamiltonian circuit π (a permutation of the nodes) that minimizes the total length

$$d(\pi) = \sum_{i=1}^{n-1} d(\pi(i), \pi(i+1)) + d(\pi(n), \pi(1))$$



Bounding functions

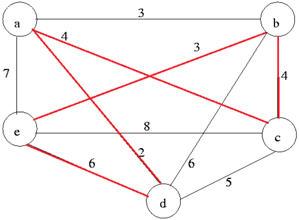
- The value of the minimum spanning tree of the graph, since the optimal TSP tour without an edge is a spanning tree.



A graph

Bounding functions

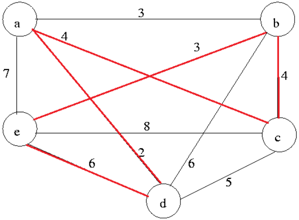
- The value of the minimum spanning tree of the graph, since the optimal TSP tour without an edge is a spanning tree.



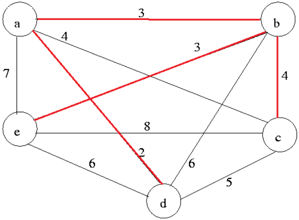
Optimal tour: 19

Bounding functions

- The value of the minimum spanning tree of the graph, since the optimal TSP tour without an edge is a spanning tree.



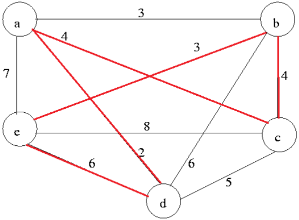
Optimal tour: 19

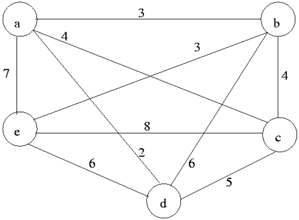


Lower bound: 12

Bounding functions

- Sum up the distances of the two closest nodes to each node *i* and divide the total by two.



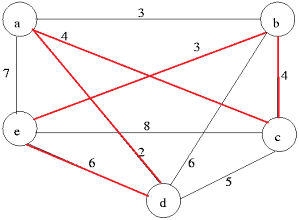


Optimal tour: 19

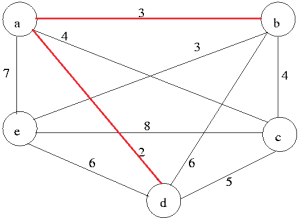
Lower bound:

Bounding functions

- Sum up the distances of the two closest nodes to each node *i* and divide the total by two.



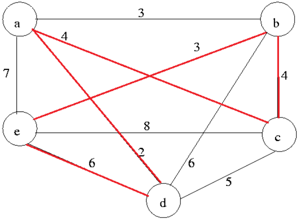
Optimal tour: 19



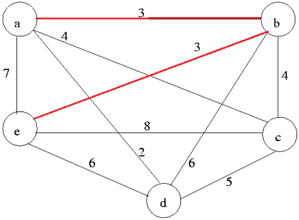
Lower bound: 2.5

Bounding functions

- Sum up the distances of the two closest nodes to each node *i* and divide the total by two.



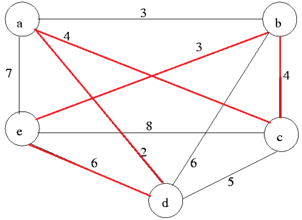
Optimal tour: 19



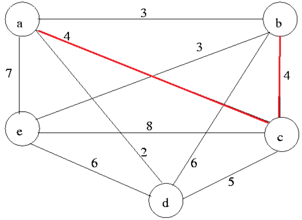
Lower bound: 5.5

Bounding functions

- Sum up the distances of the two closest nodes to each node i and divide the total by two.



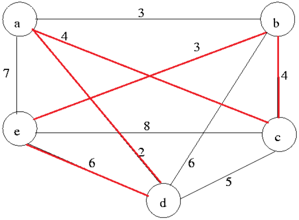
Optimal tour: 19



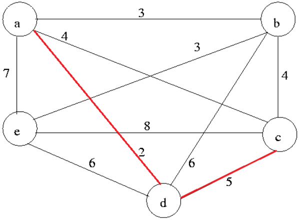
Lower bound: 9.5

Bounding functions

- Sum up the distances of the two closest nodes to each node i and divide the total by two.



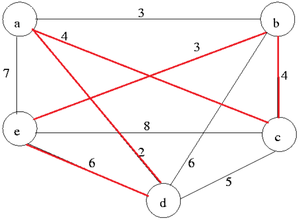
Optimal tour: 19



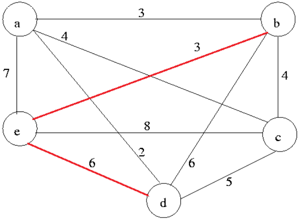
Lower bound: 13

Bounding functions

- Sum up the distances of the two closest nodes to each node *i* and divide the total by two.



Optimal tour: 19



Lower bound: 17.5