UNIVERSITY OF COPENHAGEN

Branch-and-bound

Advanced Algorithms 2012

Rasmus Fonseca

DIKU - 22/5 2012
Dias 1

How would you solve an NP-complete problem?

Significant other: I want to take a picture of every pretty monument in this city tomorrow!

You: But then we have to walk all day.

Significant other: But I really want to. Cant you just measure the distances on the map and then solve the traveling salesperson problem tonight?

You: Fine .. Why else did I take a CS degree.

How would you solve an NP-complete problem?

- **Full enumeration**: Test all possible solutions and check which one is best. *(For example: Brute force, branch-and-bound, branch-and-cut, branch-and-price …)*

- **Approximation algorithms**: Might not find the optimal solution, but guarantees that your solution is not worse than a certain factor. *(You will see plenty examples next lecture)*

- **Meta-heuristic**: If you are ok with a good solution that does not give you any guarantee. *(For example: Hill-climbing local search, tabu-search, simulated annealing, genetic algorithms, ant colony optimization …)*

- **Try not to think about it**: Restate the problem so it becomes solvable *(For example: Expected time, parameterized complexity, change to a physics degree …)*

Brute force

Consider a set, *S*, of feasible solutions to a problem. Brute force requires two tools:

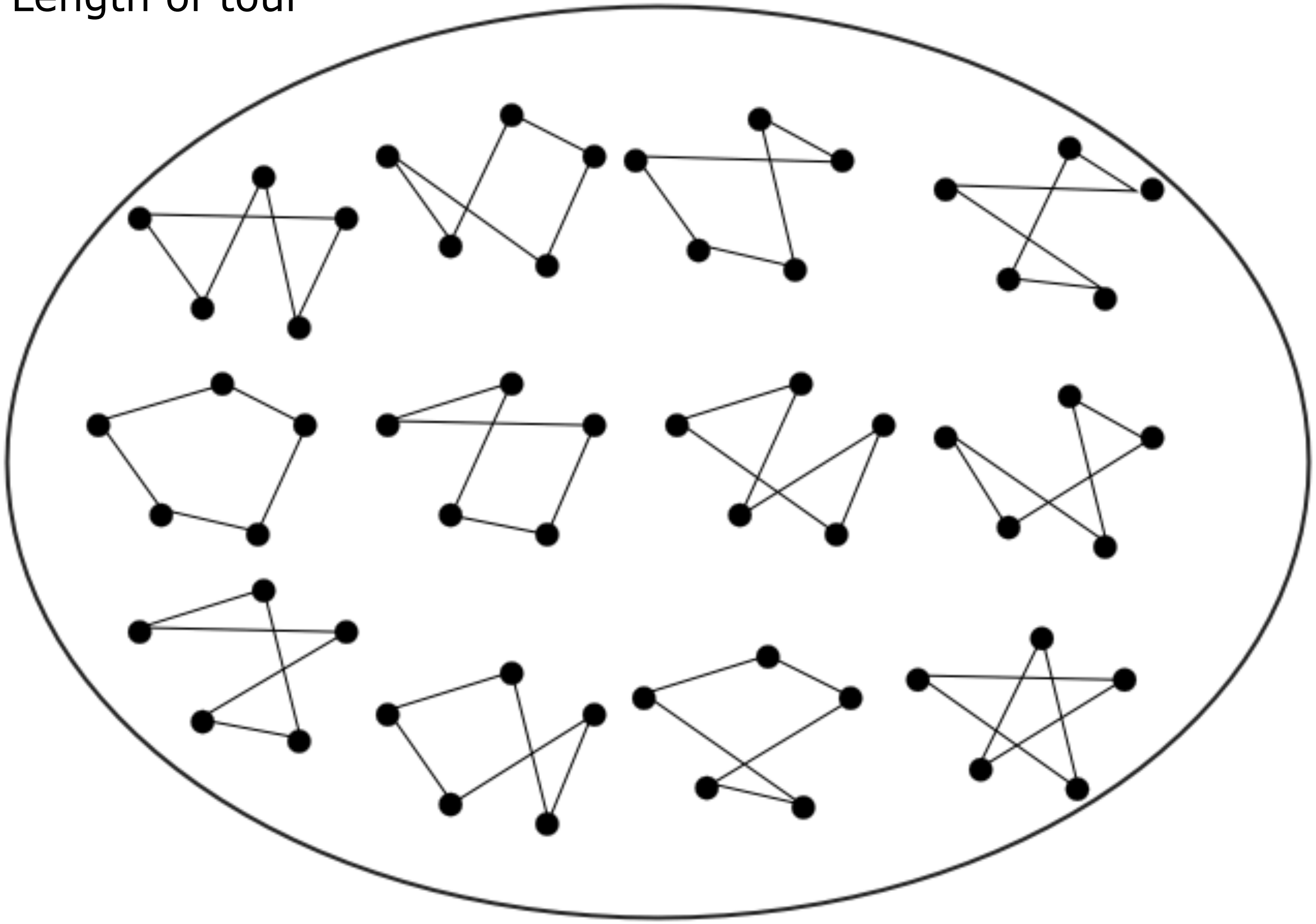
- A method to split *S* into strictly smaller sets: **`branch(S)`**
- A method to evaluate the objective of each solution: **`f(s)`**

```
Brute-force-minimize(S)
Q = queue containing only S
best = null
while(Q is not empty)
   Si = pop an element from Q
   if ( Si contains 1 element: s )
     if ( f(s) better than f(best) )
       best = s
   else
      add branch(Si) to Q
```

Brute force

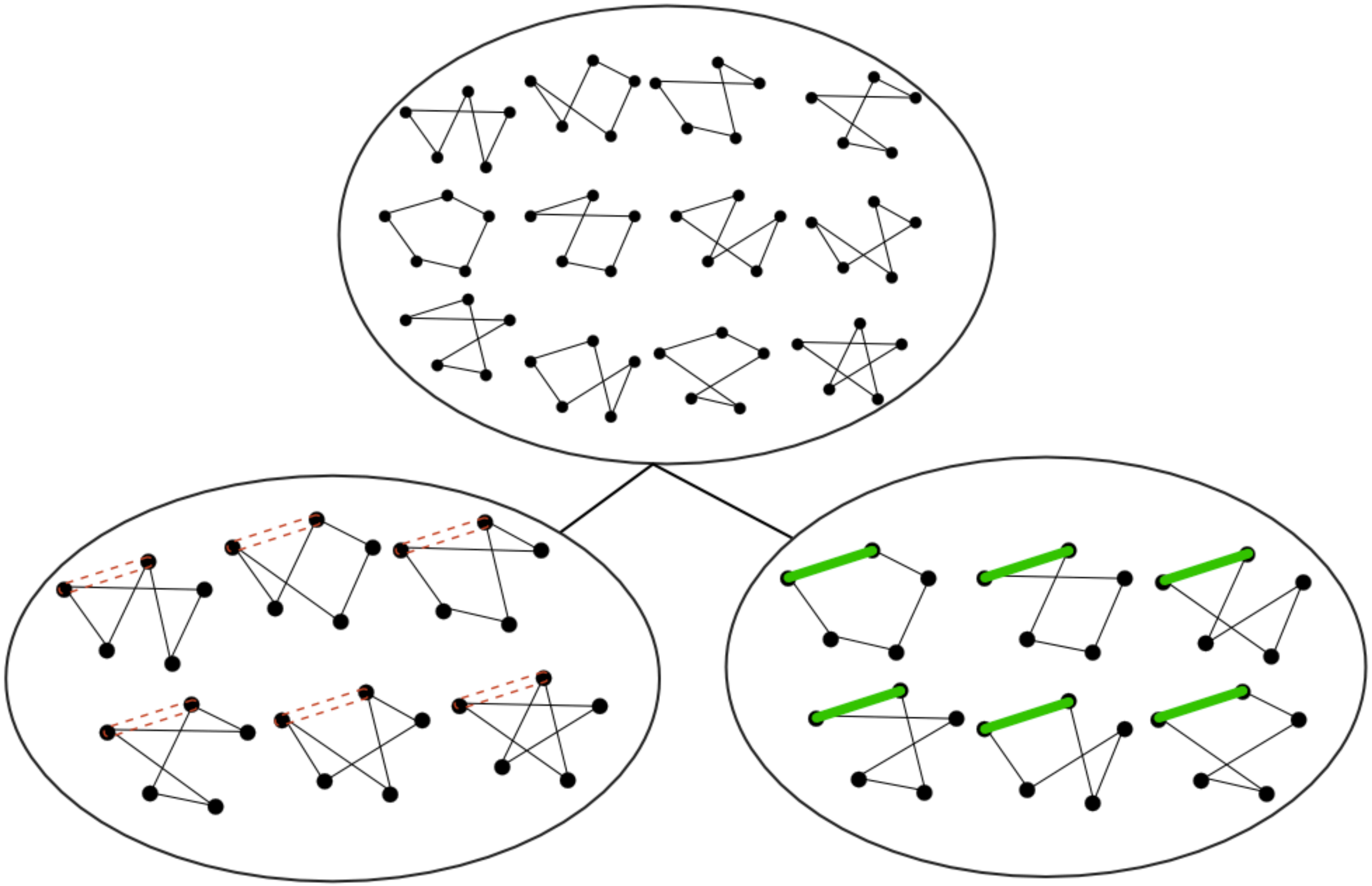
`best = null`

`branch(S)`: Include or exclude a single edge
`f(s)`: Length of tour



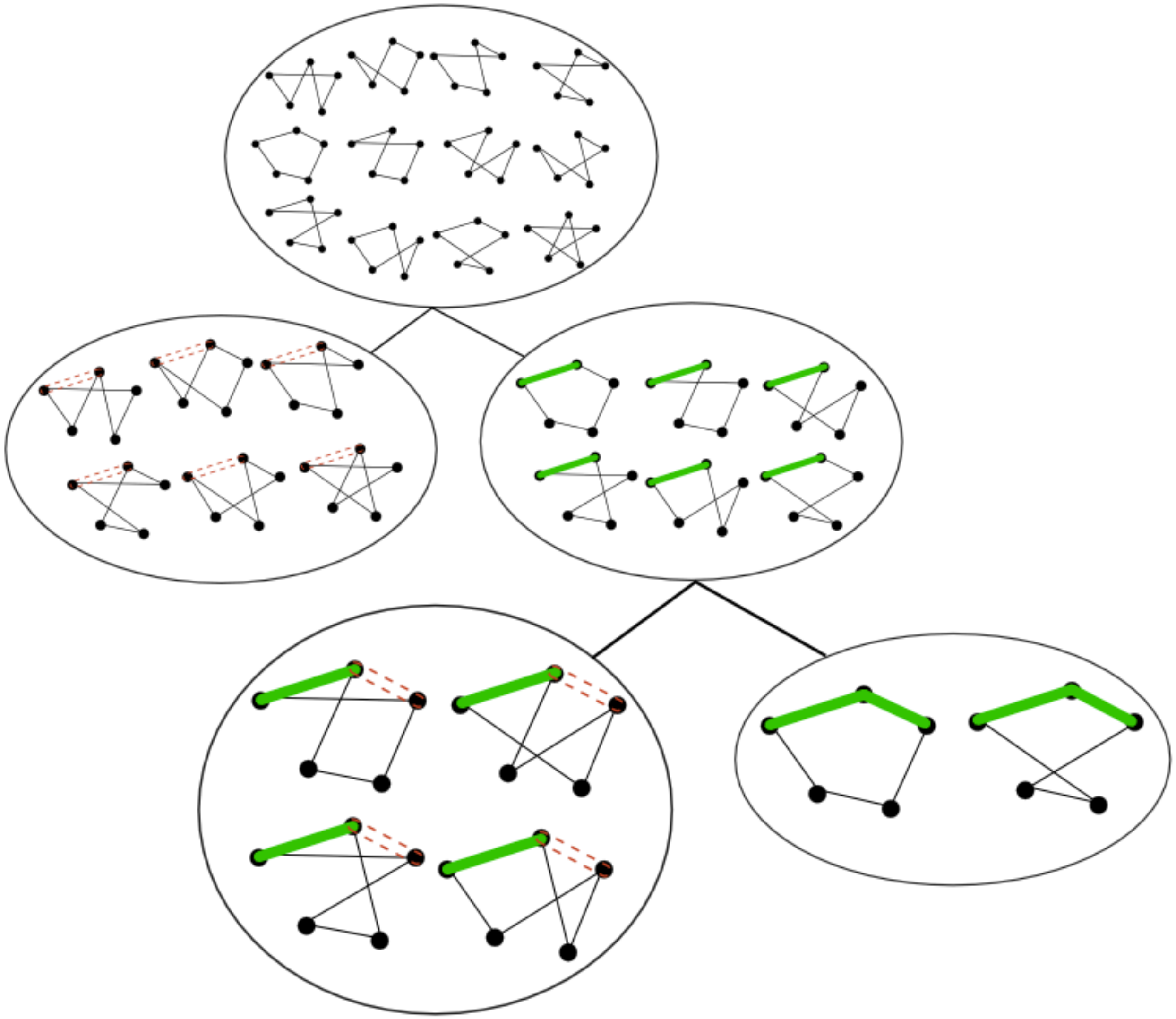
Brute force

`best = null`



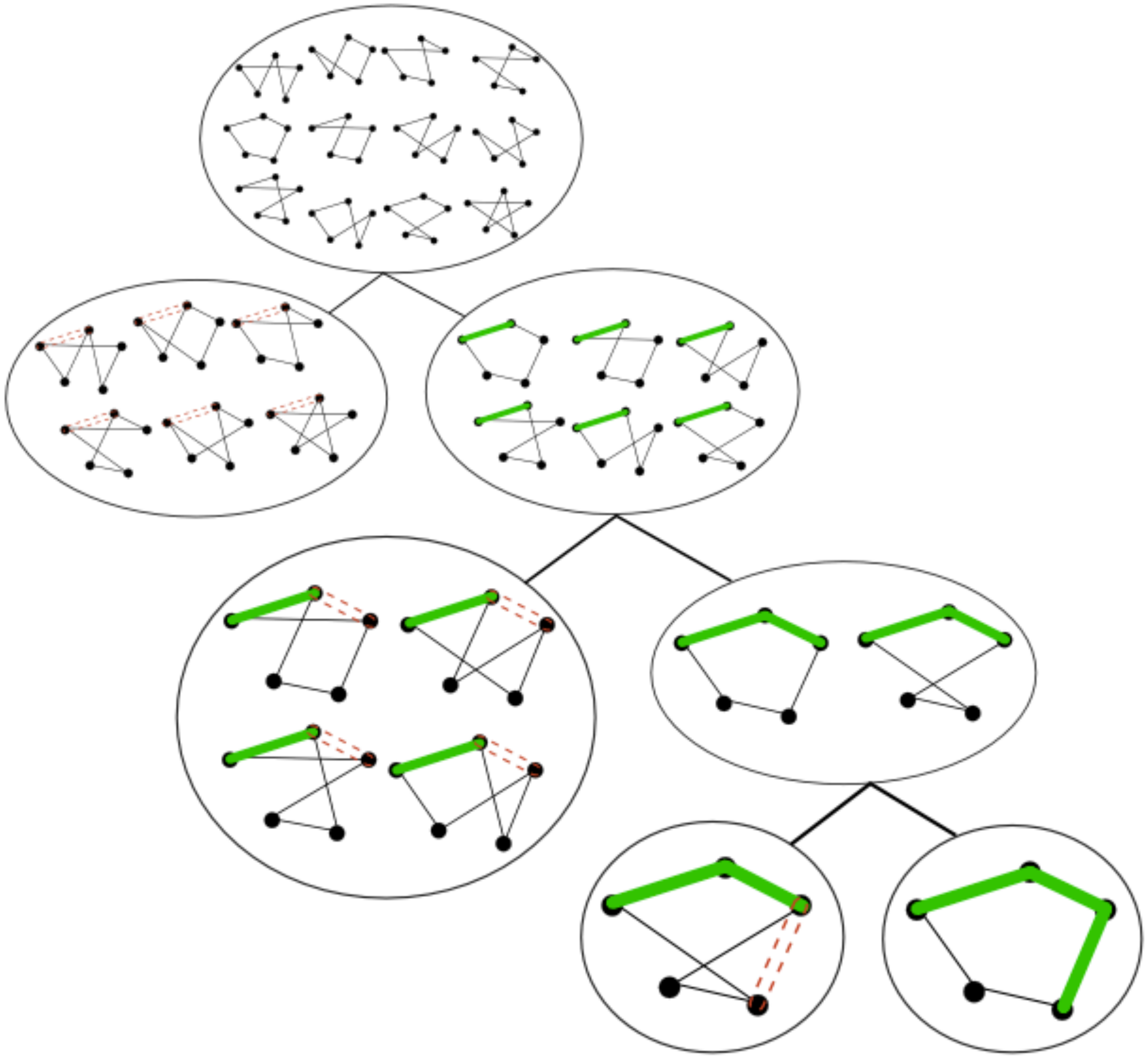
Brute force

`best = null`



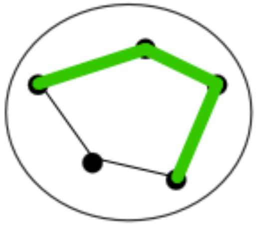
Brute force

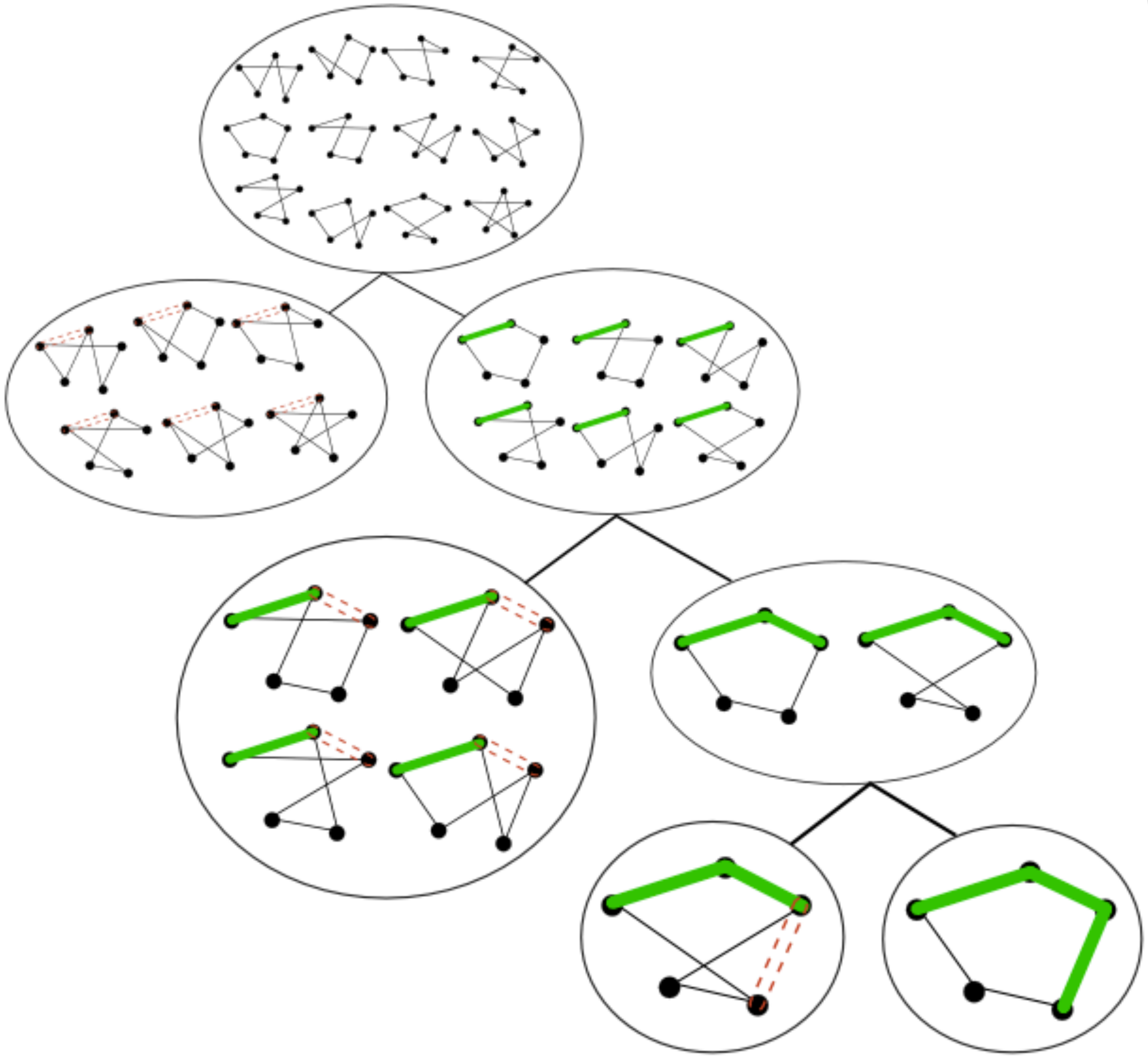
`best = null`



Brute force

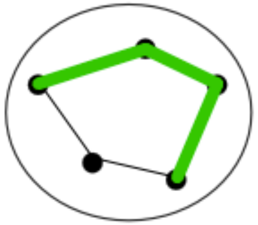
best =

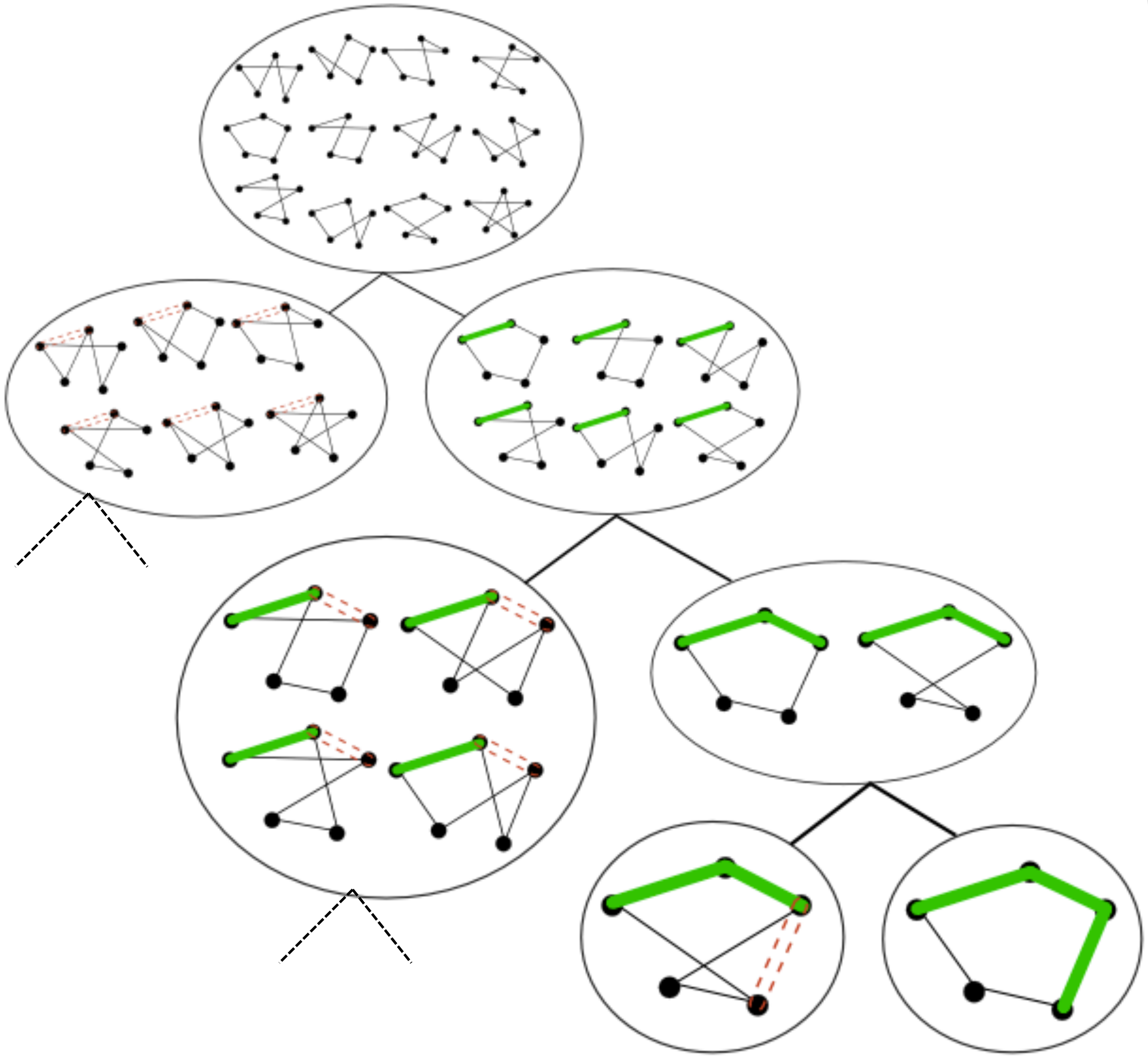




Brute force

best =





Brute force

Consider a set, *S*, of feasible solutions to a problem. Brute force requires two tools:

- A method to split *S* into strictly smaller sets: **`branch(S)`**
- A method to evaluate the objective of each solution: **`f(s)`**

```
Brute-force-minimize(S)
   Q = queue containing only S
   best = null
   while(Q is not empty)
      Si = pop an element from Q
      if ( Si contains 1 element: s )
         if ( f(s) better than f(best) )
            best = s
      else
         add branch(Si) to Q
```

Branch and bound

Consider a set, *S*, of feasible solutions to a problem. Branch and bound requires three tools:

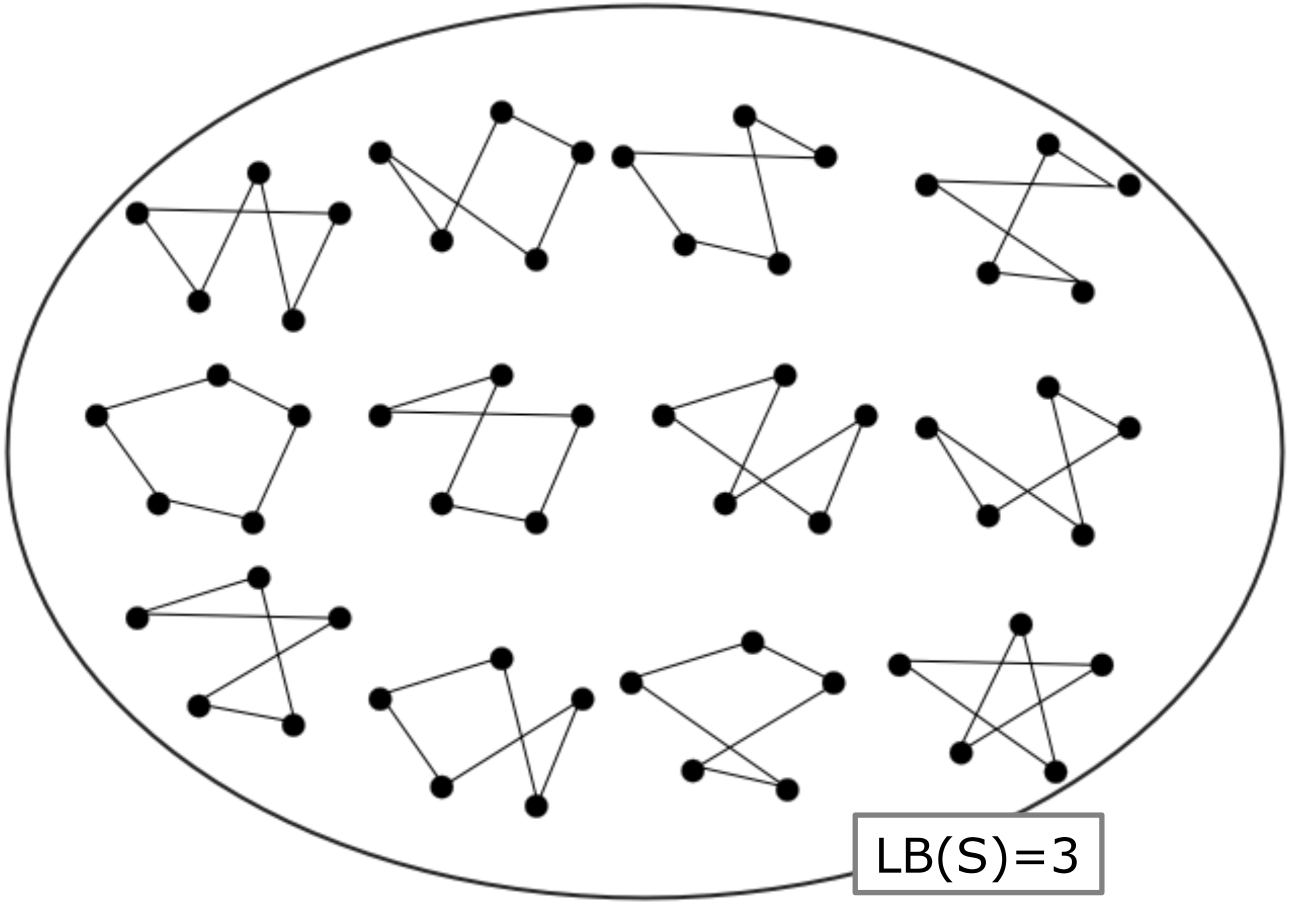
- A method to split *S* into strictly smaller sets: **`branch(S)`**
- A method to evaluate the objective of each solution: **`f(s)`**
- A function **`lowerBound(S)`** such that for any *s* in *S*: **`lowerBound(S) ≤ f(s)`**

```
Branch-and-bound-minimize(S)
   Q = queue containing only S
   best = null
   while(Q is not empty)
      Si = pop an element from Q
      if ( Si contains 1 element: s )
         if ( f(s) better than f(best) )
            best = s
      else
         if ( lowerBound(Si) < f(best) )
            add branch(Si) to Q
```

Branch-and-bound

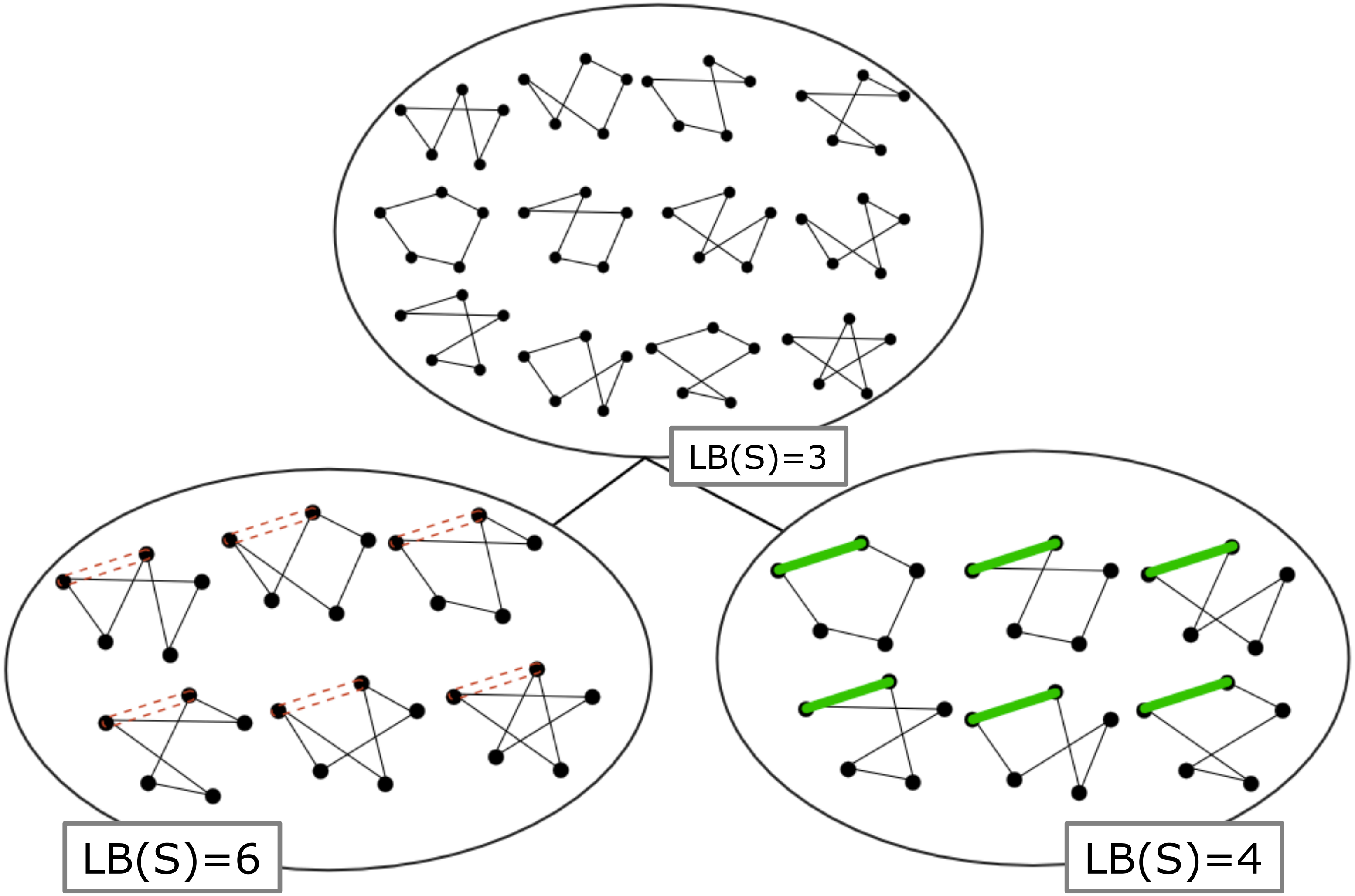
best = null

branch(S): Include or exclude a single edge
f(s): Length of tour
Queue popping-strategy: Best-first



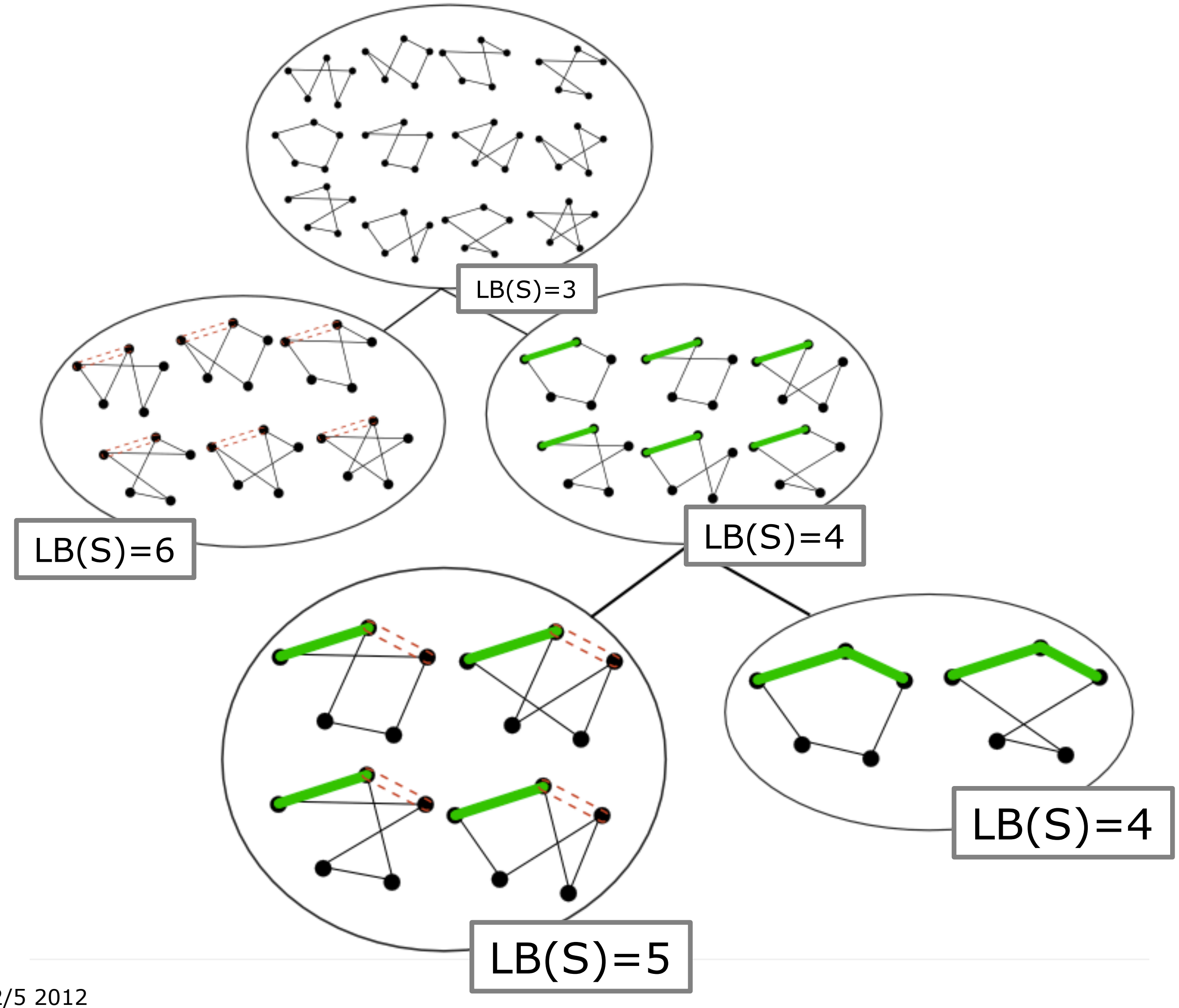
Branch-and-bound

`best = null`



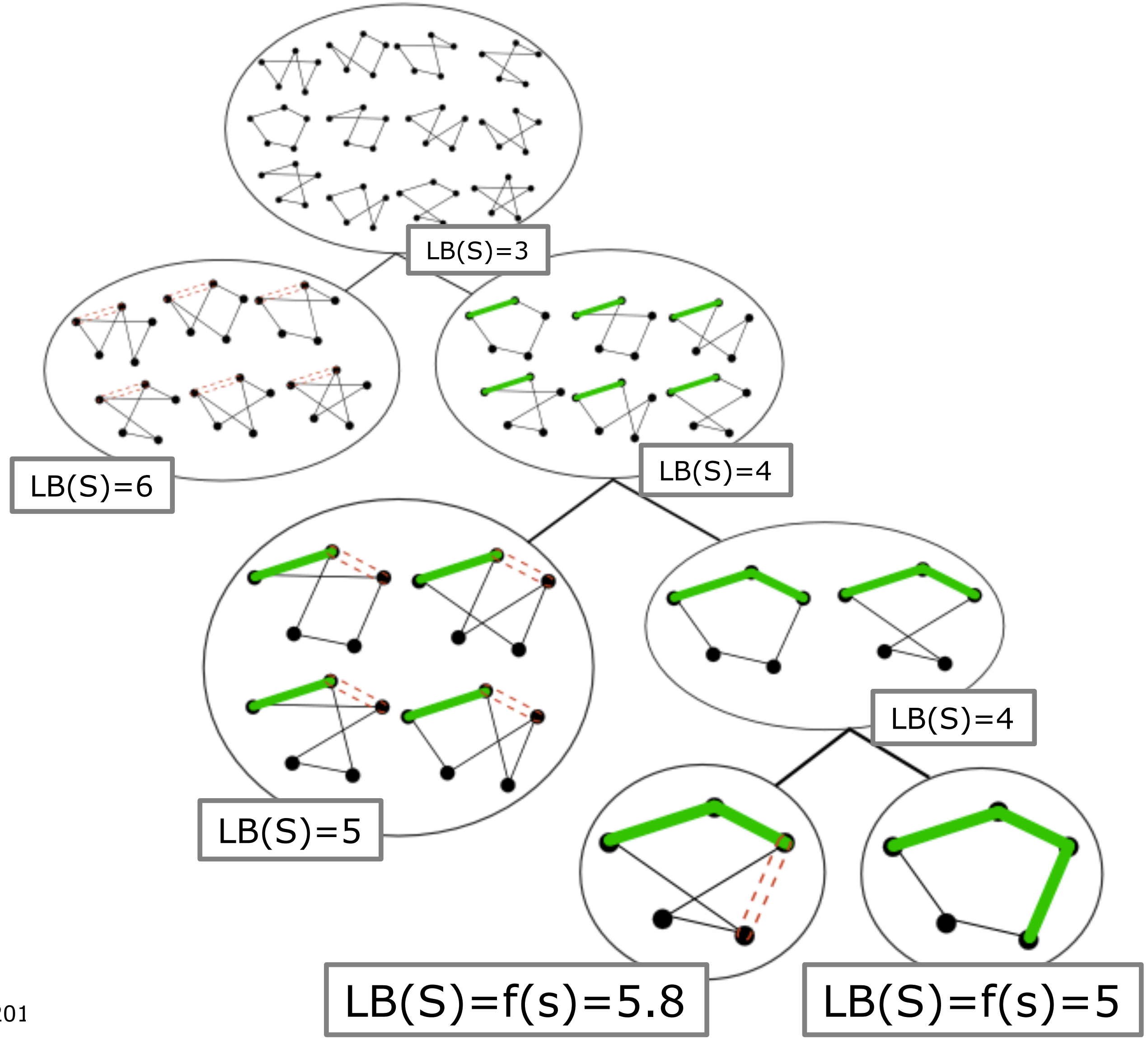
Branch-and-bound

best = null



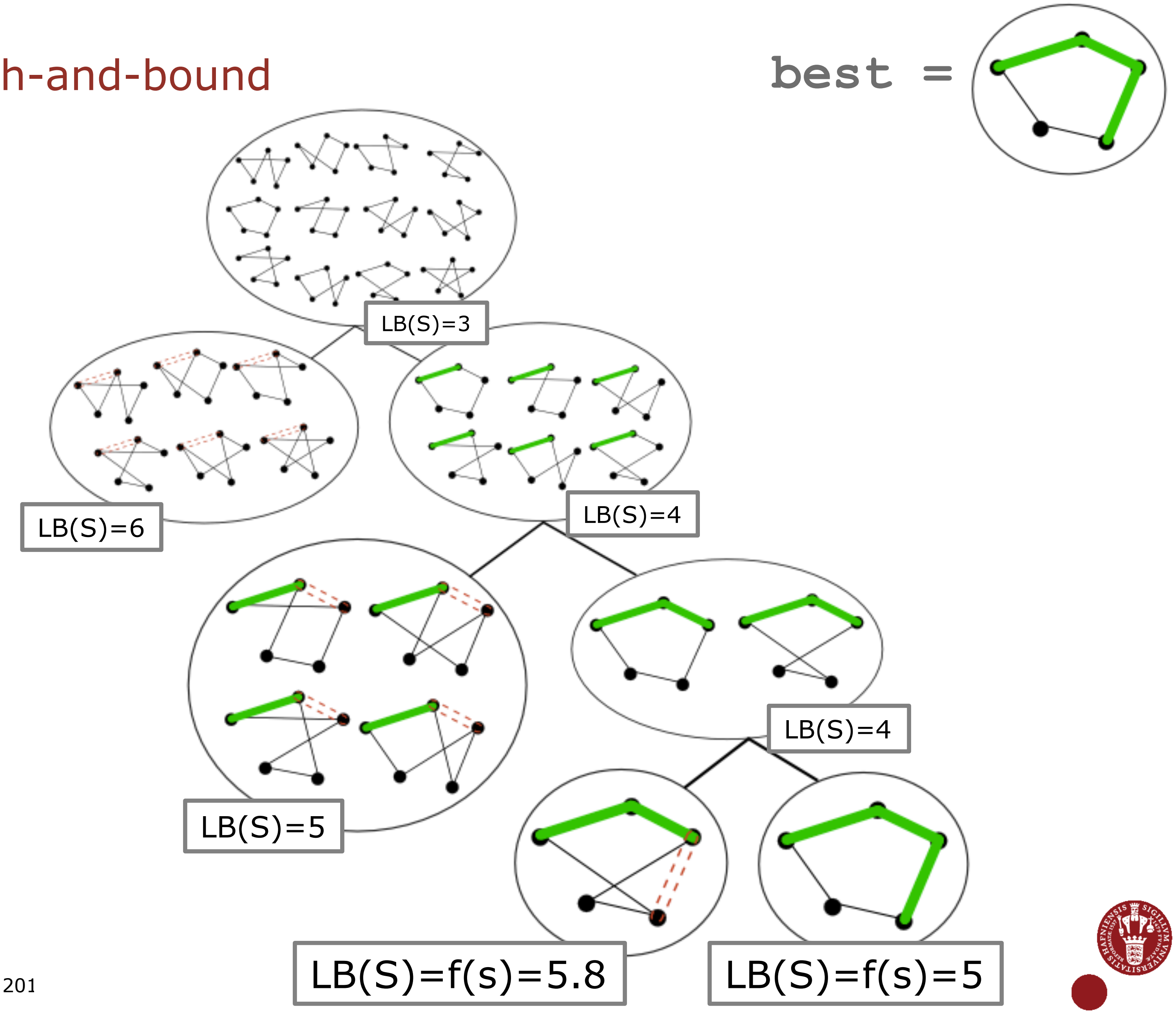
Branch-and-bound

`best = null`

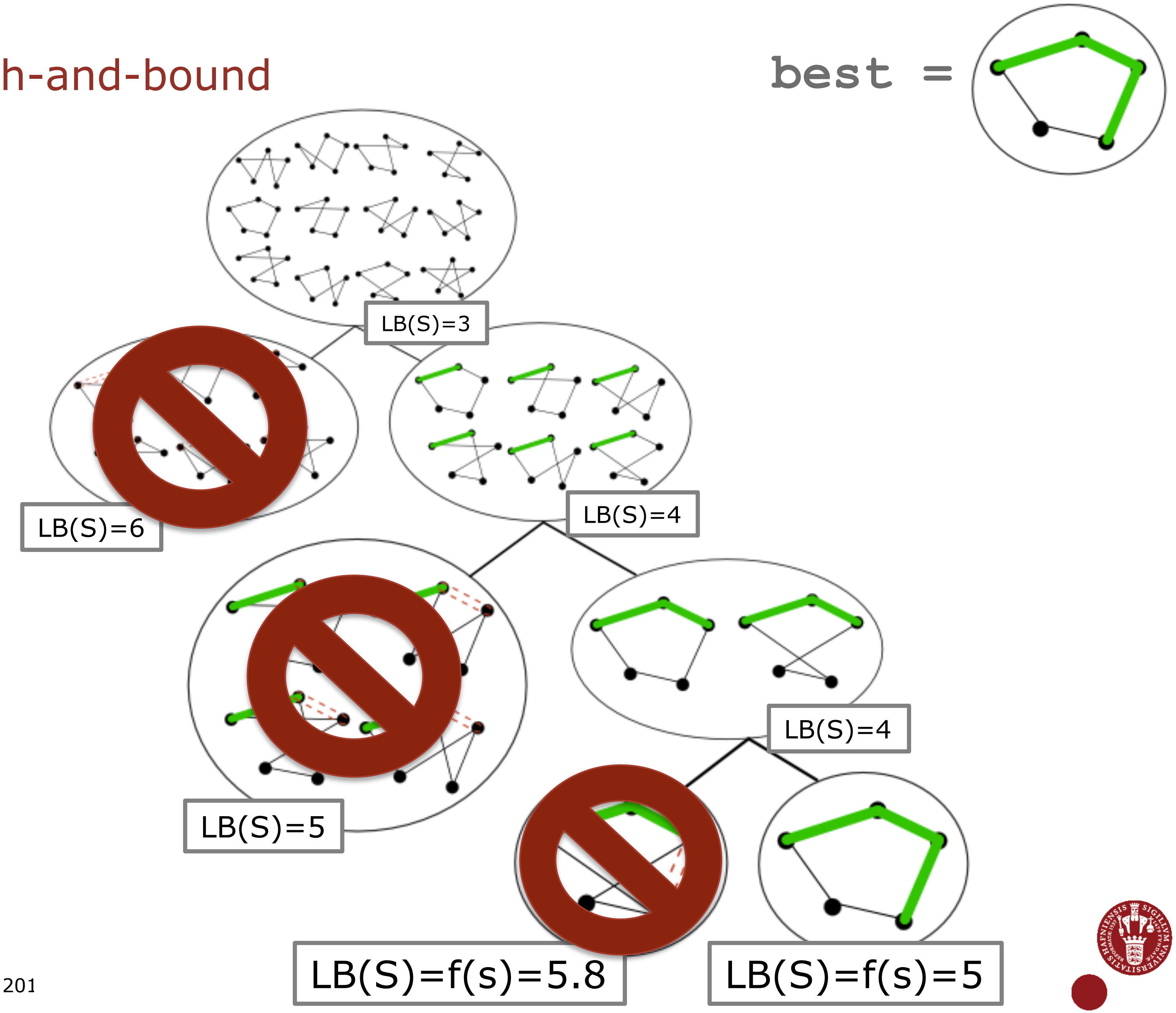


Branch-and-bound

best =



Branch-and-bound



Branch-and-bound for ILP

- What do we branch on?

- How do we branch?

- What is **`f(s)`**?

- What can we use as **`lowerBound(S)`**?

$$
\begin{array}{lll}
\min & \sum_{i=1}^{m} c_i x_i & \\
\text{st.} & \sum_i a_{ij} x_i \geq b_j & \forall j = 1 \ldots n \\
 & x_i \in \{0, 1\} & \forall i = 1 \ldots m
\end{array}
$$

Branch-and-bound for ILP

- What do we branch on?
  - Node with fewest fixed decision variables (breadth-first)
  - Node with most fixed decision variables (depth-first)
  - Node with lowest lower-bound (best-first)
- How do we branch?
  - Branch on decision variable (fractional variable x_{ij} in relaxed solution).
- What is `f(s)`?
  - Objective value
- What can we use as `lowerBound(S)`?
  - Relax integrality of decision variables and use objective value of LP solution

$$
\begin{array}{lll}
\min & \sum_{i=1}^{m} c_i x_i & \\
\text{st.} & \sum_i a_{ij} x_i \geq b_j & \forall j = 1 \ldots n \\
 & x_i \in \{0,1\} & \forall i = 1 \ldots m
\end{array}
$$

Lower-bounds

- The above method can be applied to any ILP formulation.
- All NP-complete problems you have seen have ILP formulations
  - Examples: Knapsack, subset-sum, vertex-cover, TSP
- Ill finish with an example how to find lower-bound to TSP without using ILP.

Integer programming examples

Knapsack problem: Given a knapsack with capacity *C*, and *n* items each with profit p_i and weight w_i, find the most profitable selection of items to put in knapsack.

x_i: indicates if object *i* is brought

$$
\begin{array}{ll}
\max & \sum_{i=1}^{n} p_i x_i \\
\text{st.} & \sum_{i=1}^{n} w_i x_i \leq C \\
 & x_i \in \{0,1\}
\end{array}
$$

Subset sum problem: As above, but with $p_i = w_i$.

Integer programming examples

Minimum vertex cover: Given an undirected graph and a cost c_i for every vertex i, find the smallest weighted subset of vertices such that every edge has an endpoint in this subset.

x_i: indicates if vertex i is in this subset

$$\begin{array}{lll} \min & \sum_{i \in V} c_i x_i & \\ \text{st.} & x_u + x_v \geq 1 & \forall (u,v) \in E \\ & x_i \in \{0,1\} & \forall i \in V \end{array}$$

Integer programming examples

Graph coloring: Given an undirected graph, determine the fewest vertex-colors that are needed such that no two adjacent vertices have the same color.

C: set of n colors

w_c: color c is used

x_{ic}: vertex i has color c

$$
\begin{array}{lll}
\min & \sum_{c=1}^{n} w_c & \\
\text{st.} & \sum_{c=1}^{n} x_{ic} = 1 & \forall i \in V \\
& x_{uc} + x_{vc} \leq 1 & \forall (u,v) \in E, c \in C \\
& x_{ic} \leq w_c & \forall i \in V, c \in C \\
& x_i \in \{0,1\} &
\end{array}
$$

TSP as Integer Linear Program

- $x_{ij} = \begin{cases} 1 & \text{if edge } (i,j) \text{ is included} \\ 0 & \text{otherwise} \end{cases}$
- $d_{ij} = \text{distance from vertex } i \text{ to } j$
- $$\begin{aligned} \min \textstyle\sum_{i,j} d_{ij} x_{ij} & \\ \textstyle\sum_{j} x_{ij} = 1 & \quad \forall i \in V \\ \textstyle\sum_{i} x_{ij} = 1 & \quad \forall j \in V \\ \textstyle\sum_{i,j \in S} x_{ij} \leq |S| - 1 & \quad \forall S \subset V, |S| > 1 \\ x_{ij} \in \{0,1\} & \end{aligned}$$
- Symmetric TSP: $d_{ij} = d_{ji}$
- Metric TSP: $d_{ik} \leq d_{ij} + d_{jk}$

TSP as Integer Linear Program

$$
\begin{aligned}
\min \textstyle\sum_{i,j} d_{ij} x_{ij} & \\
\textstyle\sum_{j} x_{ij} = 1 & \quad \forall i \in V \\
\textstyle\sum_{i} x_{ij} = 1 & \quad \forall j \in V \\
\textstyle\sum_{i,j \in S} x_{ij} \leq |S| - 1 & \quad \forall S \subset V, |S| > 1 \\
x_{ij} \in \{0, 1\} &
\end{aligned}
$$

- Dantzig, Fulkerson and Johnson (1954) Solutions of a large scale travelling salesman problem, Ops. Res., 2, 393-410
- n cities
- 2^n+2n-2 constraints
- n(n-1) variables

TSP as Mixed Integer Linear Program

$$
\begin{aligned}
\min \textstyle\sum_{i,j} d_{ij} x_{ij} & \\
\textstyle\sum_j x_{ij} = 1 & \qquad \forall i \in V \\
\textstyle\sum_i x_{ij} = 1 & \qquad \forall j \in V \\
u_i - u_j + |V| \cdot x_{ij} \leq |V| - 1 & \qquad \forall i, j \in V - \{1\} \\
x_{ij} \in \{0,1\} \quad u_i \geq 0 & \qquad \forall i, j \in V
\end{aligned}
$$

- Miller, Tucker and Zemlin (1960) Integer programming formulation of travelling salesman problems, J. ACM, 3, 326-329.
- n cities
- n^2-n+2 constraints
- n(n-1) 0-1 variables, (n-1) continuous

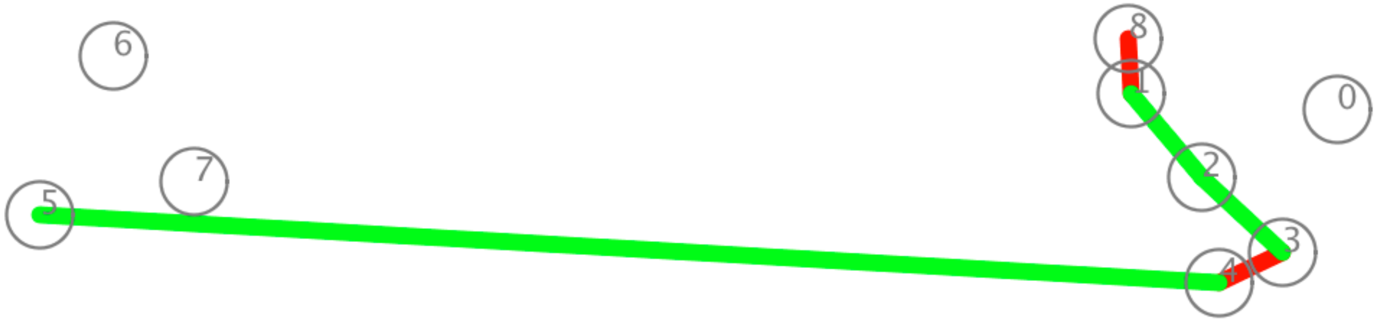
- This representation (MTZ) might be more practical than the previous one by DFJ.

MST lower-bound

- A minimum spanning tree is a lower bound on the optimal length of a TSP-tour.
    - Proof: Remove one edge from the TSP-tour, then the remaining path is a spanning tree. The minimum spanning tree will trivially have less or equal length.

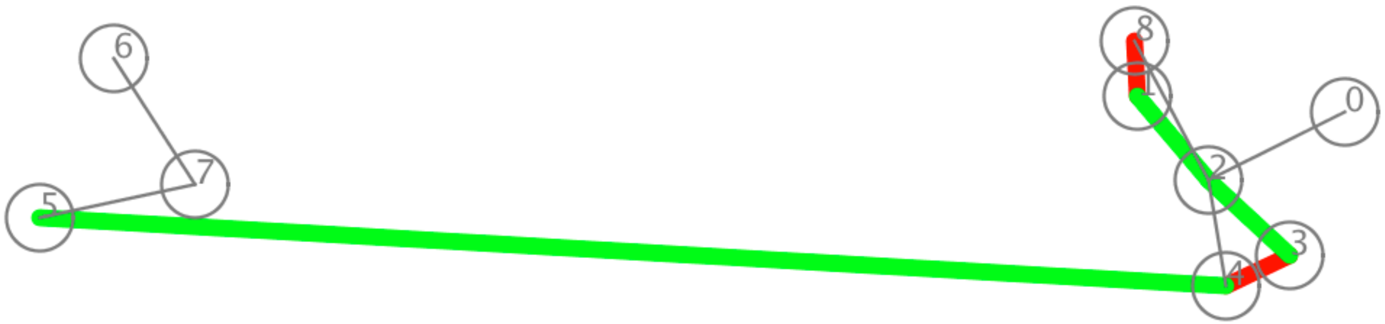
MST lower-bound

- Given a partial solution to TSP: Determine a lower-bound.
  - Contract included edges (green)
  - Disregard excluded edges (red)
  - Find minimum spanning tree



MST lower-bound

- Given a partial solution to TSP: Determine a lower-bound.
  - Contract included edges (green)
  - Disregard excluded edges (red)
  - Find minimum spanning tree (gray)

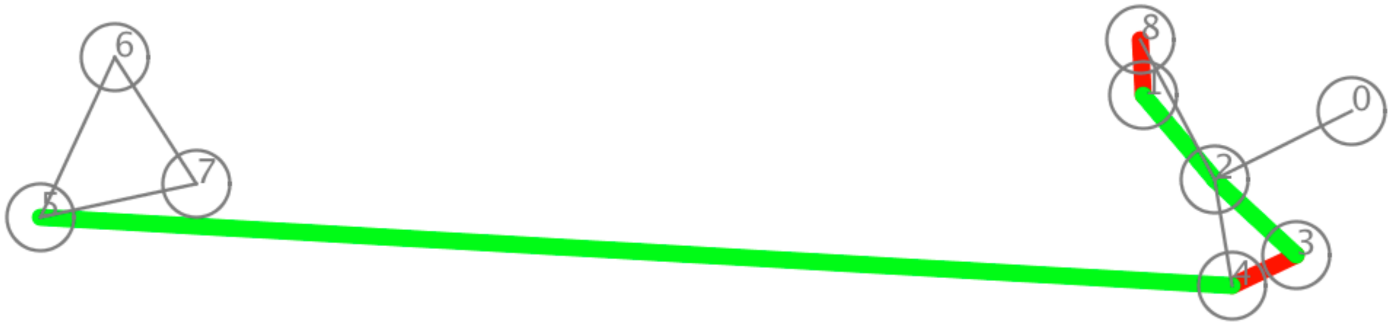


- Add green and gray edges to get a lower-bound

1-tree lower-bound

- A **1-tree** is a subgraph consisting of two edges adjacent to node 1, plus the edges of a tree on nodes {2,…,n}
  - Wolsey L.A. Integer Programming (1998) Wiley-Interscience publication

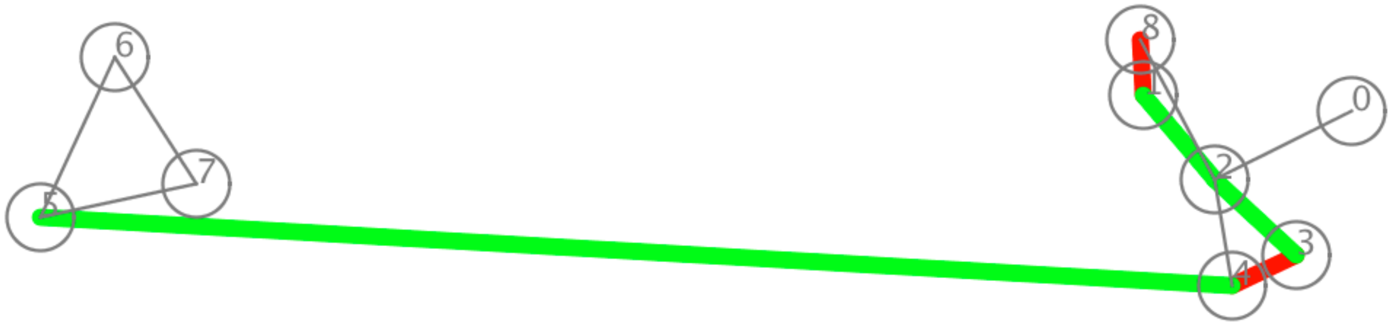
- The smallest 1-tree is a lower-bound on the TSP-tour (proof in assignment)



- Choose any node *i*, find a MST on the remaining nodes and connect *i* to the remaining using the two smallest edges.

1-tree lower-bound

- A **1-tree** is a subgraph consisting of two edges adjacent to node 1, plus the edges of a tree on nodes {2,…,n}
    - Wolsey L.A. Integer Programming (1998) Wiley-Interscience publication



- Actually the 1-tree is a lagrangian relaxation of the sub-tour elimination constraint in the ILP (see Wolsey for details)