Branch and bound - Approximations
Advanced Algorithms

Master CPS2/DSC/MLDM

Amaury Habrard

amaury.habrard@univ-st-etienne.fr

Laboratoire Hubert Curien, UMR CNRS 5516
Université Jean Monnet Saint-Étienne
amaury.habrard@univ-st-etienne.fr

Semester 1

Question 1 **Task Assignment**

Q1 cost and search tree

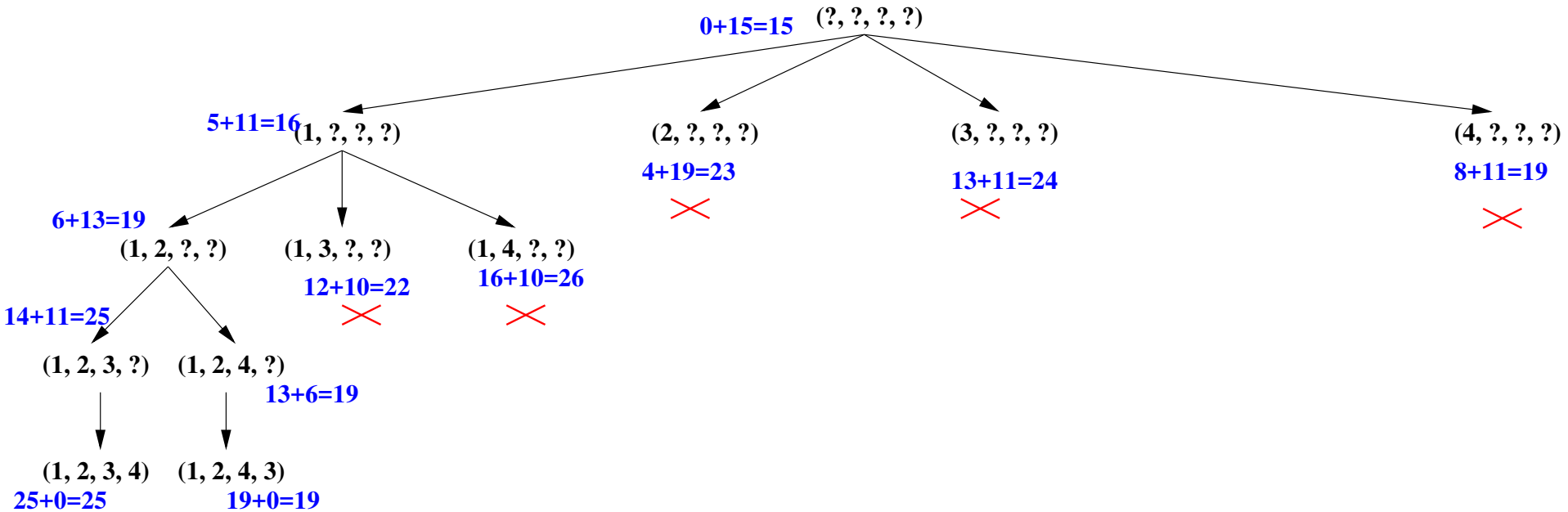
Similarly as in the lecture, imagine that we have currently affected a task for k agents out of n, for a corresponding solution vector $\mathbf{v}$, we can define the following quantities:

- $g^*(\mathbf{v}) = \sum_{i=1}^{k} c[i, \mathbf{v}[i]]$
  the sum of the costs of the tasks affected to the first k agents
- $$f(\mathbf{v}) = \sum_{i=k+1}^{n} \min_{\substack{1 \leq j \leq n \\ j \neq \mathbf{v}[l] \text{ for } 1 \leq l \leq k}} c[i, j]$$

  we take the minimum for each agent (line) among the available quantities

Q1: Search Tree

Here is the new search tree obtained:



Similarly, each node defines a (partial) solution written in black. We

associate the quantity $g^* + h = f$ in blue next (or below) to each node (first value before $=$ is g^*, second is h). The red crosses indicate when the search can be cut. You can see that more branches can cut when a finer evaluation function is used.

Q2: example

Consider the following cost table with 2 agents and 2 functions:

Agent\Task	1	2
1	2	1
2	8	2

The strategy implies to assign task 2 to agent 1, then we need to associate task 1 to agent 2. The final cost is 8+1=9.
However, the optimal solution consists in assigning task 1 to agent 1 and task 2 to agent 2, leading to a final cost of 2+2=4.
This strategy is clearly not optimal. In particular, with this example you can see that $h = 8$ because the other values are removed leading to an evaluation function that can be higher than the optimal result.
$\Rightarrow$ **your evaluation function must always provide a smaller result than the optimal solution.**

Question 2 **TSP - Approximation**

Traveling salesman problem

TSP is known to be an NP-hard problem, thus hard to solve. It corresponds to look for an hamiltonian path with smallest cost in the graph (hamiltonian path is also known to be NP-hard as well).
We consider here an approach allowing us to compute an approximation of the best solution. Given the graph $G = (V, E)$, first apply a minimum spanning tree (MST) algorithm to find a MST. Then, pick a node as the root and perform a depth-first-search. Consider the nodes according to the order defined by the DFS and create an hamiltonian path, this is easy to find you only need to follow the sequence of nodes according to the DFS order. You obtain then an hamiltonian path and thus a potential solution of the problem. The global complexity of the algorithm depends on the search of the MST which can be done in $O(|E| \log |E|)$ depending on the data structure used ($O(|V|^2)$ worst case, the DFS and the construction of the path can be done in $O(|E|)$ time). The procedure is illustrated on the next slide.

Illustration of all the steps of the procedure (step 1)

Let root r be a in following given set of points (graph).

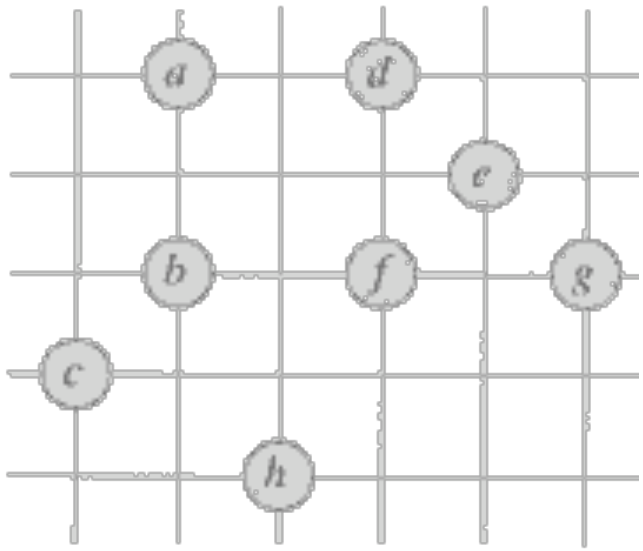


Illustration of all the steps of the procedure (step 2)

Construct MST from root a using MST-PRIM (G, c, r).

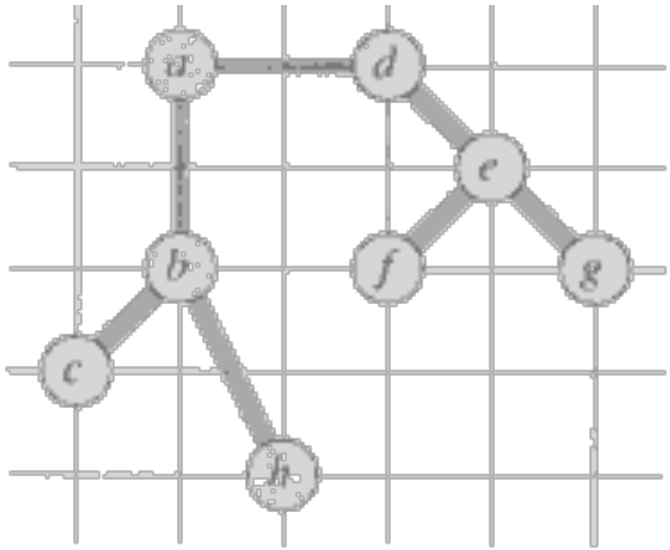


Illustration of all the steps of the procedure (step 3)

List vertices visited in preorder walk. $L = \{a, b, c, h, d, e, f, g\}$

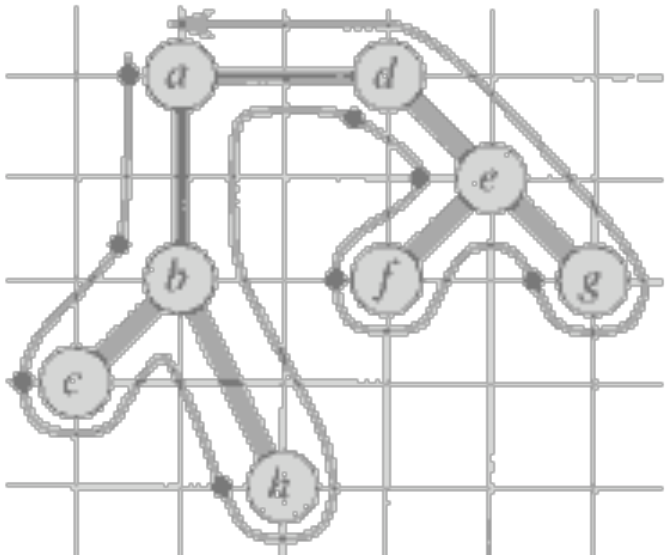
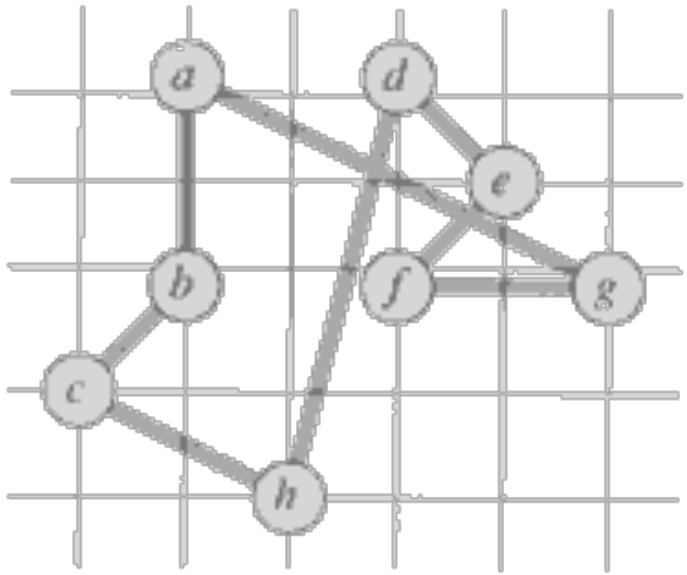


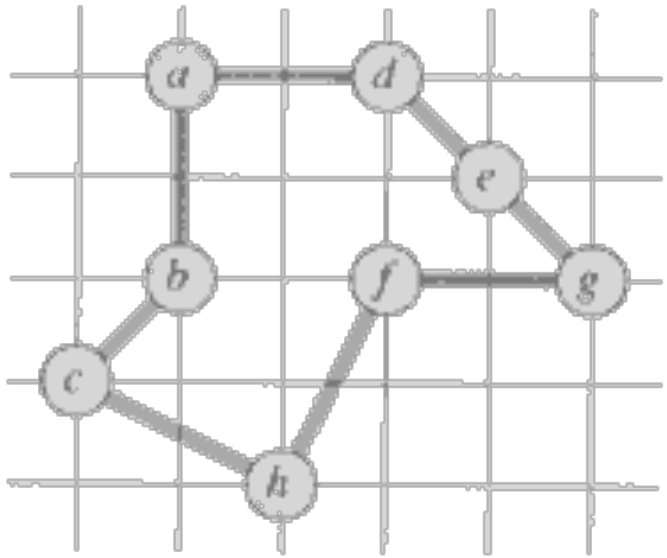
Illustration of all the steps of the procedure (step 4)

Return Hamiltonian cycle.



A possible optimal solution (the costs corresponds to the euclidean distance between two linked nodes and this solution is about 23% shorter).

Optimal TSP tour for a given problem (graph) would be



The solution is at most twice the best solution

Proof.

Let H^* be the optimal tour. Note that a TSP tour with one edge removed is a spanning tree (but not necessarily a minimum spanning tree). Let T be the MST found by the minimum spanning tree algorithm.
This implies that the cost of the MST T is a lower bound on the cost of an optimal tour:

$$cost(T) \leq cost(H^*).$$

Using only the edges of the MST T to do the complete tour implies to traverse every edge of T exactly twice (we need to go forward and to come back from and to each node). Let w be this walk over T, then:

$$cost(w) = 2cost(T).$$

Using the first inequality before, this implies that:

$$cost(w) \leq 2cost(H^*).$$

The solution is at most twice the best solution (ctd)

Now, let s be the solution found by the procedure. Since we assume the weights to respect the triangle inequality, the cost of s must be smaller than the cost of w since going directly to a node directly is less costly than passing through another node, we have then that:

$$cost(s) \leq cost(w) \leq 2cost(H^*)$$

which proves the claim.

Question 3

TSP and branch and bound with edges.

1- An optimal solution has at most one element per row/column

Easy (by contradiction) what happens if one solution has two elements of one row/column? $\Rightarrow$ A town is seen twice.

2- If we substract a value from a whole line/column we do not change the solution

Consider a matrix c and a matrix c' obtained by removing a quantity α from all the values of a line of the matrix c.
Let γ be an optimal tout in c of cost $cost_c(\gamma)$ and γ' an optimal tour in c' of cost $cost_{c'}(\gamma')$.
Use the fact that γ and γ' are optimal solutions, define a relationship between $cost_c(\gamma)$ and $cost_{c'}(\gamma')$ by proving that $cost_c(\gamma) = cost_{c'}(\gamma') + \alpha$.

- We have: $cost_c(\gamma) = cost_{c'}(\gamma) + \alpha$ and $cost_{c'}(\gamma') = cost_c(\gamma') - \alpha$
- Since γ is optimal in c, then $cost_c(\gamma) \leq cost_c(\gamma')$ and thus $cost_c(\gamma) \leq cost_{c'}(\gamma') + \alpha$
- Since γ' is optimal in c', then $cost_{c'}(\gamma') \leq cost_{c'}(\gamma)$ and thus $cost_{c'}(\gamma') \leq cost_c(\gamma) - \alpha$
- By combining the last two lines $cost_c(\gamma) = cost_{c'}(\gamma') + \alpha$.

3- Example with 3 cities: A, B, C

- Original data matrix M :

	A	B	C
A	∞	4	2
B	6	∞	5
C	2	6	∞

- After 1st operator $\mathcal{M}_l$:

	A	B	C
A	∞	2	0
B	1	∞	0
C	0	4	∞

- After 2nd operator $\mathcal{M}_c$:

	A	B	C
A	∞	0	0
B	1	∞	0
C	0	2	∞

$\Rightarrow f(M) = 11$, with $M_{row} = 9$ and $M_{column} = 2$

4- What is the best edge to select at each step?

In general, there is more solution in the right subtree. Thus, the idea is to choose the edge that maximizes the best possible value that will be used by the branch and bound process in the right subtree.
$\Rightarrow$ In the previous example, take edge `(C-A)` of weight 2, you increase $f(M) = 14 = 2 + 5 + 6 + 1$ for the right subtree since `(C-A)` appears to be ∞ for this subtree

Example with 3 cities: A, B, C - matrix for right branch

- Original data matrix M :

	A	B	C
A	∞	4	2
B	6	∞	5
C	∞	6	∞

- After 1st operator $\mathcal{M}_l$:

	A	B	C
A	∞	2	0
B	1	∞	0
C	∞	0	∞

- After 2nd operator $\mathcal{M}_c$:

	A	B	C
A	∞	0	0
B	0	∞	0
C	∞	2	∞

$f(M) = 14 = 2 + 5 + 6 + 1$

Example with 3 cities: A, B, C - Matrix for left branch

	A	B	C
A	∞	4	∞
B	∞	∞	5
C	2	∞	∞

5- How can we improve the solution on the left part of the tree?

On the new example, using the criterion we choose the edge $[4, 6]$, so we put ∞ in the matrix of the right child for $[4, 6]$.

Now, for the left subtree, we can try to put more ∞ to avoid unconsistent solution.

Example: suppose we have the following partial solution at a given step 4−6, 3−5 et 2−1, we have 3 paths not connected (and we already removed edges 6−4, 5−3 and 1−2). If for the next step we choose 1−4, we remove 4−1, but the partial solution is now 2−1−4−6 and 3−5. To avoid a cycle of length strictly lower than n, we need to remove the edge 6−2.