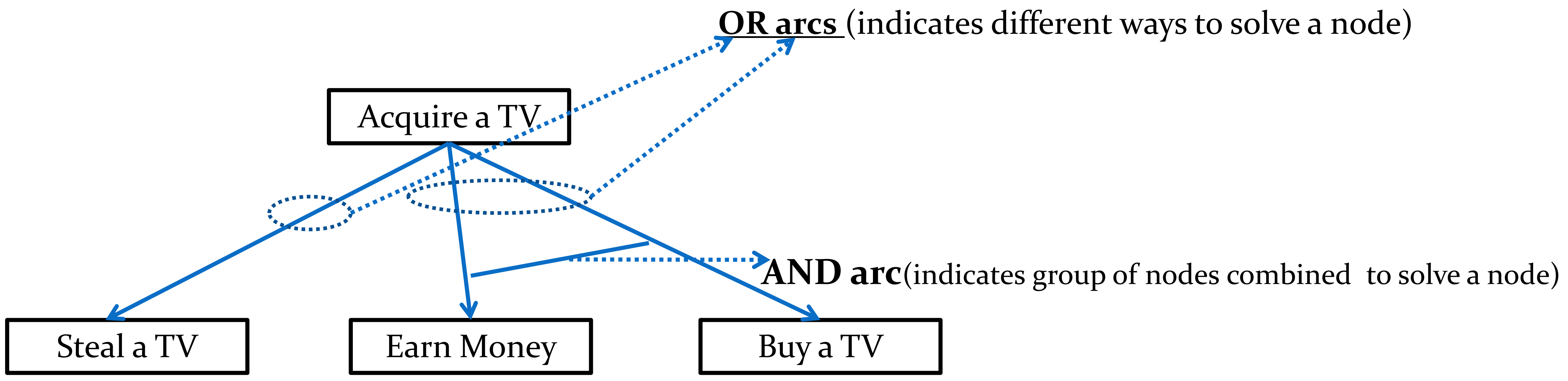
CS-208:Artificial Intelligence
Lectures-09
Problem Reduction



OR arcs (indicates different ways to solve a node)
Acquire a TV
AND arc(indicates group of nodes combined to solve a node)
Steal a TV
Earn Money
Buy a TV

According to this example,

- one way to acquire TV is to steal a TV
- another way to acquire a TV is to earn sufficient money and buy the TV. Here a line is used to connect the arcs pointing earn money and Buy a TV indicating that both are the sub-problems. This connecting line is called **AND arc**

Problem Reduction

The searching techniques, we have discussed so far DFS, BFS, Hill Climbing, Best First Search and A* algorithm are applicable only for OR-graph.

AND-OR Graphs

The AND-OR Graph (or tree) is useful for representing the solution of problems that can solved by decomposing them into a set of smaller problems, all of which must then be solved.

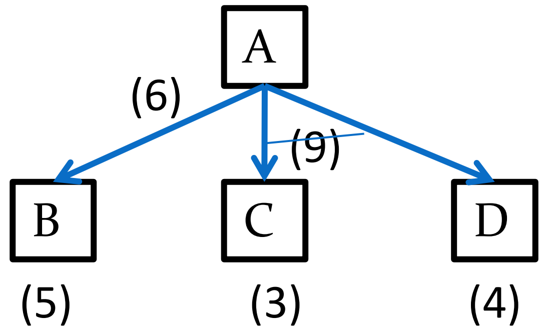
This decomposition, or reduction, generates arcs that are called AND arcs. One AND arc may point to any number of successor nodes, all of which must be solved in order for the arc to point to a solution.

Just as in an OR graph, several arcs may emerge from a single node, indicating a variety of ways in which the original problem might be solved. This is why the structure is called not simply an AND-graph but rather an AND-OR graph (tree)

Why A* algorithm is not adequate for searching AND-OR Graph?

Explanation through an Illustration:

Consider the following **AND-OR** graph:



Assumptions

1. The number at each node represents ***f*** value. Here we will consider ***f***(**n**) = ***h***(***n***) only and ignore ***g***(***n***).
2. Every operation has a uniform cost for simplicity
   - Each arc with single successor has the cost of 1
   - Each AND arc with multiple successors has a cost of 1 for each of its component arcs it connects.

Here the problem is that the choice of which node to expand not only depends on the *f* value of the node, but also on whether that node is part of the current best path from the initial node.

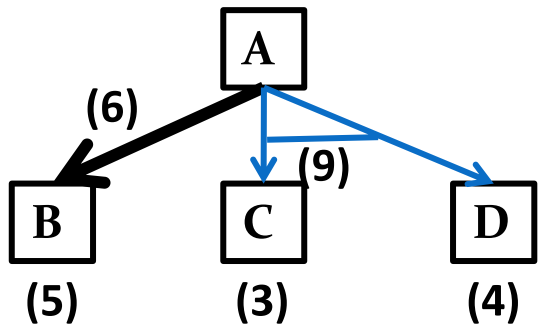


Figure-01

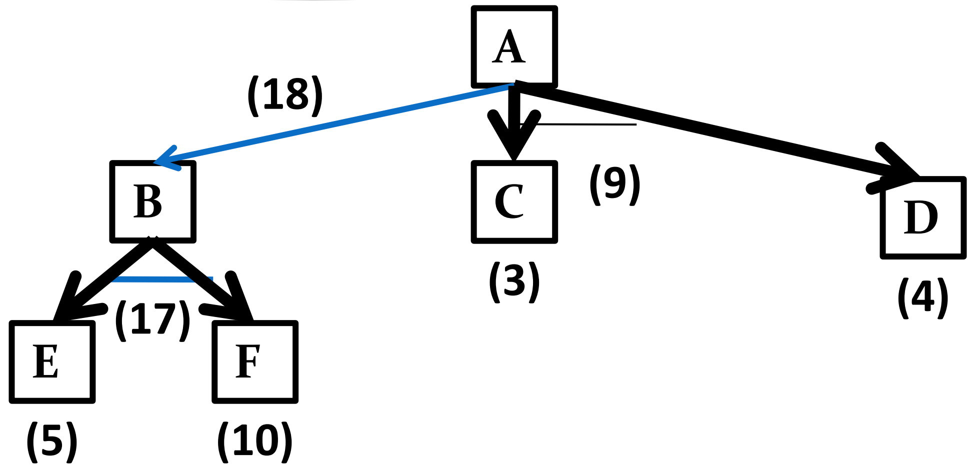


Figure-02

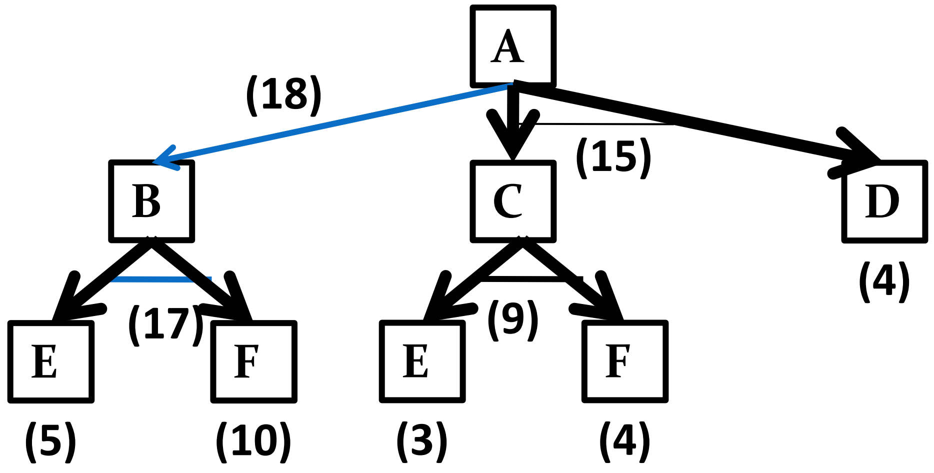


Figure-03

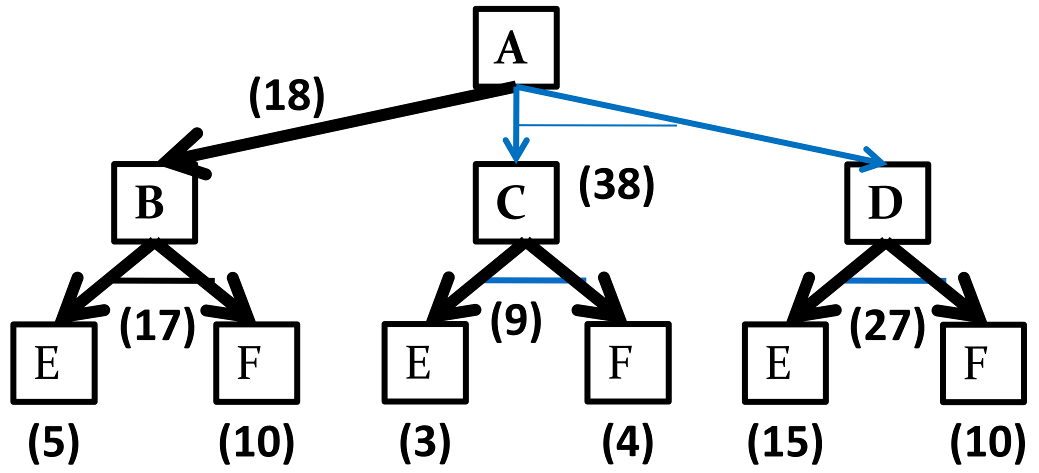


Figure-04

- Darkened Arrows indicate the current best path from each of the expanded nodes.
- In Figure-2 the current best path from the initial node A→ B where B is the only unexpanded node that was being expanded. After this expansion the cost of this path is changed from 6 to 18 and become no longer the current best path from the initial node(because the other path from A have the cost of 9.
- In Figure-3 the current best path from the initial node are A→ C and A→ D where C and D are the unexpanded nodes. Here C is selected for expansion first. After the expansion, this current best remains the same because its new cost is 15 which is still less than the cost of the other path 18.
- In Figure-4 After the expansion of D, the cost of this path becomes 38 and this makes the change in the current best path to the path which having lesser cost of 18.

Before describing an algorithm for searching AND_OR graph, we need to exploit a value called *FUTILITY*. This corresponds to a threshold value , any solution with cost above it, is too expensive to be practical. when the estimated cost of the solution becomes greater than the value of *FUTILITY,* then the search will be abandoned.

Problem Reduction Algorithm:

1. Initialize a graph **G** with only *starting node* named as *INIT.*
2. Repeat the followings until INIT is labeled as SOLVED or h(INIT) becomes greater than *FUTILITY***.**
    a) Traverse the graph G starting from INIT and following the current best path, accumulate the set of nodes that are on that path and have not expanded or labeled as SOLVED
    b) Pick up one of these unexpanded node and call it as NODE. Generate the successors of NODE. If there are none, set h(NODE) = FUTILITY (i.e., NODE is unsolvable); otherwise for each successor that is not an ancestor of NODE do the following:
        i. Add successor to G.
        ii. If the successor is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0, otherwise compute its h.

c) Change the h(NODE) to reflect the new information provided by its successors. Propagate this newly discovered information up through the graph G by doing the following:

   i. If any node contains a successor arc whose descendent are all SOLVED then label the node itself labeled as SOLVED.

   ii. At each node that is visited while going up in the graph G, decide which of its successor arc is most promising and mark it as the part of the current best path. This may cause the current beat path subject to change.