Artificial Intelligence:
Search Methods for Problem Solving

Problem Decomposition

A First Course in Artificial Intelligence: Chapter 6

Deepak Khemani
Department of Computer Science & Engineering
IIT Madras

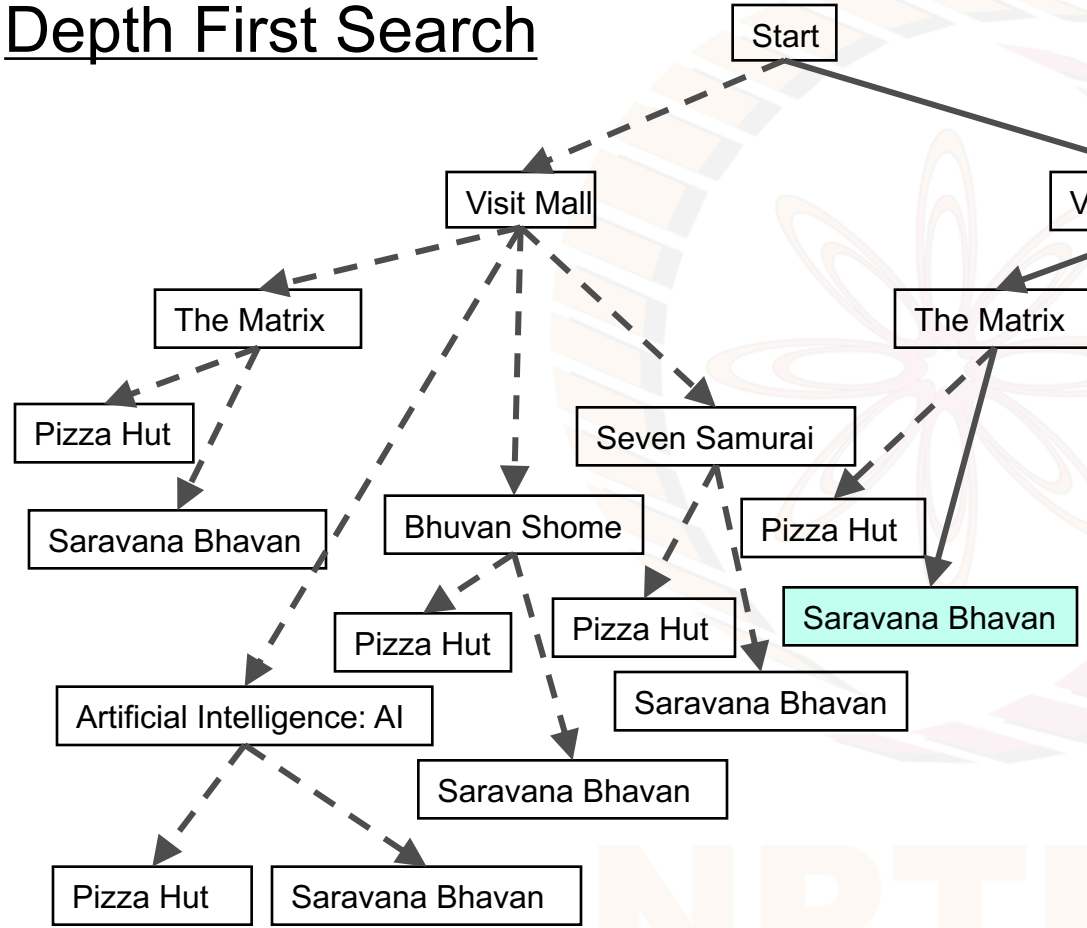
Problem Decomposition

- So far our view of *problem solving* using search has been *state centered*
  - the *state space* is the arena for search
  - the *solution* is expressed as a *sequence of states*
  - even when we talk of solution space, the solution, for example for the TSP problem, is expressed in terms of states.
- Problem decomposition takes a goal directed view of problem solving
  - the emphasis is on breaking up a problem into smaller problems
    - like in backward state space planning: goal → subgoals
  - primitive problems are labeled SOLVED
  - … otherwise they are LIVE and have to be refined
    - like in the SSS* game playing algorithm

Motivation

- Consider the problem of planning an evening out with friends
- To plan for an activity, a movie followed by dinner
- Let us say the agent proposing the plan works with a MoveGen as follows
  - in the start state pick an activity
  - having chosen an activity pick a movie
  - having picked a movie pick a restaurant for dinner
  - propose the plan
    - if not accepted then backtrack
    - essentially doing depth first search
- Then the search conducted by the agent may be as follows…

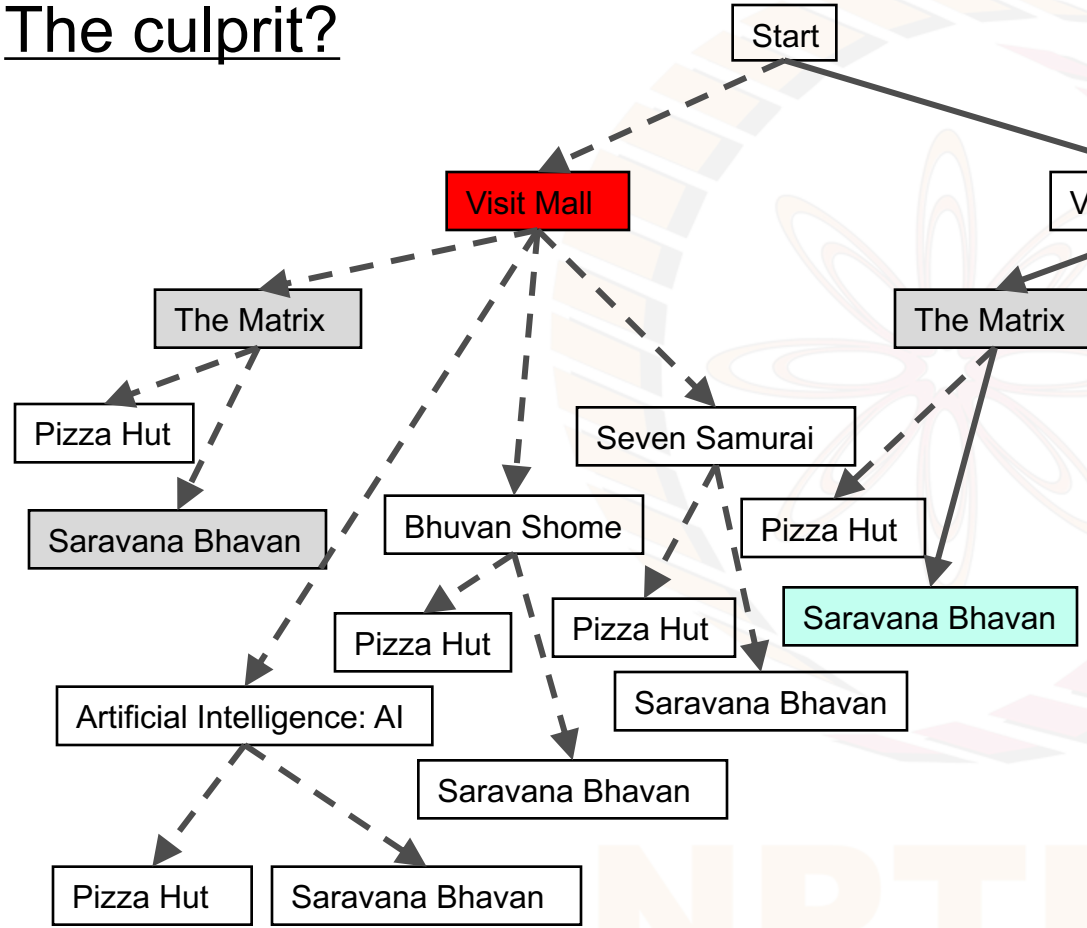
Depth First Search



Everyone accepts the solution

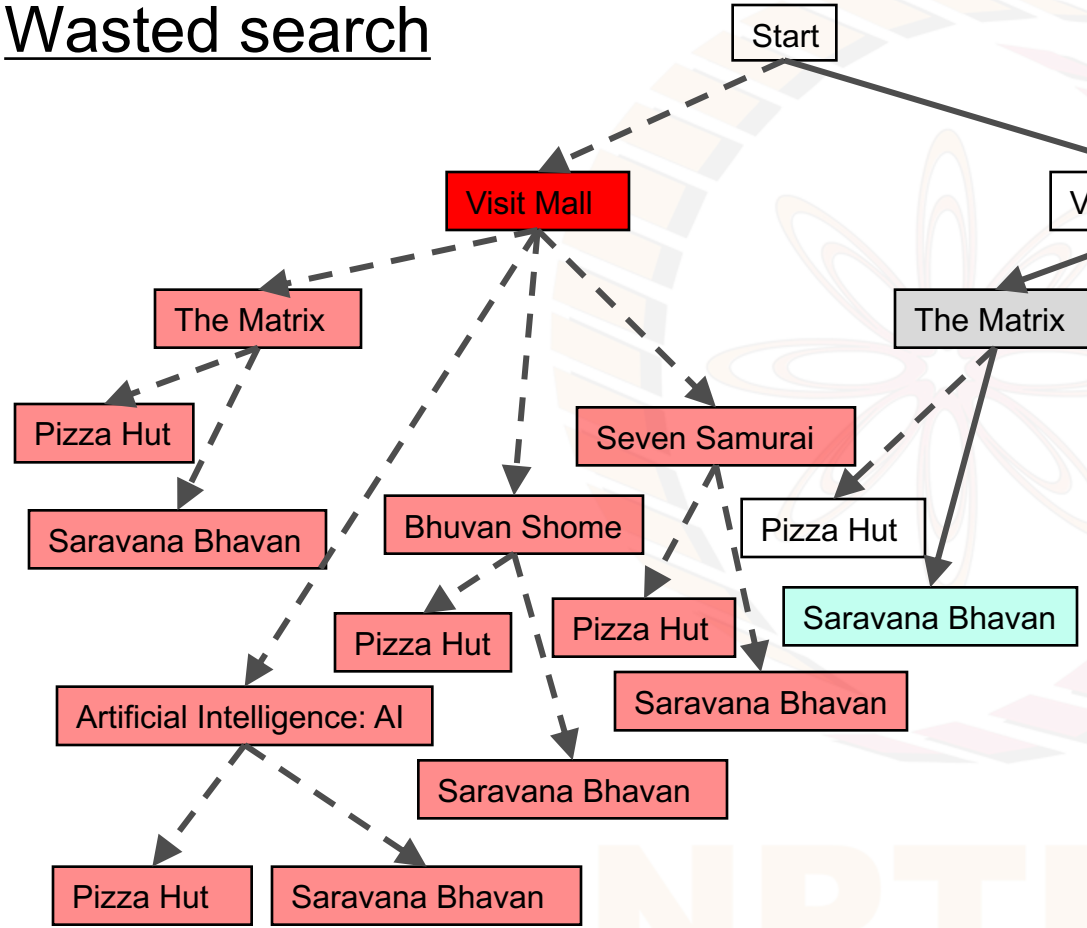
1. visit the beach
2. watch The Matrix
3. eat at Saravana Bhavan

The culprit?



It looks like the culprit is the choice to visit the mall, because,
the other two choices occurred in the left subtree as well

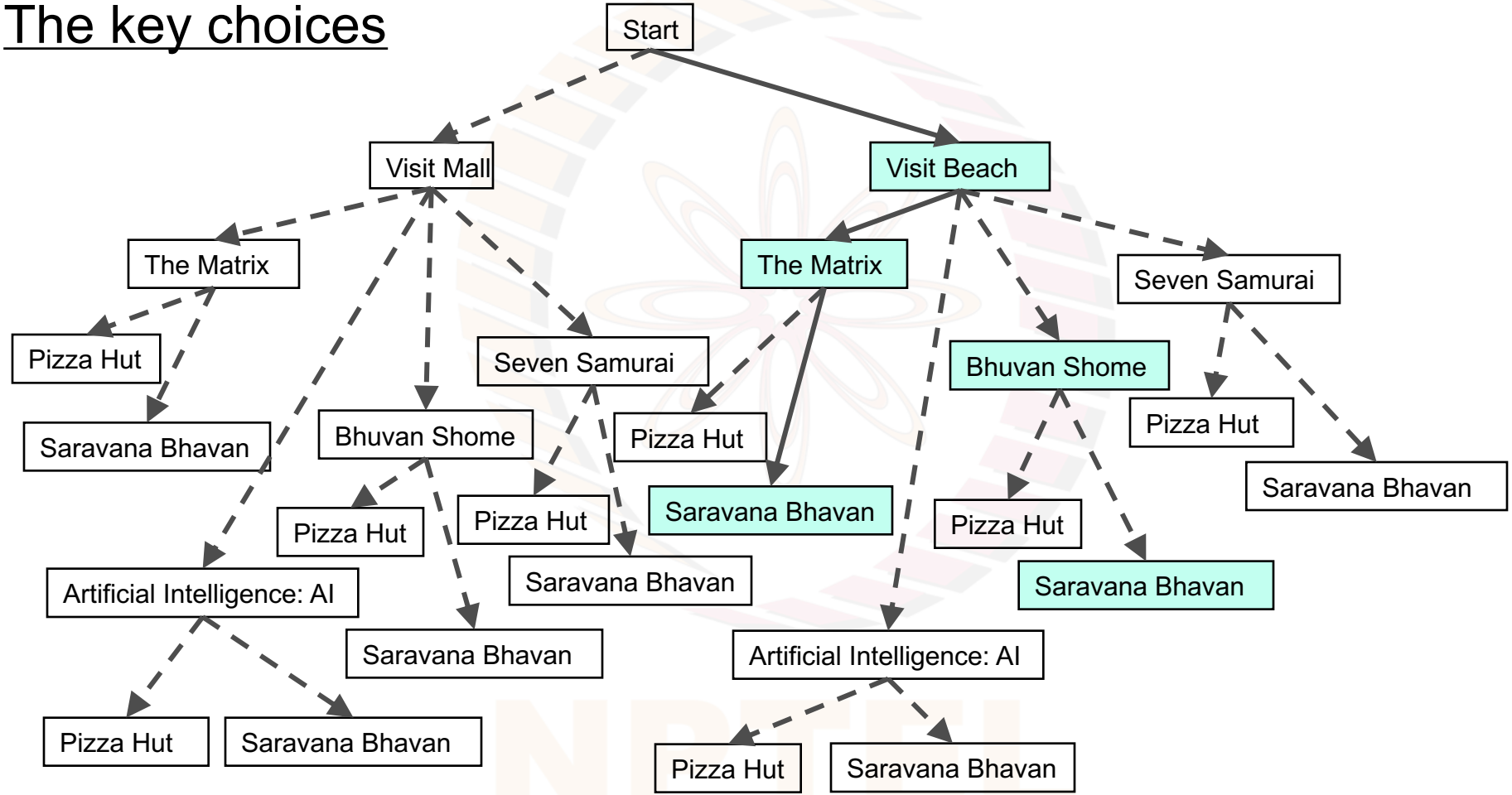
Wasted search



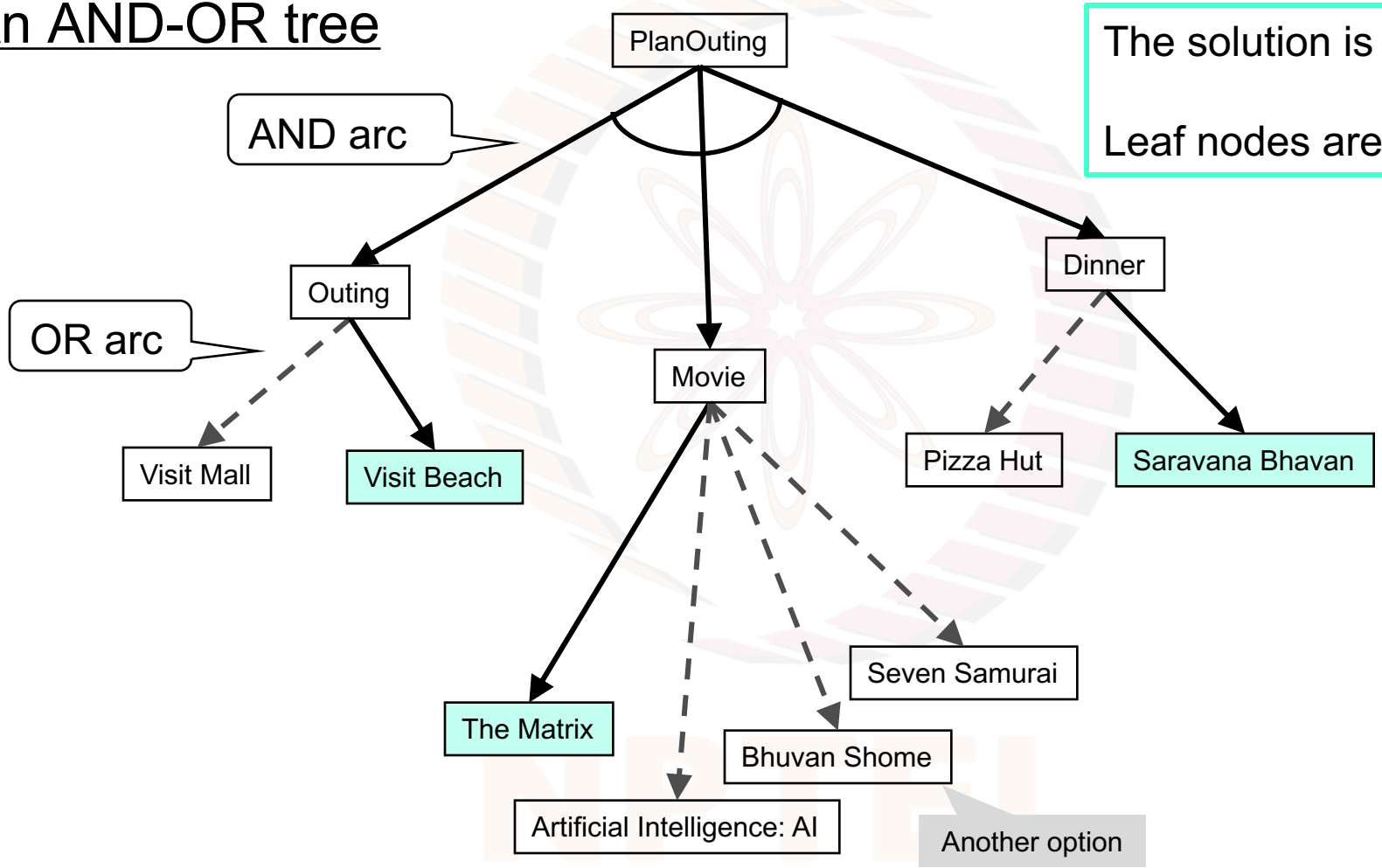
Chronological backtracking in DFS results in wasted effort below the Visit Mall node.

Dependency directed backtracking in Constraint Processing would have jumped back to the culprit variable.

The key choices



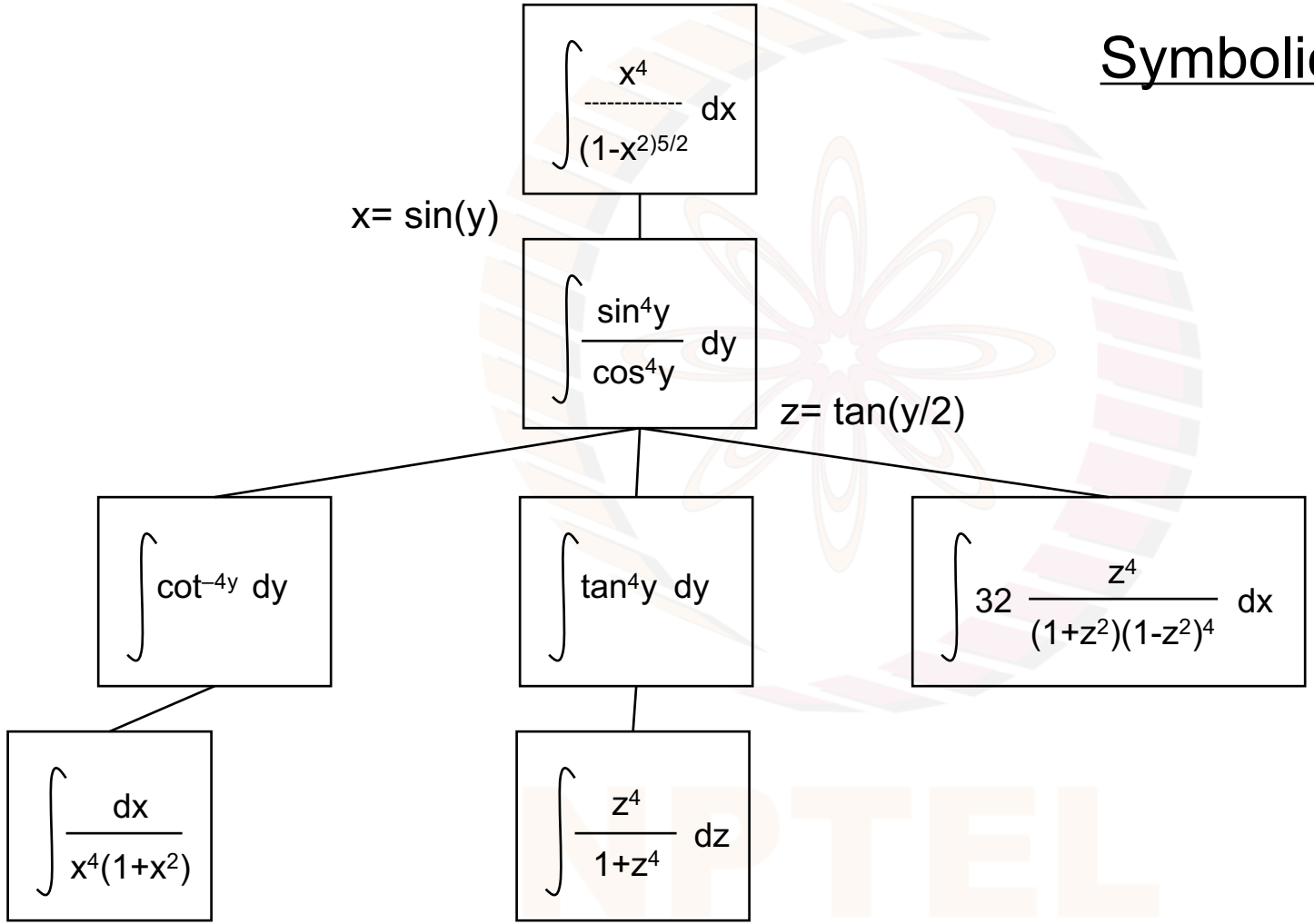
An AND-OR tree



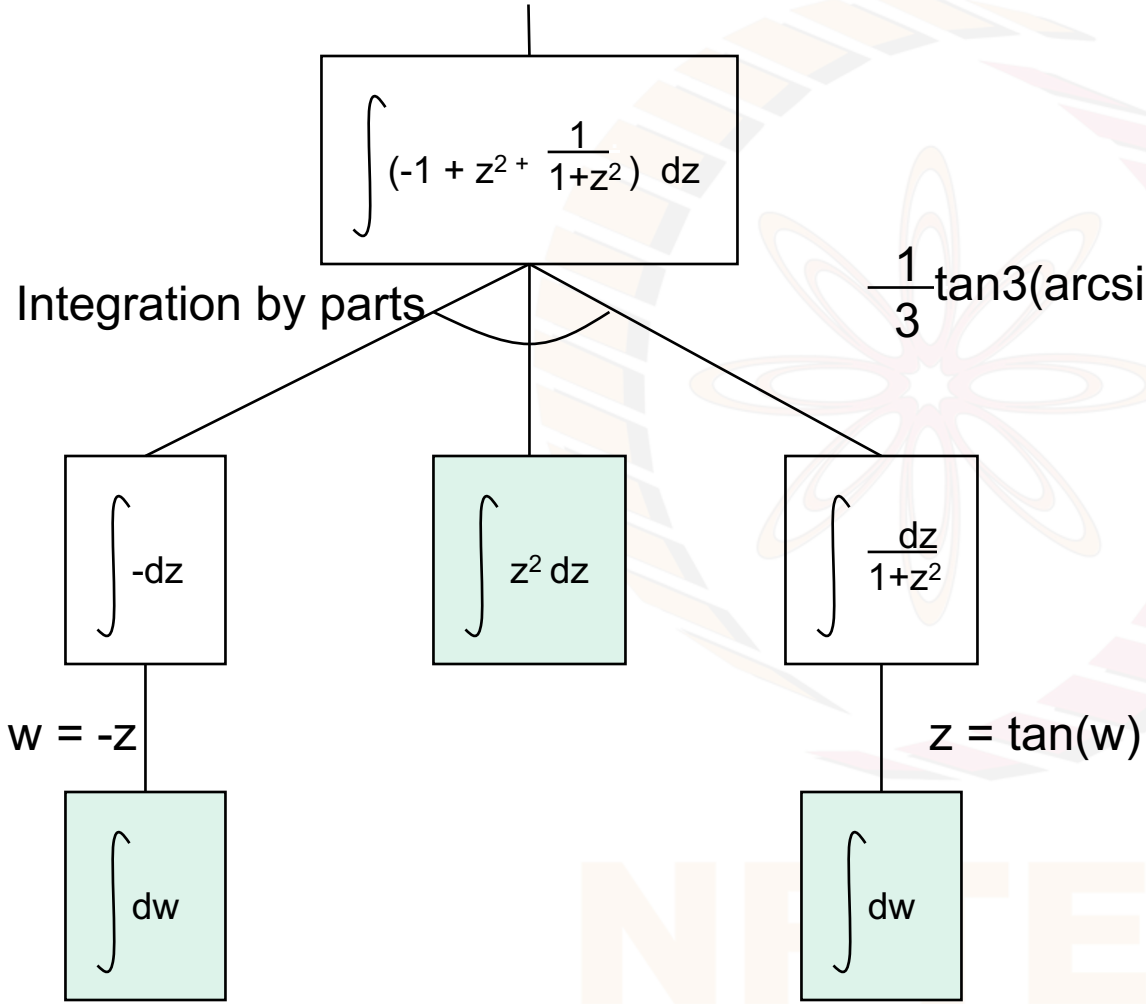
The solution is a subtree!

Leaf nodes are SOLVED

Symbolic Integration



Symbolic Integration



$\frac{1}{3}\tan 3(\arcsin x) - \tan(\arcsin x) + \arcsin x$

Dendral (1965-1980)

One of the earliest successes of AI was the program *Dendral* (for Dendritic Algorithm) developed at Stanford University.

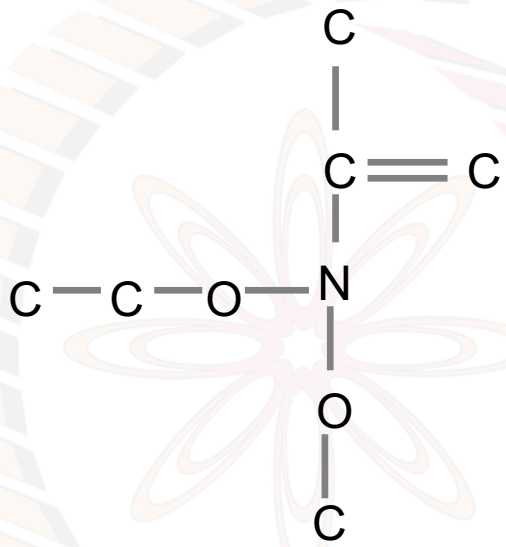
The *DENDRAL* program was the first AI program to emphasize the power of specialized knowledge over generalized problem-solving methods.

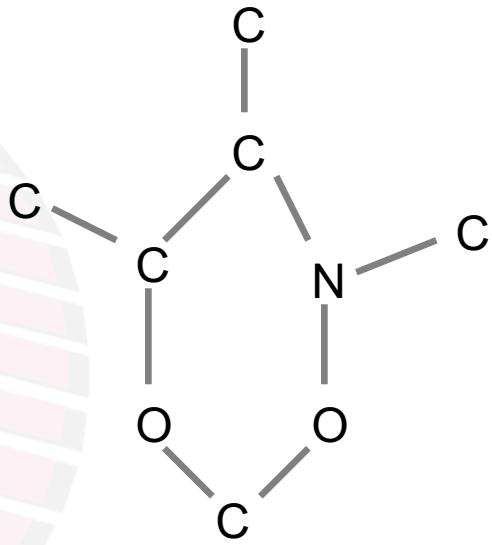
To assist chemists in the task of determining the structure of a chemical compound. The number of candidate structures for a given compound can be very large.

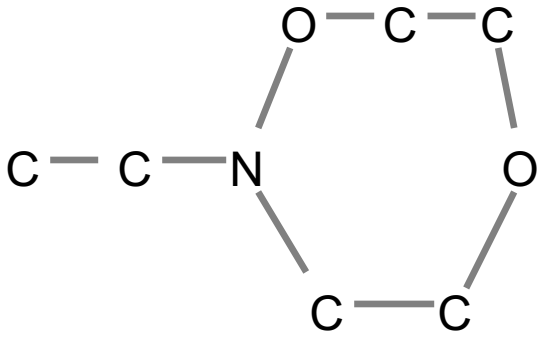
Dendral led to a program called *CONGEN* (CONstrained GENerator) that allows a chemist to constrain the generation of candidates.

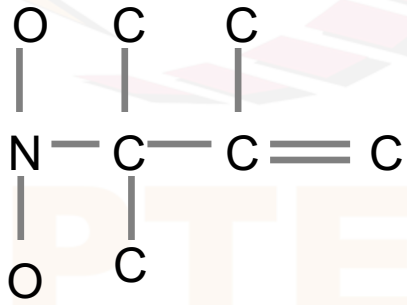
Structural formulas

Candidate structures for $C_6H_{13}NO_2$ generated by CONGEN [Buchanan 82]. The hydrogen atoms are not shown.









Dendral: an Expert System

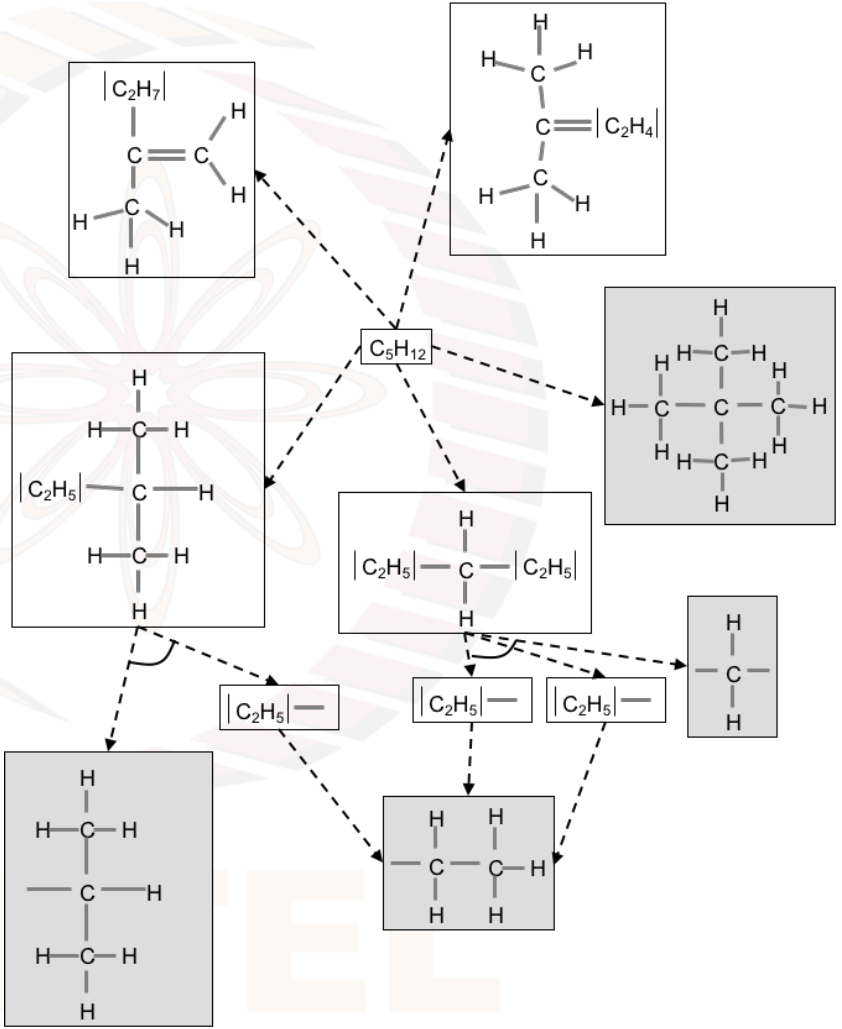
The program DENDRAL explored and And-Or graph.

It generated candidate structures and generated a synthetic spectrogram.

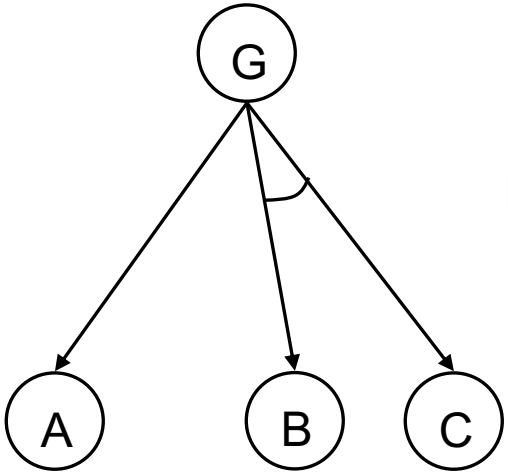
This was compared to the spectrogram of the material.

Performed better than most human chemists.

An *Expert System*



Goal Trees



The search space generated for solving And-Or (AO) problems can be seen as a goal tree.

The figure shows that a goal G may be refined in two ways

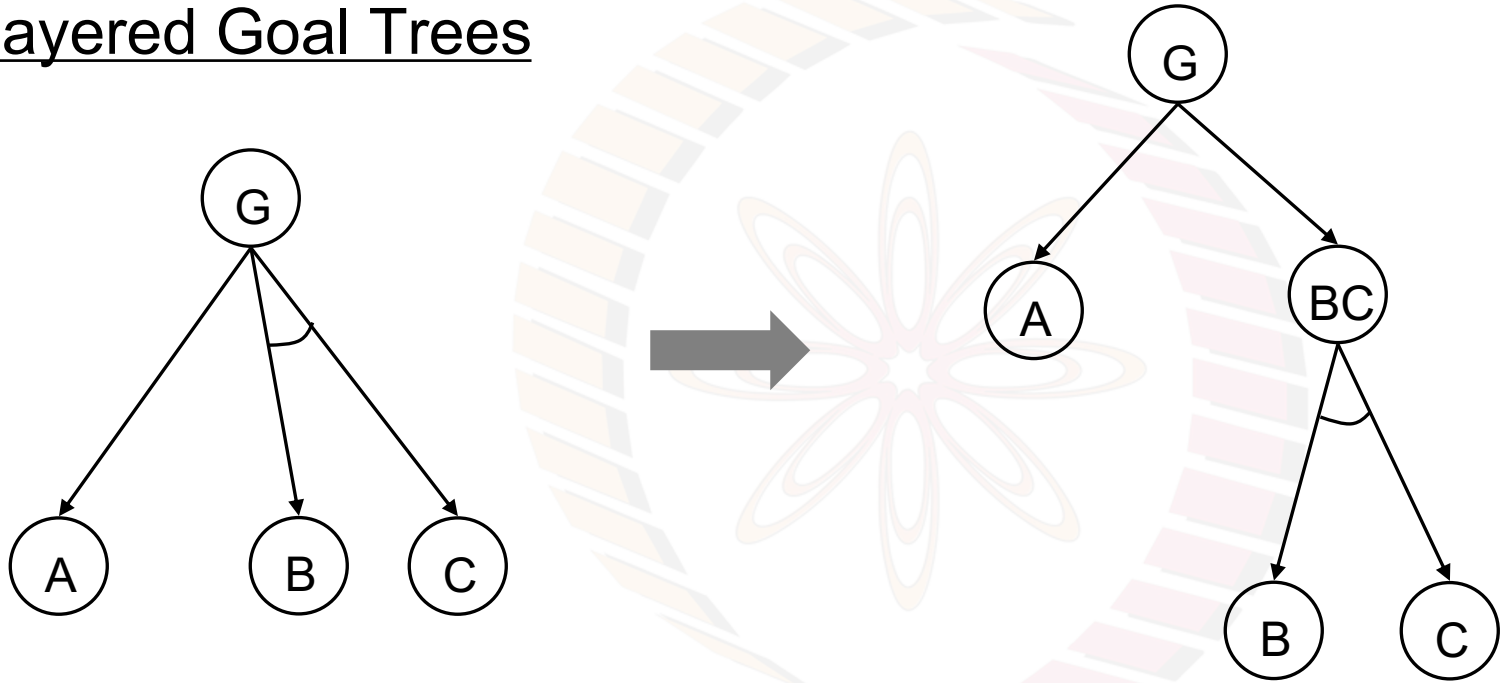
- the left branch involves solving the subgoal A
- the right branch reduces in to subgoals B *and* C
    - both have to be solved

The nodes in the search space have a *heuristic value* that is an *estimate* of the cost of solving the node.

Edge costs indicate the *cost of transforming* a problem.

Solved leaf nodes may have a non-zero cost as well.

Layered Goal Trees



An AO graph with mixed nodes can be converted into a graph with pure AND and OR nodes.

Solving Goal Trees

At any point the algorithm AO* maintains a graph generated so far.

Every choice point in the graph has a marker
marking the best choice as it appears at that point of time.

The algorithm follows the marked path leading to a set of live nodes.
It refines one of the LIVE nodes by expanding it.

It backs up the cost of the best hyper arc and propagates it upwards.

If the best choice leads to a SOLVED node
the node is also labeled SOLVED

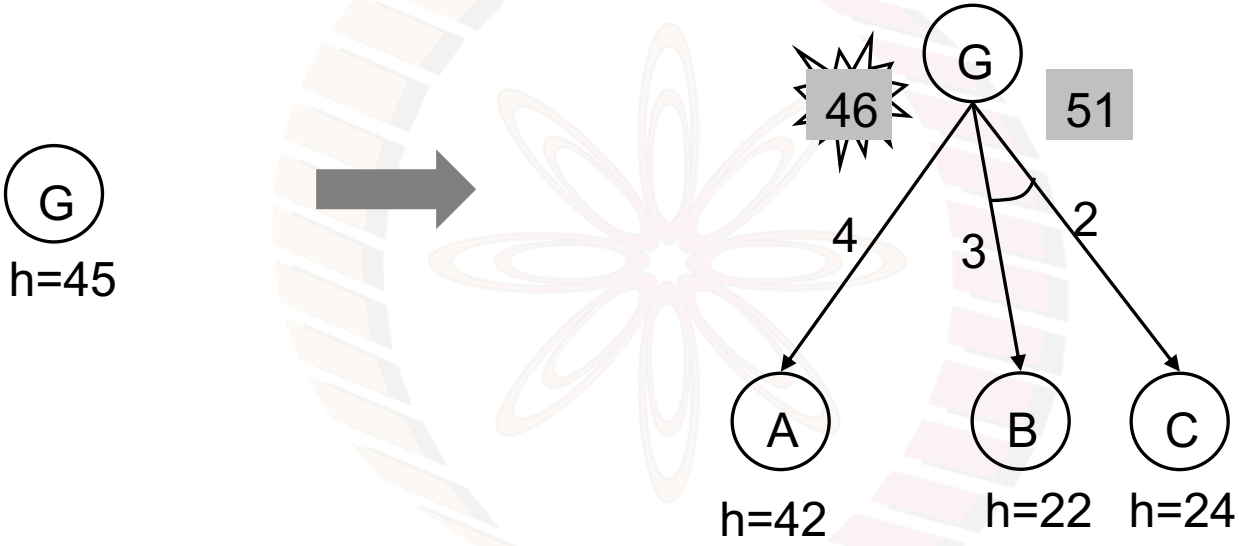
The algorithm terminates when the root is labeled SOLVED
or there is no solution

AO*

The algorithm for solving the goal tree, known as the *AO** algorithm (Martelli and Montanari, 1978; Nilsson, 1980) has the following cycle.

- Starting at the root traverse the graph along marked paths till the algorithm reaches a set of unsolved nodes *U*.
- Pick a node *n* from *U* and refine it.
- Propagate the revised estimate of *n* up via all ancestors.
- If for a node all *AND* successors along the marked path are marked *SOLVED* mark it *SOLVED* as well.
- If a node has *OR* edges emanating from it, and the cheapest successor is marked *SOLVED* then mark the node *SOLVED*.
- Terminate when the root node is marked *SOLVED*.

An illustration



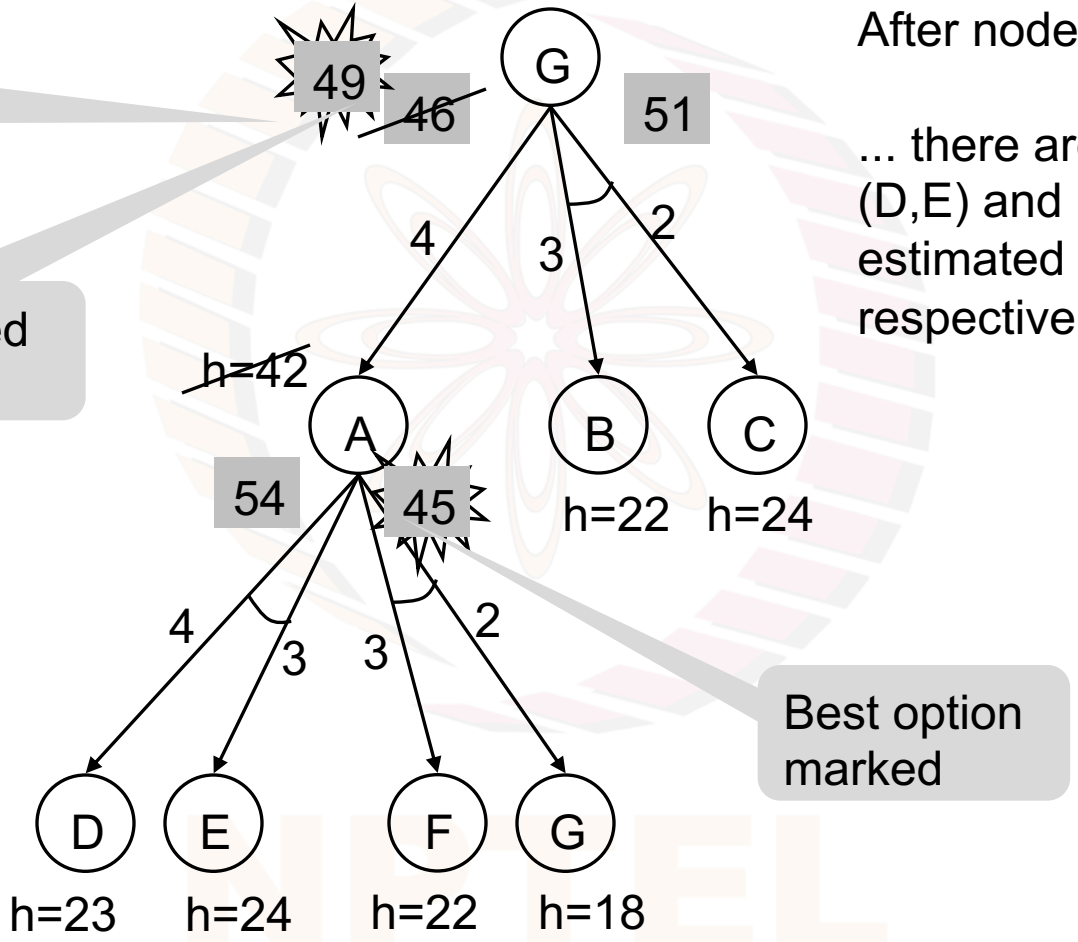
Which is the best node to expand next?

Even though node B has the lowest heuristic value it is a part of a more expensive looking option. Note that the choice is made on the basis of backed up values 46 and 51.

An illustration

Best option marked

New estimated total cost.



After node A is expanded...

... there are two options (D,E) and (F,G) with estimated costs 54 and 45 respectively.

Best option marked

AO*: forward phase

```
AO*(start, Futility)        ▷ Futility is the maximum cost solution acceptable
 1  add start to G          ▷ Use a graph G instead of OPEN, CLOSED lists
 2  compute h(start)
 3  solved(start) ← FALSE
 4  while solved(start) = FALSE and h(start) ≤ Futility
        ▷ FORWARD PHASE
 5      U ← trace marked paths in G to a set of unexpanded nodes
 6      N ← select a node from U
 7      children ← SUCCESSORS(N)
 8      if children is empty
 9          h(N) ← Futility
10      else check for looping in the members of children
11          remove any looping members from children
12          for each S ∈ children
13              add S to G
14              if S is primitive
15                  solved(N) ← TRUE
16                  compute h(N)    ▷ could be zero
```

AO*: forward phase

```
        ▷ PROPAGATE BACK
17      M ← {N}          ▷ set of modified nodes
18      while M is not empty
19          D ← remove deepest node from M
20          compute best cost of D from its children
21          mark best option at D as MARKED
22          if all nodes connected through marked arcs are solved
23              solved(D) ← TRUE
24          if D has changed
25              add all parents of D to M
26  if solved(start) = TRUE
27      return the marked subgraph from start node
28  else return null
```

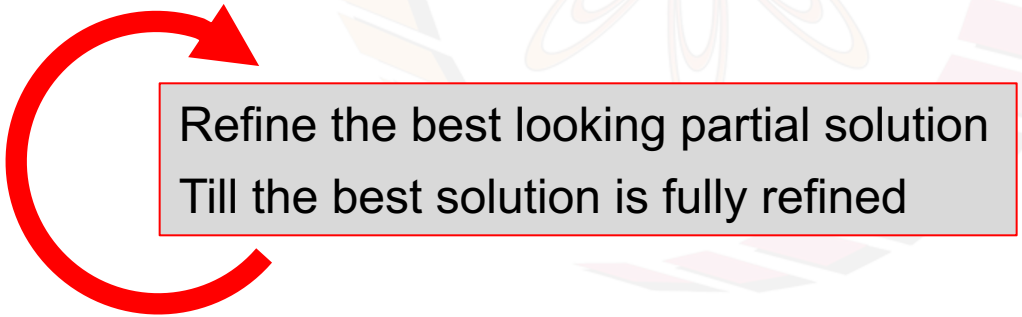
Algorithm AO*

```
AO*(start, Futility)        ▷ Futility is the maximum cost solution acceptable
 1  add start to G          ▷ Use a graph G instead of OPEN, CLOSED lists
 2  compute h(start)
 3  solved(start) ← FALSE
 4  while solved(start) = FALSE and h(start) ≤ Futility
        ▷ FORWARD PHASE
 5      U ← trace marked paths in G to a set of unexpanded nodes
 6      N ← select a node from U
 7      children ← SUCCESSORS(N)
 8      if children is empty
 9          h(N) ← Futility
10      else check for looping in the members of children
11          remove any looping members from children
12          for each S ∈ children
13              add S to G
14              if S is primitive
15                  solved(N) ← TRUE
16                  compute h(N)    ▷ could be zero
        ▷ PROPAGATE BACK
17      M ← { N }       ▷ set of modified nodes
18      while M is not empty
19          D ← remove deepest node from M
20          compute best cost of D from its children
21          mark best option at D as MARKED
22          if all nodes connected through marked arcs are solved
23              solved(D) ← TRUE
24          if D has changed
25              add all parents of D to M
26  if solved(start) = TRUE
27      return the marked subgraph from start node
28  else return null
```

Admissibility of AO*

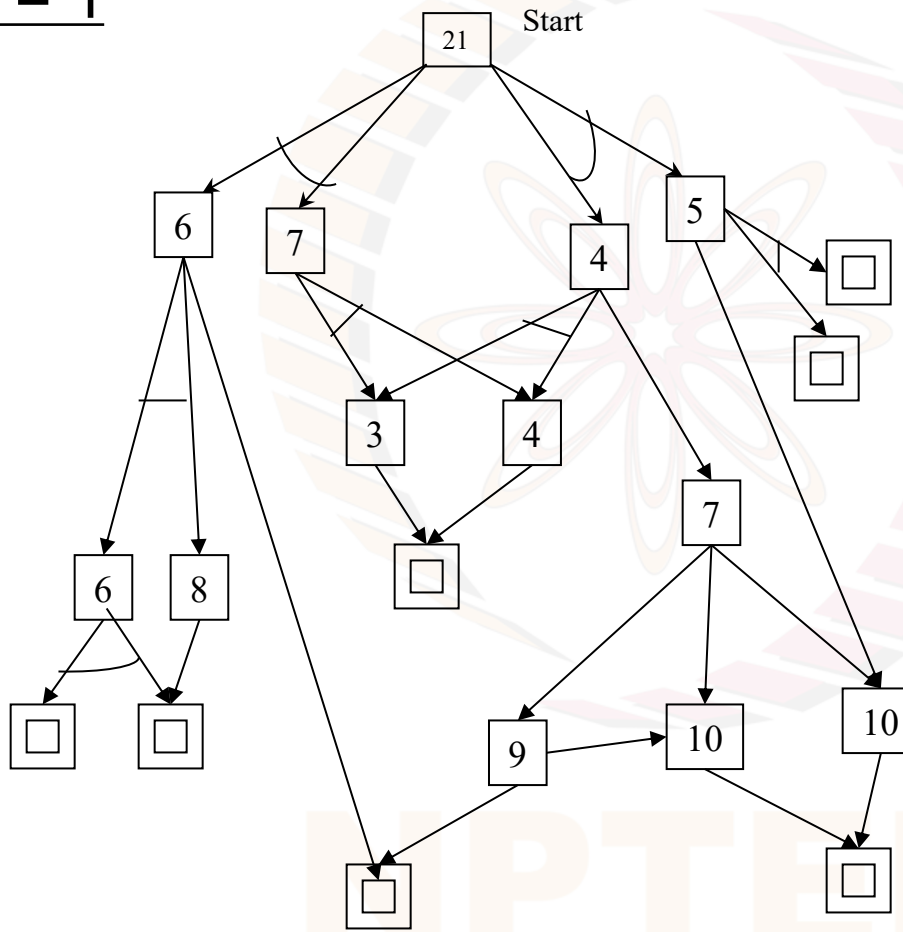
Like A* the algorithm AO* is also admissible (finds the least cost solution) when the heuristic function underestimates the actual cost.

The explanation is the same as before – as long as the partial solutions are looking better it will keep exploring them.



We illustrate this by varying the cost of edges.

Edge cost = 1

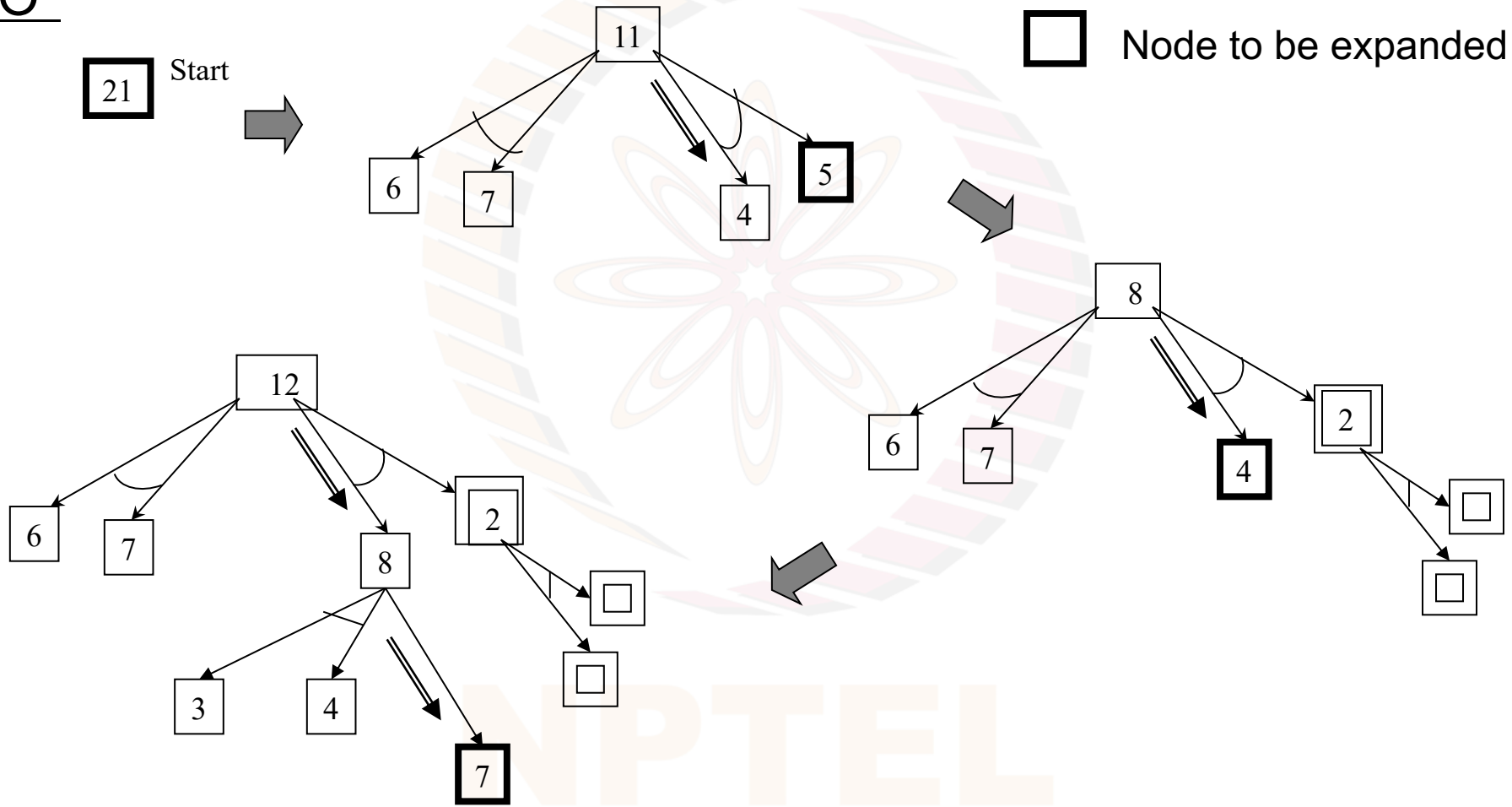


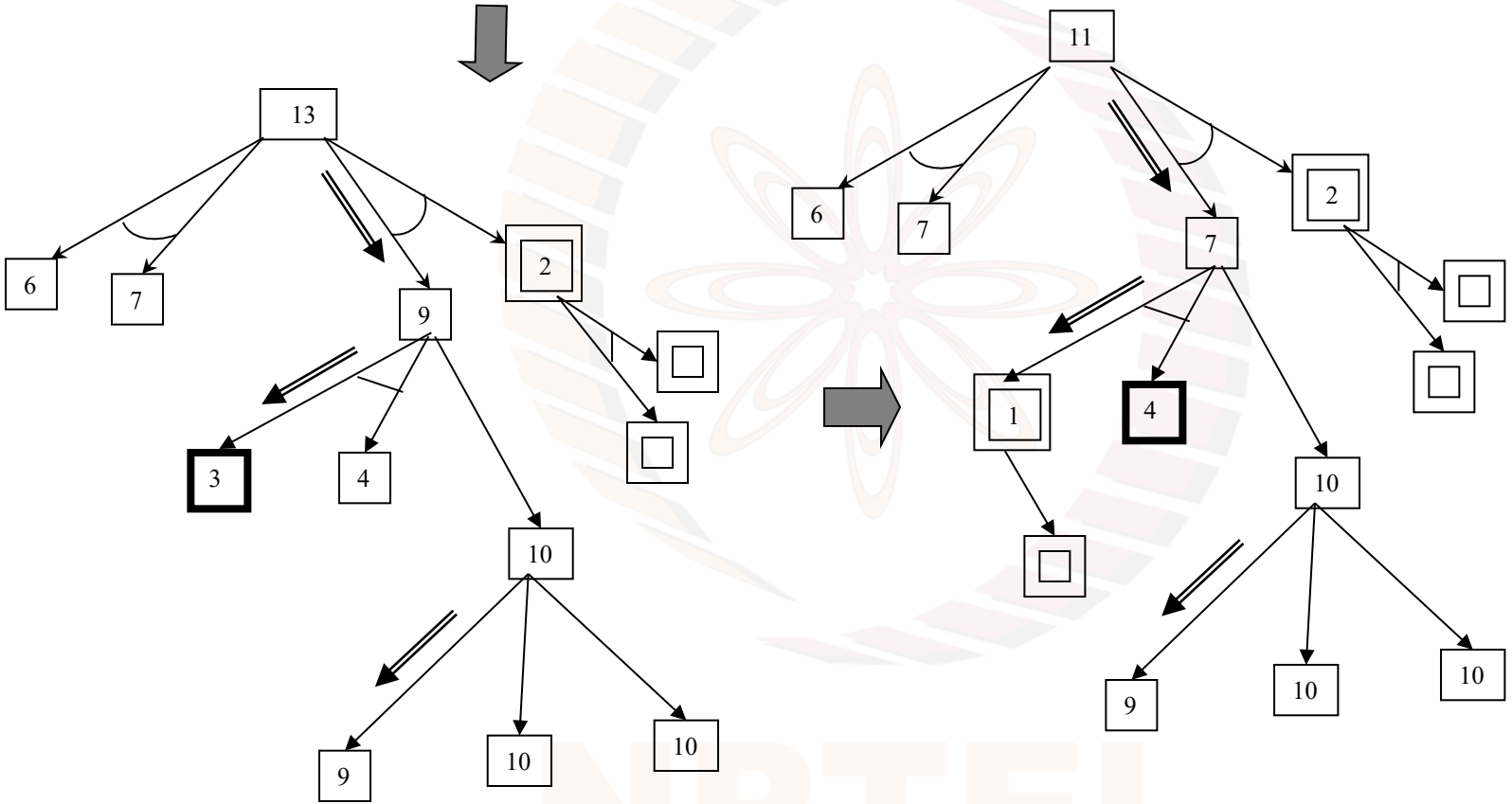
Assume every arc costs one unit to traverse.

Nodes are labeled with heuristic values.

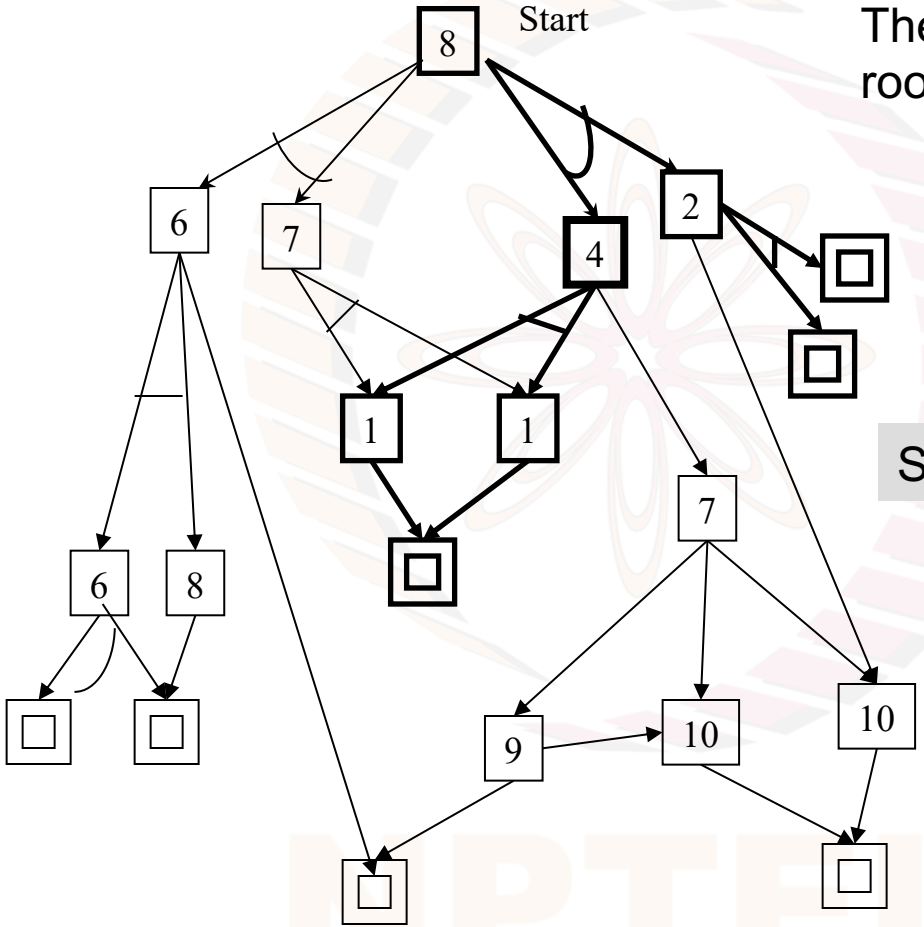
Solved nodes represented by double lined boxes have cost zero

AO*




13
6
7
9
2
3
4
10
9
10
10
11
6
7
7
2
1
4
10
9
10
10

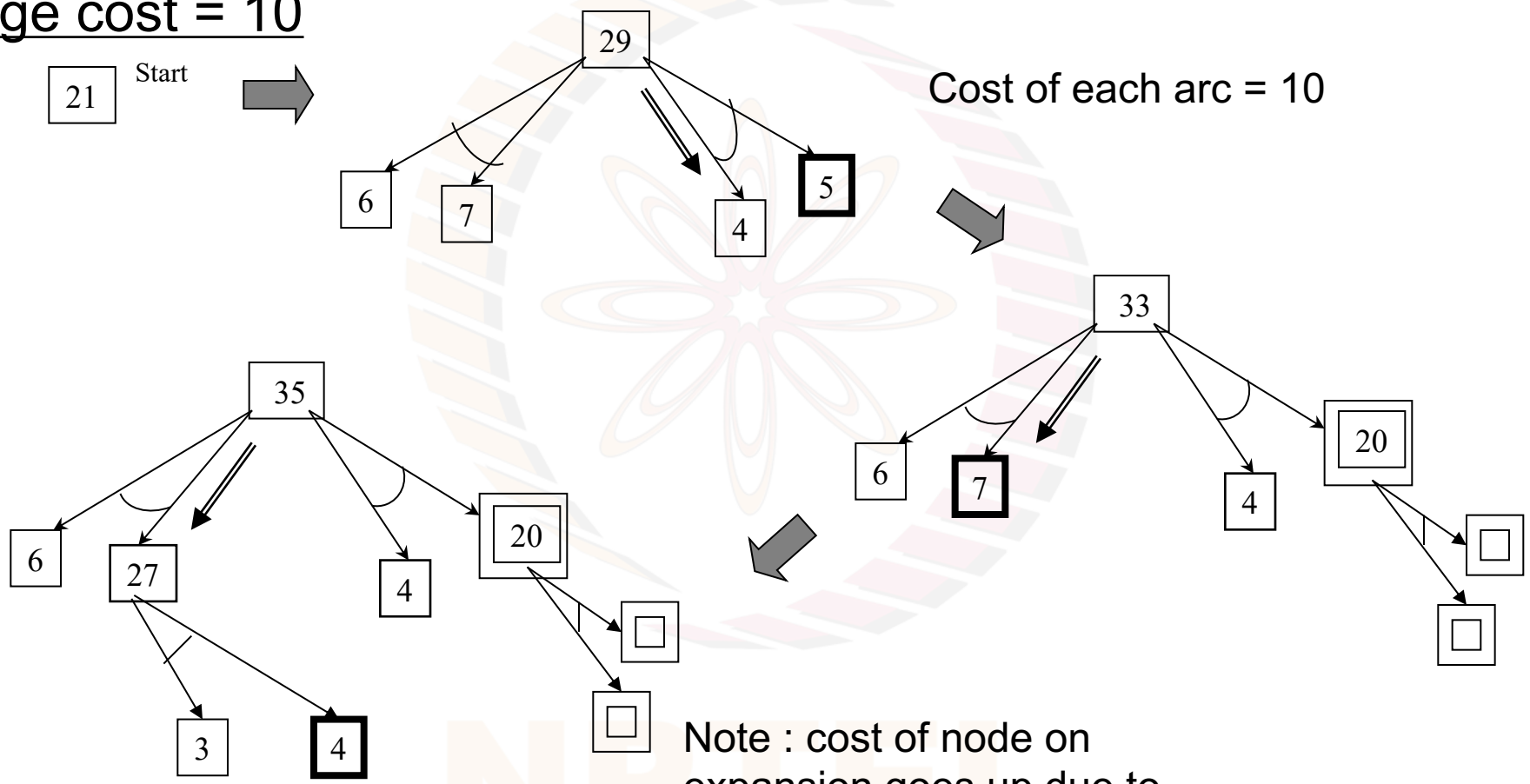
Solution



The algorithm terminates as the root (start) is labeled SOLVED

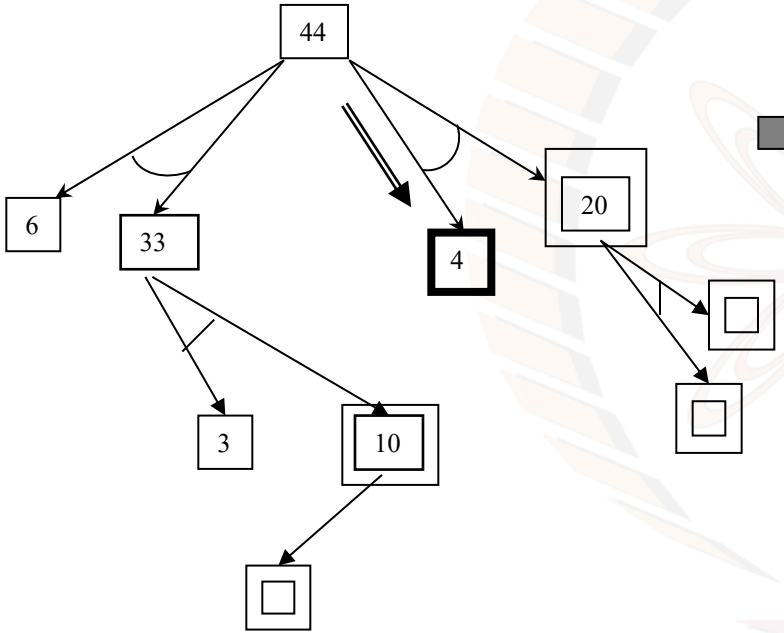
Solution found with cost = 8

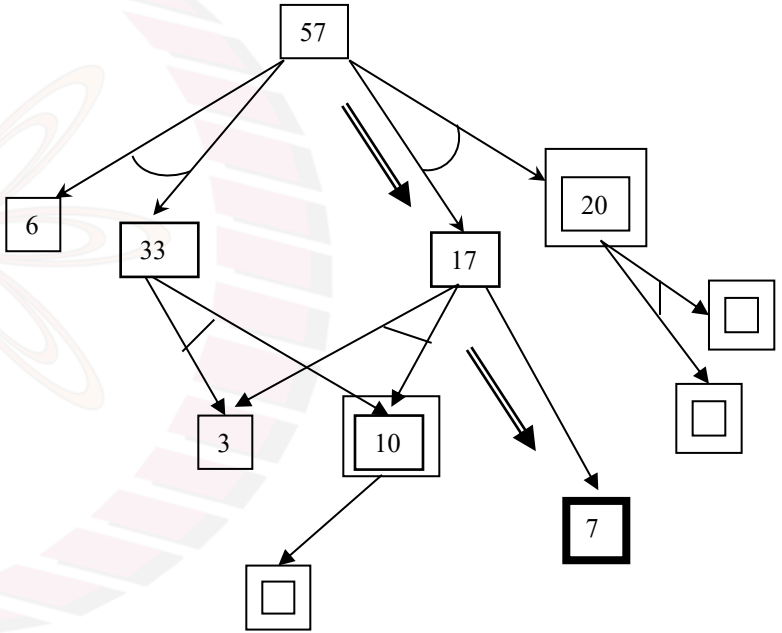
Edge cost = 10

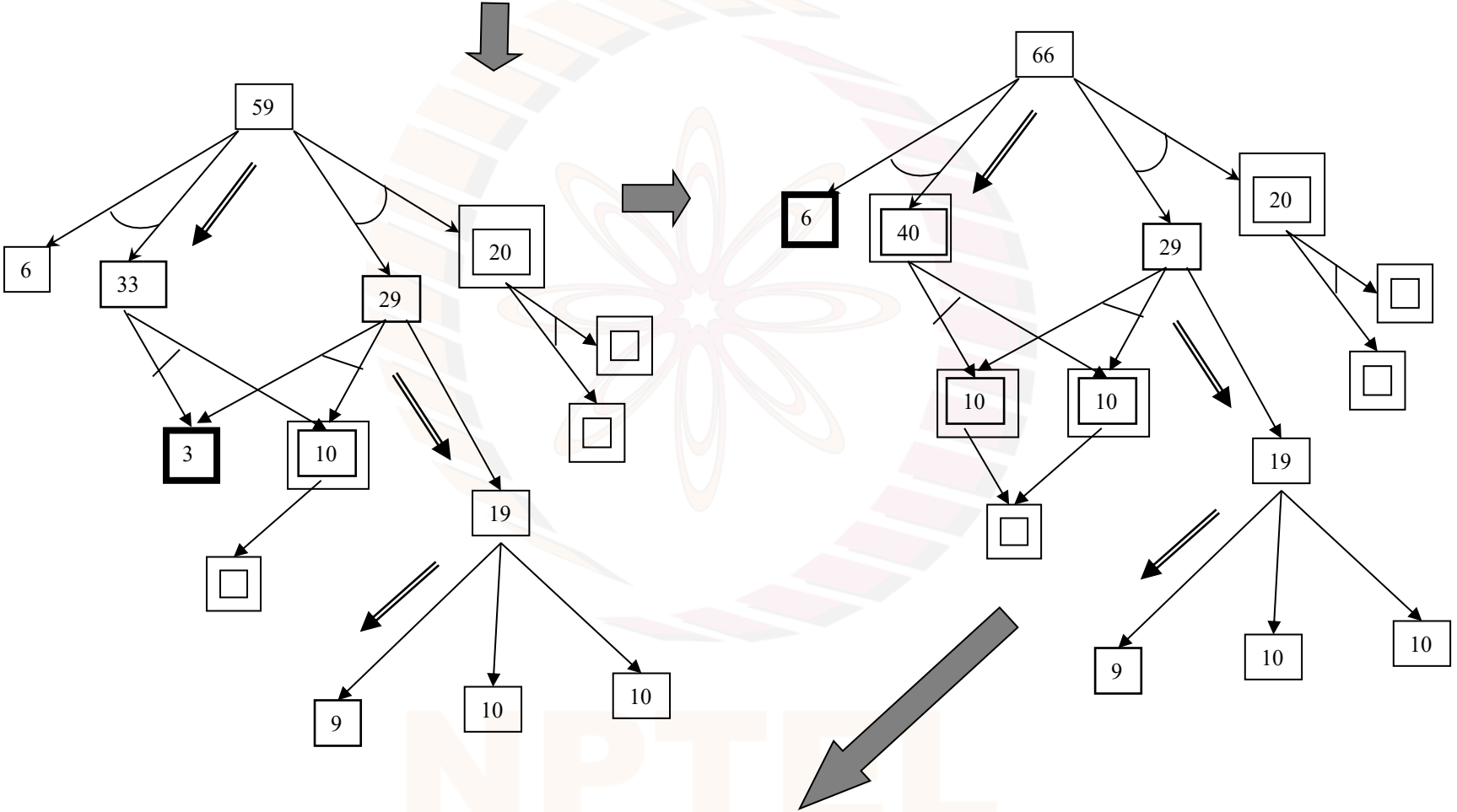


Note : cost of node on expansion goes up due to underestimation.

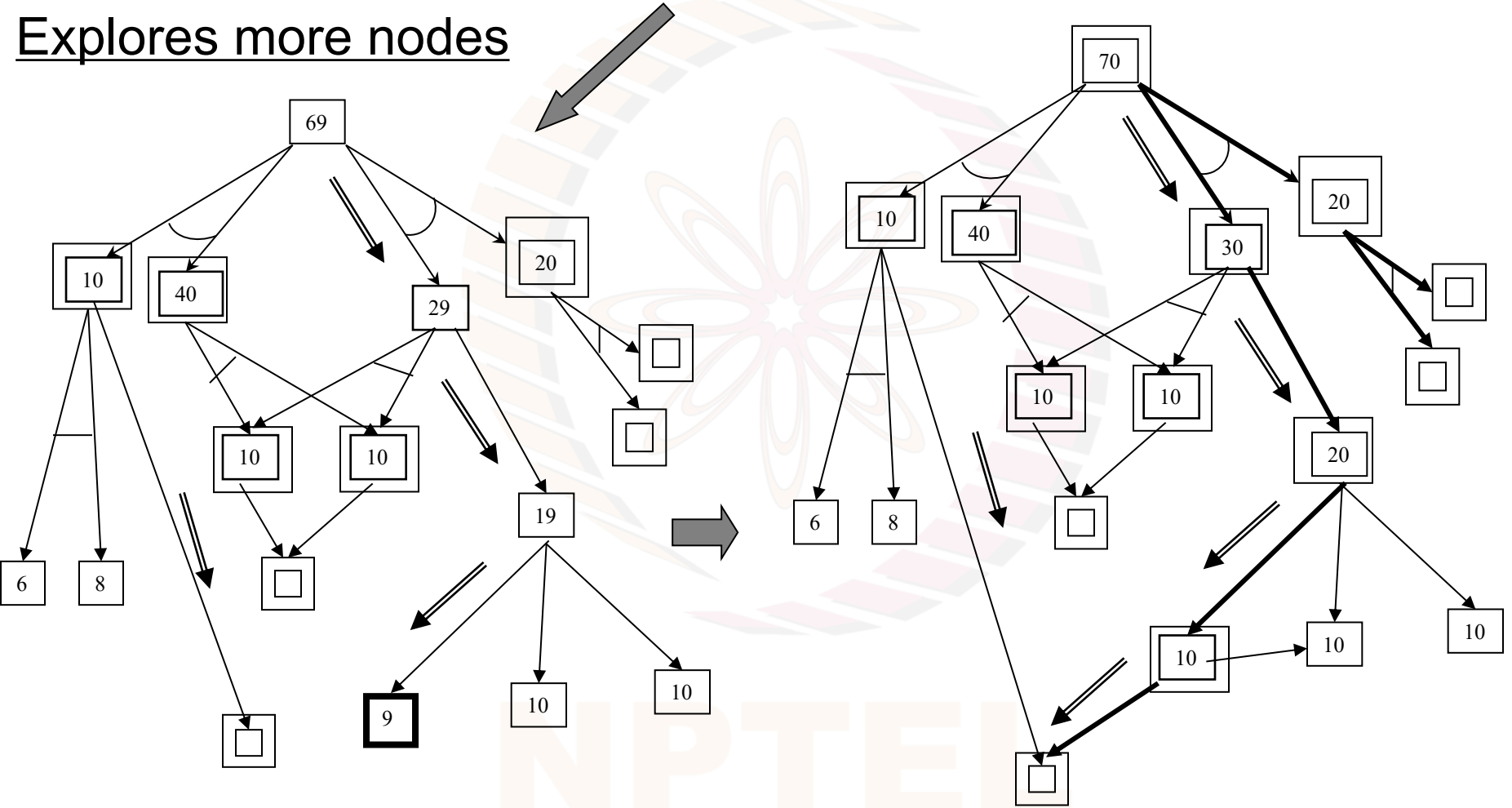
Edge cost = 10






59
6
33
29
20
3
10
19
9
10
10
66
6
40
29
20
10
10
19
9
10
10

Explores more nodes



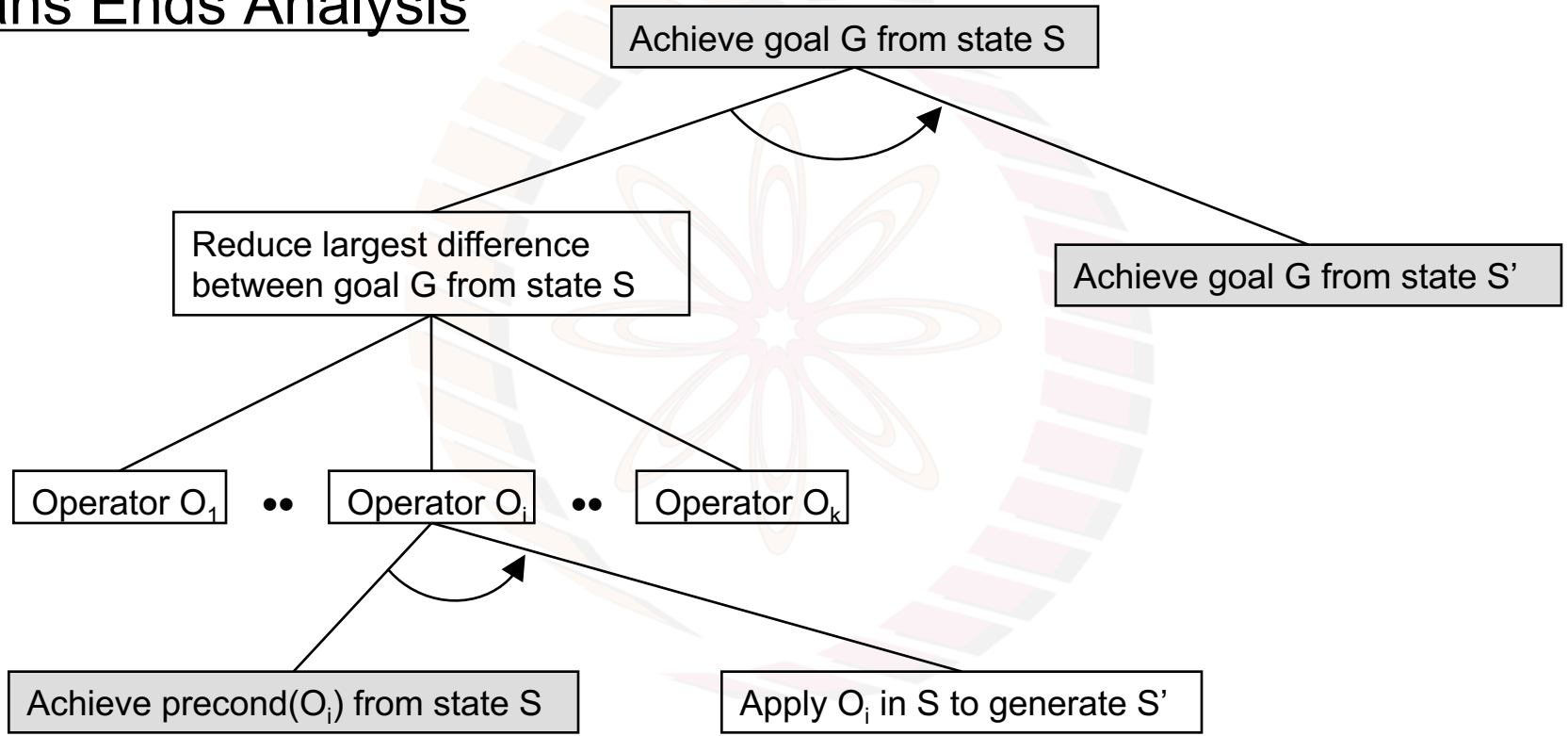
Means Ends Analysis

In their seminal work on *Human Problem Solving*, Newell and Simon proposed a general purpose strategy for problem solving, which they call the *General Problem Solver* (GPS). GPS encapsulated the heuristic approach which they called *Means Ends Analysis*.

- compare the current state with the desired state, and
- list out the *differences* between them
- evaluate the differences in terms of magnitude (in some way)
- consult an *operator-difference table*
- reduce *largest* (most important) *differences first*

- the *differences* characterize the *ends* that need to be achieved
- the *operators* define the *means* of achieving those ends.

Means Ends Analysis



The MEA strategy generates an AND/OR tree by replicating the this structure below the shaded nodes when a recursive call is made.

End : AO*