LAO* Paper Presentation

Jonathan Eidelman
CSC2542
University of Toronto
Fall 2016

Acknowledgements

The slides on LAO* are those of Pascal Poupart.

The slides on AO* are those of Gholamreza Ghassem-Sani.

Thank you to both researchers for sharing their slides on the web.

Module 9
LAO*

CS 886 Sequential Decision Making and Reinforcement Learning

University of Waterloo

Large State Space

- Value Iteration, Policy Iteration and Linear Programming
  - Complexity at least quadratic in $|S|$

- Problem: $|S|$ may be very large
  - Queuing problems: infinite state space
  - Factored problems: exponentially many states

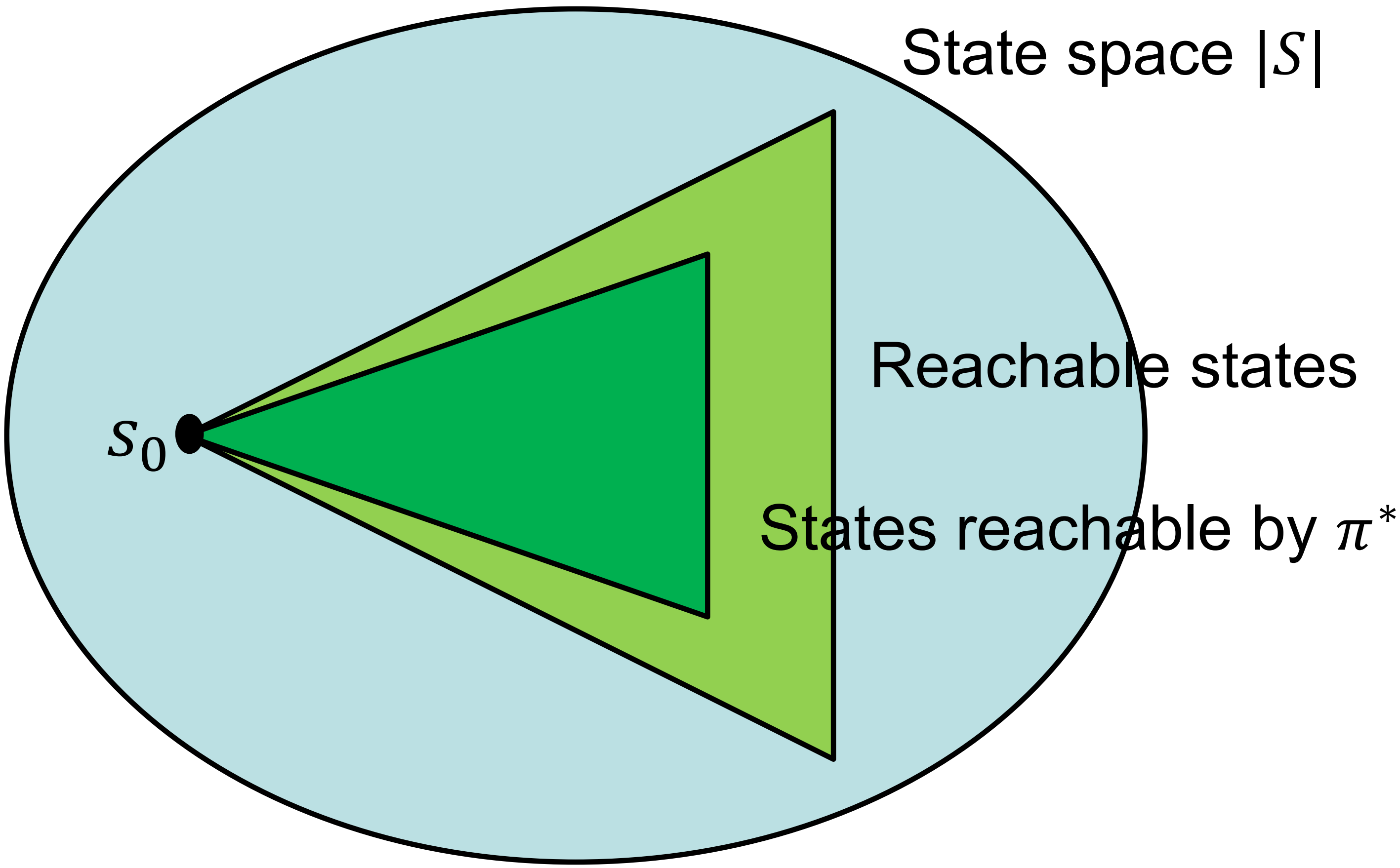
Mitigate Size of State Space

- Two ideas:

- Exploit initial state
  - Not all states are reachable

- Exploit heuristic h
  - approximation of optimal value function
  - usually an upper bound $h(s) \geq V^*(s)\ \forall s$

State Space



LAO* Algorithm

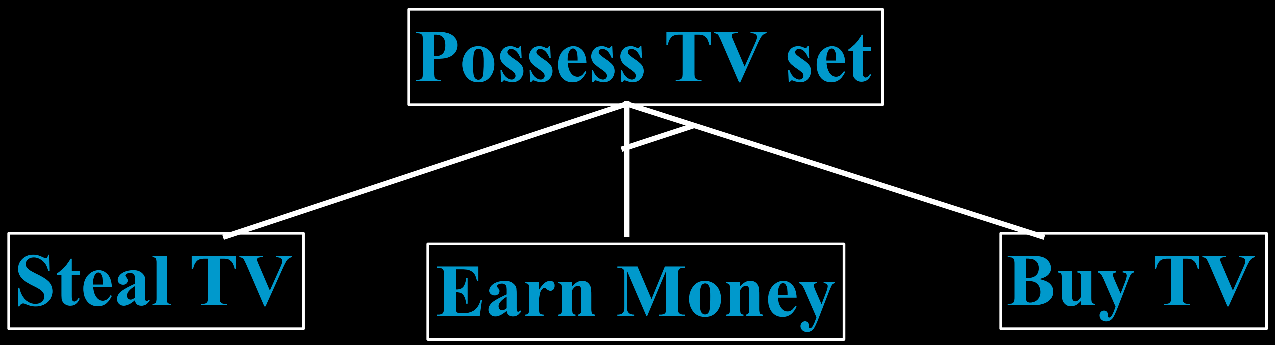
- Related to
  - A*: path heuristic search
  - AO*: tree heuristic search
  - LAO*: cyclic graph heuristic search
- LAO* alternates between
  - State space expansion
  - Policy optimization
    - value iteration, policy iteration, linear programming

Slides by: Gholamreza Ghassem-Sani

AO* REVIEW

AND/OR graphs

- Some problems are best represented as achieving subgoals, some of which achieved simultaneously and independently (AND)
- Up to now, only dealt with OR options



Searching AND/OR graphs

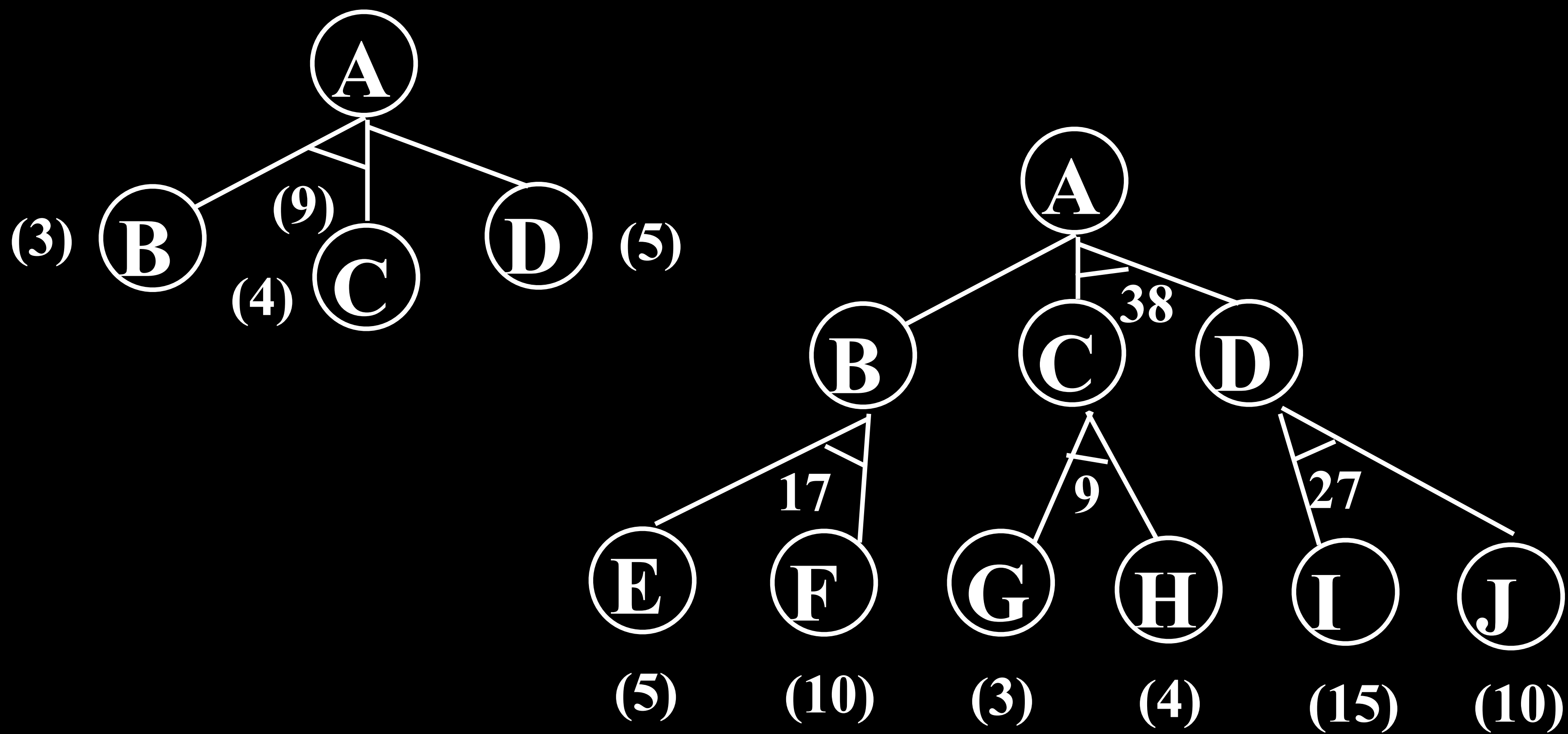
- A solution in an AND-OR tree is a *sub tree* whose *leafs* are included in the goal set

- Cost function: sum of costs in AND node
  $f(n) = f(n_1) + f(n_2) + \ldots. + f(n_k)$

- How can we extend A* to search AND/OR trees? The AO* algorithm.

AND/OR search

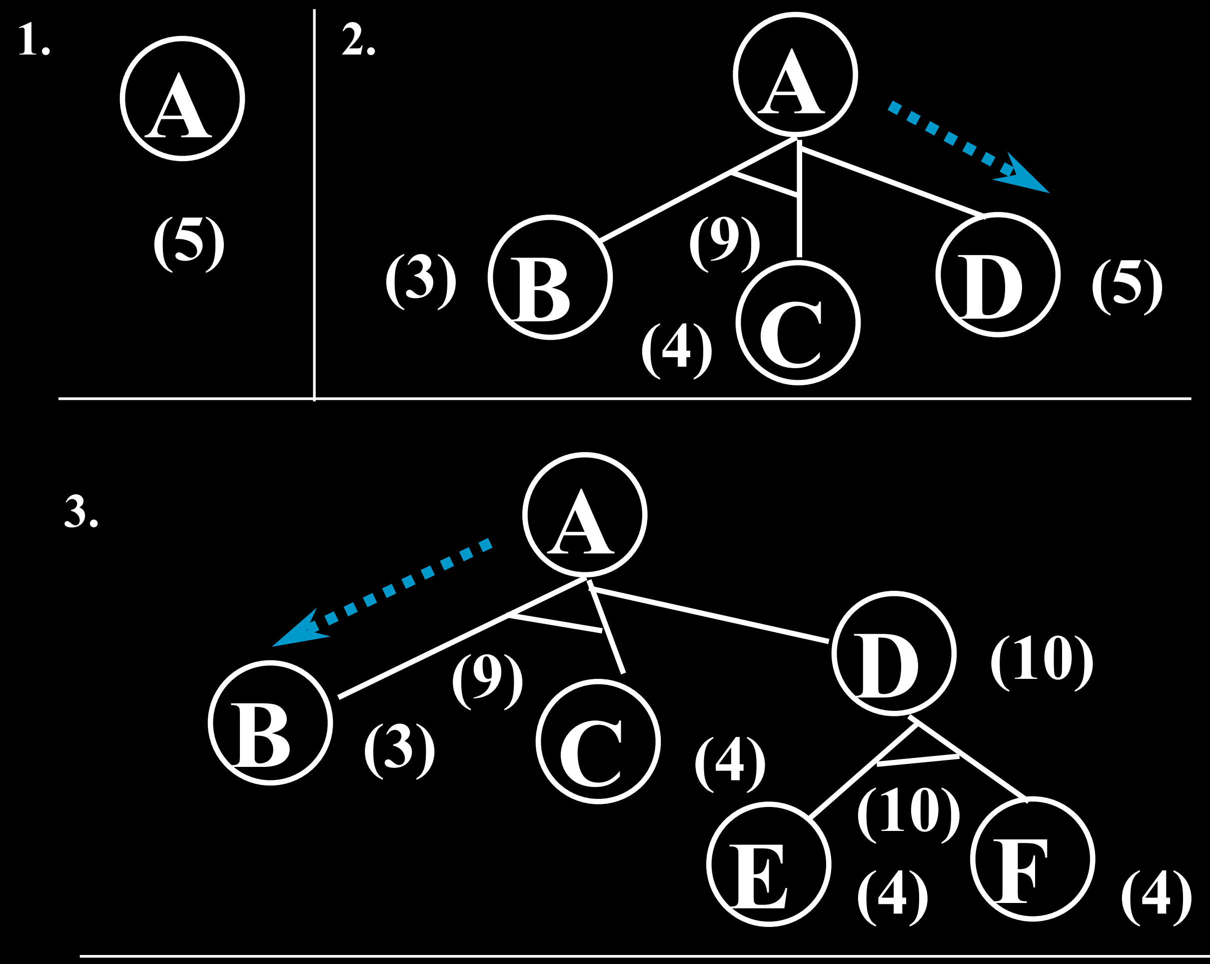
- We must examine several nodes simultaneously when choosing the next move



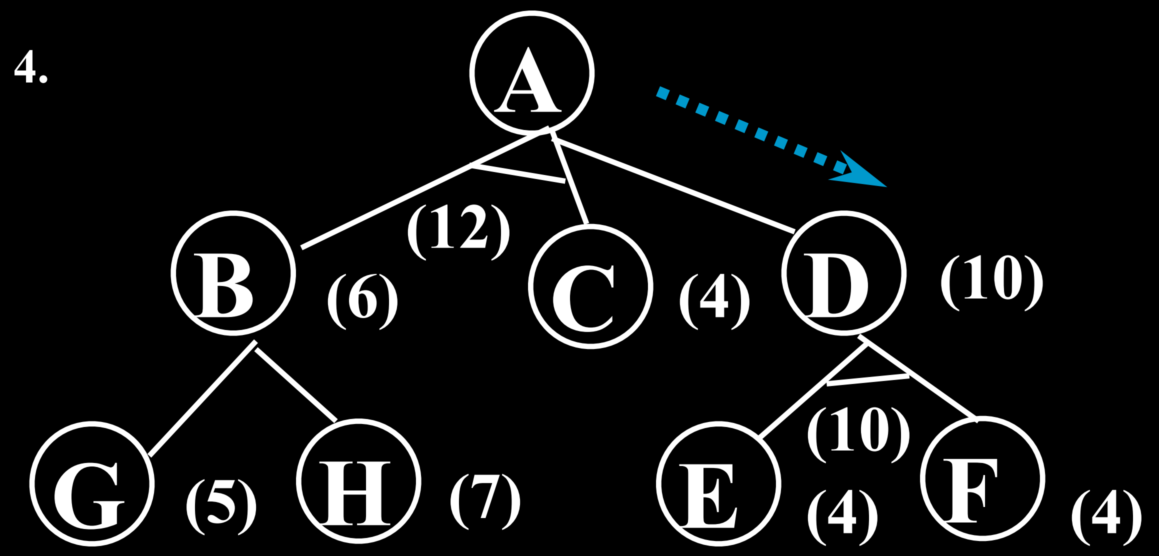
AND/OR Best-First-Search

- Traverse the graph (from the initial node) following the best current path.
- Pick one of the unexpanded nodes on that path and expand it. Add its successors to the graph and compute f for each of them
- Change the expanded node's f value to reflect its successors. Propagate the change up the graph.
- Reconsider the current best solution and repeat until a solution is found

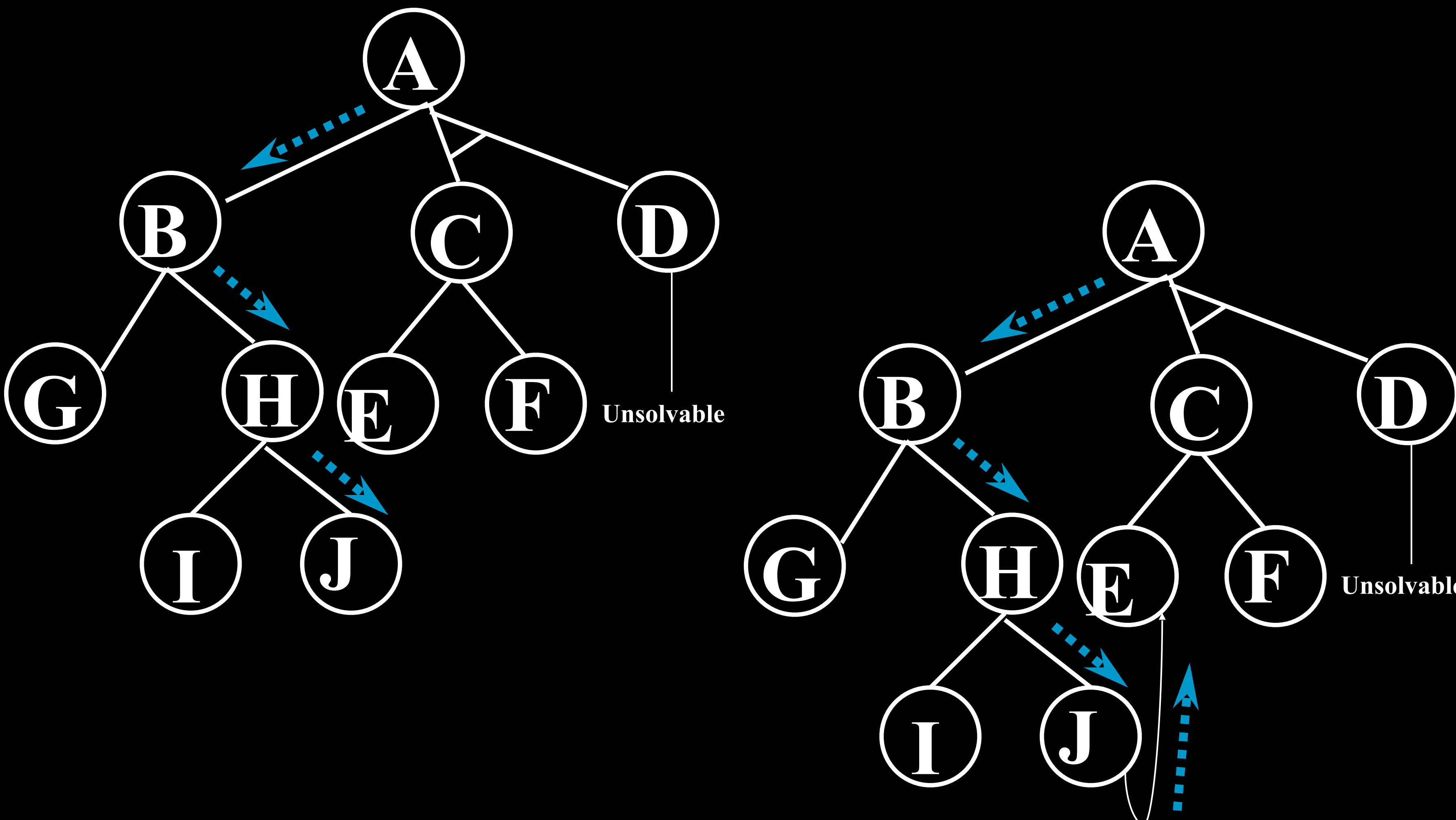
AND/OR Best-First-Search example



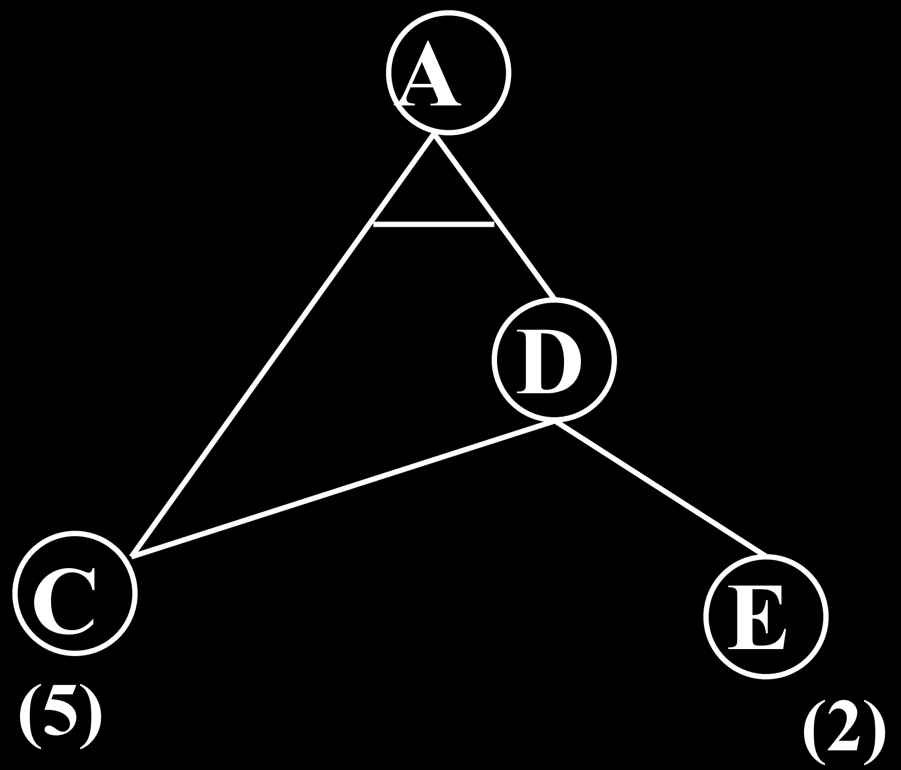
AND/OR Best-First-Search example



A Longer path may be better



Interacting Sub goals



AO* algorithm

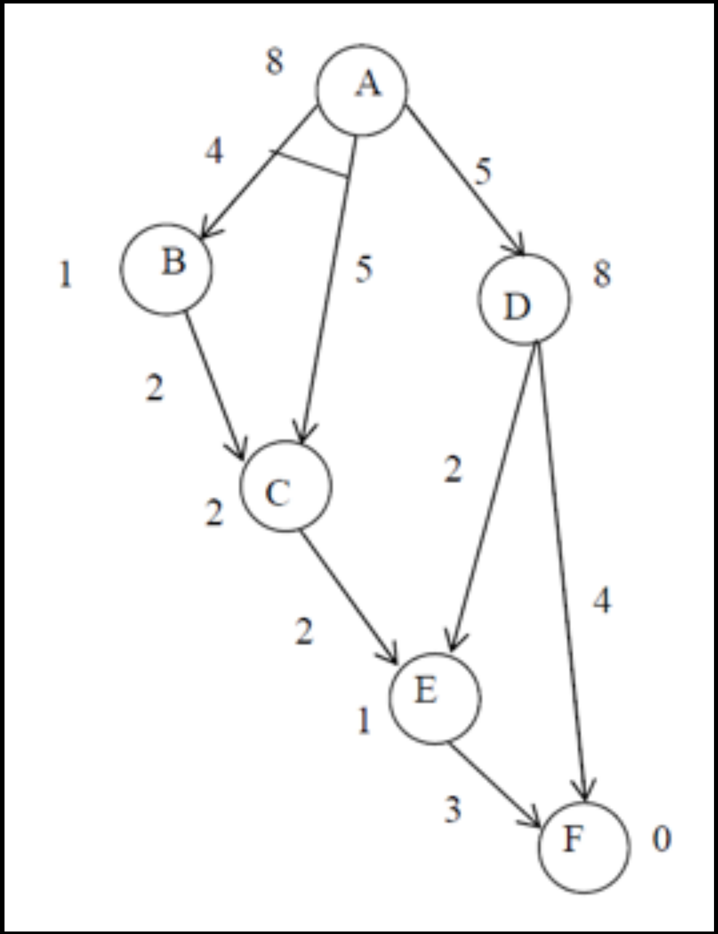
1. Let *G* be a graph with only starting node *INIT.*
2. Repeat the followings until INIT is labeled SOLVED or h(INIT) > FUTILITY
   a) *Select an unexpanded node from the most promising path from INIT (call it NODE)*
   b) Generate successors of NODE. If there are none, set h(NODE) = FUTILITY (i.e., NODE is unsolvable); otherwise for each SUCCESSOR that is not an ancestor of NODE do the following:
      i. Add SUCCESSSOR to G.
      ii. If SUCCESSOR is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0.
      iii. If SUCCESSPR is not a terminal node, compute its h

AO* algorithm (Cont.)

c) Propagate the newly discovered information up the graph by doing the following: let S be set of SOLVED nodes or nodes whose h values have been changed and need to have values propagated back to their parents. Initialize S to Node. Until S is empty repeat the followings:

   i. Remove a node from S and call it CURRENT.
   ii. Compute the cost of each of the arcs emerging from CURRENT. Assign minimum cost of its successors as its h.
   iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost in step ii
   iv. Mark CURRENT as SOLVED if all of the nodes connected to it through new labeled arc have been labeled SOLVED
   v. If CURRENT has been labeled SOLVED or its cost was just changed, propagate its new cost back up through the graph. So add all of the ancestors of CURRENT to S.

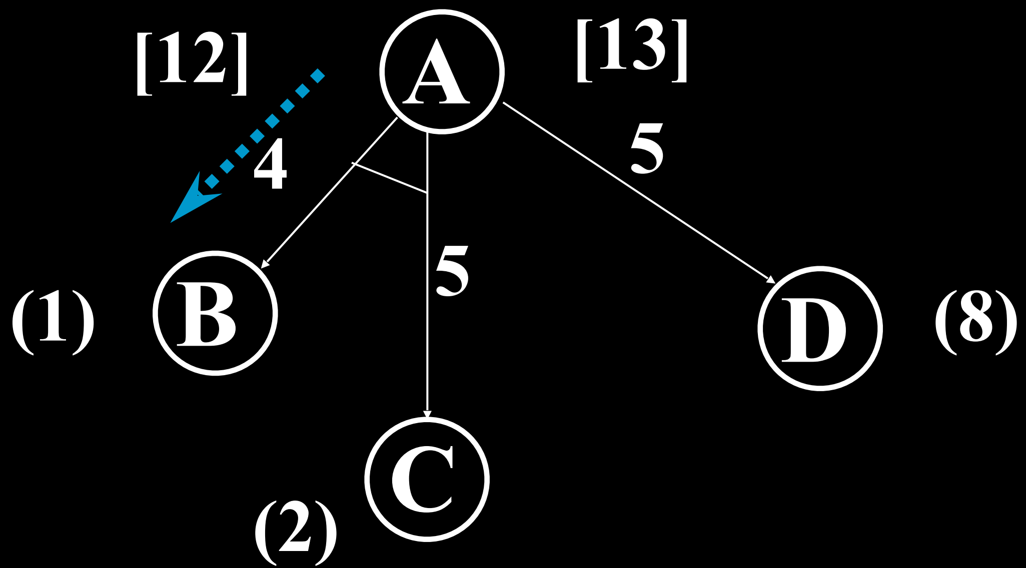
An Example



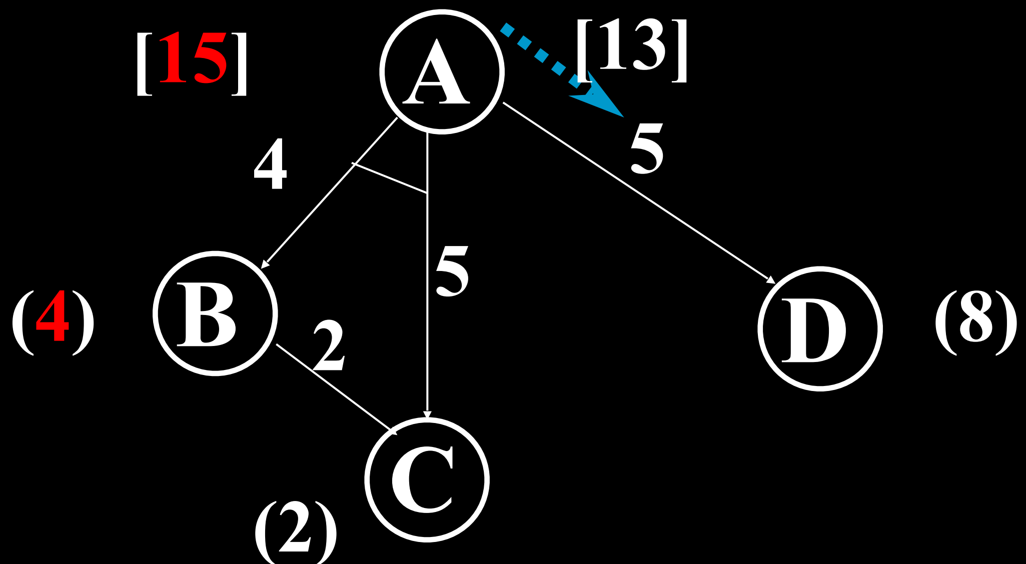
An Example

(8) A

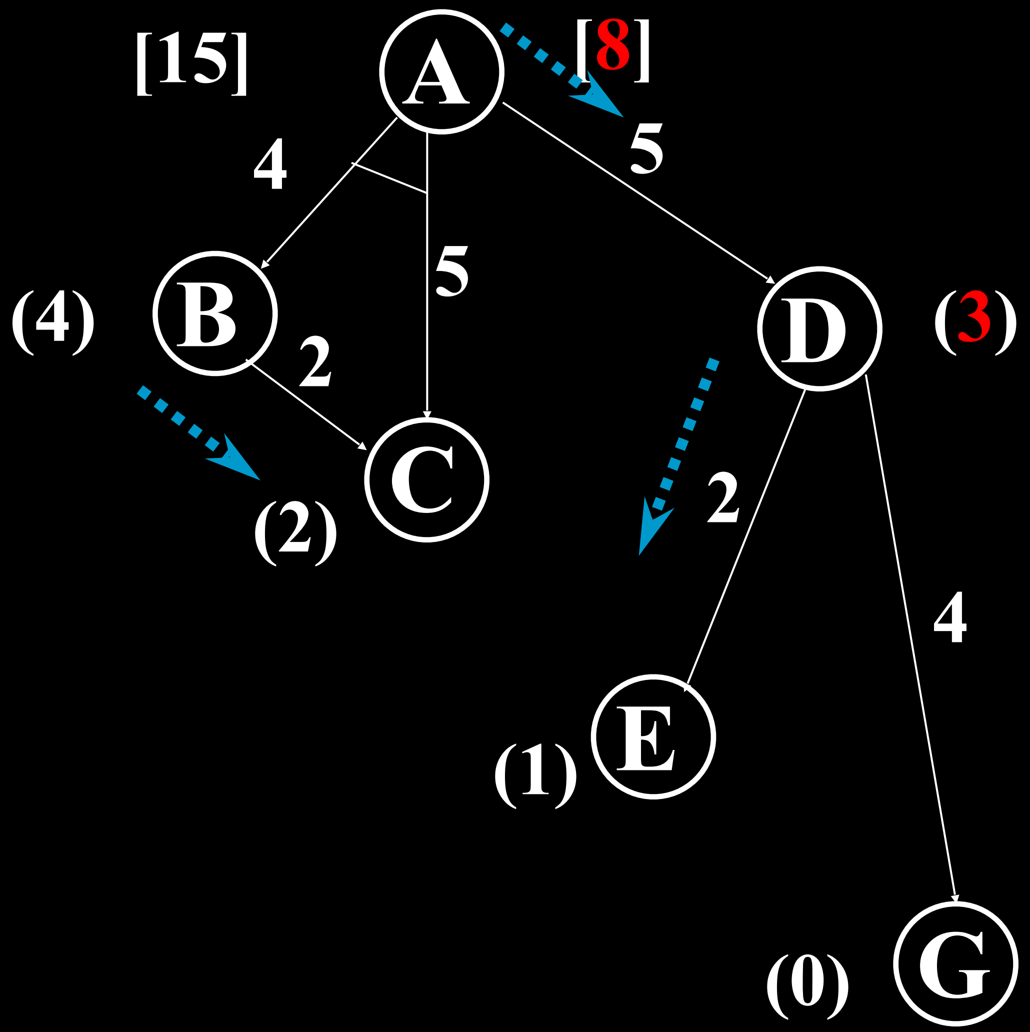
An Example



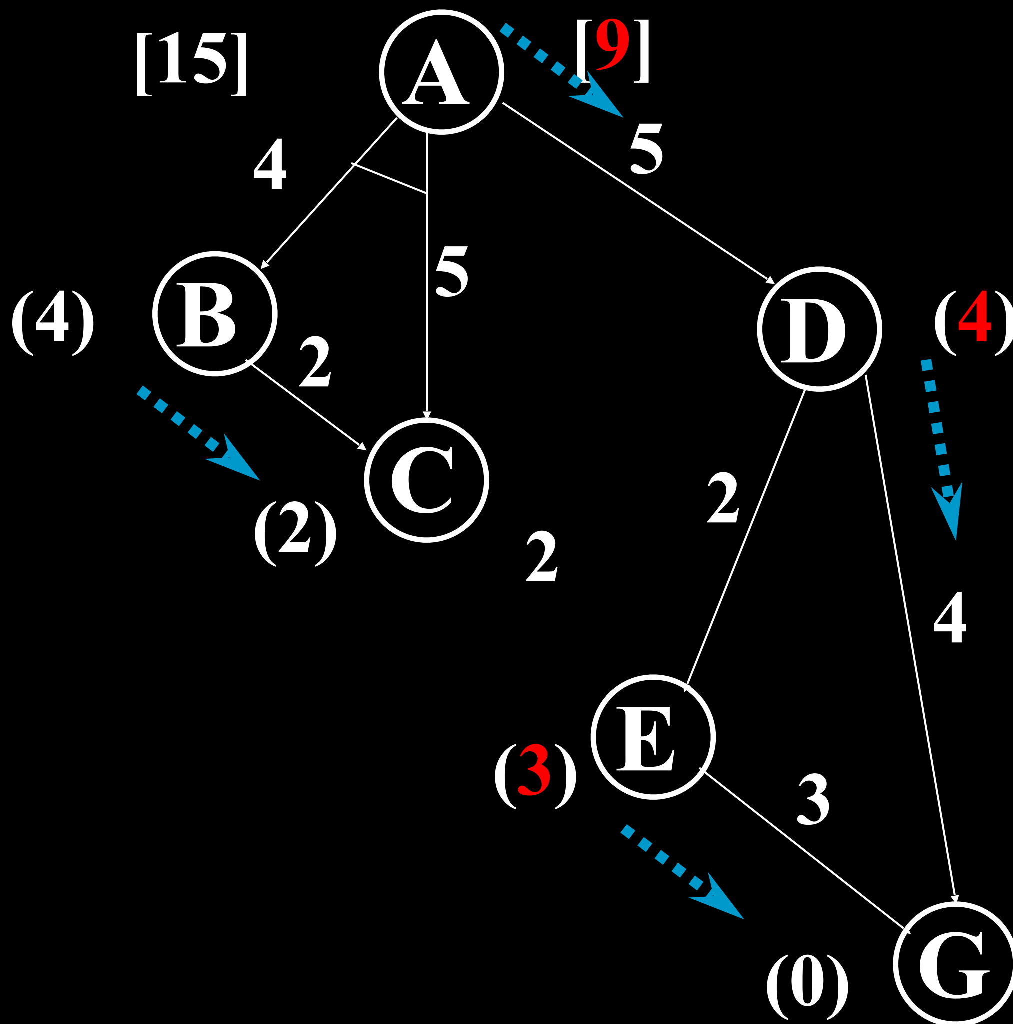
An Example



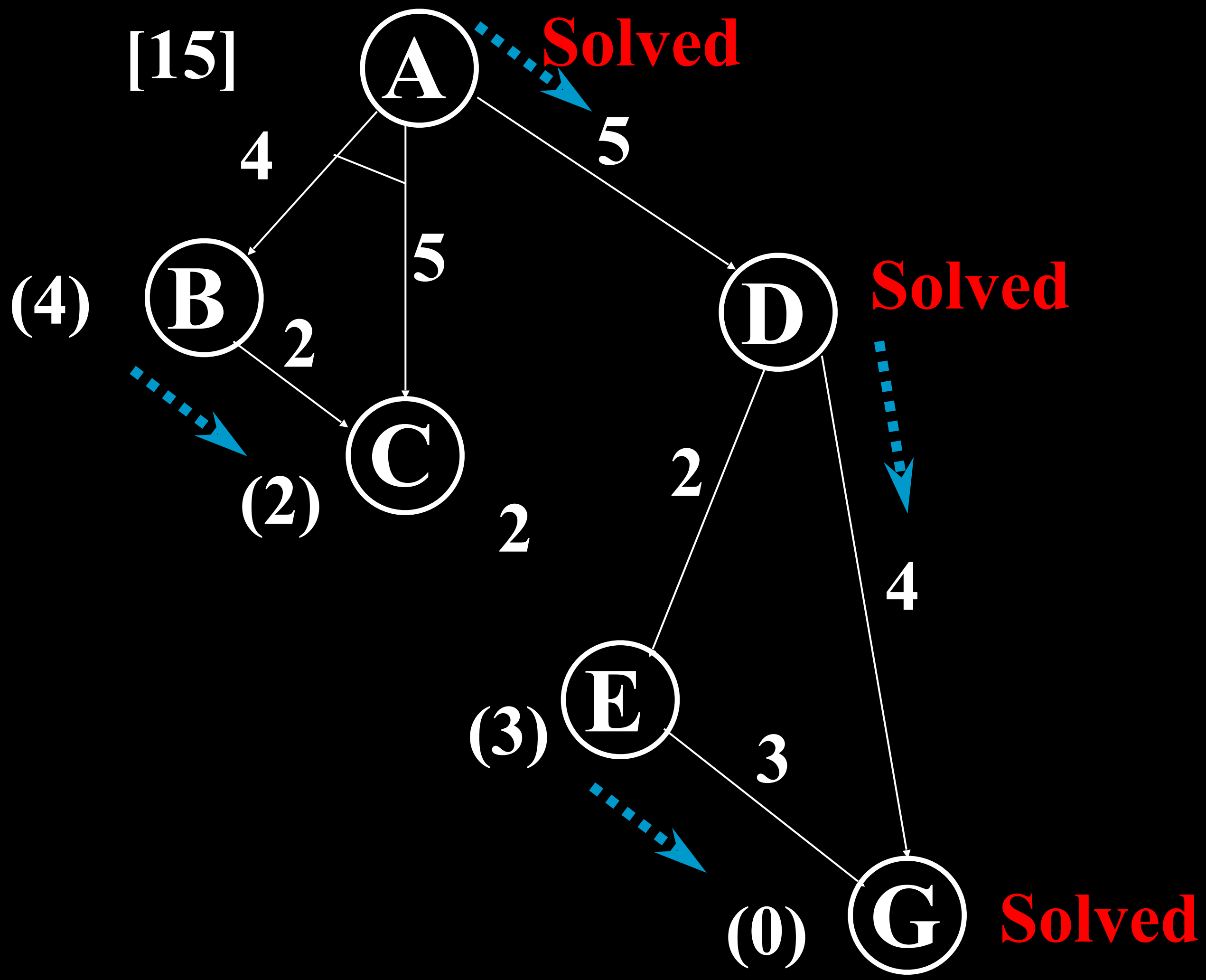
An Example



An Example



An Example



Real Time A*

- Considers the cost (> 0) for switching from one branch to another in the search
- Example: path finding in real life

11	4	1		2	7	16
F	D	B	A	C	E	G

f(B) = 1 + 1 = 2	(A)	f(C) = 1 + 2 = 3
f(D) = 1 + 4 = 5	(B)	f(A) = 1 + 3 = 4
f(B) = 1 + 5 = 6	(A)	f(C) = 1 + 2 = 3
f(A) = 1 + 6 = 7	(C)	f(E) = 1 + 7 = 8
f(B) = 1 + 5 = 6	(A)	f(C) = 1 + 8 = 9
f(D) = 1 + 4 = 5	(B)	f(A) = 1 + 9 = 10
f(F)=1+11= 12	(D)	f(B) = 1 + 10 = 11

Another Example

Current State = S

f(A) = 3 + 5 = 8

f(B) = 2 + 4 = 6

Current State = B

f(S) = 2 + 8 = 10

f(A) = 4 + 5 = 9

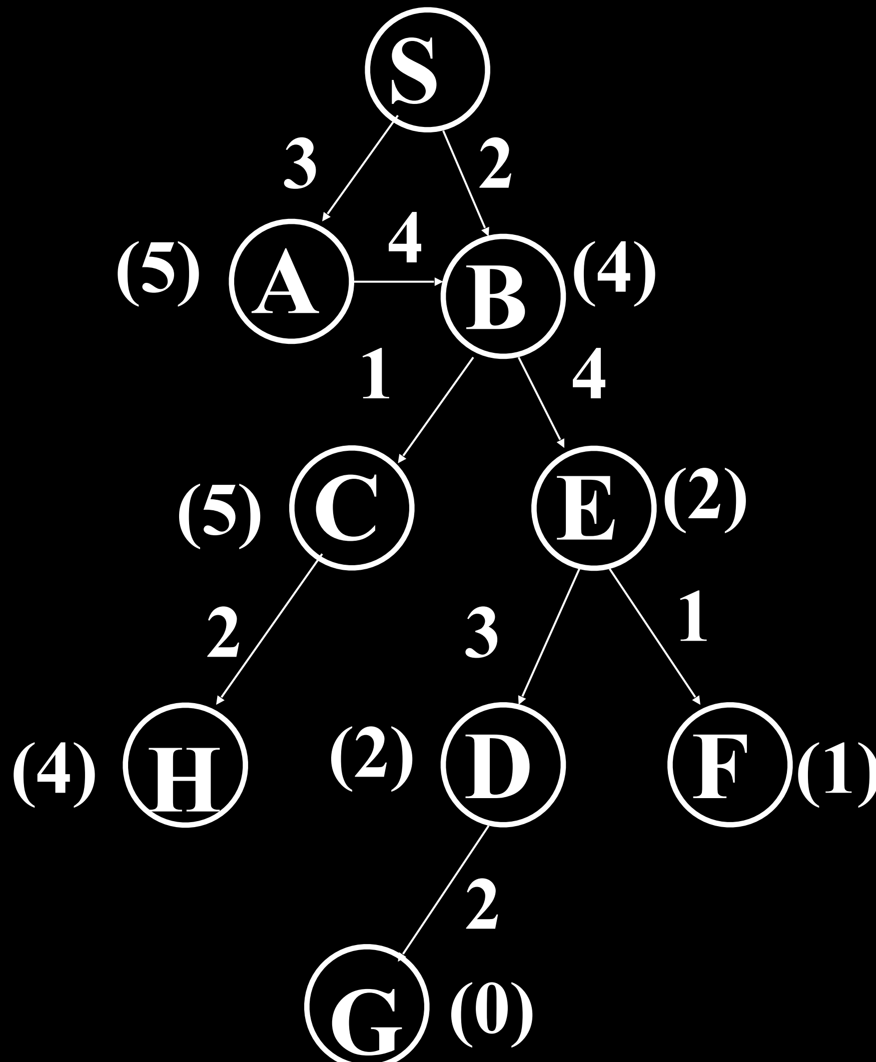
f(C) = 1 + 5 = 6

f(E) = 4 + 2 = 6

Current State = C

f(H) = 2 + 4 = 6

f(B) = 1 + 6 = 7



Another Example

Current State = H

f(C) = 2 + 7 = 9

Current State = C

f(B) = 1 + 6 = 7

f(H) = ∞

Current State = B

f(S) = 2 + 8 = 10

f(A) = 4 + 5 = 9

f(E) = 4 + 2 = 6

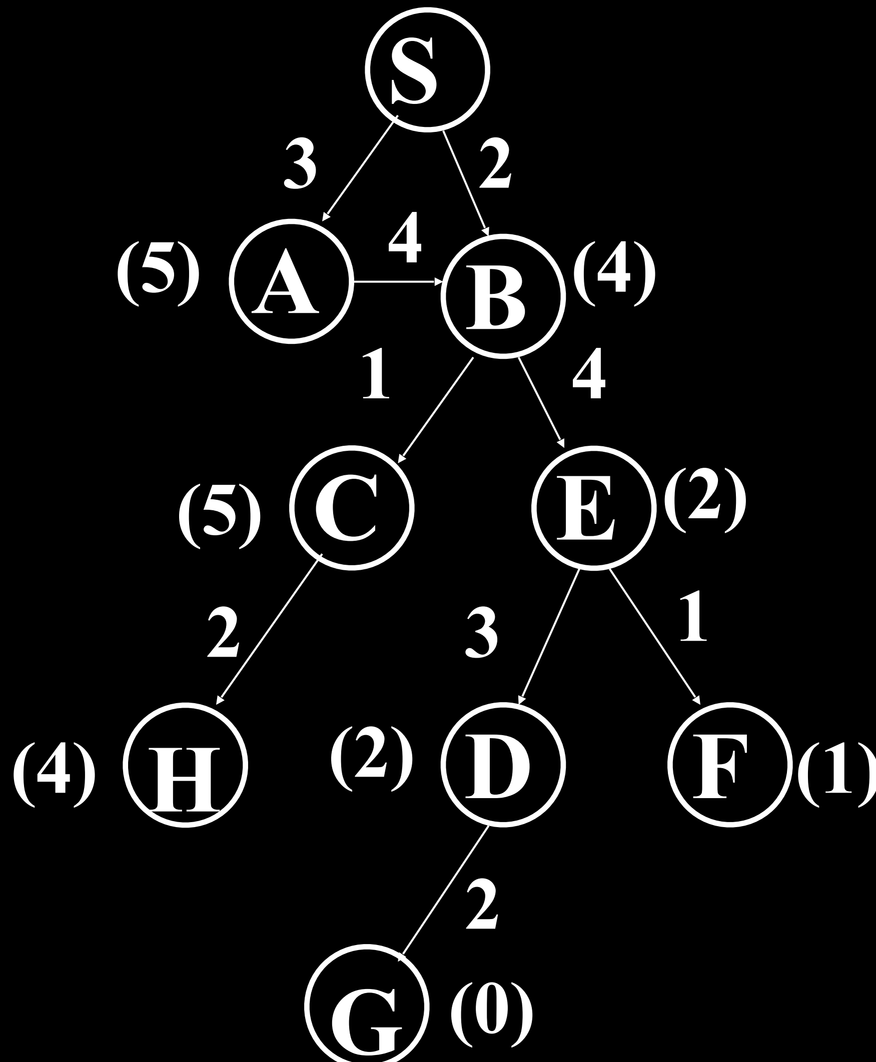
f(C) = ∞

Current State = E

f(B) = 4 + 9 = 13

f(D) = 3 + 2 = 5

f(F) = 1 + 1 = 2



Another Example

Current State = F

f(E) = 1 + 5 = 6

Current State = E

f(D) = 3 + 2 = 5

f(B) = 4 + 9 = 13

f(F) = ∞

Current State = D

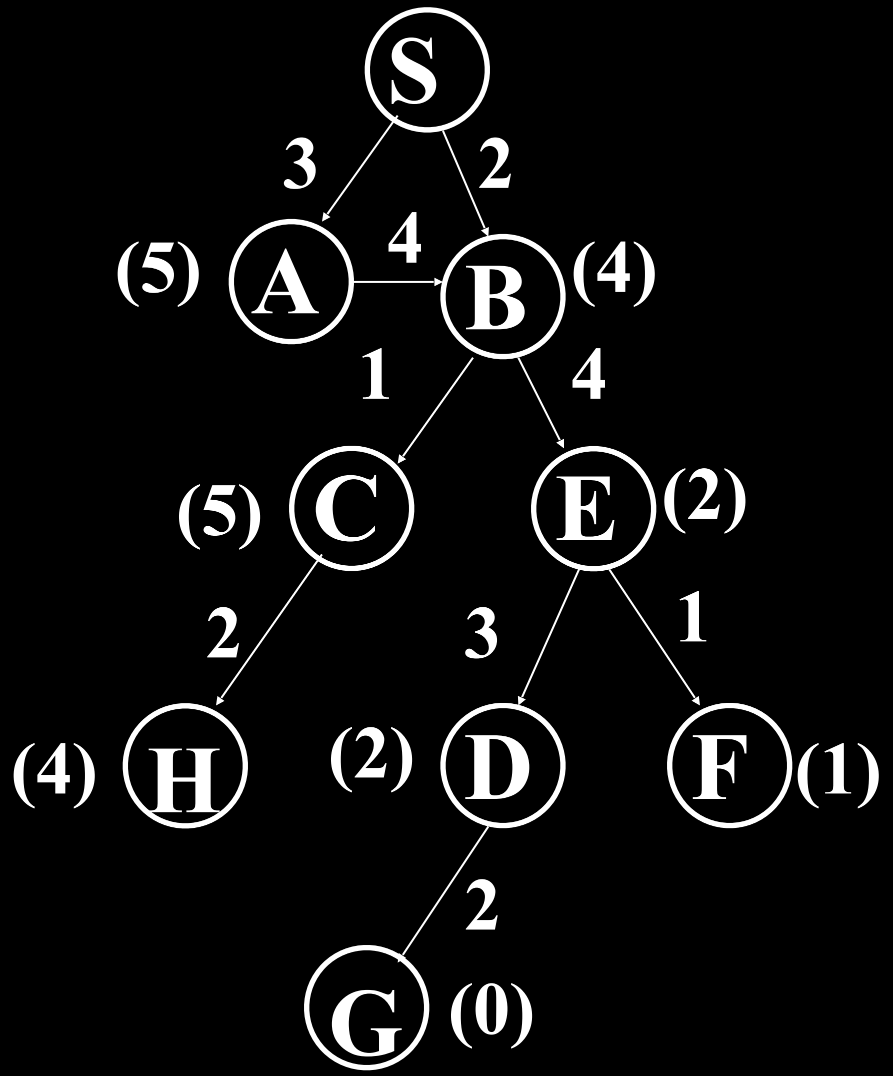
f(G) = 2 + 0 = 2

f(E) = 3 + 13 = 16

Visited Nodes =

S, B, C, H, C, B, E, F, E, D, G

Path = S, B, E, D, G



Terminology

- S: state space
- $S_E \subseteq S$: envelope
  - Growing set of states
- $S_T \subseteq S_E$: terminal states
  - States whose children are not in the envelope
- $S_{s_0}^{\pi} \subseteq S_E$: states reachable from s_0 by following π
- $h(s)$: heuristic such that $h(s) \geq V^*(s)\ \forall s$
  - E.g., $h(s) = \max_{s,a} R(s,a)/(1-\gamma)$

LAO* Algorithm

LAO*(MDP, heuristic h)

$S_E \leftarrow \{s_0\}$, $S_T \leftarrow \{s_0\}$

Repeat

Let $R_E(s,a) = \begin{cases} h(s) & s \in S_T \\ R(s,a) & \text{otherwise} \end{cases}$

Let $T_E(s'|s,a) = \begin{cases} 0 & s \in S_T \\ \Pr(s'|s,a) & \text{otherwise} \end{cases}$

Find optimal policy π for $\langle S_E, R_E, T_E \rangle$

Find reachable states $S_{s_0}^{\pi}$

Select reachable terminal states $\{\mathrm{s}_1, \dots, \mathrm{s}_\mathrm{k}\} \subseteq S_{s_0}^{\pi} \cap S_T$

$S_T \leftarrow (S_T \setminus \{s_1, \dots, s_k\}) \cup (children(\{s_1, \dots, s_k\}) \setminus S_E)$

$S_E \leftarrow S_E \cup children(\{s_1, \dots, s_k\})$

Until $S_{s_0}^{\pi} \cap S_T$ is empty

Efficiency

Efficiency influenced by

1. Choice of terminal states to add to envelope

2. Algorithm to find optimal policy
   - Can use value iteration, policy iteration, modified policy iteration, linear programming
   - Key: reuse previous computation
     - E.g., start with previous policy or value function at each iteration

Convergence

- Theorem: LAO* converges to the optimal policy
- Proof:
  - Fact: At each iteration, the value function V is an upper bound on V^* due to the heuristic function h
  - Proof by contradiction: suppose the algorithm stops, but π is not optimal.
    - Since the algorithm stopped, all states reachable by π are in $S_E \setminus S_T$
    - Hence, the value function V is the value of π and since π is suboptimal then $V < V^*$, which contradicts the fact that V is an upper bound on V^*

Summary

- LAO*
  - Extension of basic solution algorithms (value iteration, policy iteration, linear programming)
  - Exploit initial state and heuristic function
  - Gradually grow an envelope of states
  - Complexity depends on # of reachable states instead of size of state space