Hill Climbing Algorithm

20/01/2025

Koustav Rudra

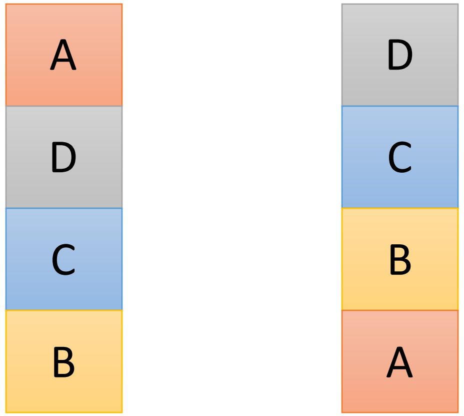
Hill Climbing Algorithm

- Evaluate the INITIAL state
  - If it is GOAL return it
  - Else CURRENT← INITIAL

- Loop until the solution is found or no new operators could be applied to CURRENT:
  - Select an operator that has not been applied to the current state [CURRENT] and apply it to produce new state [NEW]

  - Evaluate NEW:
    - If it is GOAL return it
    - Else If NEW > CURRENT, CURRENT← NEW
    - Else go to Loop

Hill Climbing Algorithm



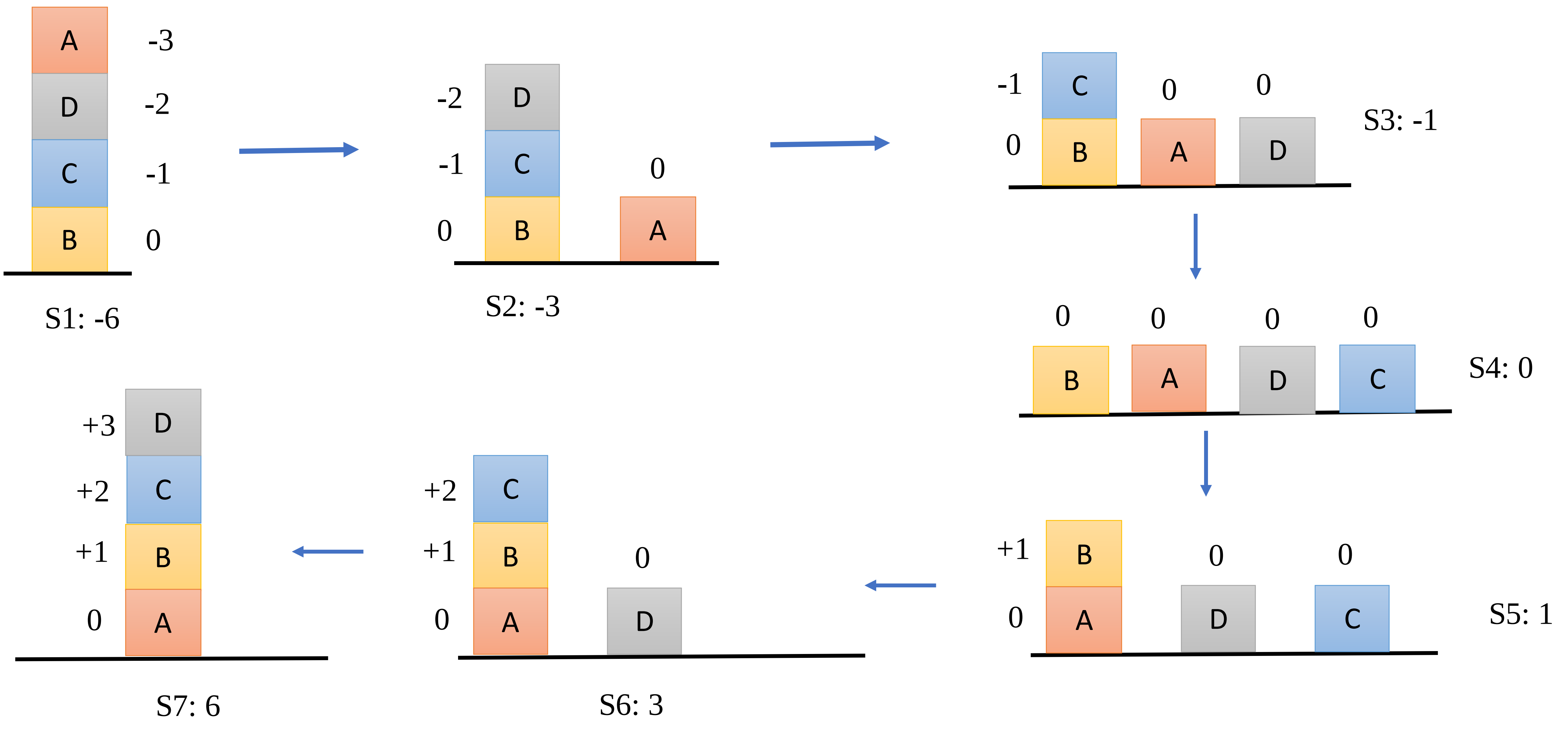
Start Goal

- H(x):
- For each block that has the correct support structure: +1 to every block in the support structure

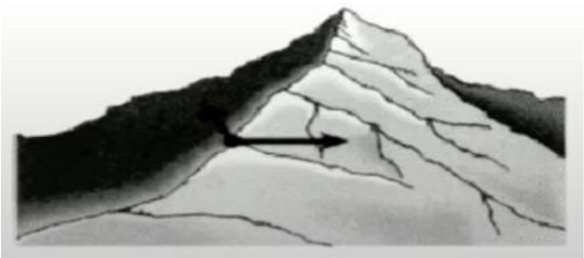
- For each block that has a wrong support structure: -1 to every block in the support structure

Drawbacks

Hill Climbing Algorithm

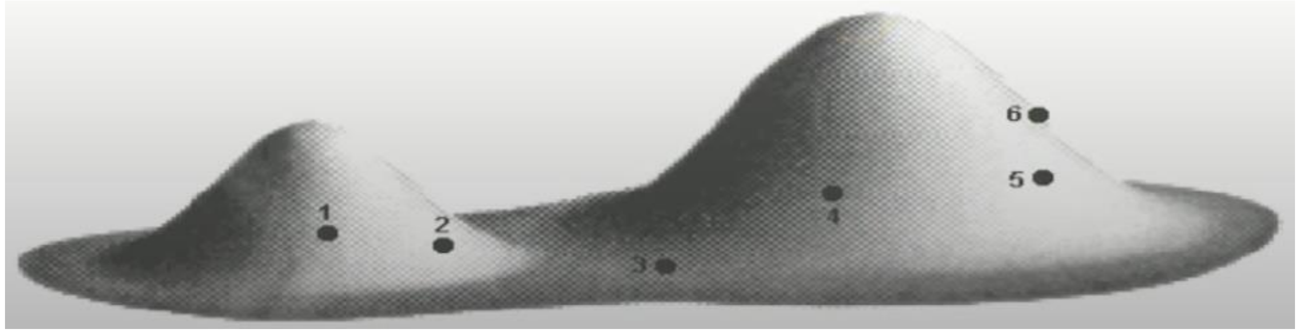


Hill Climbing Algorithm: Drawbacks

- Local Maxima:
  
  - A local maxima is supposed to be global maxima
- Plateaus:
  
  - Area of sear h space where evaluation function is flat
  - Requiring random walk
- Ridge:
  
  - Steep slopes
  - Search direction is not towards the top but towards the side

Drawback: Solution

- In each of the previous cases (local maxima, plateaus, & ridge), the algorithm reaches a point at which no progress is being made



- Solution
  - Random-restart-hill-climbing
  - Random initial states are generated
  - Running each until it halts or makes no discernible progress
  - Best result is chosen

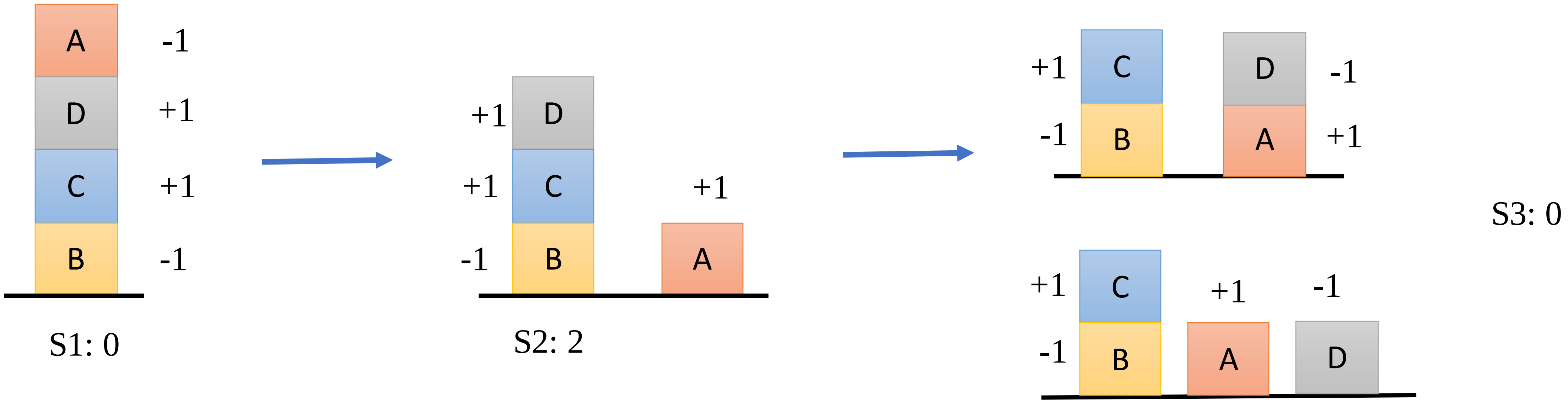
Hill Climbing Algorithm: Disadvantages

- Hill Climbing uses local information:
  - Decides what to do next by looking only at the “immediate” consequences of its choices
  - Will terminate when at local optimum
  - The order of application of operators can make a big difference
- Global information might be encoded in heuristic functions

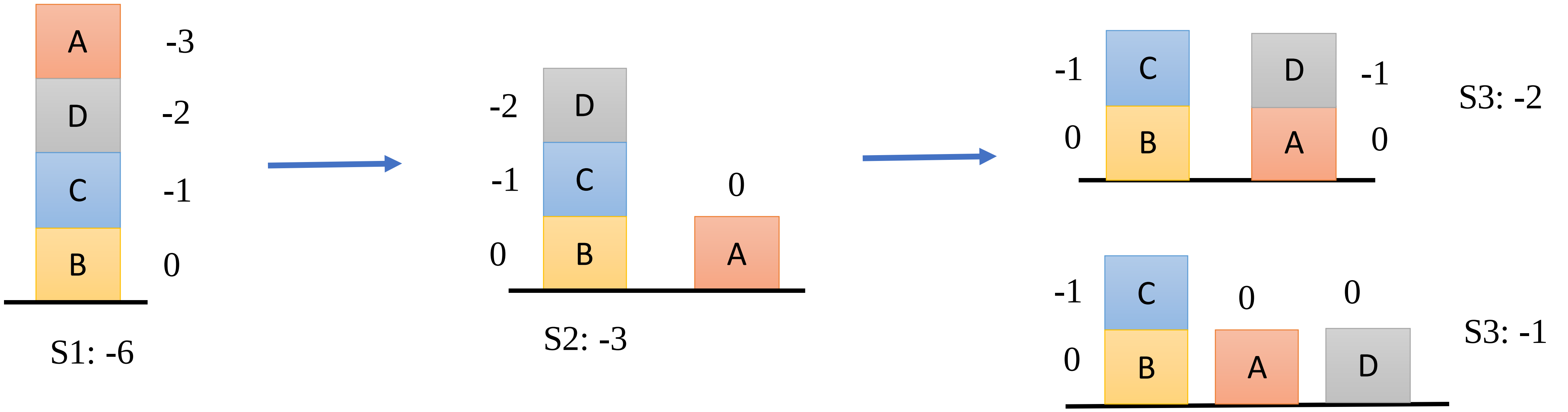
Hill Climbing Algorithm: Disadvantages

- Local Heuristics
  - +1 for each block that is resting on the thing it is supposed to be resting on
  - -1 for each block that is resting on a wrong thing

Hill Climbing Algorithm: Local Heuristics

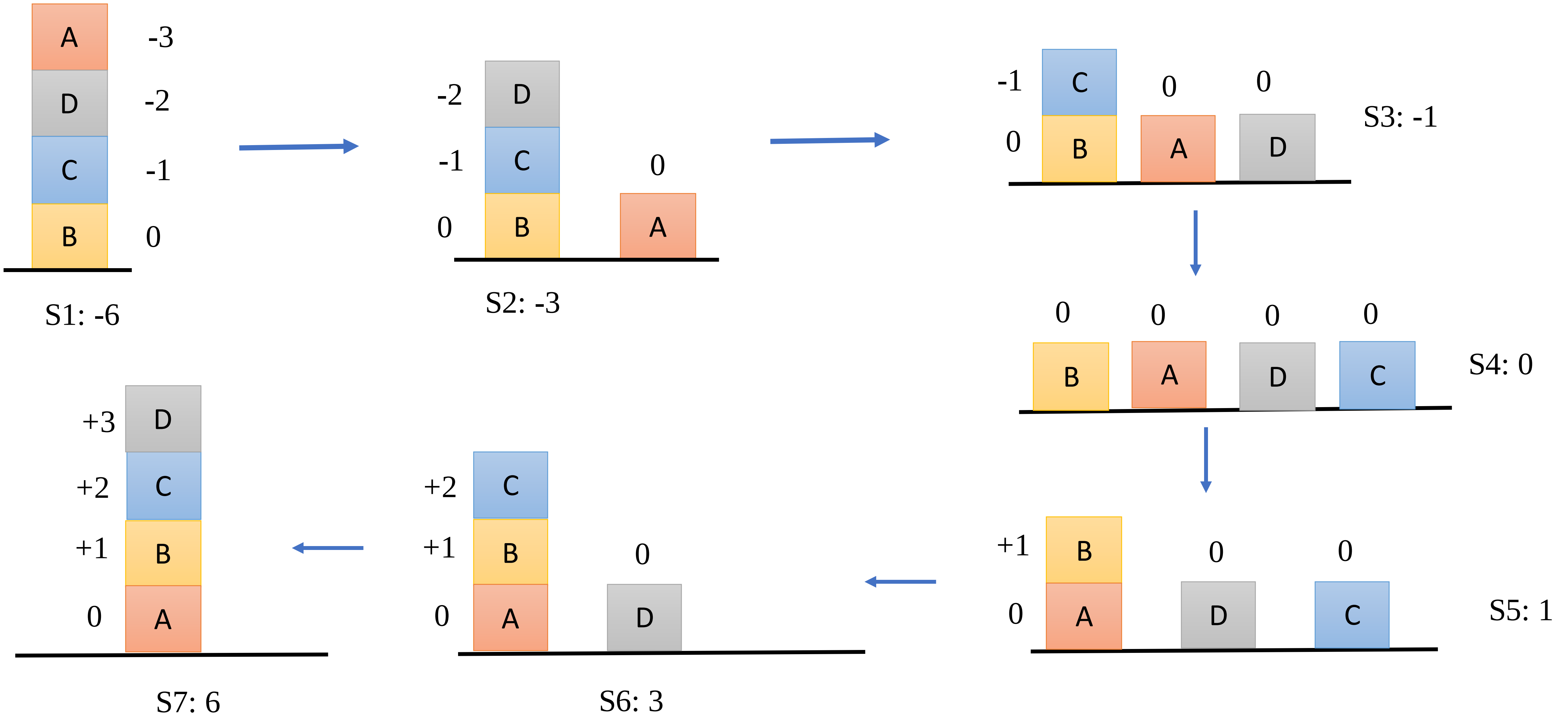


Hill Climbing Algorithm: Global Heuristics



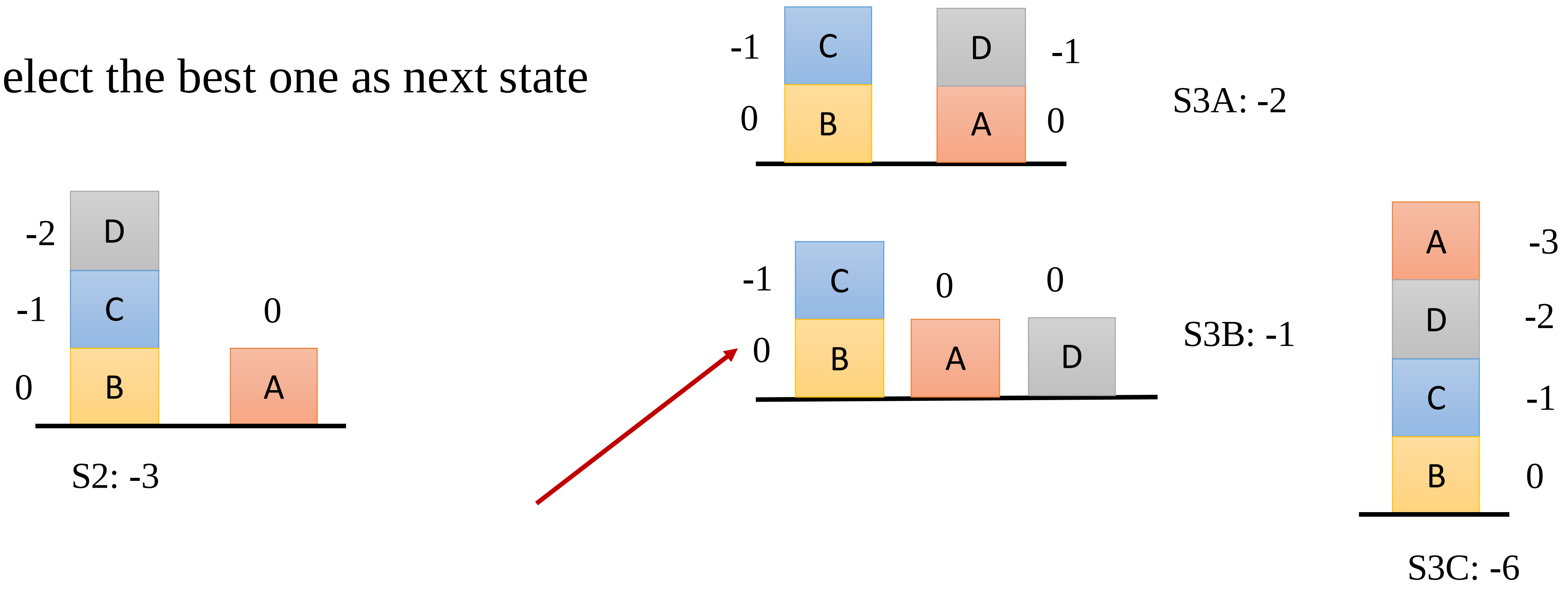
- H(x):
- For each block that has the correct support structure: +1 to every block in the support structure
- For each block that has a wrong support structure: -1 to every block in the support structure
- There is no local maximum
- Takeaway
  - Sometimes changing the heuristic function is all we need

Hill Climbing Algorithm: Global Heuristics



Steepest-Ascent Hill Climbing Algorithm

- Basic Hill Climbing first applies one operator and gets new state
- Steepest-Ascent Hill Climbing considers all the moves from the current state
- Select the best one as next state



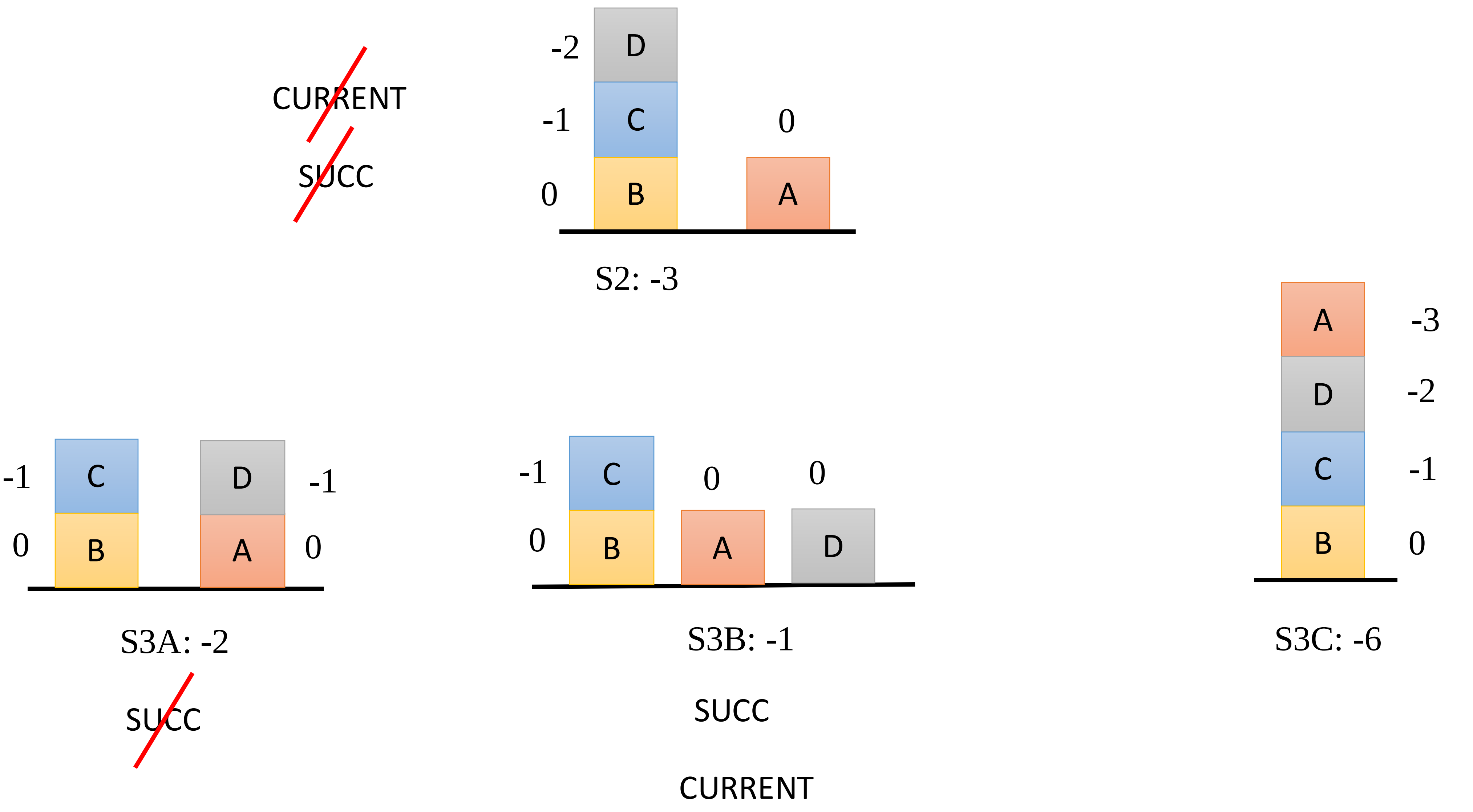
Steepest-Ascent Hill Climbing Algorithm

- Evaluate the INITIAL state
  - If it is GOAL return it
  - Else CURRENT← INITIAL

- Loop until the solution is found or until a complete iteration produces no change to CURRENT:
  - Let SUCC be a state such that any possible successor of the current state will be better than SUCC
  - For each operator that applies to the current state [CURRENT] do
    - Apply the operator and generate a new state [NEW]

    - Evaluate NEW:
      - If it is GOAL return it and quit
      - Else If NEW > SUCC, SUCC← NEW
  - If SUCC>CURRENT
    - CURRENT ← SUCC

Steepest-Ascent Hill Climbing Algorithm



Memory Bound A*

- Whenever $|\text{OPEN} \cup CLOSED|$ approaches M
  - Some of the least promising states are removed

Search with Limited Memory and Time

- Mechanisms for discarding nodes
  - Pruning
  - DFBB
  - MA*

- Mechanisms for redoing nodes
  - IDA*

Domain Relaxation

- A problem with fewer restrictions on the actions is called a **relaxed problem**

- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem

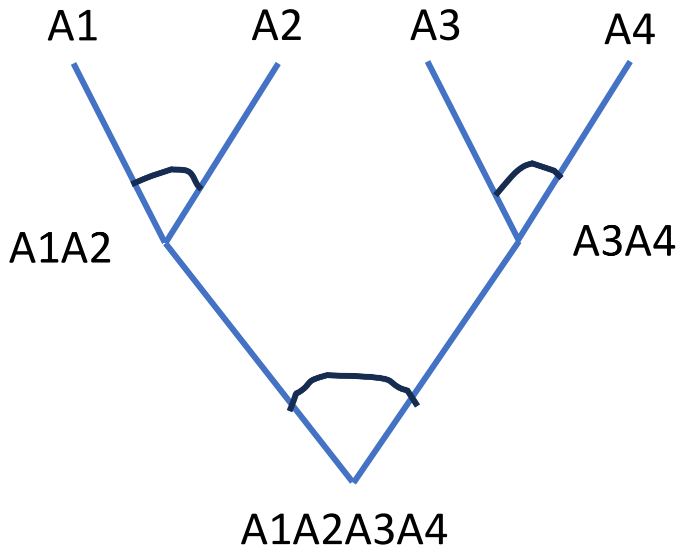
AIFA
Problem Reduction Search

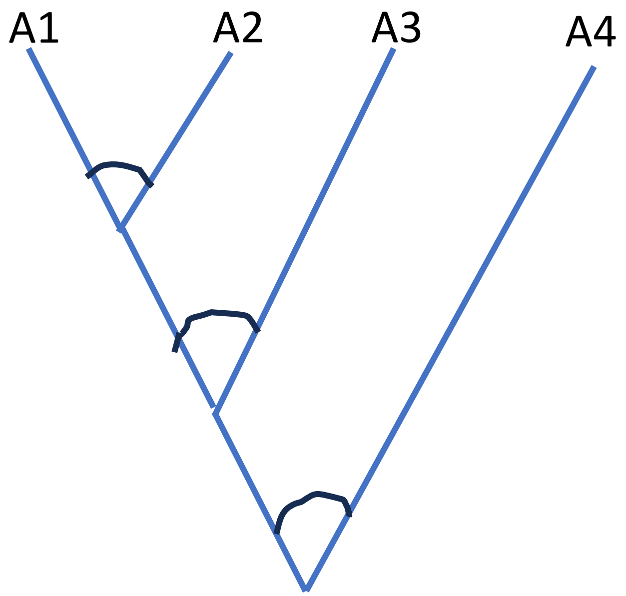
20/01/2025

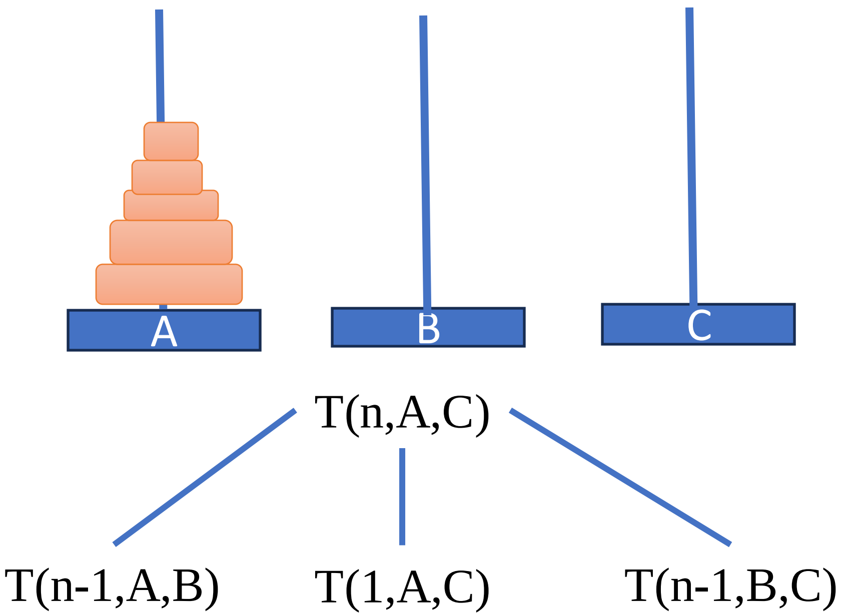
Koustav Rudra

Problem Reduction Search

- Planning how best to solve a problem that can be recursively decomposed into sub-problems in multiple ways
  - Matrix multiplication problem
  - Tower of Hanoi
  - Theorem proving





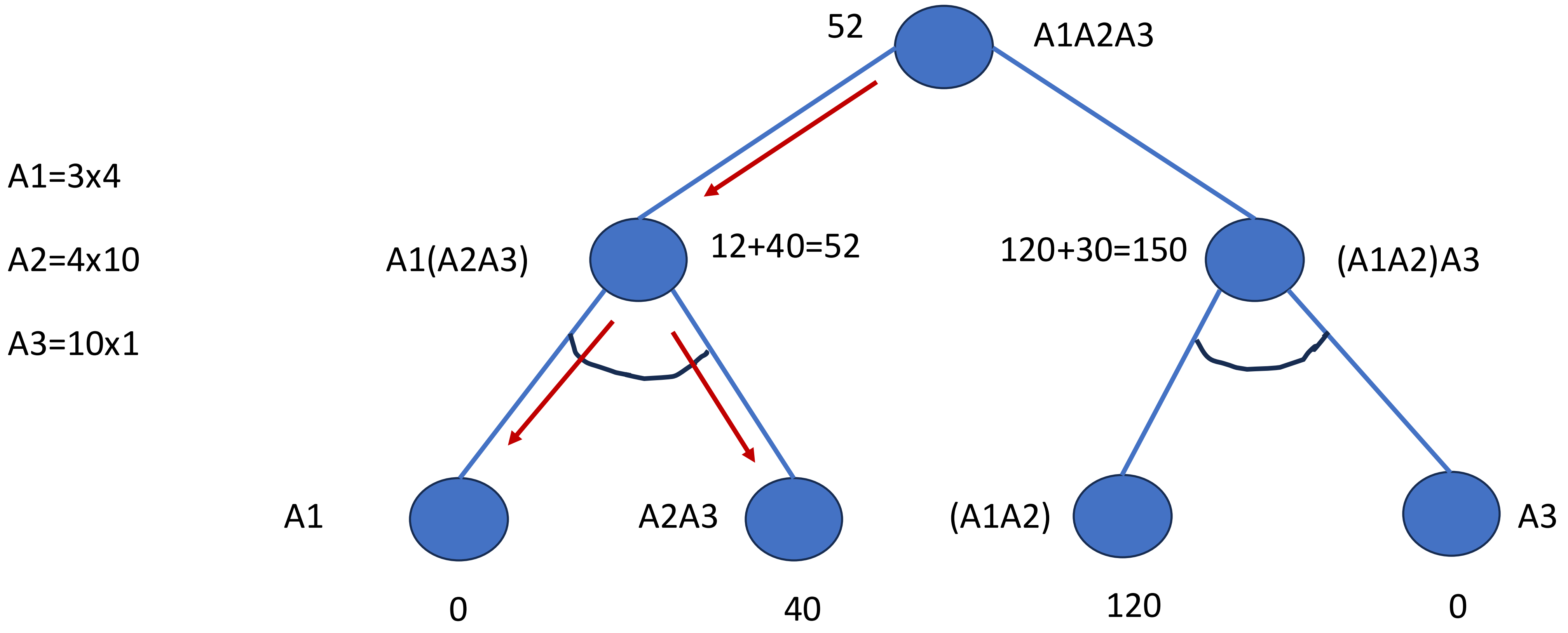


Formulation

- AND/OR Graph
  - An OR node represents a choice between possible decompositions
  - An AND node represents a given decomposition

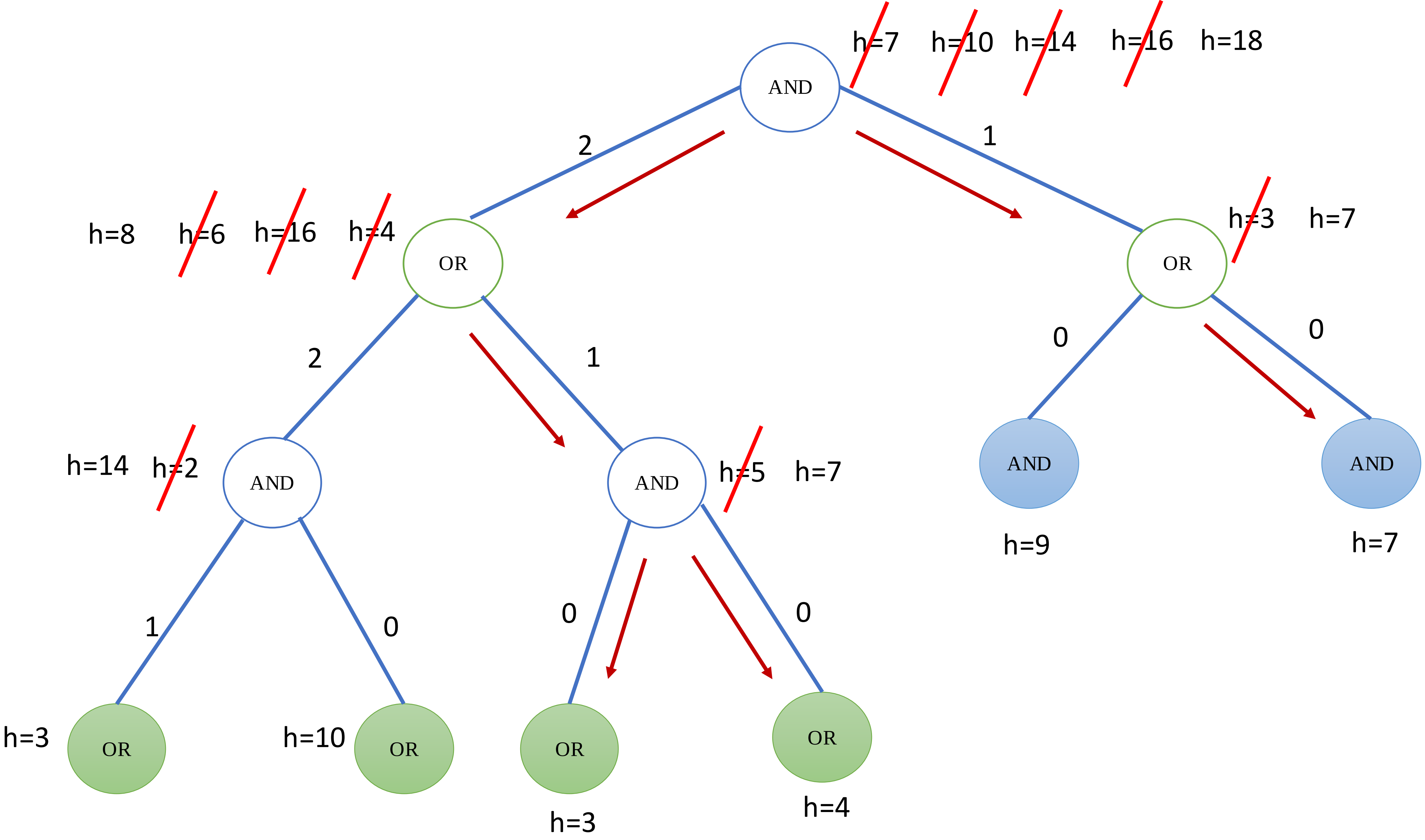
- Game Trees
  - Max/Min nodes
  - Max nodes represent the choice of my opponent
  - Min nodes represent my choice

Each node has a separate optimization criteria



- This is when heuristics is not present

AO*: Example



AND
~~h=7~~ ~~h=10~~ ~~h=14~~ ~~h=16~~ h=18
2
1
OR
h=8 ~~h=6~~ ~~h=16~~ ~~h=4~~
OR
~~h=3~~ h=7
2
1
0
0
AND
h=14 ~~h=2~~
AND
~~h=5~~ h=7
AND
h=9
AND
h=7
1
0
0
0
OR
h=3
OR
h=10
OR
h=3
OR
h=4

Algorithm AO*

- Initialize: Set G*={s}, f(s) = h(s)
  - If s ∈T, terminate and label s as solved

- Terminate:
  - If s is solved, then Terminate

- Select: Select a nonterminal leaf node n from the marked subtree
- Expand:
  - Make explicit the successors of n
  - For each new successor, m:
    - Set f(m)=h(m)
  - If m is terminal, label m solved
- Cost revision: Call cost-revision(n)
- Loop: Go to Step 2

Cost Revision in AO*: cost-revise(n)

- Create Z={n}

- If Z = {} return

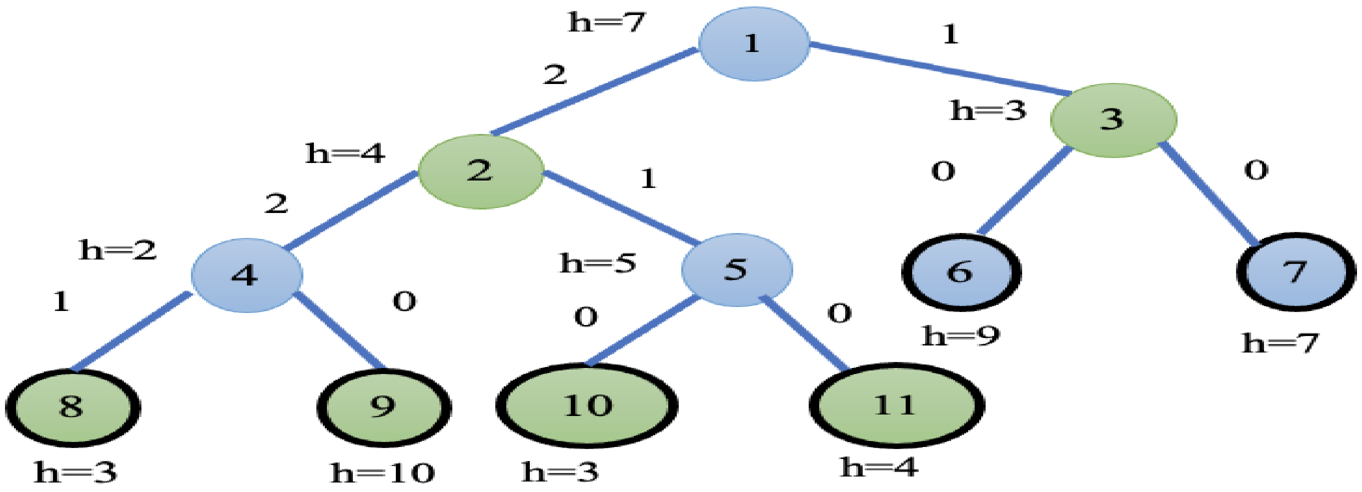
- Select a node m from Z such that m has no descendants in Z

- If m is an AND node with successors $r_1, r_2, \ldots, r_k$:
  - Set $f(m) = \sum(f(r_i) + C(m, r_i))$
  - Mark the edge to each successor of m
  - If each successor is labeled SOLVED
    - Then label m as SOLVED

Cost Revision in AO*: cost-revise(n)

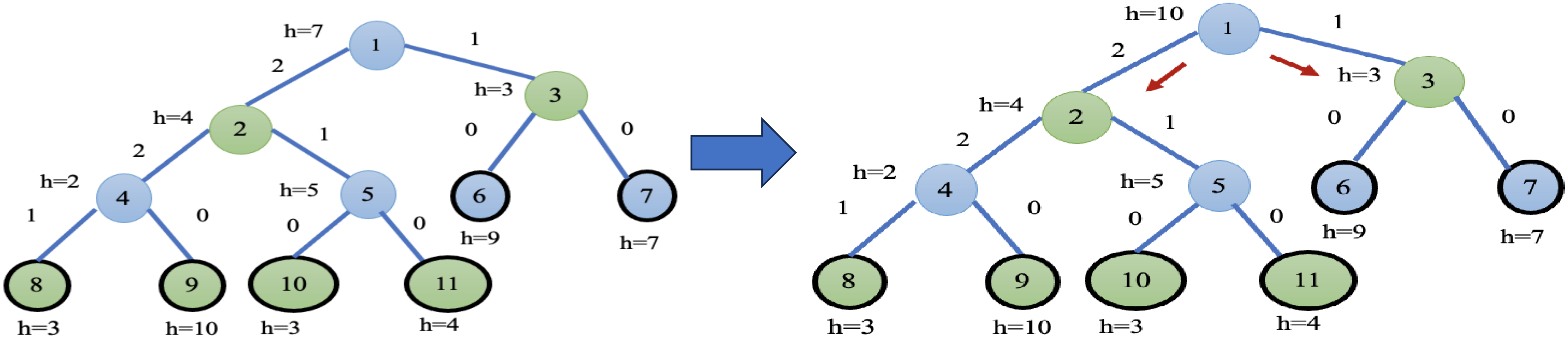
- If m is an OR node with successors $r_1, r_2, \ldots, r_k$:
  - Set $f(m) = \min\{f(r_i) + C(m, r_i)\}$
  - Mark the edge to the best successor of m
  - If the marked successor is labeled SOLVED
    - Then label m as SOLVED

- If the cost of label m has changed,
  - Then insert those parents of m into Z for which m is a marked successor



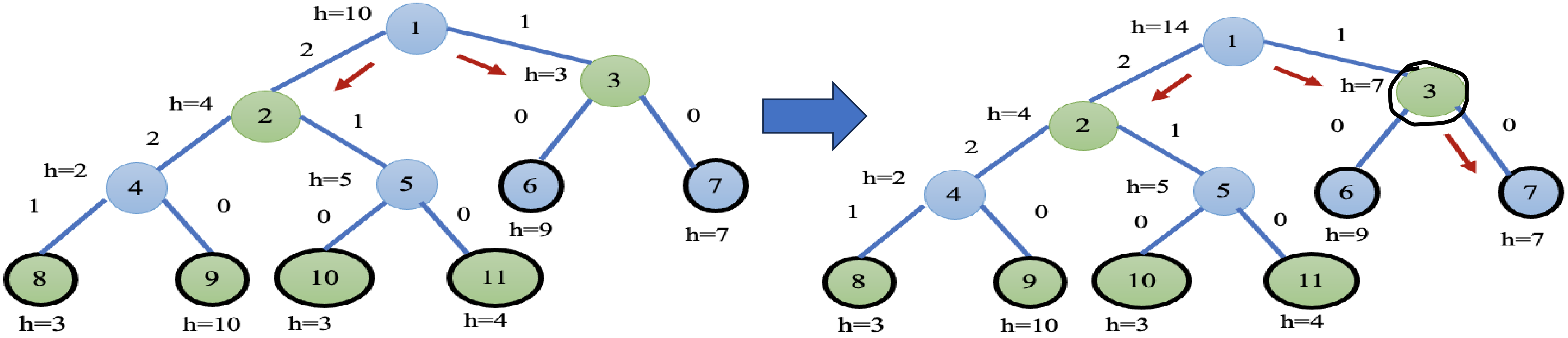
OPEN SET	SELECT	TER	EXPANDED	COST REVISION
[1(7)]	1(7)	N	[2(4,N),3(3,N)]	[1(7)]

START LIST	RET	SELECT	UPDATE	EDGE MARK	SOLVED	NEW LIST
[1(7)]	N	1(7)	[1(10)]	[(1,2),(1,3)]	N	[]



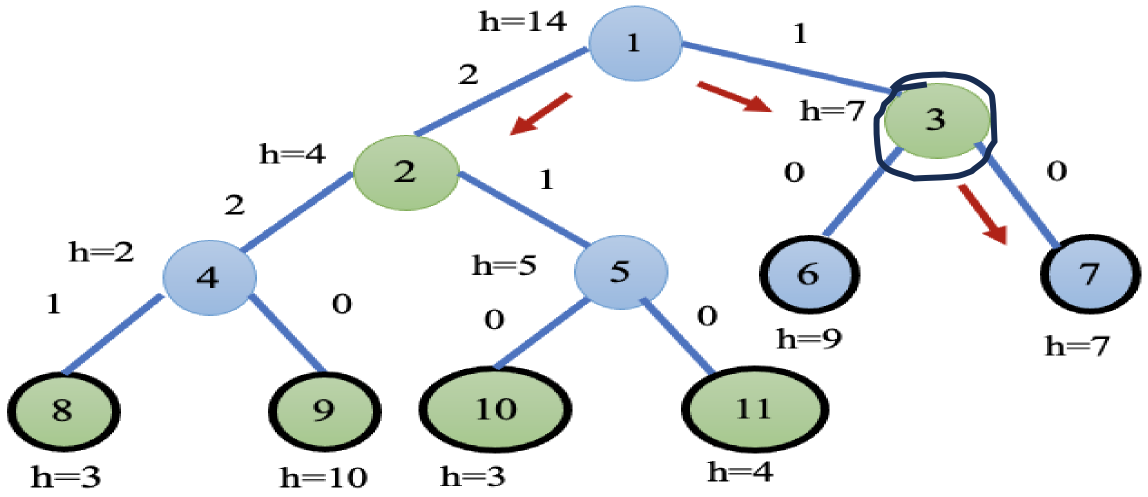
OPEN SET	SELECT	TER	EXPANDED	COST REVISION
[2(4),3(3)]	3(3)	N	[2(4,N),6(9,S),7(7,S)]	[3(3)]

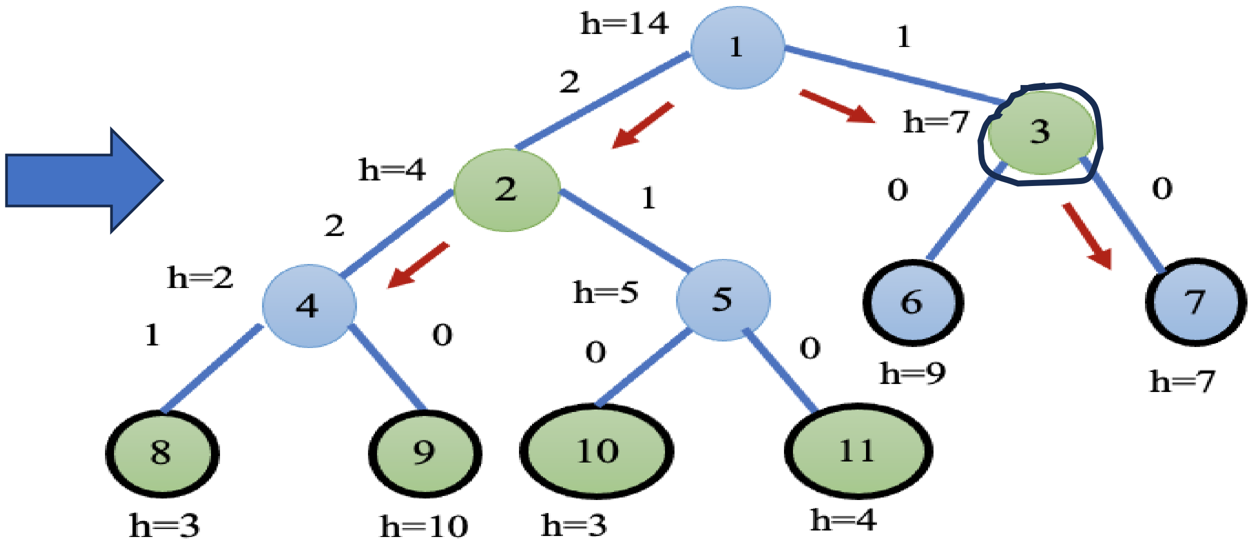
START LIST	RET	SELECT	UPDATE	EDGE MARK	SOLVED	NEW LIST
[3(3)]	N	3(3)	[3(7)]	[(3,7)]	S	[1(10)]
[1(10)]	N	1(10)	[1(14)]	[(1,2),(1,3)]	N	[]



OPEN SET	SELECT	TER	EXPANDED	COST REVISION
[2(4),6(9),7(7)]	2(4)	N	[4(2,N),5(5,N),6(9,S),7(7,S)]	[2(4)]

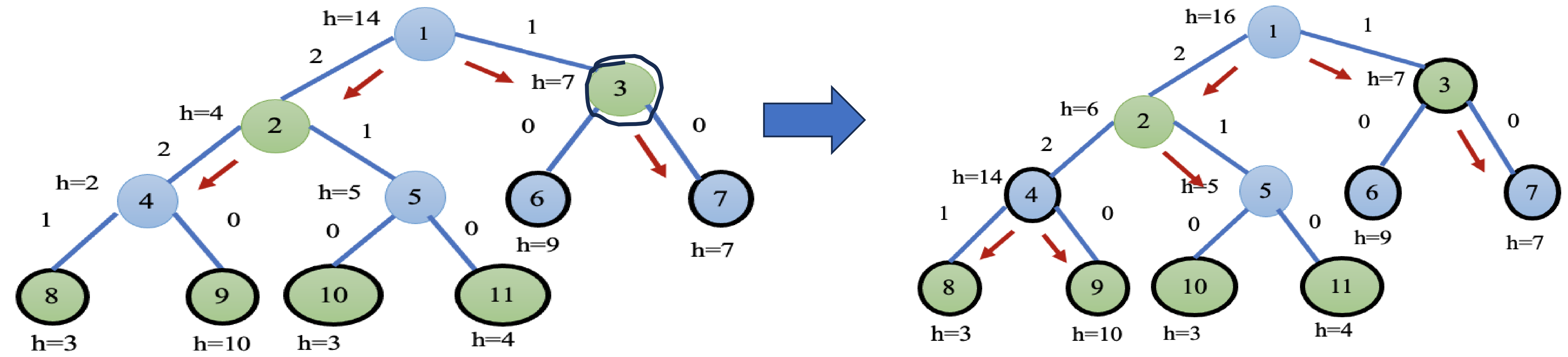
START LIST	RET	SELECT	UPDATE	EDGE MARK	SOLVED	NEW LIST
[2(4)]	N	2(4)	[2(4)]	[(2,4)]	N	[]





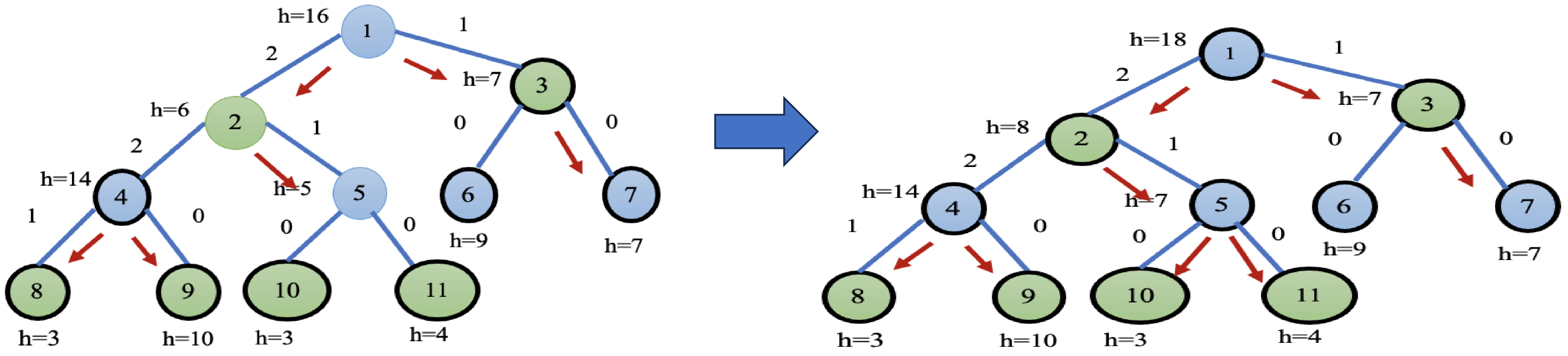
OPEN SET	SELECT	TER	EXPANDED	COST REVISION
[4(2),5(5),6(9),7(7)]	4(2)	N	[8(3,S),9(10,S),5(5,N),6(9,S),7(7,S)]	[4(2)]

START LIST	RET	SELECT	UPDATE	EDGE MARK	SOLVED	NEW LIST
[4(2)]	N	4(2)	[4(14)]	[(4,8),(4,9)]	S	[2(4)]
[2(4)]	N	2(4)	[2(6)]	[(2,5)]	N	[1(14)]
[1(14)]	N	1(14)	[1(16)]	[(1,2),(1,3)]	N	[]



OPEN SET	SELECT	TER	EXPANDED	COST REVISION
[8(3),5(5),9(10),6(9),7(7)]	5(5)	N	[8(3,S),9(10,S),10(3,S),11(4,S),6(9,S),7(7,S)]	[5(5)]

START LIST	RET	SELECT	UPDATE	EDGE MARK	SOLVED	NEW LIST
[5(5)]	N	5(5)	[5(7)]	[(5,10),(5,11)]	S	[2(6)]
[2(6)]	N	2(6)	[2(8)]	[(2,5)]	S	[1(16)]
[1(16)]	N	1(16)	[1(18)]	[(1,2),(1,3)]	S	[]



OPEN SET	SELECT	TER	EXPANDED	COST REVISION
[8(3),9(10),10(3),11(4),6(9),7(7)]	N	Y		

START LIST	RET	SELECT	UPDATE	EDGE MARK	SOLVED	NEW LIST

Thank You