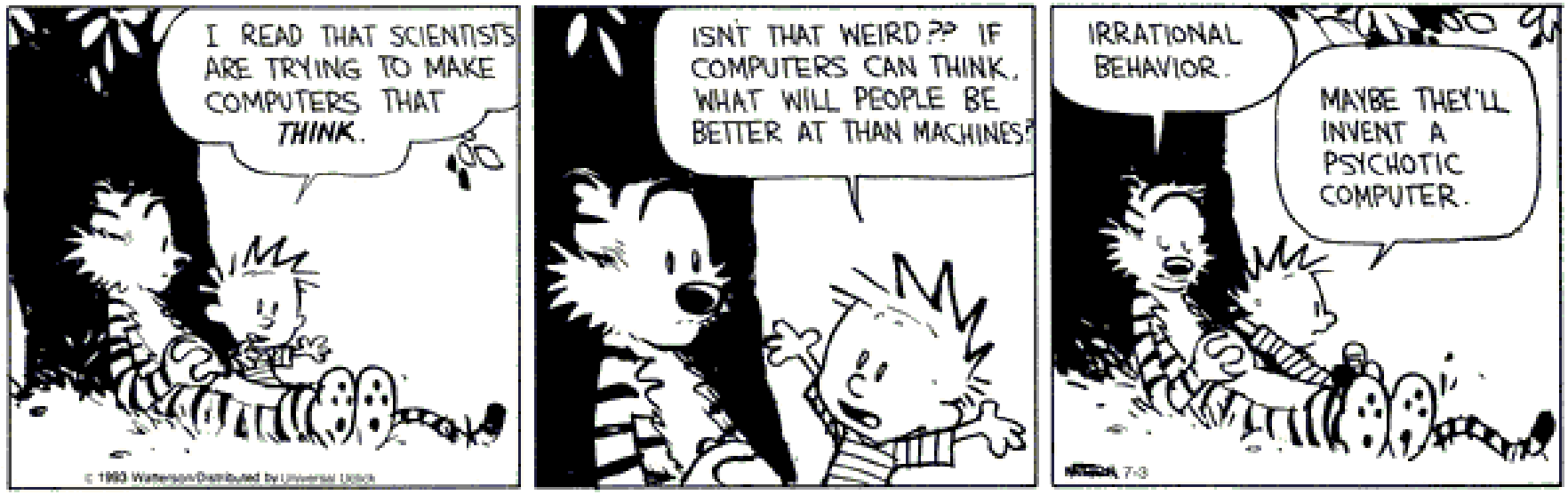

I READ THAT SCIENTISTS ARE TRYING TO MAKE COMPUTERS THAT THINK.
© 1993 Watterson/Distributed by Universal Uclick
ISN'T THAT WEIRD?? IF COMPUTERS CAN THINK, WHAT WILL PEOPLE BE BETTER AT THAN MACHINES?
IRRATIONAL BEHAVIOR.
MAYBE THEY'LL INVENT A PSYCHOTIC COMPUTER.
7-3

Heuristic Search

Chapter 3

Failure is the opportunity to begin again more intelligently

-- Moshe Arens

Outline

- Generate-and-test
- Hill climbing
- Best-first search
- Problem reduction
- Constraint satisfaction
- Means-ends analysis

Generate-and-Test

Algorithm

1. Generate a possible solution.
2. Test to see if this is actually a solution.
3. Quit if a solution has been found.
   Otherwise, return to step 1.

Generate-and-Test

- Acceptable for simple problems.
- Inefficient for problems with large space.

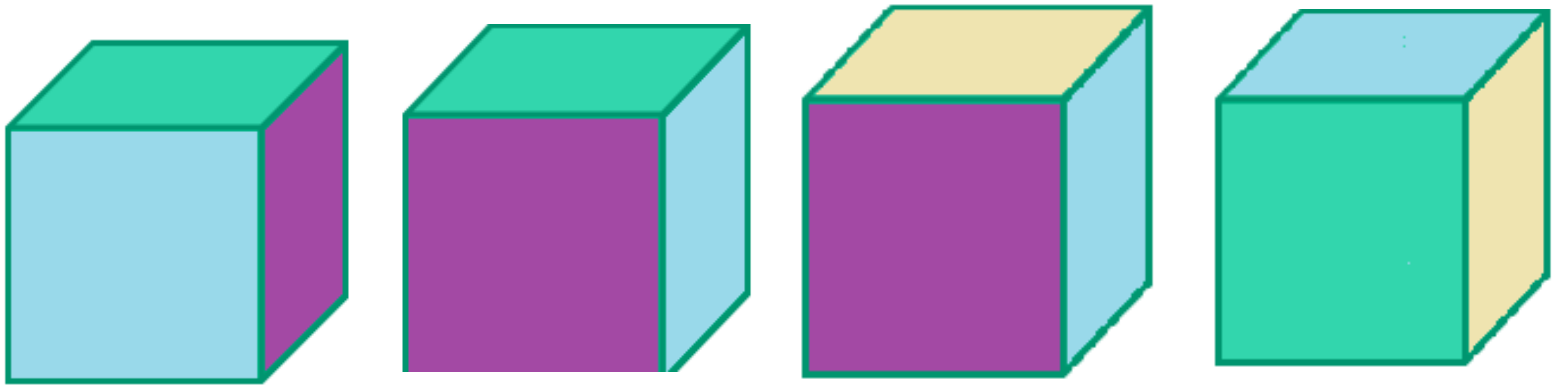
Generate-and-Test

- Exhaustive generate-and-test.
- Heuristic generate-and-test: not consider paths that seem unlikely to lead to a solution.
- Plan generate-test:
  - Create a list of candidates.
  - Apply generate-and-test to that list.

Generate-and-Test

Example: coloured blocks

"Arrange four 6-sided cubes in a row, with each side of each cube painted one of four colours, such that on all four sides of the row one block face of each colour is showing."



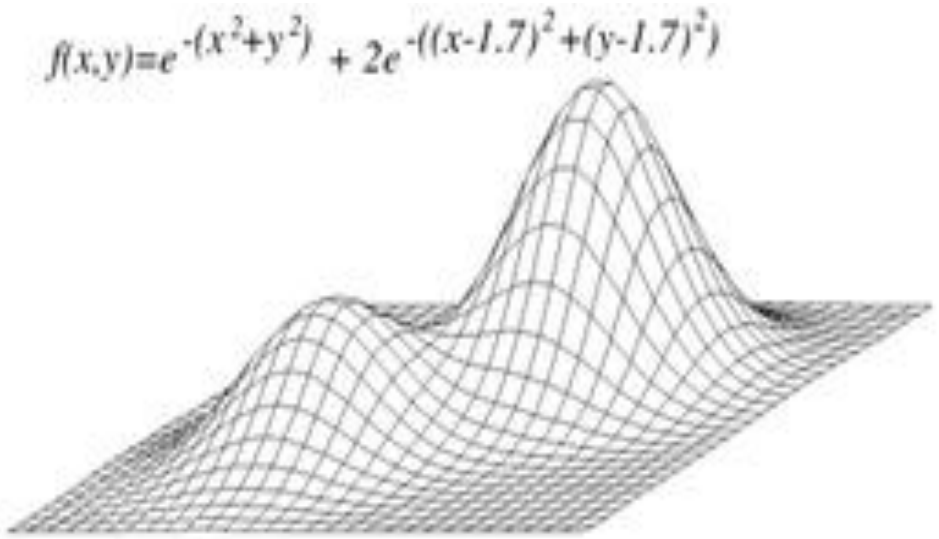
Generate-and-Test

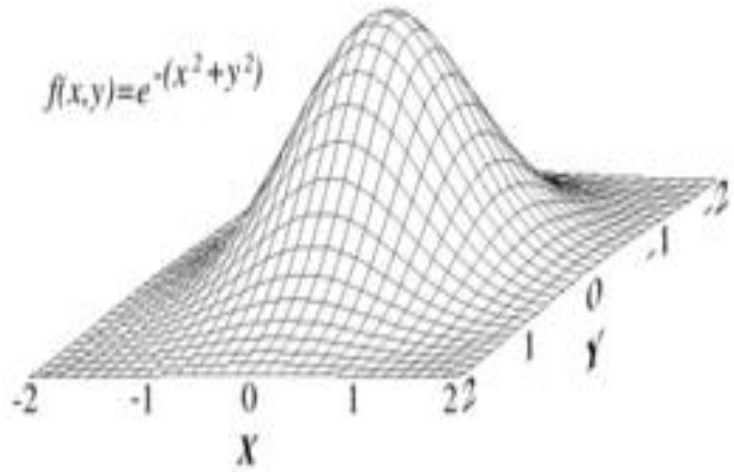
Example: coloured blocks

Heuristic: if there are more red faces than other colours then, when placing a block with several red faces, use few of them as possible as outside faces.

Hill Climbing

- Searching for a goal state = Climbing to the top of a hill



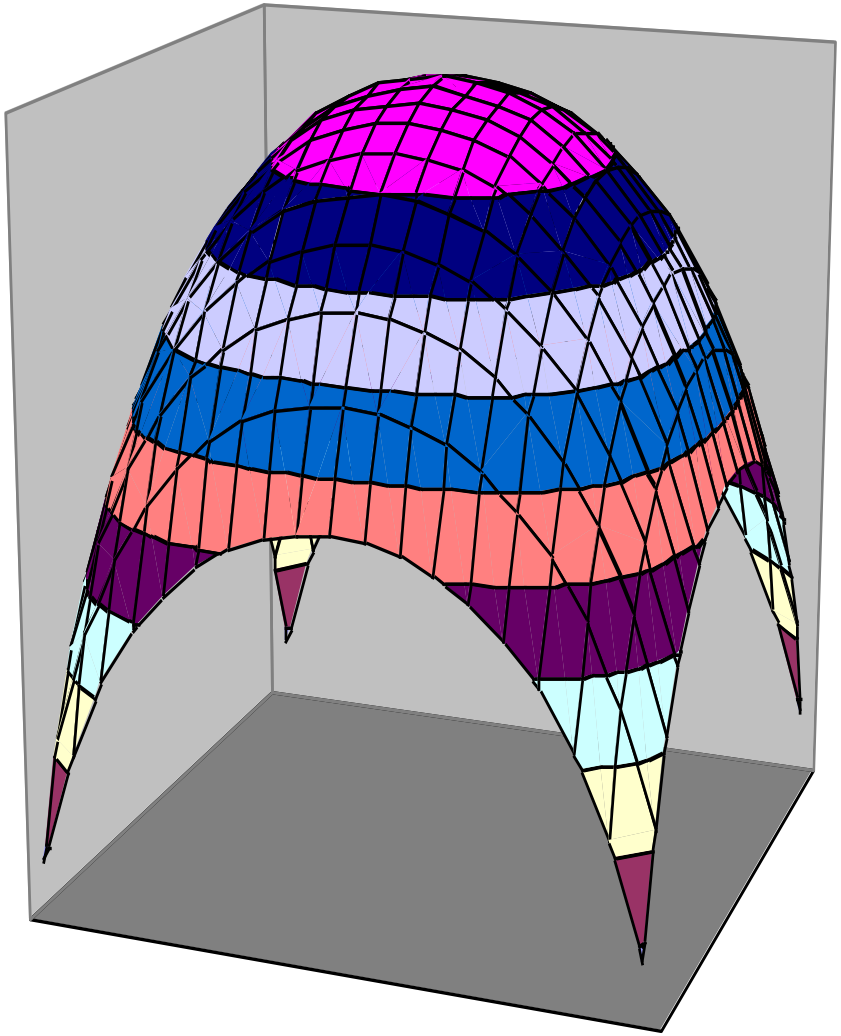


Hill Climbing

- Generate-and-test + direction to move.
- Heuristic function to estimate how close a given state is to a goal state.

Hill Climbing

Hill Climbing



Simple Hill Climbing

Algorithm

1. Evaluate the initial state.
2. Loop until a solution is found or there are no new operators left to be applied:
   - Select and apply a new operator
   - Evaluate the new state:

     goal $\rightarrow$ quit

     better than current state $\rightarrow$ new current state

Simple Hill Climbing

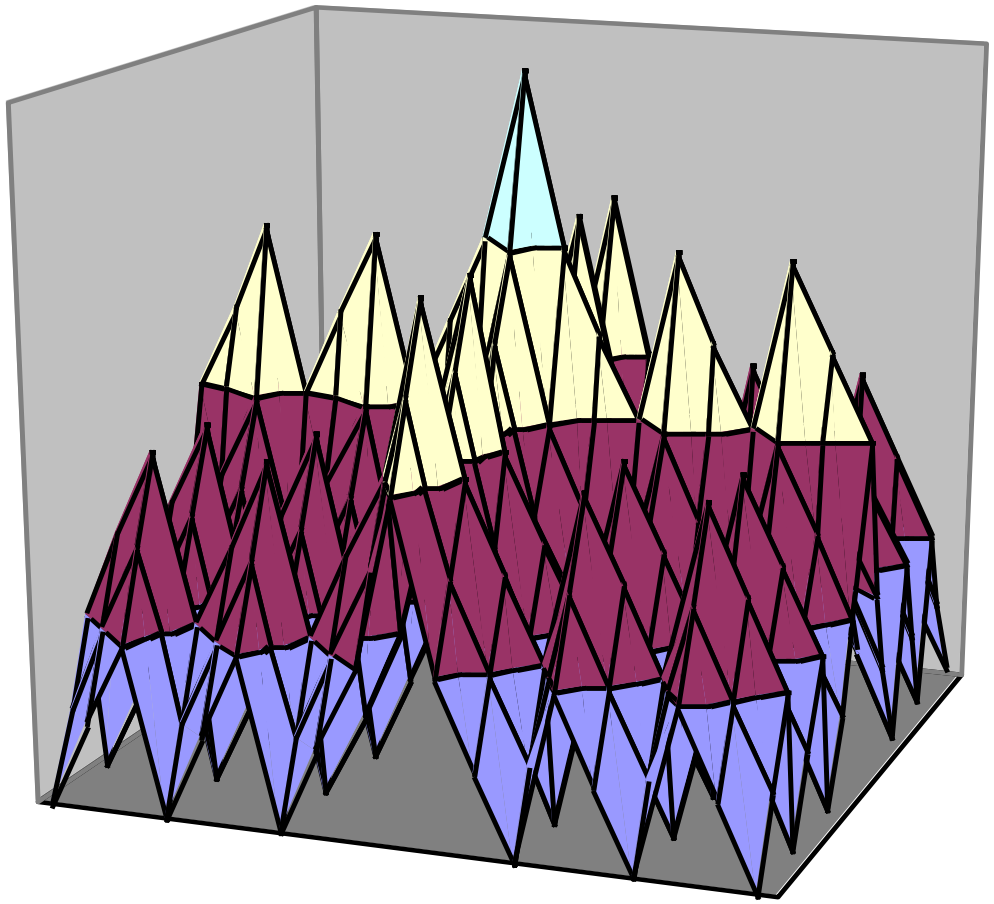
- Evaluation function as a way to inject task-specific knowledge into the control process.

Simple Hill Climbing

Example: coloured blocks

Heuristic function: the sum of the number of different colours on each of the four sides (solution = 16).

Hill Climbing



Steepest-Ascent Hill Climbing (Gradient Search)

- Considers all the moves from the current state.
- Selects the best one as the next state.

Steepest-Ascent Hill Climbing (Gradient Search)

Algorithm

1. Evaluate the initial state.
2. Loop until a solution is found or a complete iteration produces no change to current state:
   - SUCC = a state such that any possible successor of the current state will be better than SUCC (the worst state).
   - For each operator that applies to the current state, evaluate the new state:

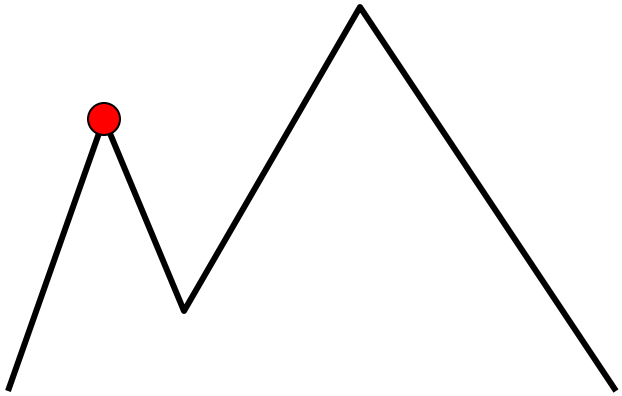
     goal $\rightarrow$ quit

     better than SUCC $\rightarrow$ set SUCC to this state
   - SUCC is better than the current state $\rightarrow$ set the current state to SUCC.

Hill Climbing: Disadvantages

Local maximum

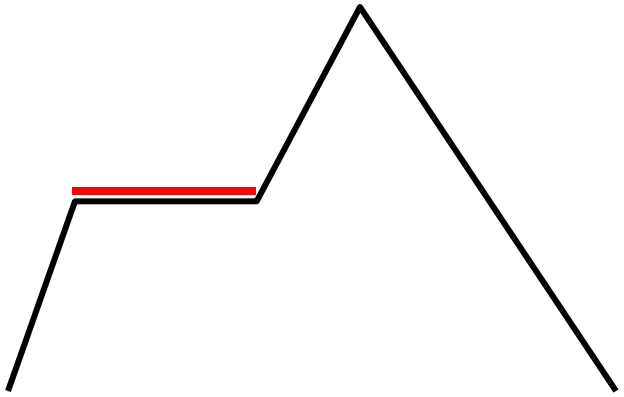
A state that is better than all of its neighbours, but not better than some other states far away.



Hill Climbing: Disadvantages

Plateau

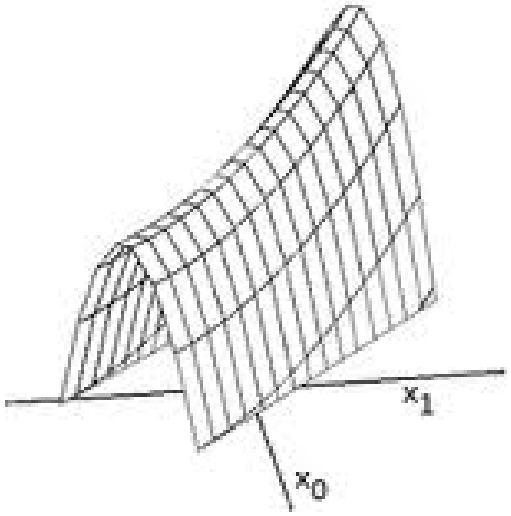
A flat area of the search space in which all neighbouring states have the same value.

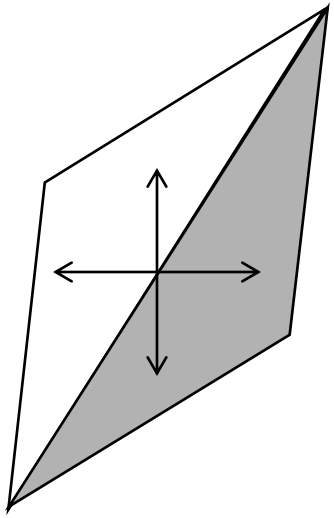


Hill Climbing: Disadvantages

Ridge

The orientation of the high region, compared to the set of available moves, makes it impossible to climb up. However, two moves executed serially may increase the height.





Ridges: dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.

Hill Climbing: Disadvantages

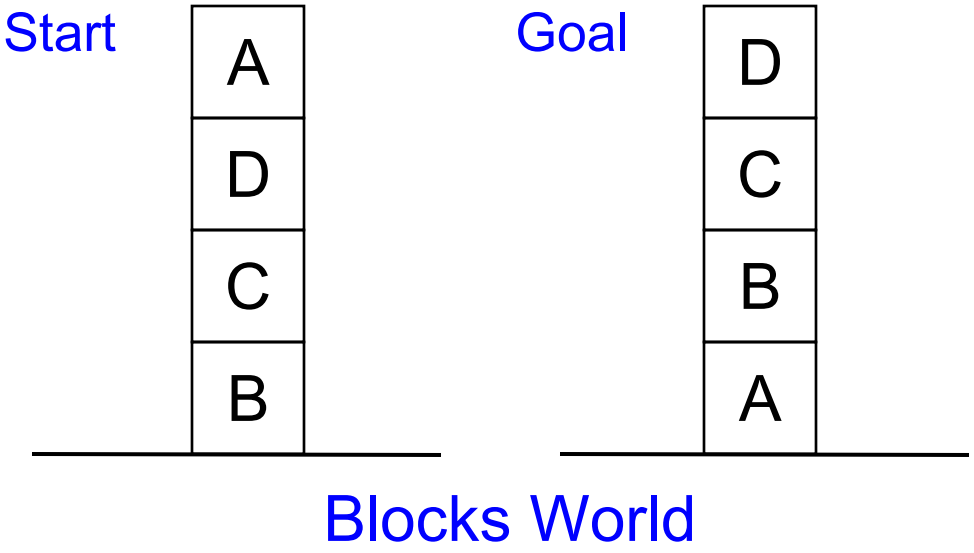
Ways Out

- Backtrack to some earlier node and try going in a different direction.
- Make a big jump to try to get in a new section.
- Moving in several directions at once.

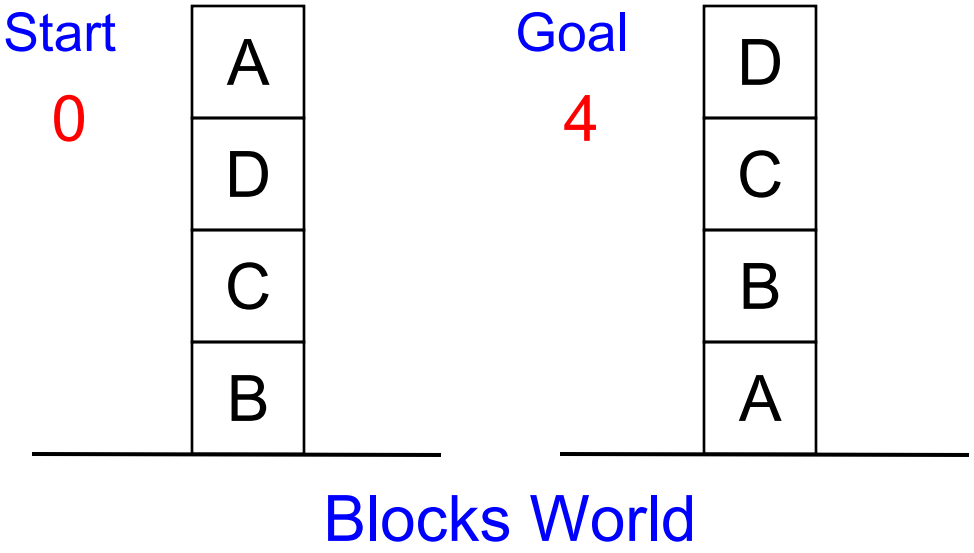
Hill Climbing: Disadvantages

- Hill climbing is a local method:
  Decides what to do next by looking only at the "immediate" consequences of its choices.
- Global information might be encoded in heuristic functions.

Hill Climbing: Disadvantages



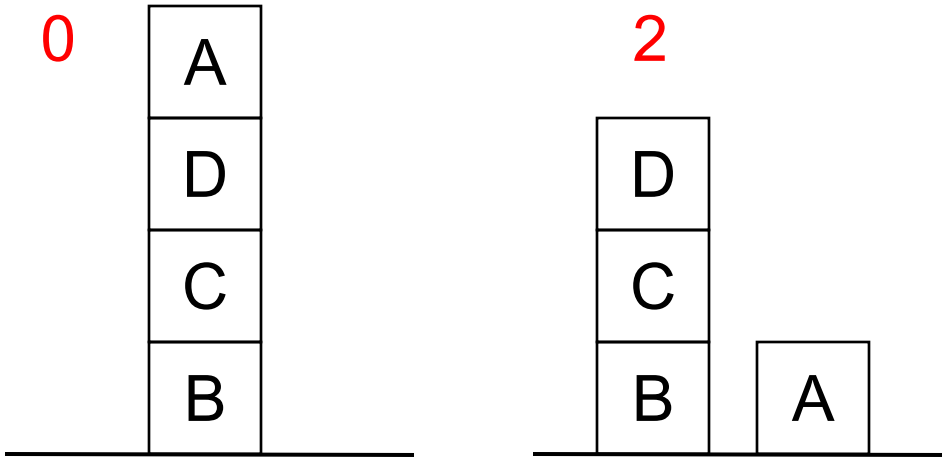
Hill Climbing: Disadvantages



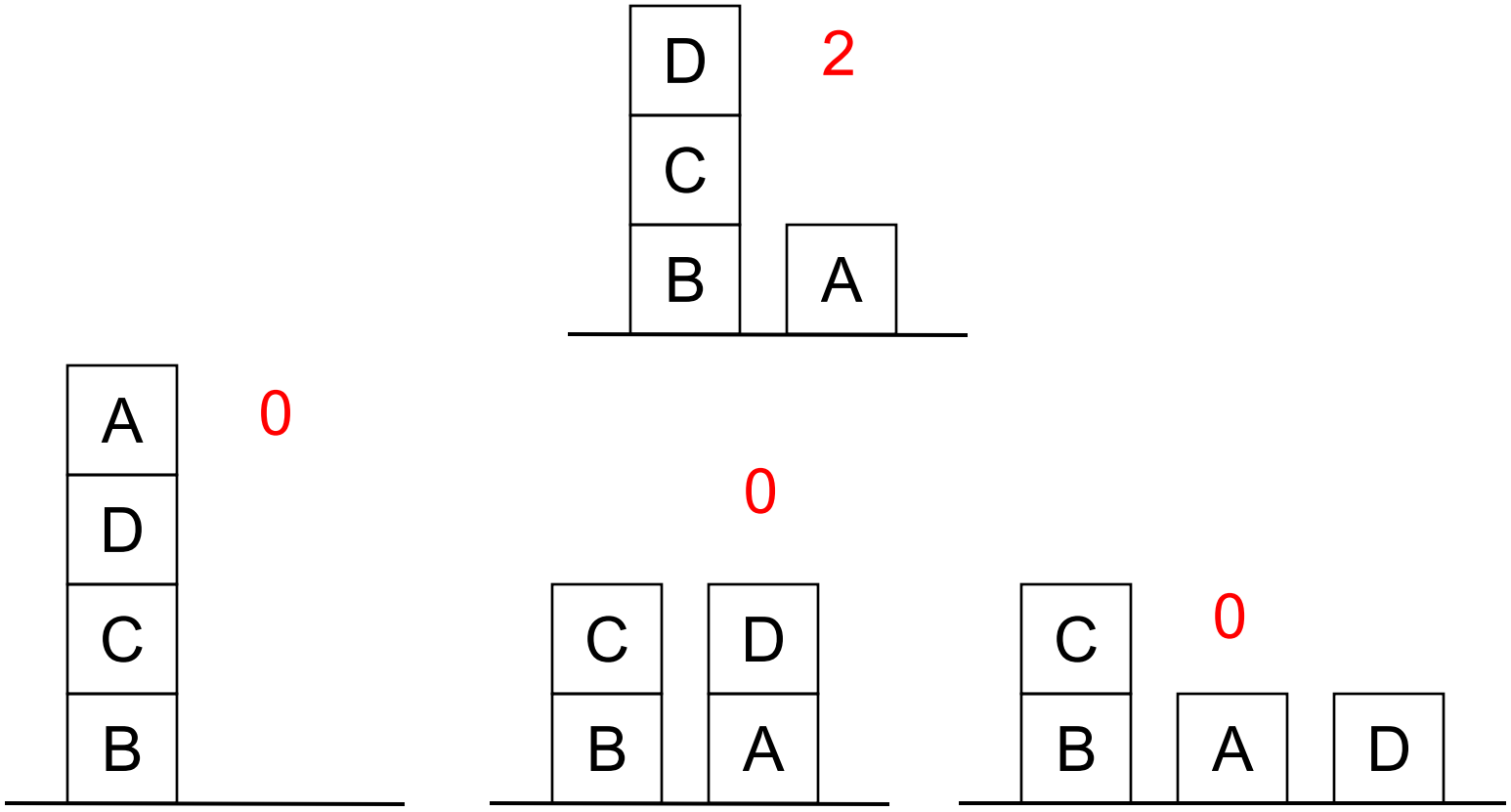
Blocks World

Local heuristic:
+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.

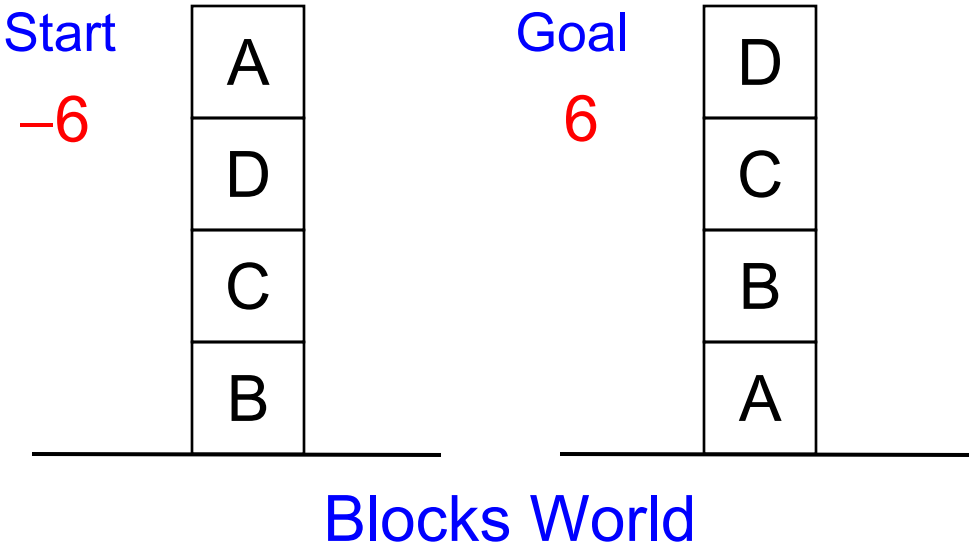
Hill Climbing: Disadvantages



Hill Climbing: Disadvantages



Hill Climbing: Disadvantages



Blocks World

Global heuristic:

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.

Example (8-puzzle)

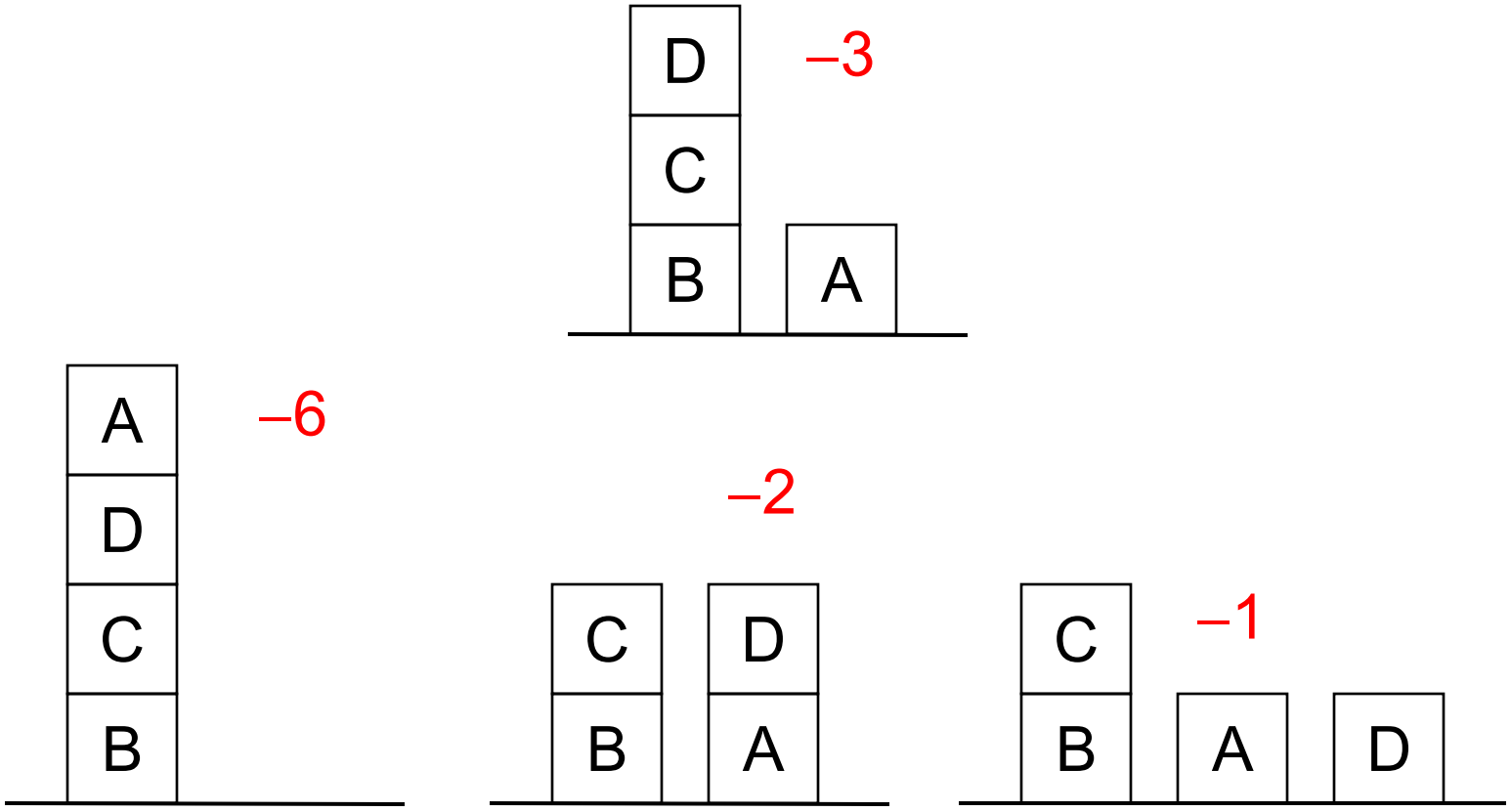
3	1	2
4	5	8
6		7

Initial

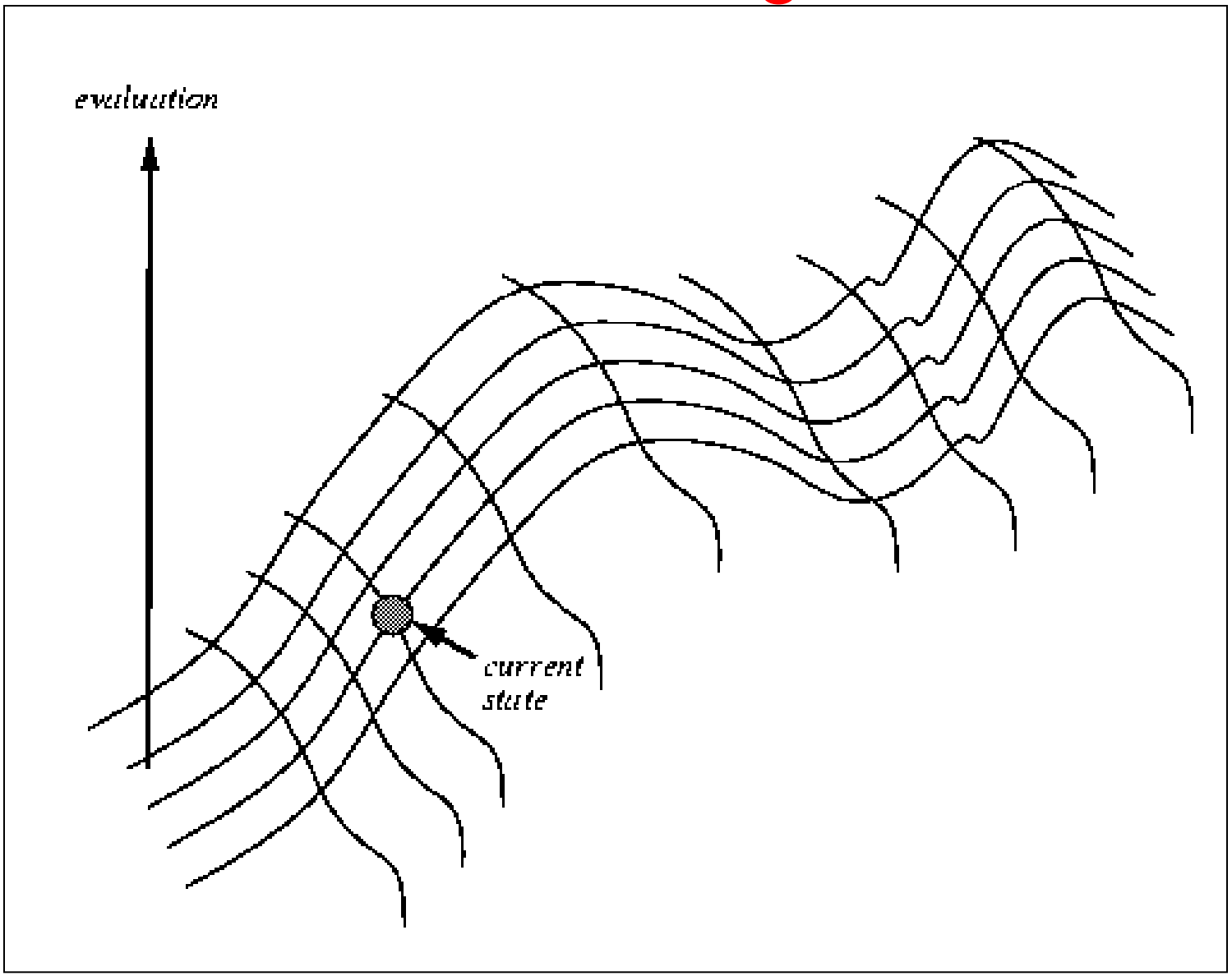
	1	2
3	4	5
6	7	8

Goal

Hill Climbing: Disadvantages



Hill Climbing



Hill Climbing: Conclusion

- Can be very inefficient in a large, rough problem space.
- Global heuristic may have to pay for computational complexity.
- Often useful when combined with other methods, getting it started right in the right general neighbourhood.

In other words, when we perform
hill-climbing, we are shortsighted.
We can’t really see far.
Thus, we make local best
decisions with the hope of
finding a global best solution.
This may be an impossible task.

Simulated Annealing

- A variation of hill climbing in which, at the beginning of the process, some downhill moves may be made.
- To do enough exploration of the whole space early on, so that the final solution is relatively insensitive to the starting state.
- Lowering the chances of getting caught at a local maximum, or plateau, or a ridge.

Simulated Annealing

Physical Annealing

- Physical substances are melted and then gradually cooled until some solid state is reached.
- The goal is to produce a minimal-energy state.
- Annealing schedule: if the temperature is lowered sufficiently slowly, then the goal will be attained.
- Nevertheless, there is some probability for a transition to a higher energy state: $e^{-\Delta E/kT}$.

Simulated Annealing

Algorithm

1. Evaluate the initial state.
2. Loop until a solution is found or there are no new operators left to be applied:
   - Set T according to an annealing schedule
   - Selects and applies a new operator
   - Evaluate the new state:

     goal $\rightarrow$ quit

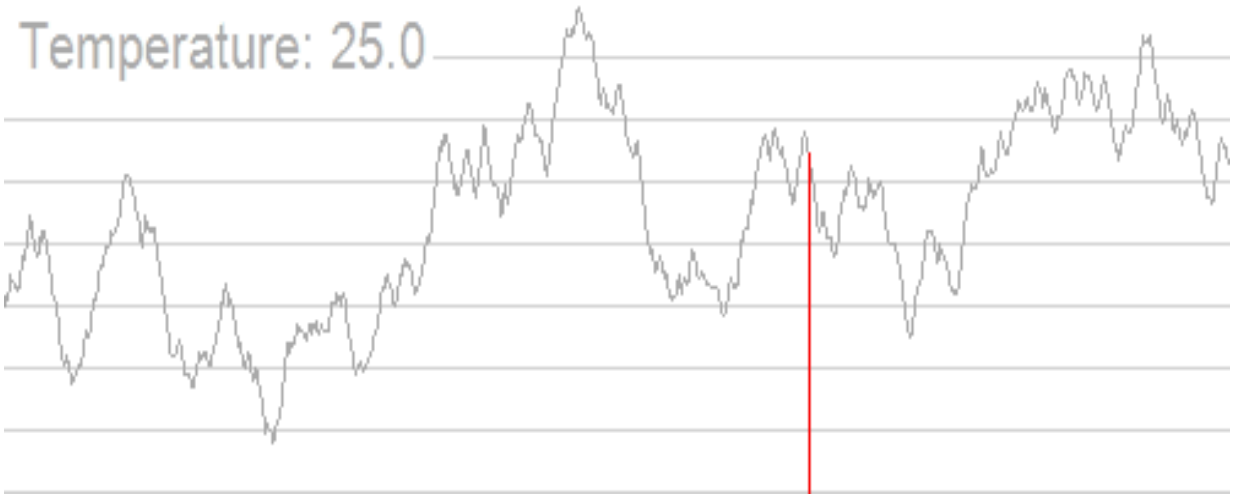
     ΔE = Val(current state) – Val(new state)

     $\Delta E < 0 \rightarrow$ new current state

     else $\rightarrow$ new current state with probability $e^{-\Delta E/T}$.

Annealing

- Based on a metallurgical metaphor
  - Start with a temperature set very high and slowly reduce it.
  - Run hillclimbing with the twist that you can occasionally replace the current state with a worse state based on the current temperature and how much worse the new state is.



Despite the many local maxima in this graph, the global maximum can still be found using simulated annealing. Unfortunately, the applicability of simulated annealing is problem-specific because it relies on finding *lucky jumps* that improve the position. In problems that involve more dimensions, the cost of finding such a jump may increase exponentially with dimensionality. Consequently, there remain many problems for which hill climbers will efficiently find good results while simulated annealing will seemingly run forever without making progress. This depiction shows an extreme case involving only one dimension.

Annealing

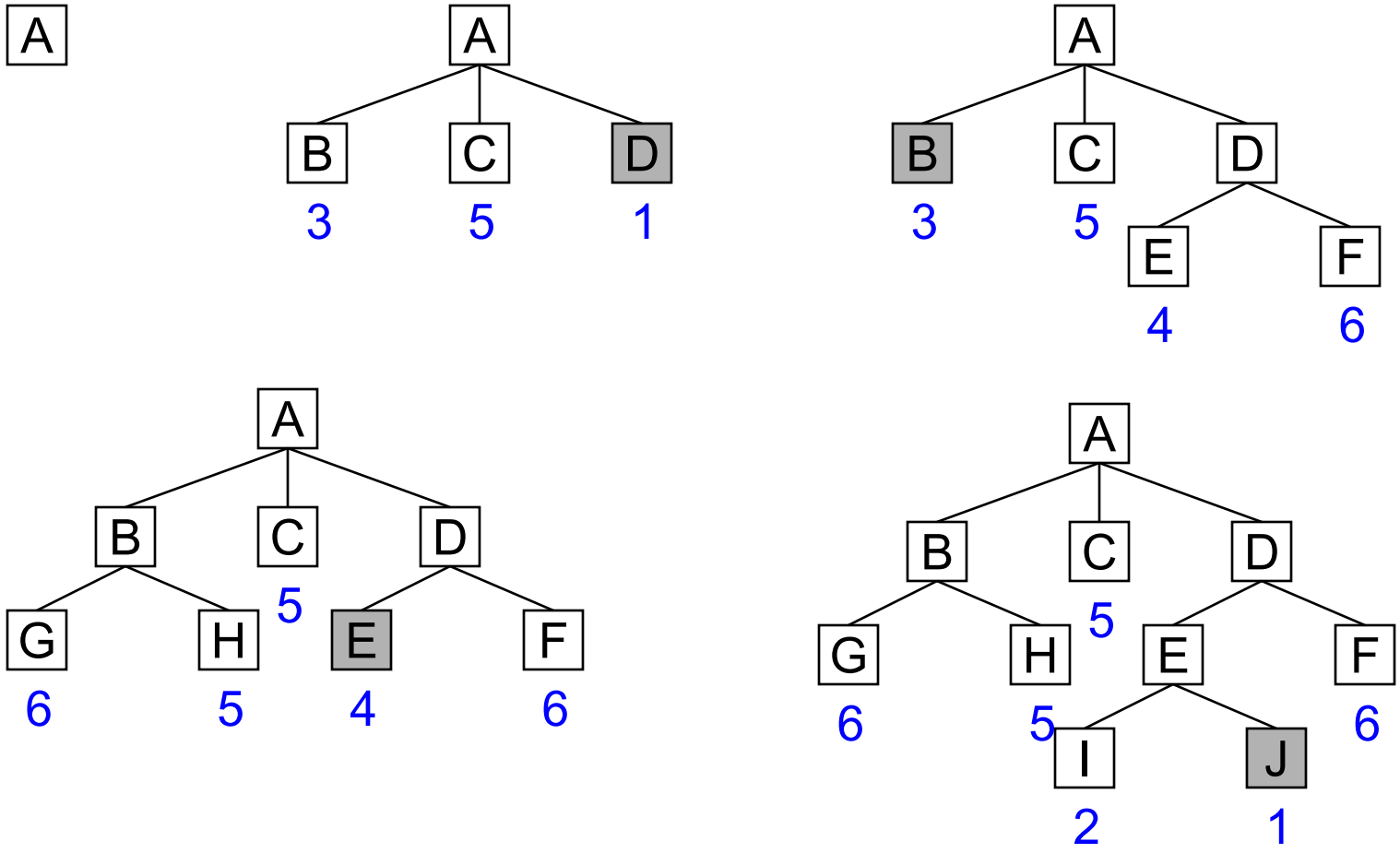
- More formally…
  - Generate a new neighbor from current state.
  - If it's better take it.
  - If it's worse then take it with some probability proportional to the temperature and the delta between the new and old states.

Best-First Search

- Depth-first search: not all competing branches having to be expanded.
- Breadth-first search: not getting trapped on dead-end paths.

  $\Rightarrow$ Combining the two is to follow a single path at a time, but switch paths whenever some competing path look more promising than the current one.

Best-First Search



Best-First Search

- OPEN: nodes that have been generated, but have not examined.

  This is organized as a priority queue.

- CLOSED: nodes that have already been examined.

  Whenever a new node is generated, check whether it has been generated before.

Best-First Search

Algorithm

1. OPEN = {initial state}.
2. Loop until a goal is found or there are no nodes left in OPEN:
   - Pick the best node in OPEN
   - Generate its successors
   - For each successor:

     new $\rightarrow$ evaluate it, add it to OPEN, record its parent

     generated before $\rightarrow$ change parent, update successors

Best-First Search

- Greedy search:
  h(n) = estimated cost of the cheapest path from node n to a goal state.

Best-First Search

- Uniform-cost search:
  g(n) = cost of the cheapest path from the initial state to node n.

Best-First Search

- Greedy search:
  h(n) = estimated cost of the cheapest path from node n to a goal state.

  Neither optimal nor complete

Best-First Search

- Greedy search:
  h(n) = estimated cost of the cheapest path from node n to a goal state.

  Neither optimal nor complete

- Uniform-cost search:
  g(n) = cost of the cheapest path from the initial state to node n.

  Optimal and complete, but very inefficient

Best-First Search

- Algorithm A* (Hart et al., 1968):

$$f(n) = g(n) + h(n)$$

$h(n)$ = cost of the cheapest path from node n to a goal state.

$g(n)$ = cost of the cheapest path from the initial state to node n.

Best-First Search

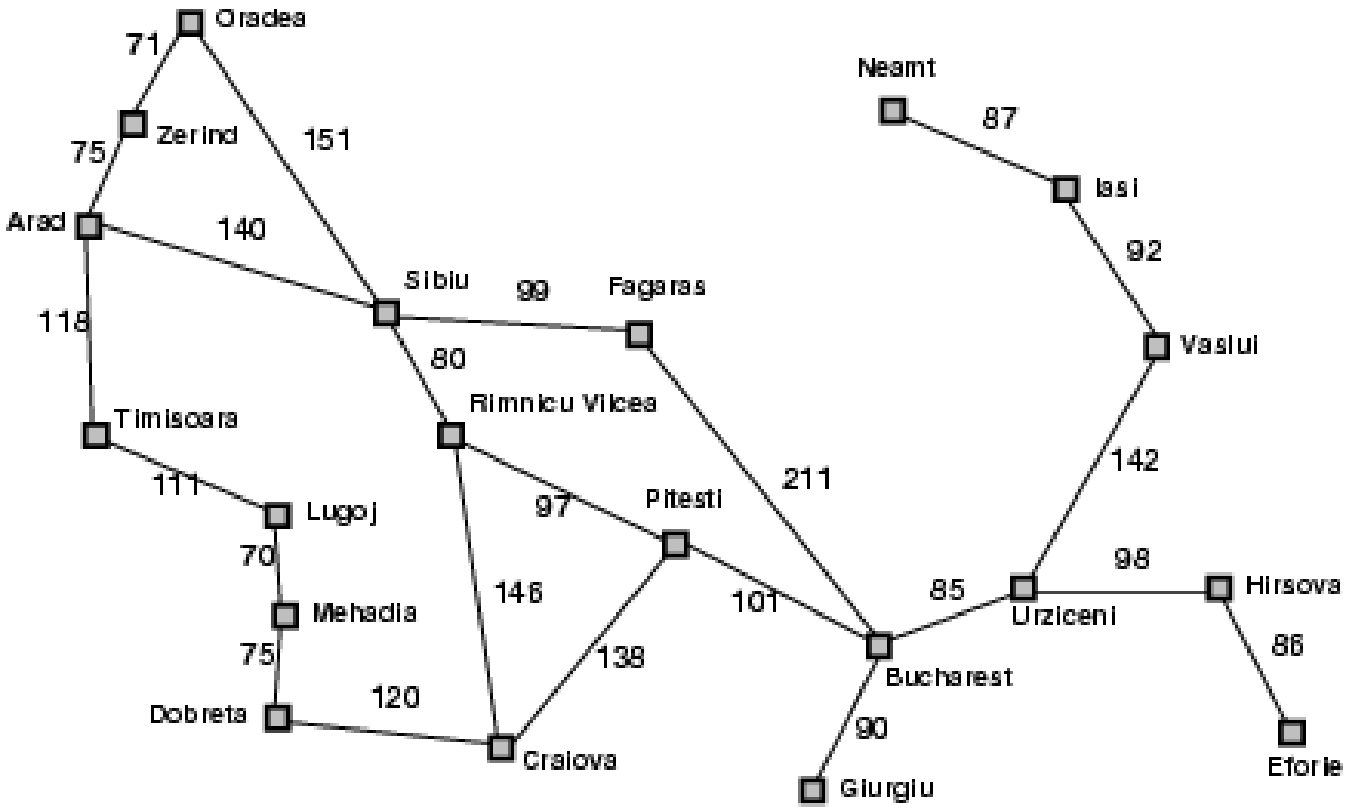
- Algorithm A*:

$$f^*(n) = g^*(n) + h^*(n)$$

$h^*(n)$ (heuristic factor) = estimate of $h(n)$.

$g^*(n)$ (depth factor) = approximation of $g(n)$ found by A* so far.

Romania with step costs in km

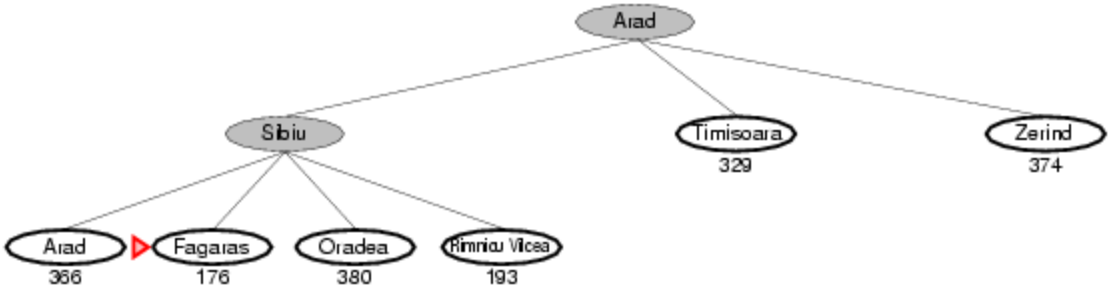


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

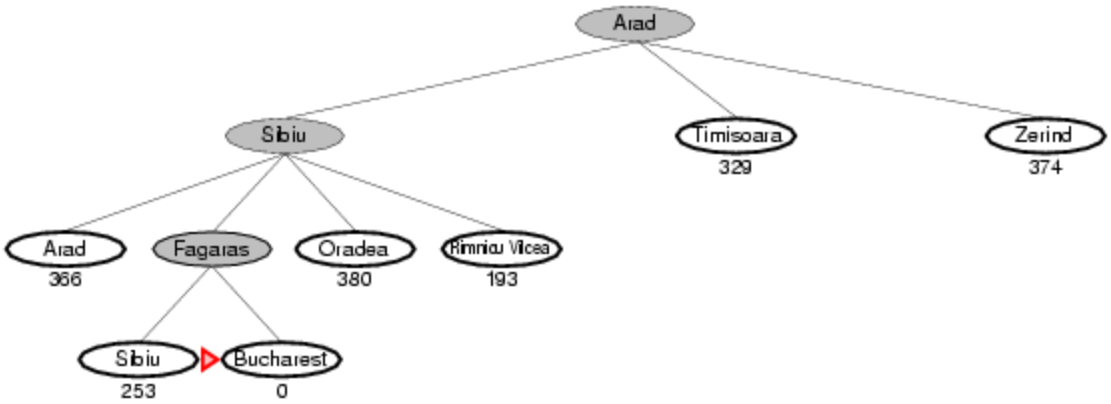
Greedy best-first search example

Greedy best-first search example

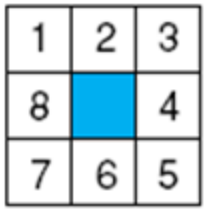
Greedy best-first search example



Greedy best-first search example



8-puzzle heuristics

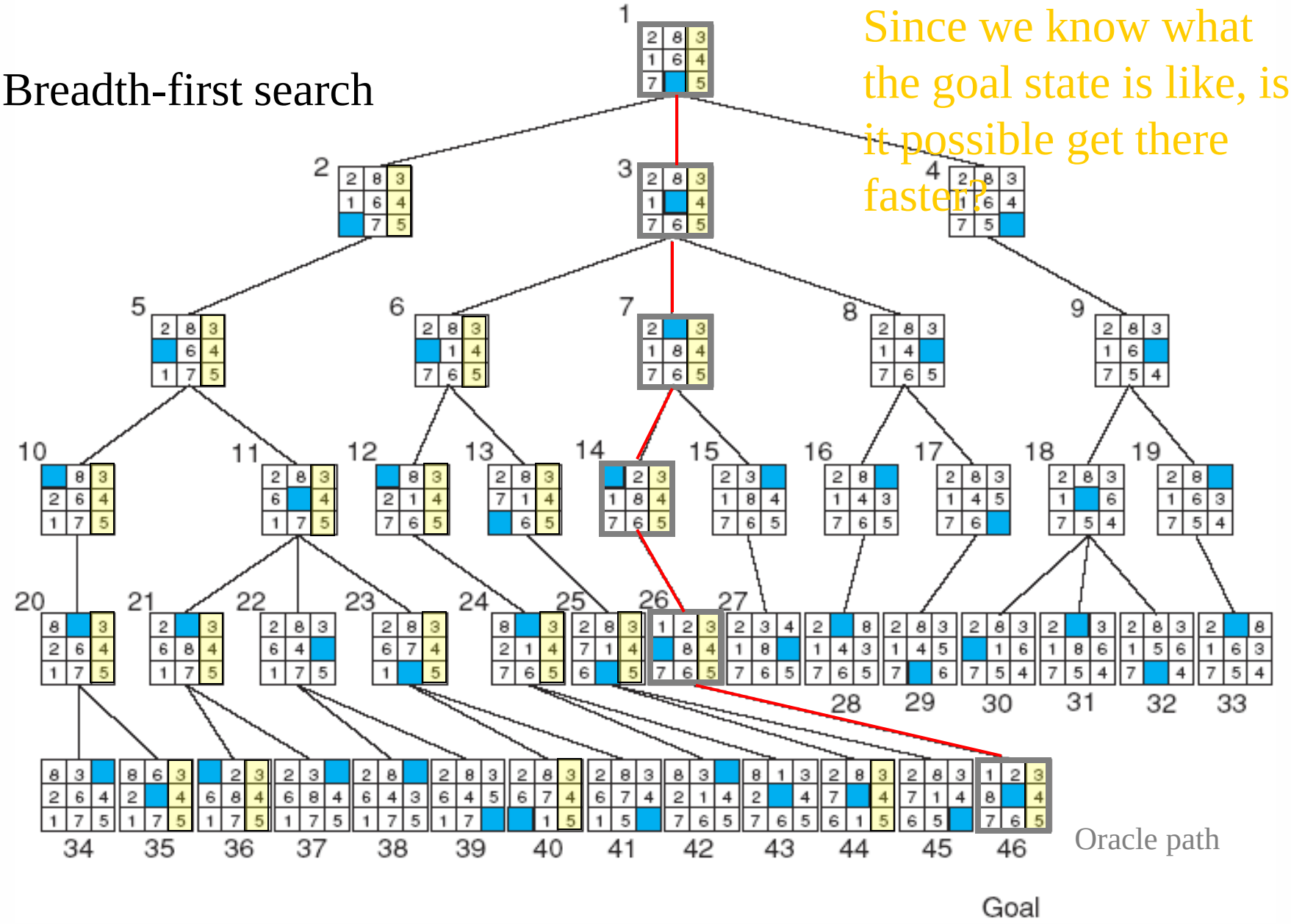


Goal

- Number of tiles out of places
  - Compared with the goal state
  - How many tiles are not in the right place?
- Sum of taxicab distances
  - For each tile, how far is it from the goal position? How many squares away? What is its distance?
- The objective is to minimize this heuristic value as you search. As it approaches 0, you are closer to the goal.

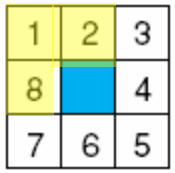
Breadth-first search

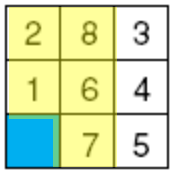
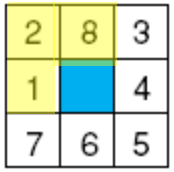
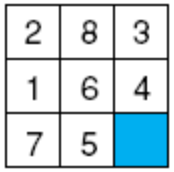
Since we know what the goal state is like, is it possible get there faster?

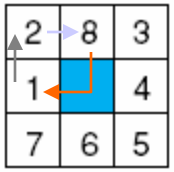


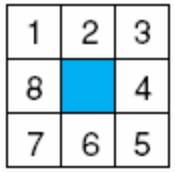
Oracle path

2 heuristics and their calculations



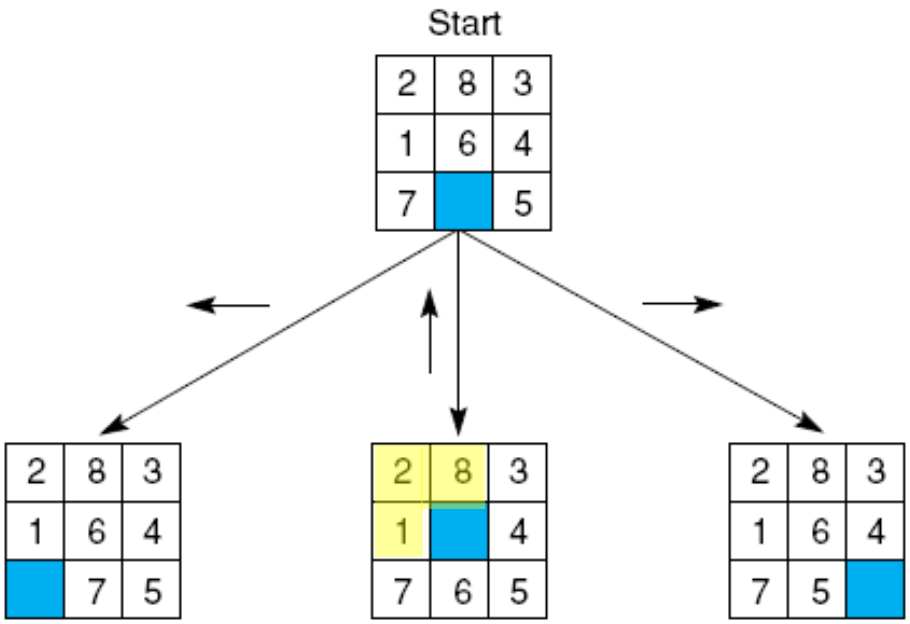
n	$h_1(n)$	$h_2(n)$
	**5**	**6**
	**3**	**4**
	**5**	**6**
	Tiles out of place	Sum of distances out of place





Algorithm A with $h_1(n)$

States that are closer to the start node are preferred because you can get there sooner.



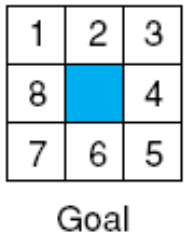
Values of f(n) for each state, 6 4 =1+3 6

where:

$f(n) = g(n) + h_1(n)$

$g(n)$ = actual distance from n to the start state, and

$h_1(n)$ = number of tiles out of place.



Trace

1. open=[a4]
   closed=[ ]

2. open=[c4,b6,d6]
   closed=[a4]

3. open=[e5,f5,g6,b6,d6]
   closed=[a4, c4]

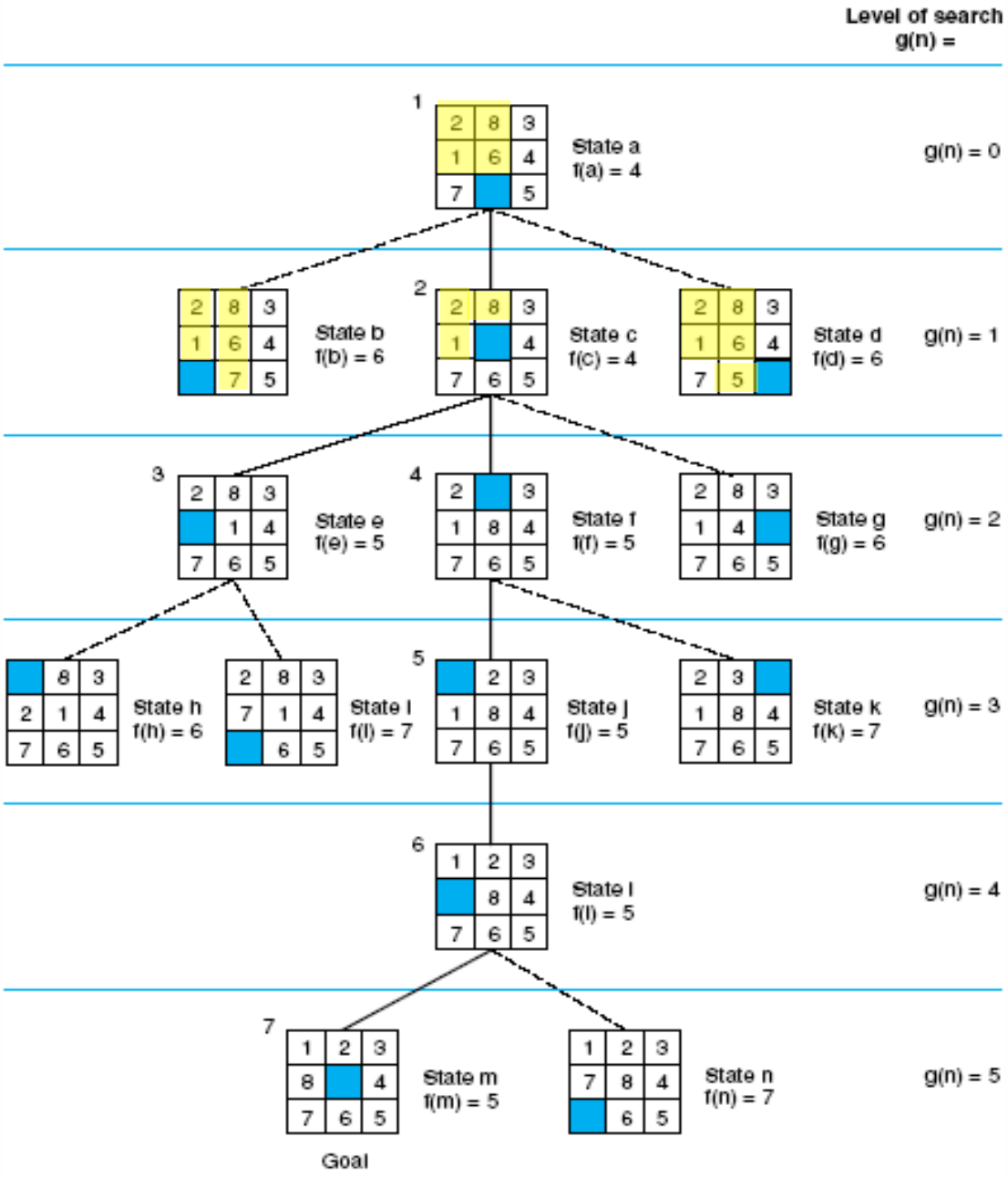
4. open=[f5,h6,g6,b6,d6,i7]
   closed=[a4,c4,e5]

5. open=[j5,h6,b6,d6,k7,i7]
   closed=[a4,c4,e5,f5]

6. open=[i5,h6,b6,d6,k7,i7]
   closed=[a4,c4,e5,f5,j5]

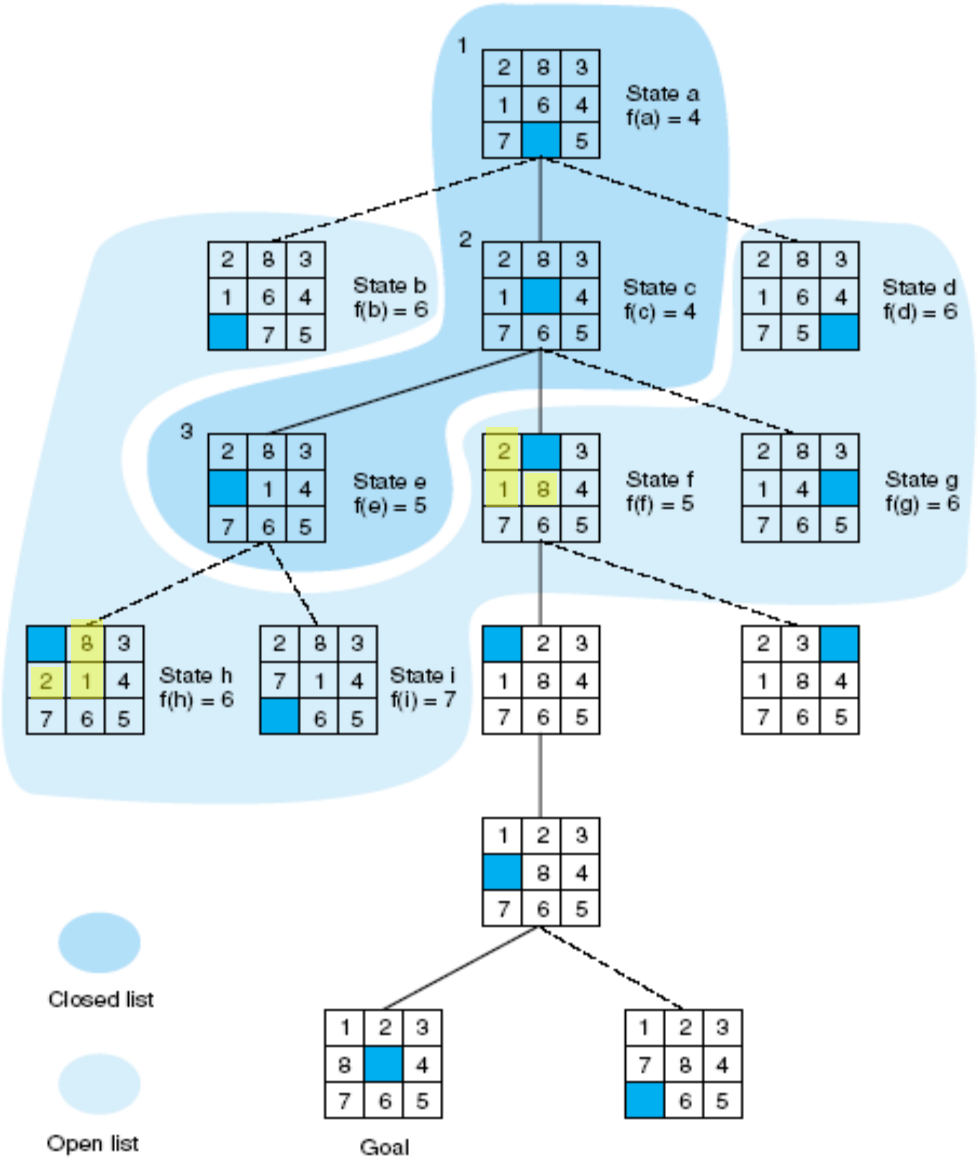
7. open=[m5,h6,b6,d6,k7,i7]
   closed=[a4,c4,e5,f5,j5,i5]

success, m = goal



At the 3rd iteration

State f and h both have 3 tiles out of place but f is preferred because you can get to f sooner.



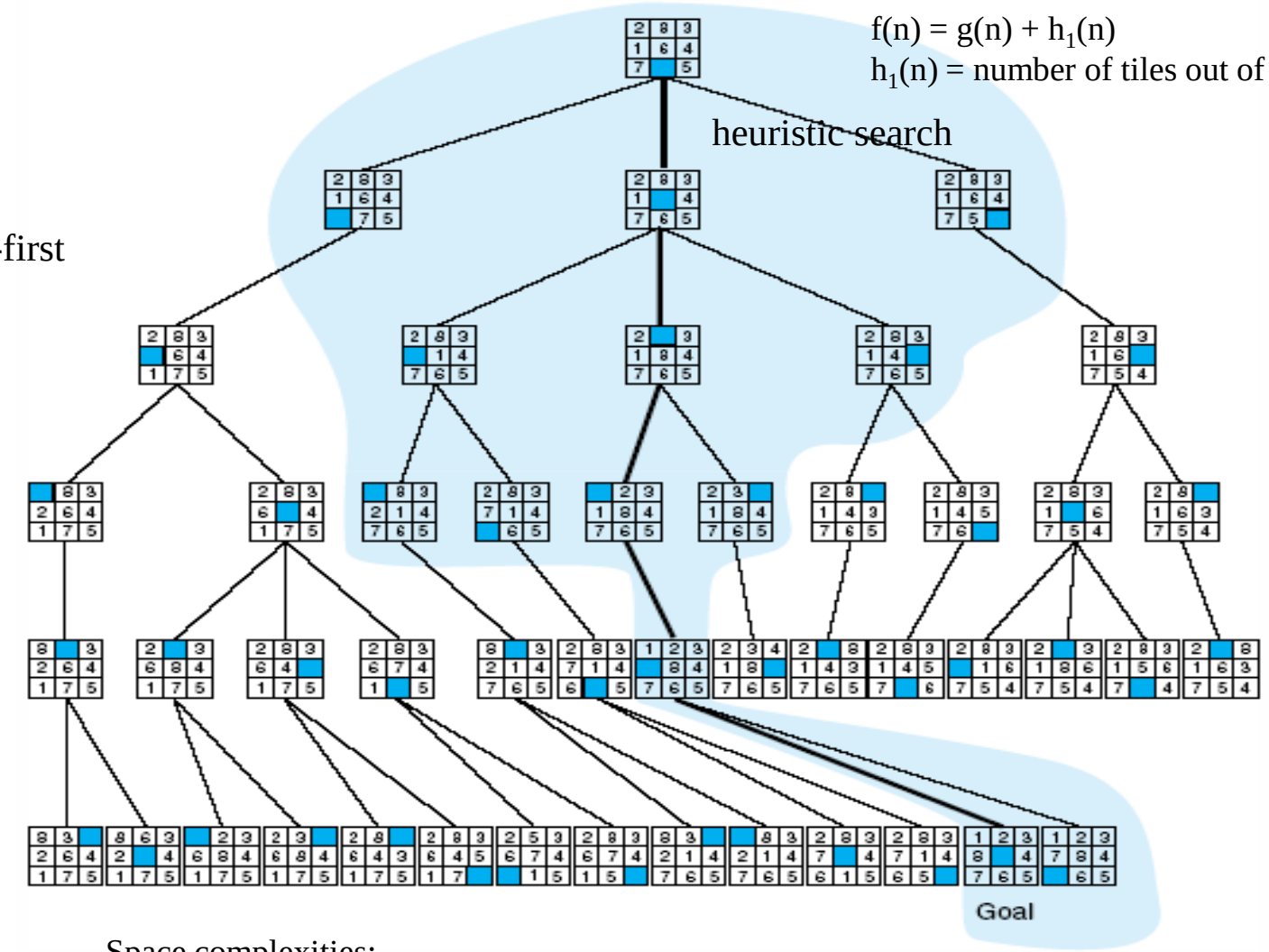
Heuristic search vs breadth-first search

$f(n) = g(n) + h_1(n)$

$h_1(n)$ = number of tiles out of place

heuristic search

breadth-first



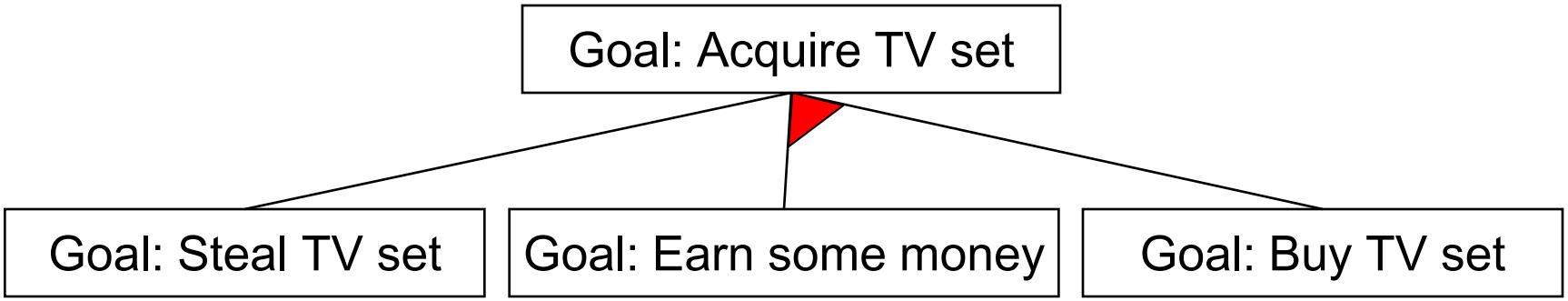
Space complexities:

Oracle path traverses 6 nodes.

Best-first with h1(n) touches 14 nodes.

Breadth-first looks at 46 nodes.

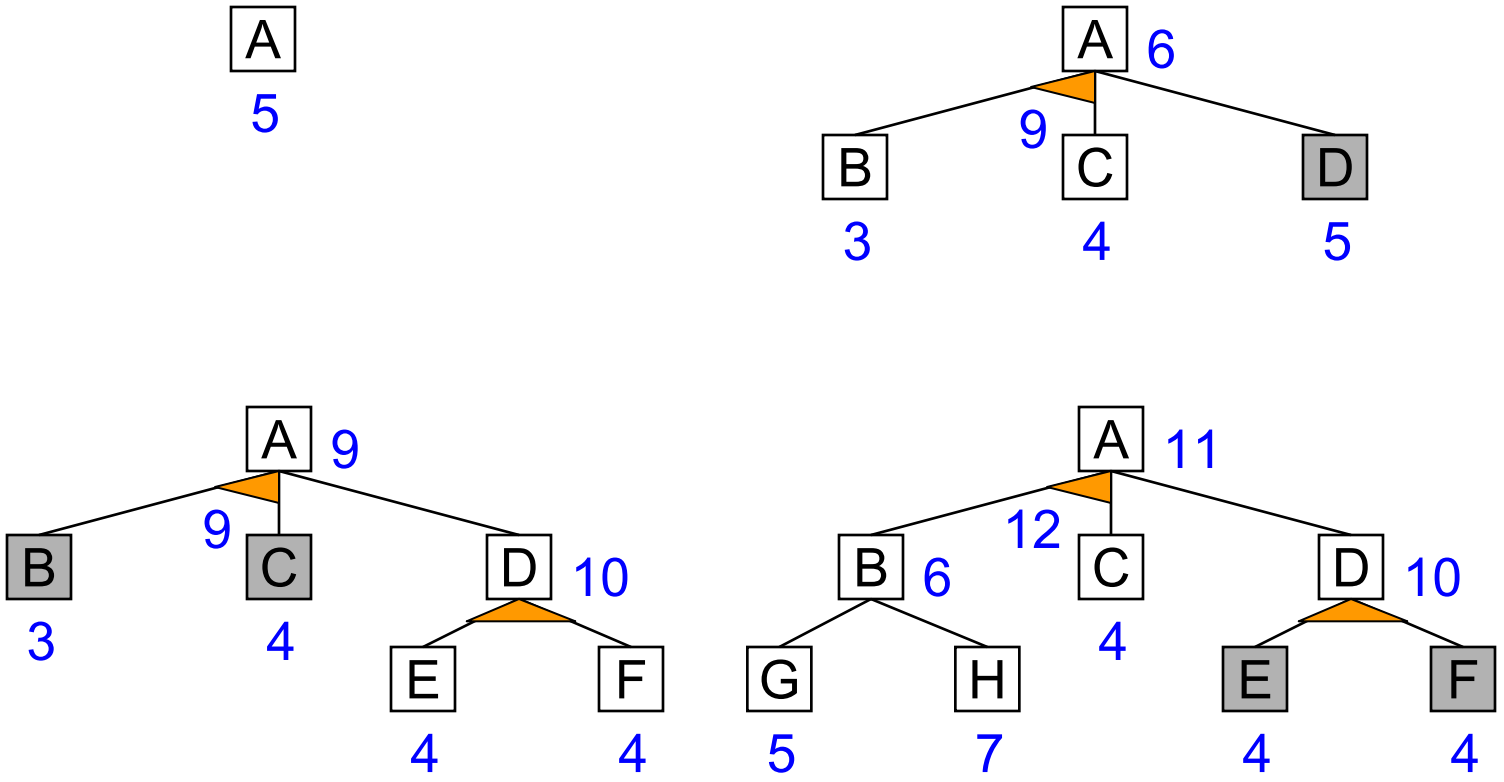
Problem Reduction



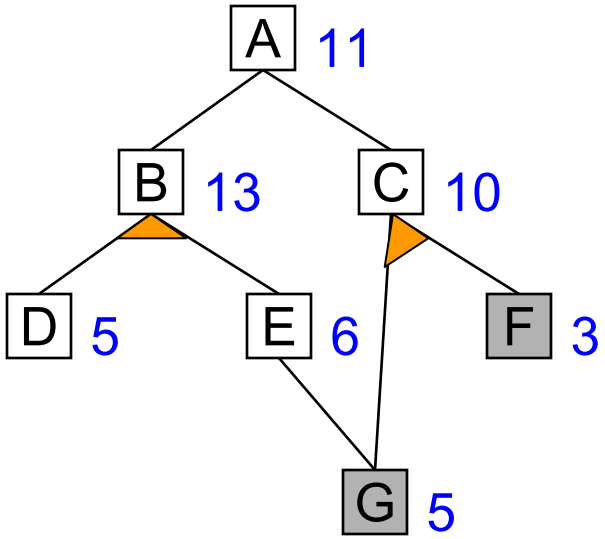
AND-OR Graphs

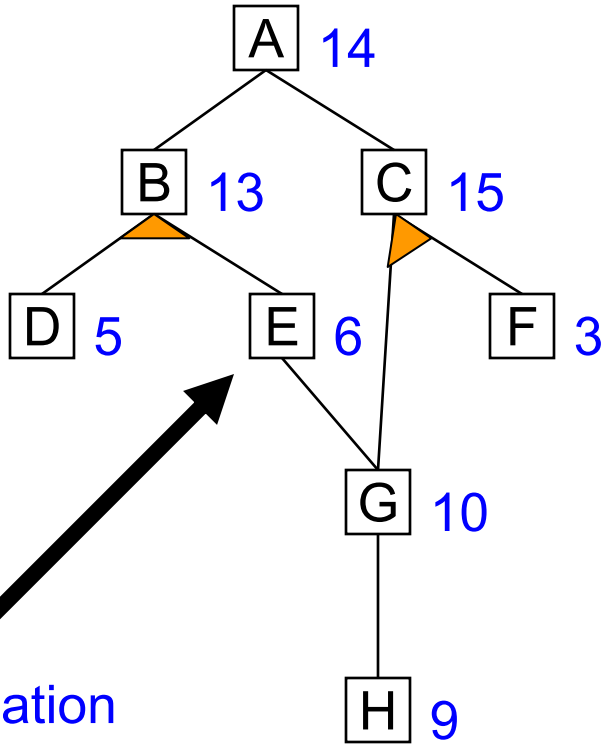
Algorithm AO* (Martelli & Montanari 1973, Nilsson 1980)

Problem Reduction: AO*



Problem Reduction: AO*





Necessary backward propagation

Constraint Satisfaction

- Many AI problems can be viewed as problems of constraint satisfaction.

Cryptarithmetic puzzle:

$$\begin{array}{r} \text{SEND} \\ + \quad \text{MORE} \\ \hline \text{MONEY} \end{array}$$

Constraint Satisfaction

- As compared with a straightforard search procedure, viewing a problem as one of constraint satisfaction can reduce substantially the amount of search.

Constraint Satisfaction

- Operates in a space of constraint sets.
- Initial state contains the original constraints given in the problem.
- A goal state is any state that has been constrained "enough".

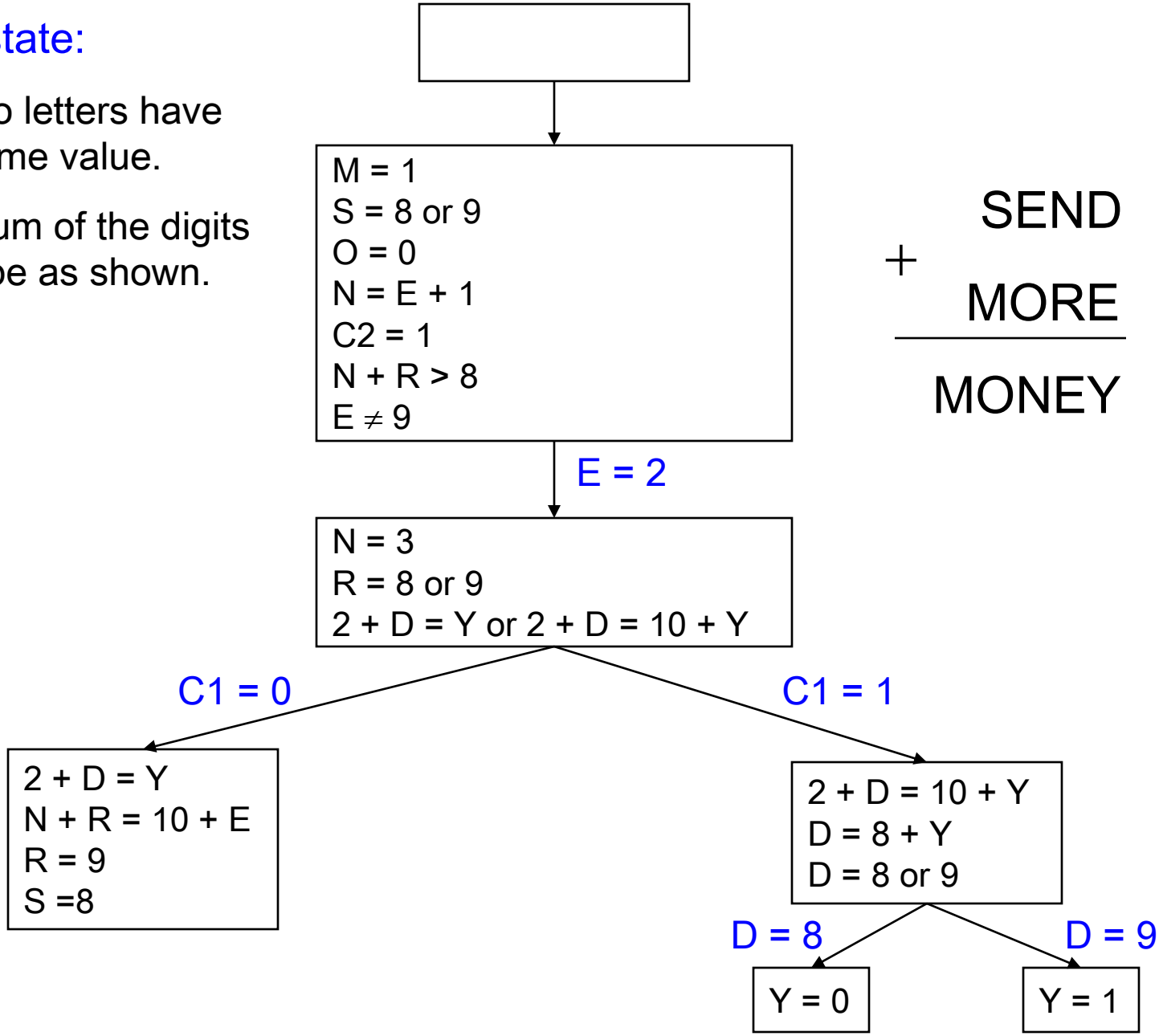
Constraint Satisfaction

Two-step process:

1. Constraints are discovered and propagated as far as possible.
2. If there is still not a solution, then search begins, adding new constraints.

Initial state:

- No two letters have the same value.
- The sum of the digits must be as shown.



SEND
\+ MORE
MONEY

Constraint Satisfaction

Two kinds of rules:

1. Rules that define valid constraint propagation.
2. Rules that suggest guesses when necessary.

Means-ends analysis

- Involves detection of difference between current state and goal state
- Once difference identified, an operator to reduce the difference must be found
- If operator cannot be applied to current state, find subproblem of getting to state where operator can be applied
- Operator may not result in goal state, solve second subproblem of getting from new state to goal state

MEA

- MEA process applied recursively
- Each rule (operator) has
  - LHS preconditions and RHS aspects of problem state changed.
- Difference table of rules and differences they can reduce.

Example

- Problem for household robot: moving desk with 2 things on it from one room to another.
- Main difference between start and goal state is location.

Move desk with 2 things on it to new room

	Push	Carry	Walk	Pickup	Putdown	Place
Move object	*	*				
Move robot			*			
Clear object				*		
Get object on object						*
Get arm empty					*	*
Be holding object				*		

Operator	Preconditions	Results
PUSH (obj, loc)	at(robot,obj) & large (obj) & clear (obj) & arm empty	at(obj, loc) & at(robot, loc)
CARRY (obj, loc)	at(robot, obj) & small(obj)	at(obj, loc) & at(robot, loc)
WALK(loc)	None	At(robot, loc)
PICKUP(obj)	At(robot, obj)	Holding(obj)
PUTDOWN(obj)	Holding(obj)	Not holding (obj)
PLACE(obj1, obj2)	At(robot,obj2) & holding (obj1)	on(obj1, obj2)

CARRY: preconditions cannot be met
PUSH: 4 preconditions
WALK to object, clear desk using PICKUP and PLACE. After PUSH objects not on desk. Must WALK to collect them and put on table using PICKUP and CARRY

Means-Ends Analysis

1. Compare CURRENT to GOAL. If no differences, return.
2. Otherwise select most important difference and reduce it by doing the following until success or failure is indicated.
   (a) Select an as yet untried operator *O* that is applicable to the current difference. If there are no such operators then signal failure.
   (b) Attempt to apply *O* to the current state. Generate descriptions of two states *O*-START a state in which *O*'s preconditions are satisfied and *O*-RESULT, the state that would result if O were applied in *O*-START.
   (c) If (FIRST-PART MEA (CURRENT, *O*-START) AND (LAST-PART MEA (*O*-RESULT, GOAL) are successful then signal success.

Homework

Exercises 1-14 (Chapter 3 – AI Rich & Knight)

Reading Algorithm A*
(http://en.wikipedia.org/wiki/A%2A_algorithm)