Heuristic Search

Outline

- Generate-and-test
- Hill climbing
- Simulated annealing
- Best-first search
- Means-ends analysis
- Constraint satisfaction

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.

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

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 → quit

     better than current state → new current state

not try all possible new states!

Simple Hill Climbing

Example: coloured blocks

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

Simple Hill Climbing

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

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:
   - Apply all the possible operators
   - Evaluate the best new state: goal → quit
     better than current state → new current 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:

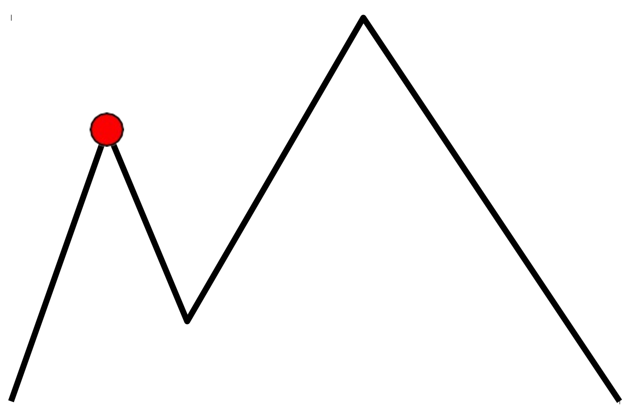
        goal → quit

        better than SUCC → set SUCC to this state
    - SUCC is better than the current state → set the new current

Hill Climbing: Disadvantages

Local maximum

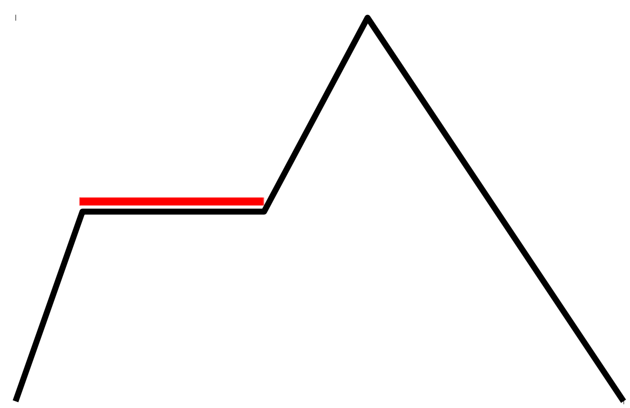
A state that is better than all of its neighbours, but not better than some other states far away.



Hill Climbing: Disadvantages

Plateau

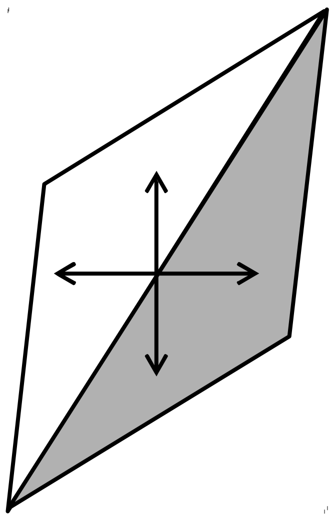
A flat area of the search space in which all neighbouring states have the same value.



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.



Hill Climbing: Disadvantages

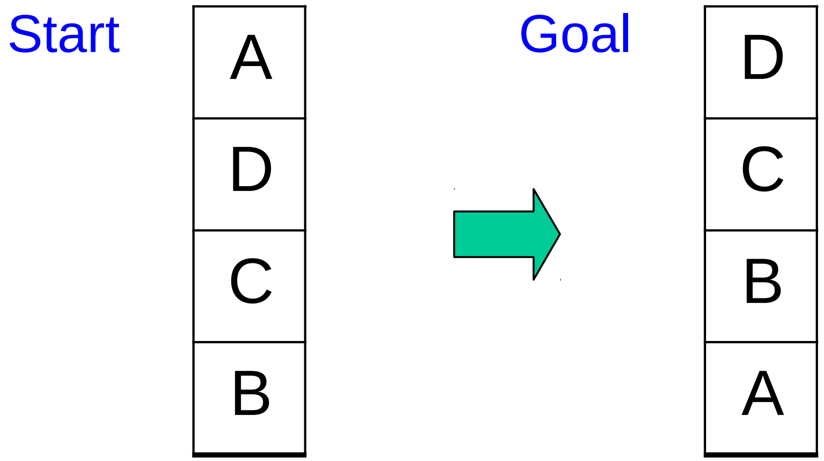
Ways Out

- 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.

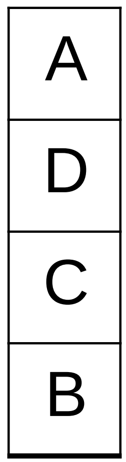
Hill Climbing: Blocks World



Hill Climbing: Blocks World

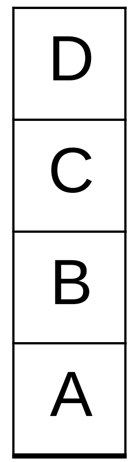
Start

0



Goal

4

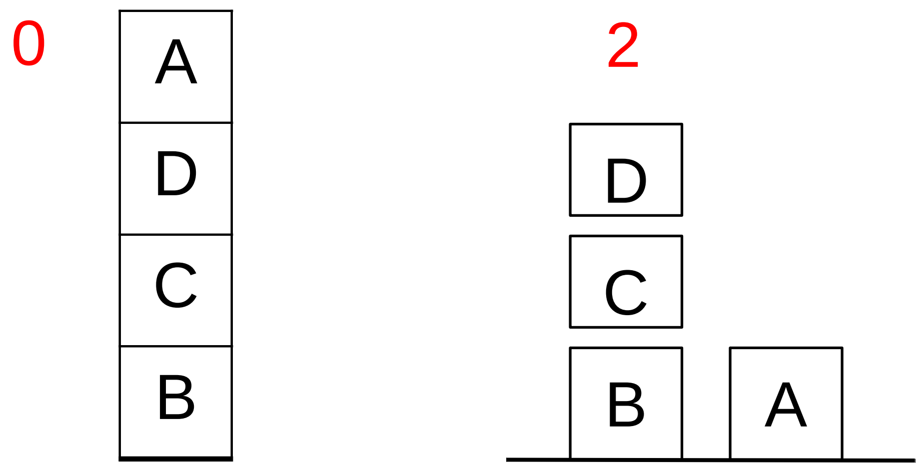


Local heuristic:

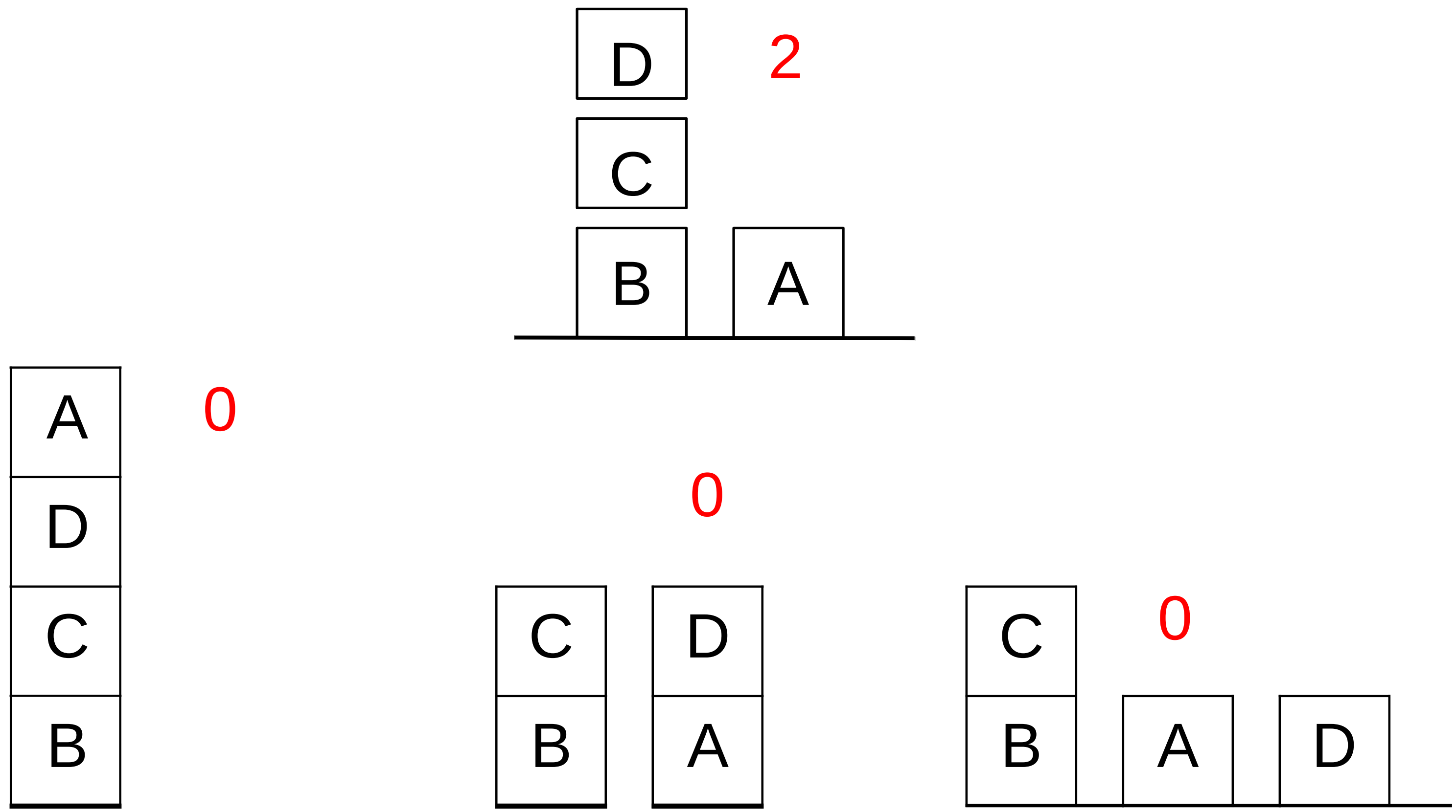
+1 for each block that is resting on the thing it is supposed to be resting on.

-1 for each block that is resting on a wrong thing.

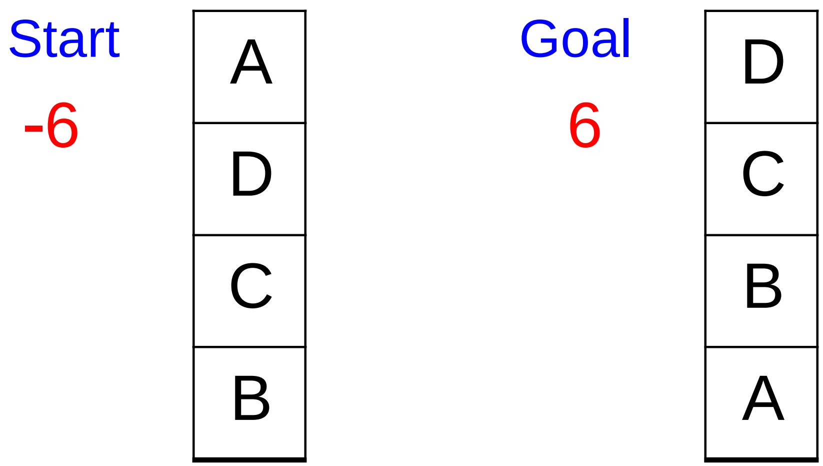
Hill Climbing: Blocks World



Hill Climbing: Blocks World

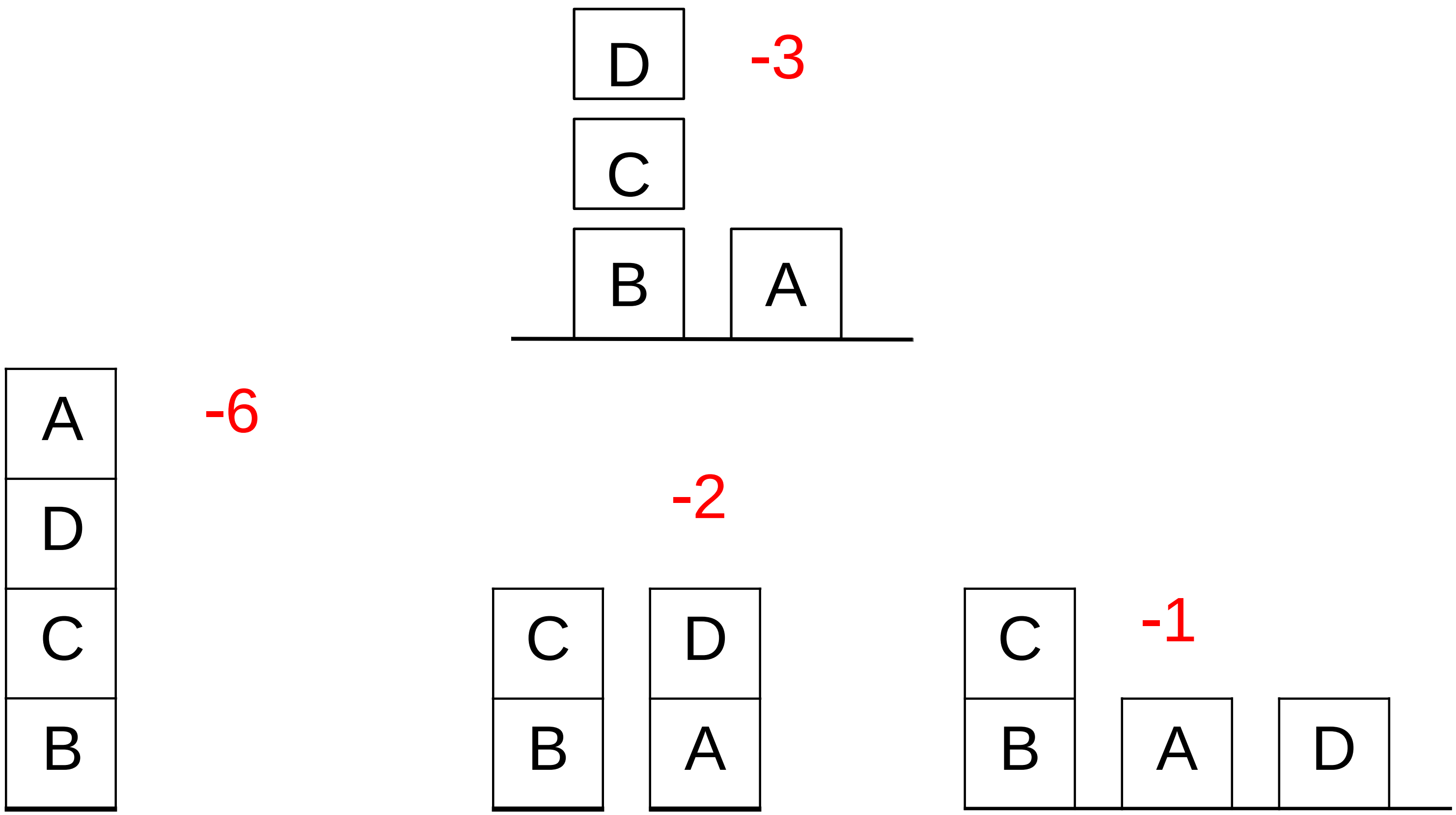


Hill Climbing: Blocks World



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.

Hill Climbing: Blocks World



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.

Simulated Annealing

- A variation of hill climbing in which, at the beginning of the process, some downhill moves may be made.

Simulated Annealing

- 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

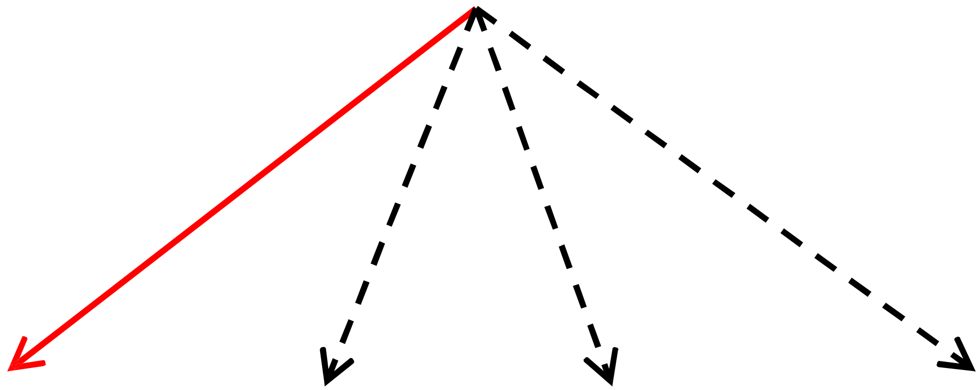
     ΔE = Val(current state) - Val(new state)

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

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

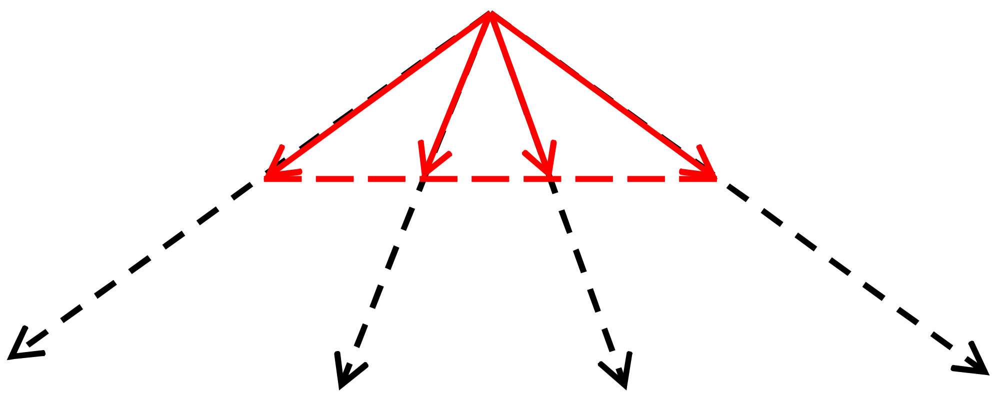
Best-First Search

- Depth-first search:
  - Pro: not having to expand all competing branches
  - Con: getting trapped on dead-end paths



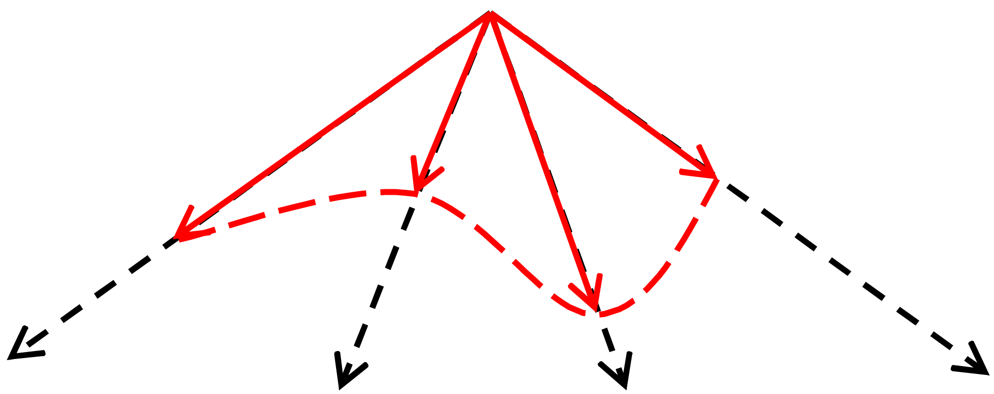
Best-First Search

- Breadth-first search:
  - Pro: not getting trapped on dead-end paths
  - Con: having to expand all competing branches

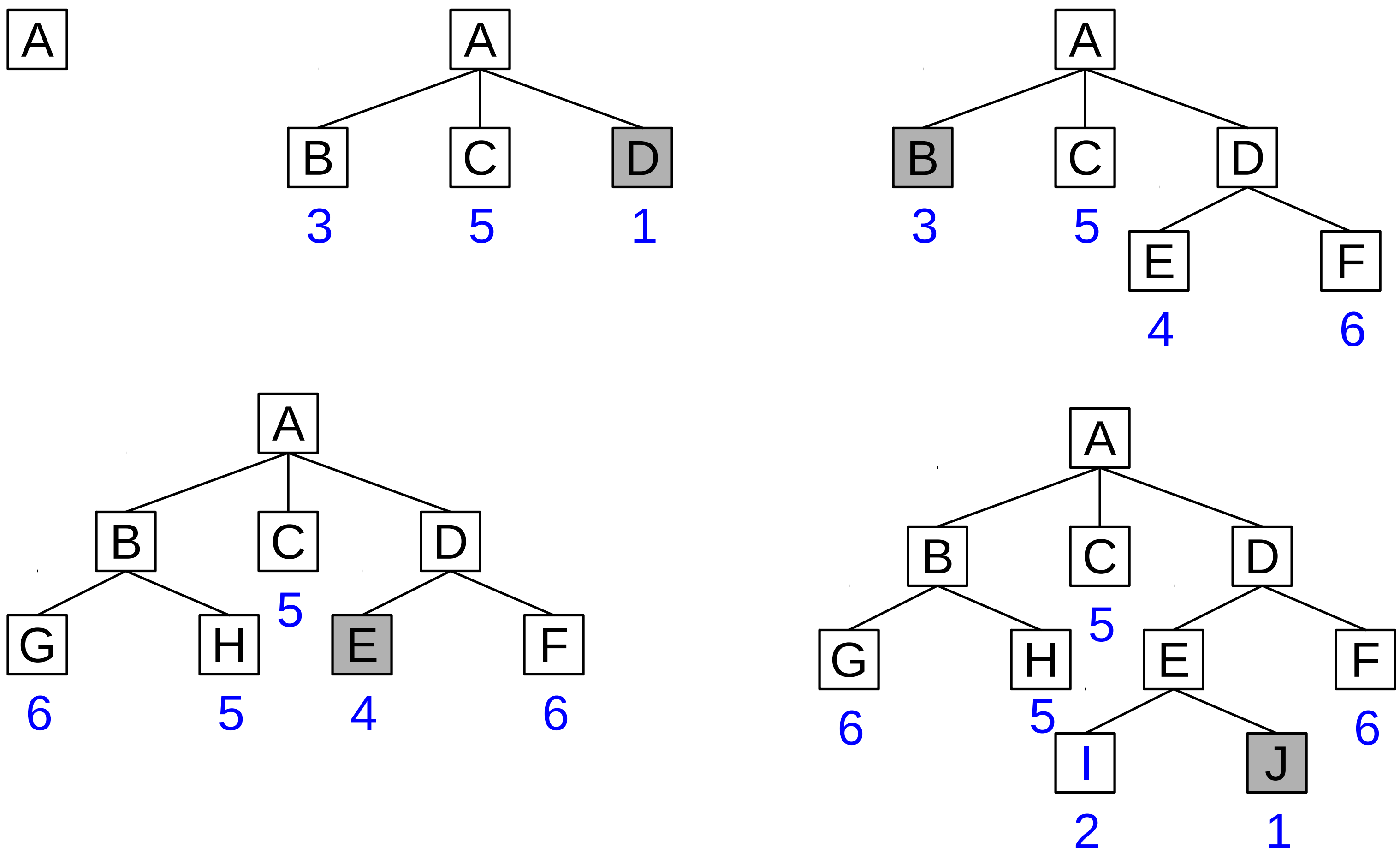


Best-First Search

⇒ Combining the two is to follow a single path at a time, but switch paths whenever some competing path looks 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 → evaluate it, add it to OPEN, record its parent

     generated before → change parent, update successors

Best-First Search

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

Best-First Search

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

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

Best-First Search

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

Neither optimal nor complete

Best-First Search

- Greedy search:
  h(n) = 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.