INFO 6205
Program Structure and Algorithms

Heuristic Search

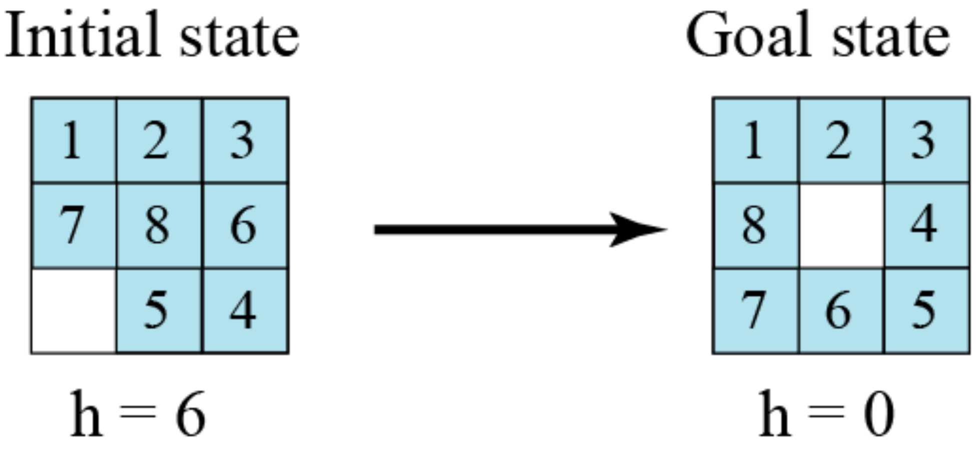
Nik Bear Brown

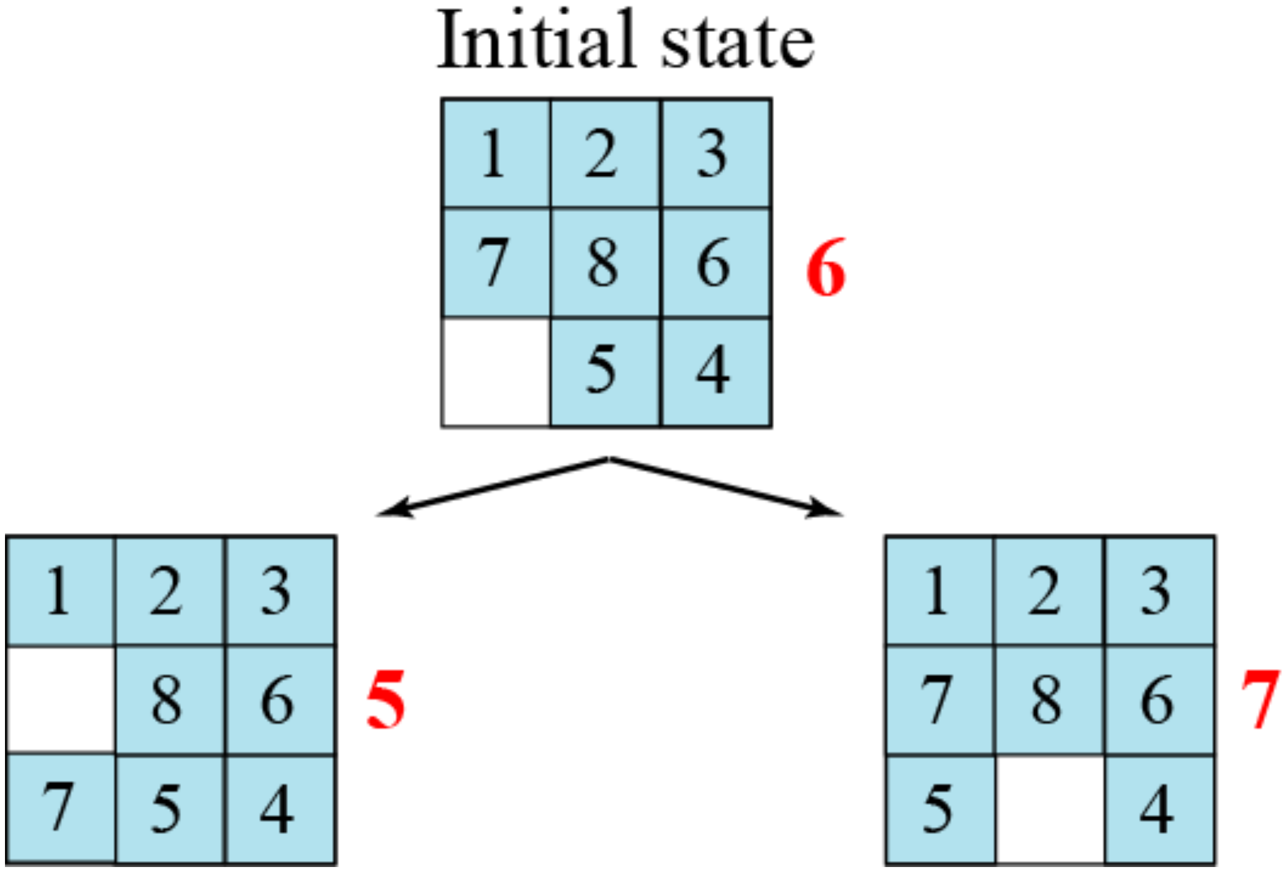
Topics

- Heuristic Search
- - Greedy Search
- - A*-Search
- - IDA*-Search

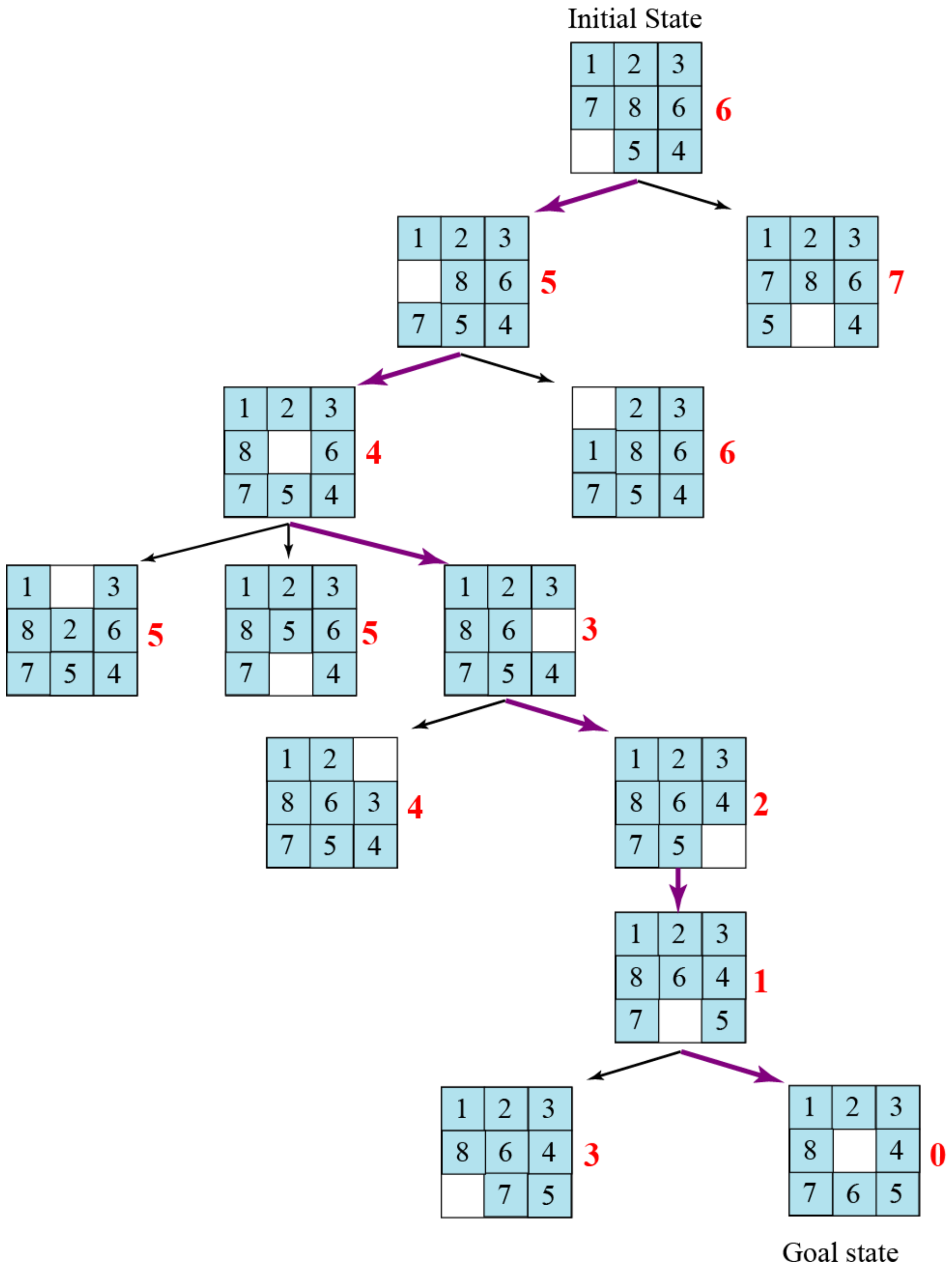
Heuristic search

Using heuristic search, we assign a quantitative value called a heuristic value (h value) to each node. This quantitative value shows the relative closeness of the node to the goal state. For example, consider solving the 8-puzzle.





Initial state
1 2 3
7 8 6
5 4
6
1 2 3
8 6
7 5 4
5
1 2 3
7 8 6
5 4
7



Initial State
1 2 3
7 8 6
5 4
6
1 2 3
8 6
7 5 4
5
1 2 3
7 8 6
5 4
7
1 2 3
8 6
7 5 4
4
2 3
1 8 6
7 5 4
6
1 3
8 2 6
7 5 4
5
1 2 3
8 5 6
7 4
5
1 2 3
8 6
7 5 4
3
1 2
8 6 3
7 5 4
4
1 2 3
8 6 4
7 5
2
1 2 3
8 6 4
7 5
1
1 2 3
8 6 4
7 5
3
1 2 3
8 4
7 6 5
0
Goal state

Uniform Cost Search

- **Uniform Cost Search**
  - orders the nodes on the Q according to path cost from S
  - always expands the node with minimum path cost from S

```
Initialize: Let Q = {S}
While Q is not empty
        pull Q1, the first element in Q
        if Q1 is a goal report(success) and quit
        else
                child_nodes = expand(Q1)
                <eliminate child_nodes which represent loops>
                put remaining child_nodes in Q
                sort Q according to path-cost to each node
        end
Continue
```

Heuristics and Search

- in general
  - a heuristic is a “rule-of-thumb” based on domain-dependent knowledge to help you solve a problem

- in search
  - one uses a heuristic function of a state where

    h(node) = estimated cost of cheapest path from the state for that node to a goal state G

    - $h(G) = 0$
    - $h(\text{other nodes}) \geq 0$
    - (note: we will assume all individual node-to-node costs are > 0)

A(*) Algorithm

- Goal: Find shortest path
- Prerequisites
  - Graph
  - Method to estimate distance between points (heuristic)
- Basic Method
  - Try all paths?
    - Takes time
  - Orient search towards target
    - Minimizes areas of the map to be examined
    - Uses heuristics that indicate the estimated cost of getting to the destination
    - Main advantage

A(*) Algorithm

- Algorithm
  - Open list
    - Nodes that need to be considered as possible starts for further extensions of the path
  - Closed list
    - Nodes that have had all their neighbors added to the open list
  - G score
    - Contains the length or weight of the path from the current node to the start node
    - Low lengths are better
    - Every node has a G score
  - H score
    - Heuristic
    - Resembles G score except it represents an estimate of the distance from the current node to the endpoint
    - To find shortest path, this score must underestimate the distance

The A* Algorithm

- A heuristic h is admissible if
  - it for any node n it does NOT overestimate the true path cost from n to the nearest goal.

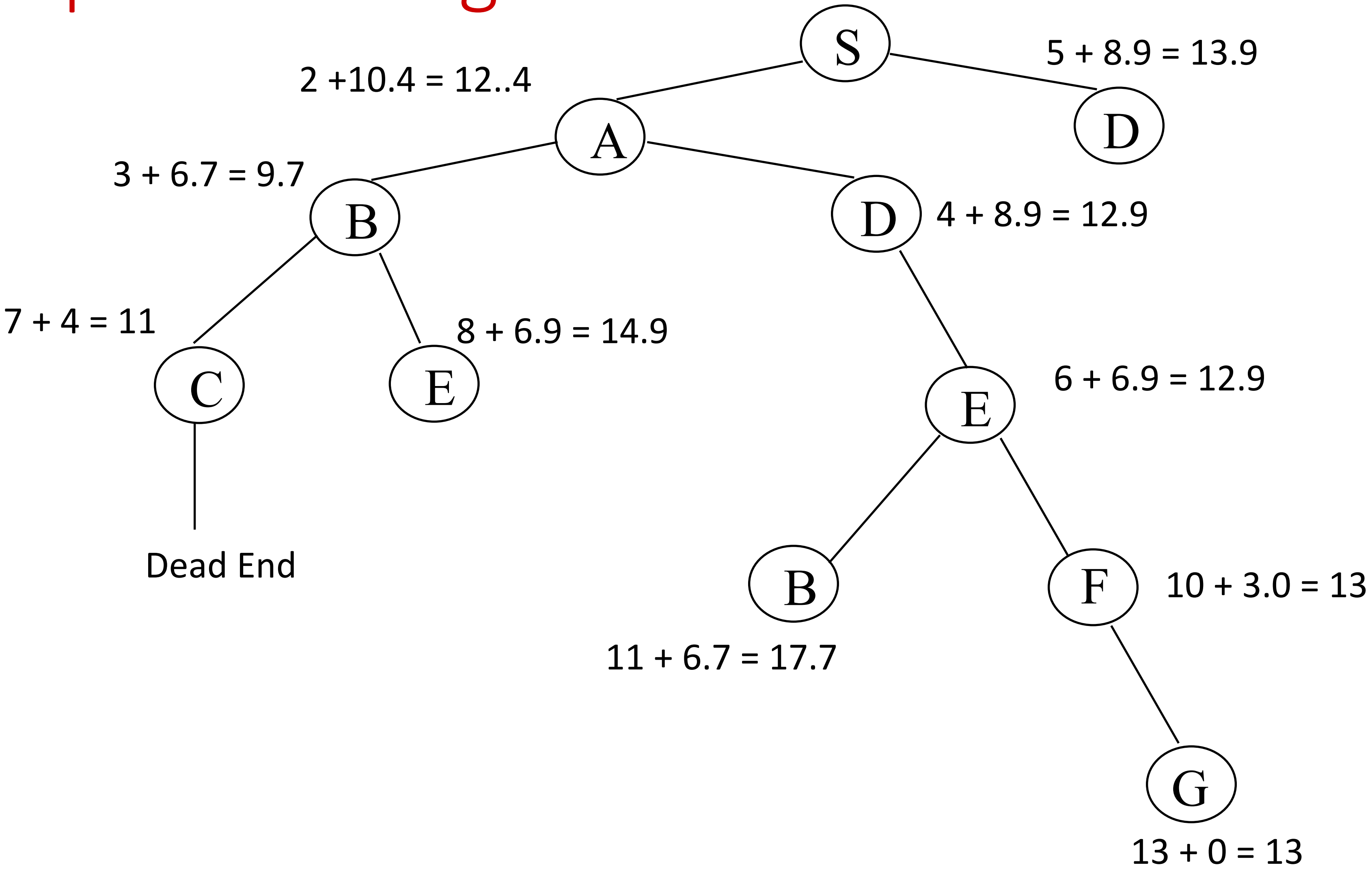
- The A* search is a search algorithm orders the nodes on the Q according to f(n)=g(n)+h(n), where h(n) is an admissible heuristic
  - i.e., it sorts nodes on Q according to an admissible heuristic h*
  - It is like uniform-cost,
    - but uses fcost(node) = path-cost(S to node) + h(node)
    - rather than just path cost(S to node)

  - note that uniform cost search can be viewed as A* search where h(n) equals 0 for all n (the latter heuristic equal to 0 for every node is clearly admissible! Why?)

Pseudo-code for the A* Search Algorithm

```
Initialize: Let Q = {S}
While Q is not empty
     pull Q1, the first element in Q
     if Q1 is a goal report(success) and quit
     else
          child_nodes = expand(Q1)
          <eliminate child_nodes which represent loops>
          put remaining child_nodes in Q
          sort Q according to ucost = pathcost(S to node) + h*(node)
     end
Continue
```

Example of A* Algorithm in action



Comments on heuristic estimation

- The estimate of the distance is called a heuristic
  - typically it comes from domain knowledge
  - e.g., the straight-line distance between 2 points

- If the heuristic never overestimates, then the search procedure using this heuristic is "admissible", i.e.,
  - h*(N) is less than or equal to realcost(N to G)

- A* is a search with admissible heuristic is optimal
  - i.e., if one uses an admissible heuristic to order the search one is guaranteed to find the optimal solution

- The closer the heuristic is to the real (unknown) path cost, the more effective it will be, ie if h1(n) and h2(n) are two admissible heuristics and $h1(n) \leq h2(n)$ for any node n then A* search with h2(n) will in general expand fewer nodes than A* search with h1(n)

Properties of A*

- A* generates an optimal solution if h(n) is an admissible heuristic and the search space is a tree:
  - h(n) is **admissible** if it never overestimates the cost to reach the destination node

- A* generates an optimal solution if h(n) is a consistent heuristic and the search space is a graph:
  - h(n) is **consistent** if for every node n and for every successor node n’ of n:

$$h(n) \leq c(n,n') + h(n')$$

- If h(n) is consistent then h(n) is admissible
- Frequently when h(n) is admissible, it is also consistent

Admissible Heuristics

- A heuristic is admissible if it is too optimistic, estimating the cost to be smaller than it actually is.

- Example:

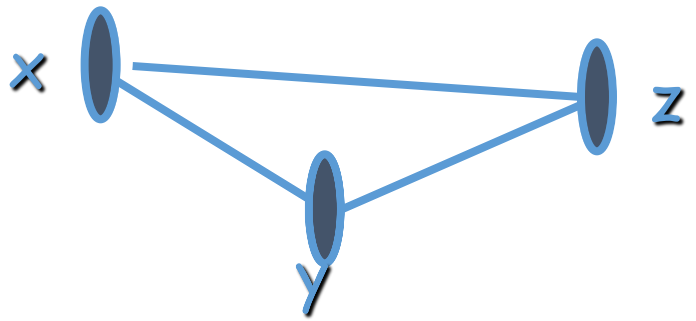
  In the road map domain,

  h(n) = “Euclidean distance to destination”

  is admissible as normally cities are not connected by roads that make straight lines

Metric Space

- A set of points X
- Distance function d(x,y)
  $d : X \rightarrow [0\ldots \infty)$
  - d(x,y) = 0 iff x=y
  - d(x,y) = d(y,x) Symmetric
  - d(x,z) ≤ d(x,y) + d(y,z) Triangle inequality



- Metric space M(X,d)

Dominance

If $h2(n) \geq h1(n)$ for all n (both admissible)

then h2 dominates h1

h2 is better for search: it is guaranteed to expand
less or equal nr of nodes.

Examples of Heuristic Functions for A*

- the 8-puzzle problem
  - the number of tiles in the wrong position
    - is this admissible?
  - the sum of distances of the tiles from their goal positions, where distance is counted as the sum of vertical and horizontal tile displacements (“Manhattan distance”)
    - is this admissible?

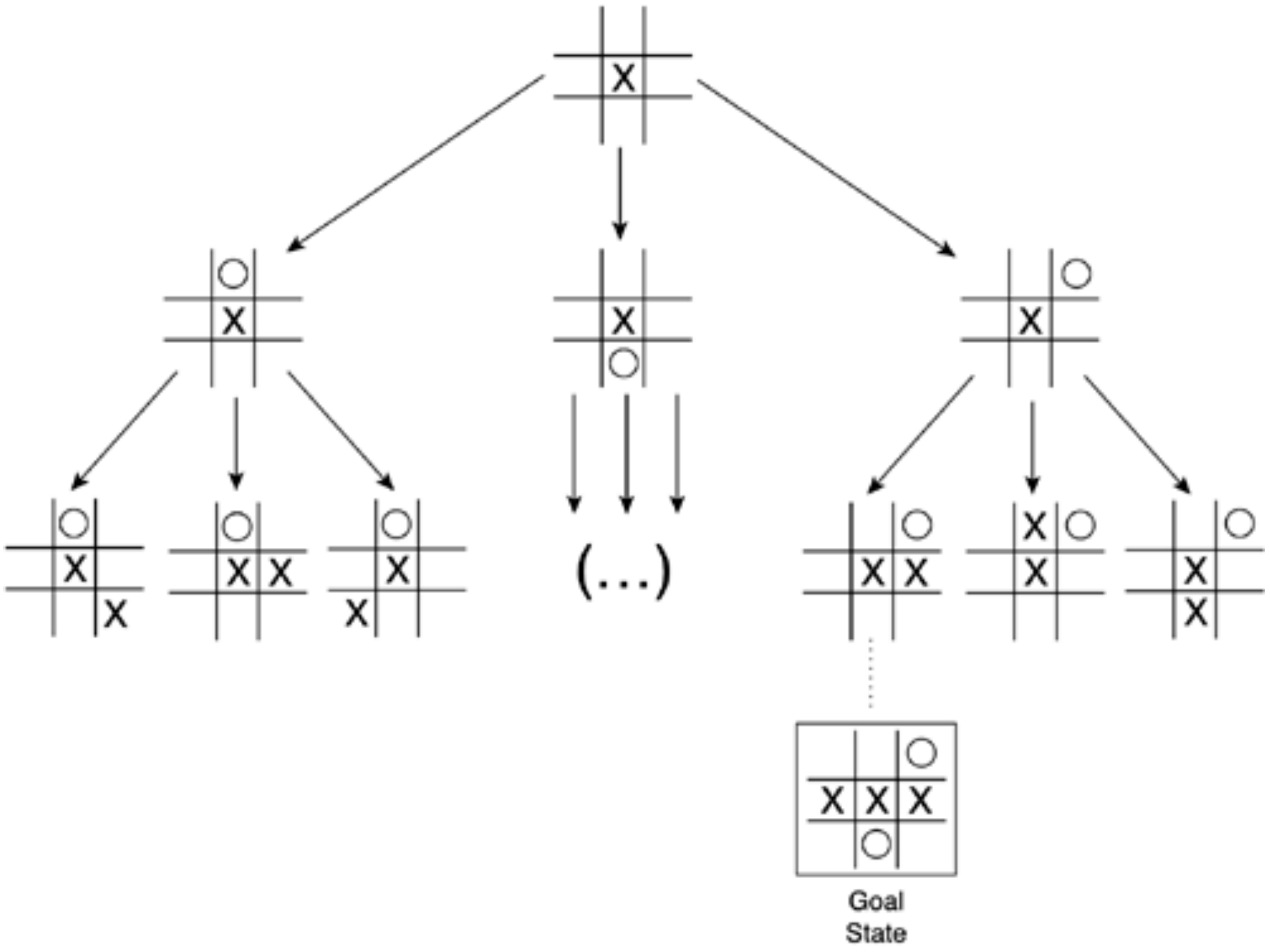
- How can we invent admissible heuristics in general?
  - look at “relaxed” problem where constraints are removed
    - e.g., we can move in straight lines between cities
    - e.g., we can move tiles independently of each other

IDA(*) Algorithm

- A*, like depth-first search, except based on increasing values of total cost rather than increasing depths
- IDA* sets bounds on the heuristic cost of a path, instead of depth
- A* always finds a cheapest solution if the heuristic is admissible
- IDA* is optimal in terms of solution cost, time, and space for admissible best-first searches on a tree

State Space

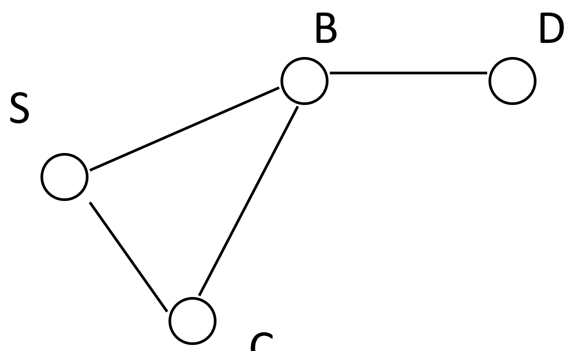
Model of a system as a set of input, output and state variables



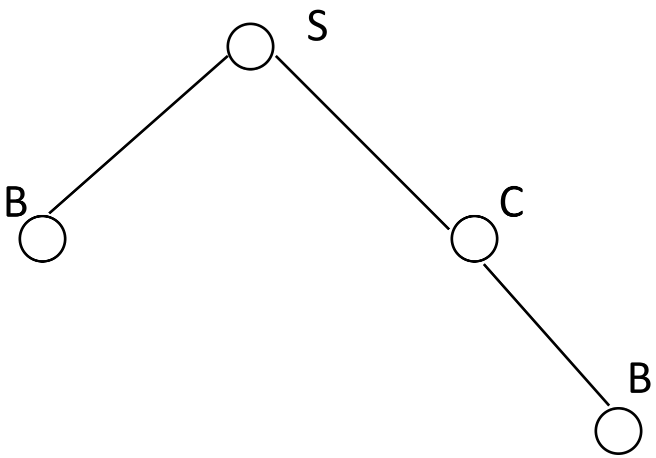
Setting Up a State Space Model

- State-space Model is a Model for The Search Problem
  - usually a set of discrete states X
    - e.g., in driving, the states in the model could be towns/cities
- Start State - a state from X where the search starts
- Goal State(s)
  - a goal is defined as a target state
  - For now: all goal states have utility 1, and all non-goals have utility 0
  - there may be many states which satisfy the goal
    - e.g., drive to a town with an airport
  - or just one state which satisfies the goal
    - e.g., drive to Las Vegas
- Operators
  - operators are mappings from X to X
    - e.g. moves from one city to another that are legal (connected with a road)

A State Space and a Search Tree are different



State Space



Example of a Search Tree

- A State Space represents all states and operators for the problem
- A Search Tree is what an algorithm constructs as it solves a search problem:
  - so we can have different search trees for the same problem
  - search trees grow in a dynamic fashion until the goal is found

Puzzle-Solving as Search

- You have a 3-gallon and a 4-gallon
- You have a faucet with an unlimited amount of water
- You need to get exactly 2 gallons in 4-gallon jug

- State representation: (x, y)
  - x: Contents of four gallon
  - y: Contents of three gallon

- Start state: (0, 0)
- Goal state(s) G = {(2, 0), (2, 1), (2, 2)}
- Operators
  - Fill 3-gallon (0,0)->(0,3), fill 4-gallon (0,0)->(0,4)
  - Fill 3-gallon from 4-gallon (4,0)->(1,3), fill 4-gallon from 3-gallon (0,3)->(3,0) or (1,3)->(4,0) or (2,3)->(4,0)….
  - Empty 3-gallon into 4-gallon, empty 4-gallon into 3-gallon
  - Dump 3-gallon down drain (0,3)->(0,0), dump 4-gallon down drain (4,0)->(0,0)