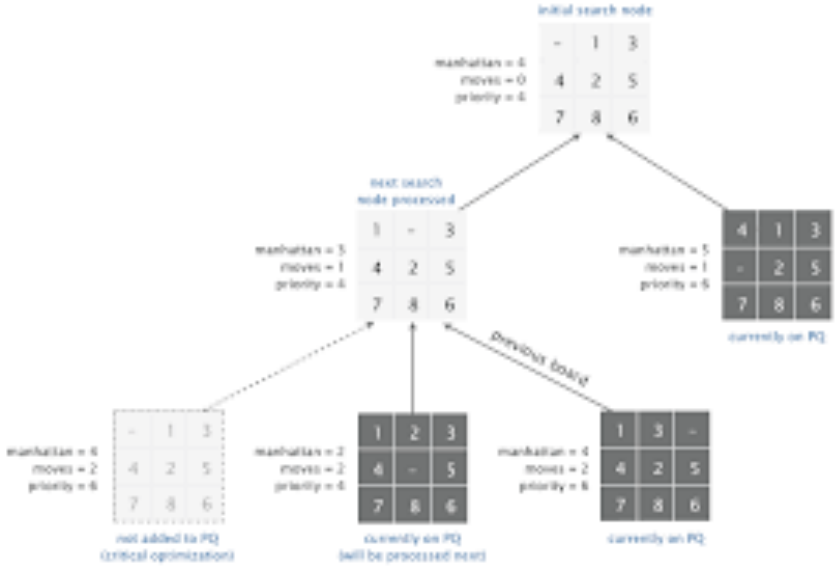
Artificial Intelligence

Class 3: Search (Ch. 3.1–3.3)



Some material adopted from notes by Charles R. Dyer, University of Wisconsin-Madison

Bookkeeping

- HW 1, pt III
  - Intro to Python
    - Sets, Tuples, Lists, Dictionaries, …
  - If you need resources, ask us!
- Pre-reading for today
  - 3.1 intro, 3.1.1, skim 3.3
- Reading after class
  - 3.1-3.3

Today's Class

- Goal-based agents
- Representing states and operators
- Example problems
- Generic state-space search algorithm

Everything in AI comes down to search.

Goal: understand search, and understand how.

Pre-Reading Review

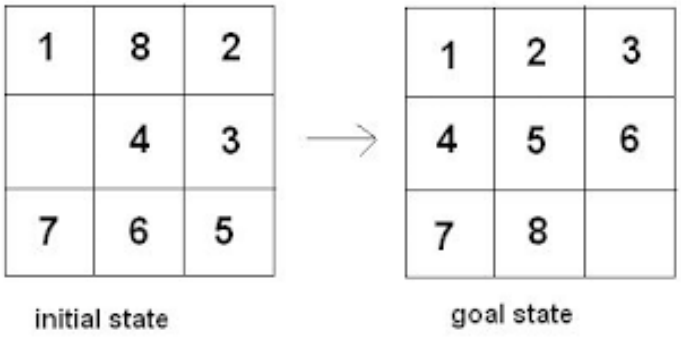
- What is **search** (a.k.a. **state-space search**)?
- What are these concepts in search?
  - Initial state
  - Actions / transition model
  - State space graph
  - Step cost / path cost
  - Goal test (cf. goal)
  - Solution / optimal solution
- What is an **open-loop system**?
- What is the difference between **expanding** and **generating** a state?
- What is the **frontier** (a.k.a. **open list**)?

Representing Actions

- Actions here are:
  - **Discrete events**
  - That occur at an **instant of time**
- For example:
  - State: "Mary is in class"
  - Action "Go home"
  - New state: "At home"
  - There is no representation of a state where she is in between (i.e., in the state of "going home").

Representing Actions

- Number of actions / operators depends on **representation** used in describing a state
  - 8-puzzle: could specify 4 possible moves for each of the 8 tiles: **4*8=32 operators**.
  - Or, could specify four moves for the "blank" square: **4 operators**



- Careful representation can simplify a problem!

Representing States

- What information about the world sufficiently describes all **relevant** aspects to solving the goal?
- That is: what knowledge must be in a state description to adequately describe the current state of the world?
- The **size of a problem** is usually described in terms of the **number of states** that are possible
  - Tic-Tac-Toe has about 3^9 states.
  - Checkers has about 10^{40} states.
  - Rubik's Cube has about 10^{19} states.
  - Chess has about 10^{120} states in a typical game.

Closed World Assumption

- We will generally use the **Closed World Assumption**:

  "All necessary information about a problem domain is available in each percept so that each state is a complete description of the world."

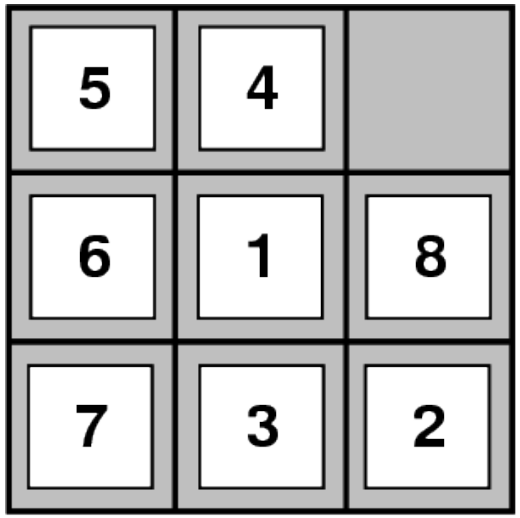
- No incomplete information at any point in time.

Some Example Problems

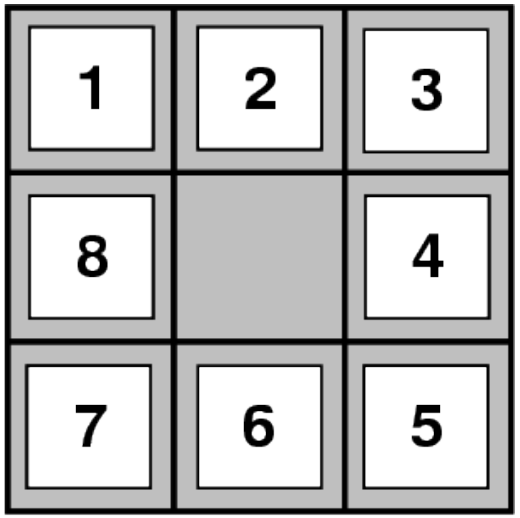
- Toy problems and micro-worlds
  - 8-Puzzle
  - Missionaries and Cannibals
  - Cryptarithmetic
  - Remove 5 Sticks
  - Water Jug Problem
- Real-world problems

8-Puzzle

Given an initial configuration of 8 numbered tiles on a 3 x 3 board, move the tiles in such a way so as to produce a desired goal configuration of the tiles.



Start State



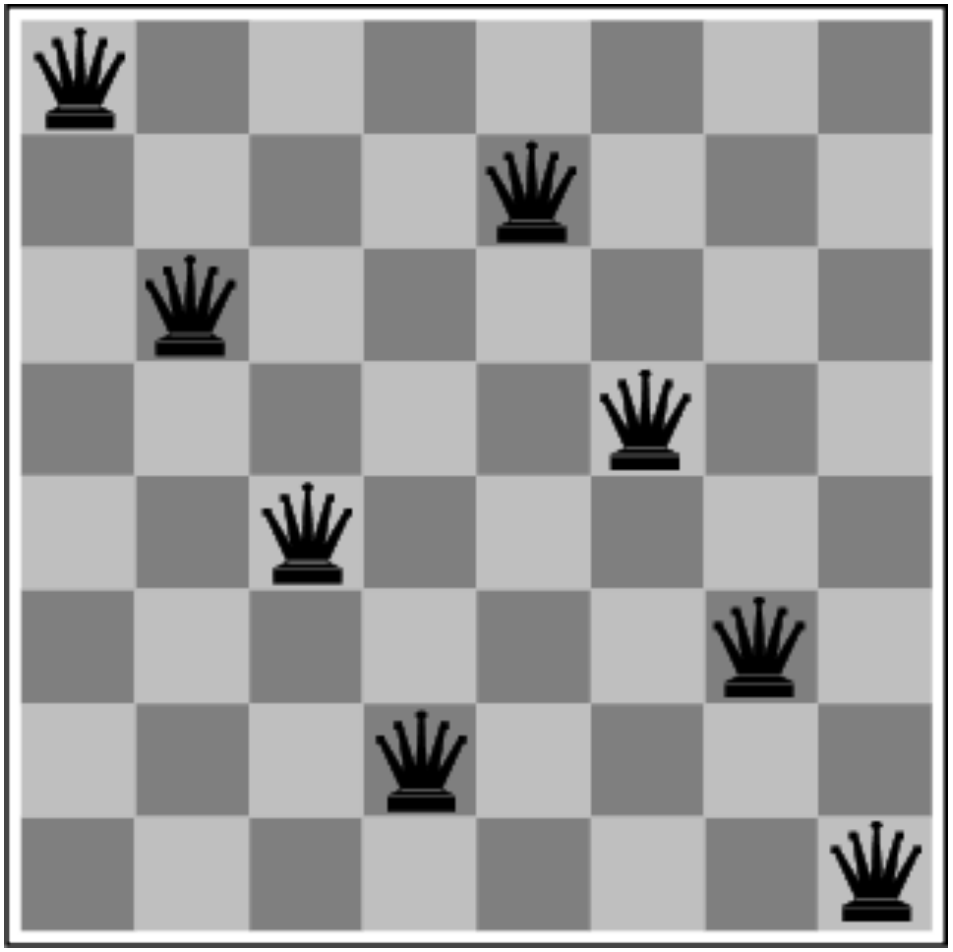
Goal State

8 puzzle

- **State:** 3 x 3 array configuration of the tiles on the board
- **Operators:**
  - Move blank square Left, Right, Up or Down.
  - This is a more efficient encoding of the operators!
- **Initial State:** Start-configuration of the board.
- **Goal:** Some configuration of the board.

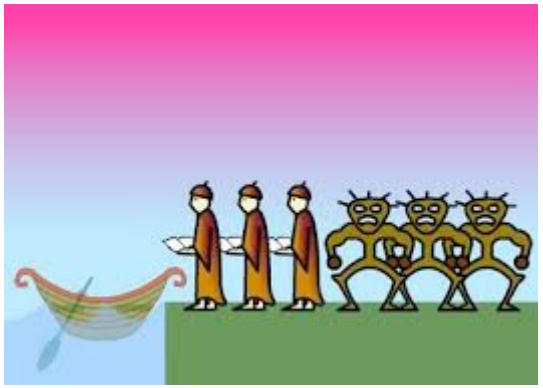
The 8-Queens Problem

Place eight queens on a chessboard such that no queen can reach any other



Missionaries and Cannibals

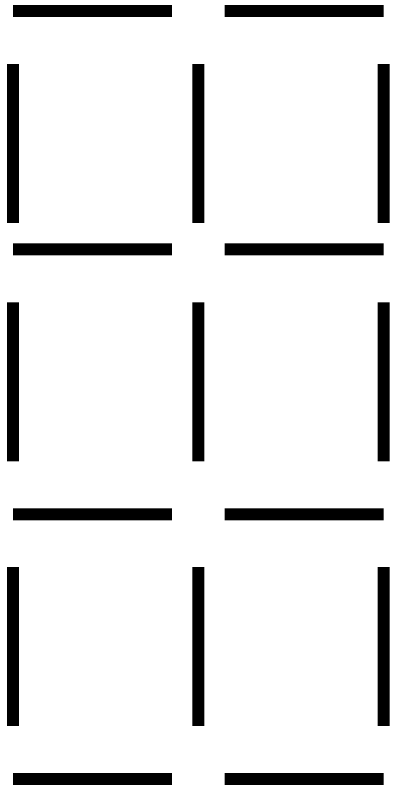
3 missionaries, 3 cannibals, and 1 boat



- **Goal**: Move everyone across the river.
- **Constraint:** Missionaries can never be outnumbered on banks.
- **State:** configuration of missionaries and cannibals and boat on each side of river.
- **Operators:** Move boat containing some set of occupants across the river (in either direction) to the other side.

Remove 5 Sticks

- Given the following configuration of sticks, remove exactly 5 sticks in such a way that the remaining configuration forms exactly 3 squares.



Some Real-World Problems

- Route finding
- Touring (traveling salesman)
- Logistics
- VLSI layout
- Robot navigation
- Learning

Knowledge Representation Issues

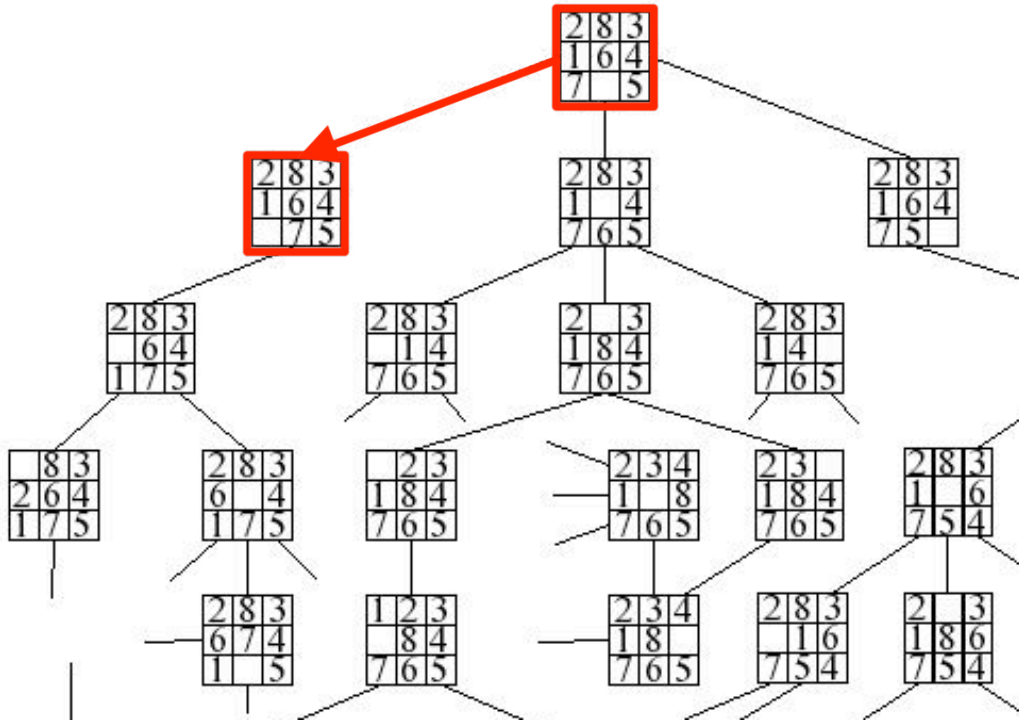
- What's in a state?
  - Is the color of the boat relevant to solving Missionaries and Cannibals problem?
  - Is sunspot activity relevant to predicting the stock market?
  - What to represent is a very hard problem!
  - Usually left to the system designer to specify.
- What **level of abstraction** to describe the world?
  - Too fine-grained and we "miss the forest for the trees"
  - Too coarse-grained and we miss critical information

Knowledge Representation Issues

- Number of states depends on
  - Representation
  - Level of abstraction
- In the Remove-5-Sticks problem:
  - If we represent individual sticks, then there are 17-choose-5 possible ways of removing 5 sticks (6188)
  - If we represent the "squares" defined by 4 sticks, there are 6 squares initially and we must remove 3
  - So, 6-choose-3 ways of removing 3 squares (20)

Formalizing Search in a State Space

- A *state space* is a **graph** (V, E):
  - V is a set of **nodes**
  - E is a set of **arcs**
  - Each arc is directed from a node to another node
- How does that work for 8-puzzle?



Formalizing Search in a State Space

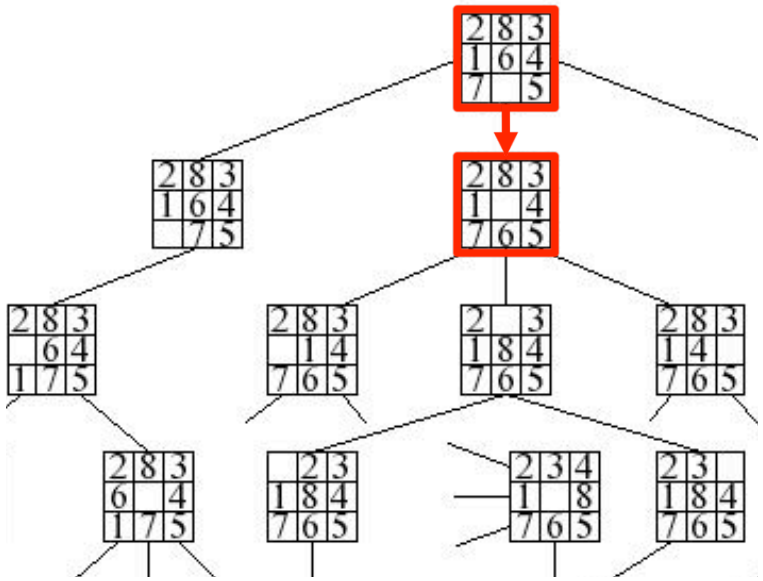
- V: A **node** is a data structure that contains a state description plus other information such as the parent of the node, the name of the operator that generated the node from that parent, and other bookkeeping data
- E: Each **arc** corresponds to an instance of one of the operators. When the operator is applied to the state associated with the arc's source node, then the resulting state is the state associated with the arc's destination node

Formalizing Search II

- Each arc has a fixed, positive **cost**
  - Corresponding to the cost of the operator
  - What is "cost" of doing that action?
- Each node has a set of **successor nodes**
  - Corresponding to all operators (actions) that can apply at source node's state
  - **Expanding** a node is **generating** successor nodes, and adding them (and associated arcs) to the state-space graph

Formalizing Search II

- One or more nodes are designated as **start nodes**
- A **goal test** predicate is applied to a *state* to determine if its associated node is a goal node



Water Jug Problem

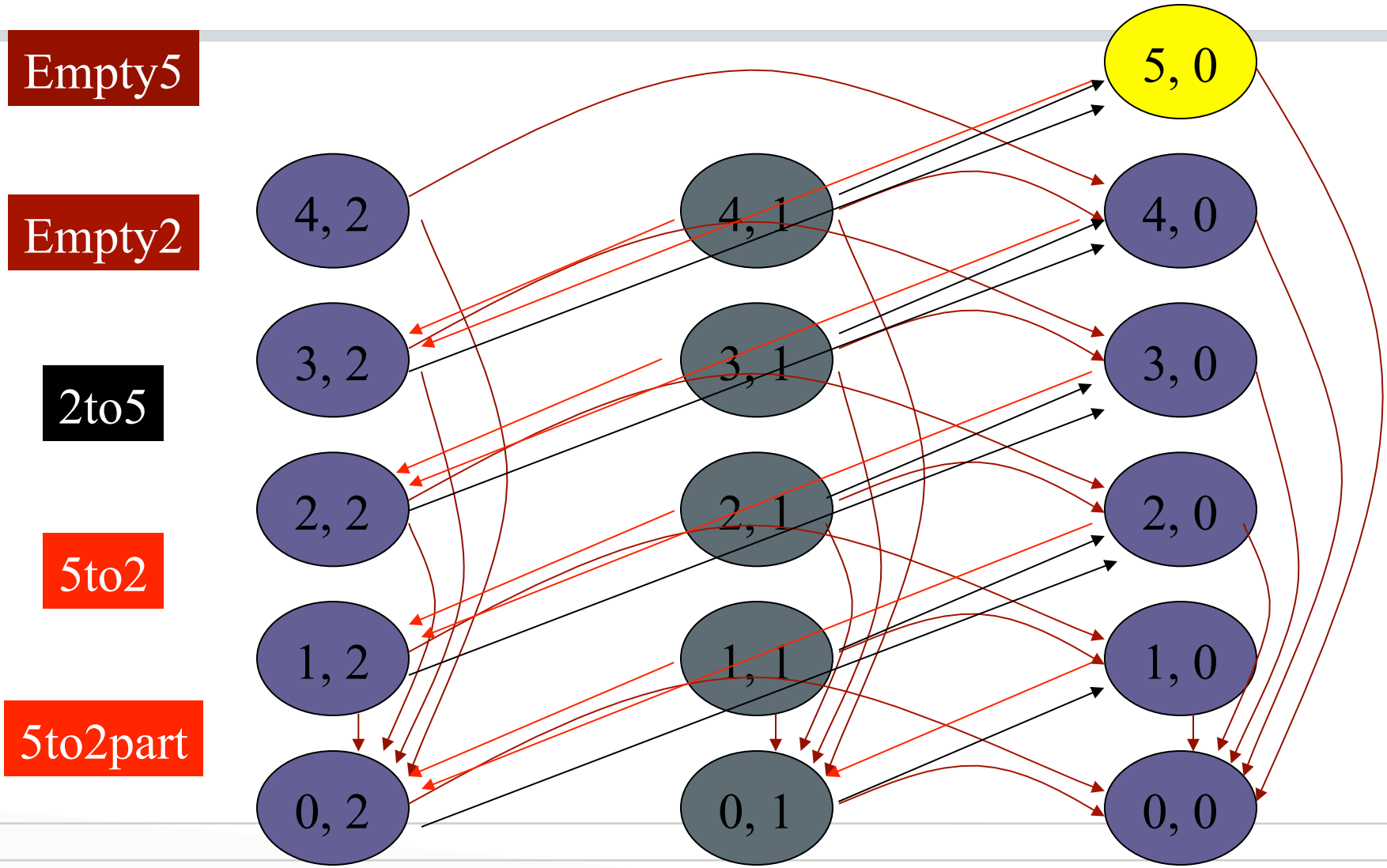
Given a full 5-gallon jug and an empty 2-gallon jug, the goal is to fill the 2-gallon jug with exactly one gallon of water.

- **State** = (x,y), where x is the number of gallons of water in the 5-gallon jug and y is # of gallons in the 2-gallon jug
- **Initial State** = (5,0)
- **Goal State** = (*,1), where * means any amount

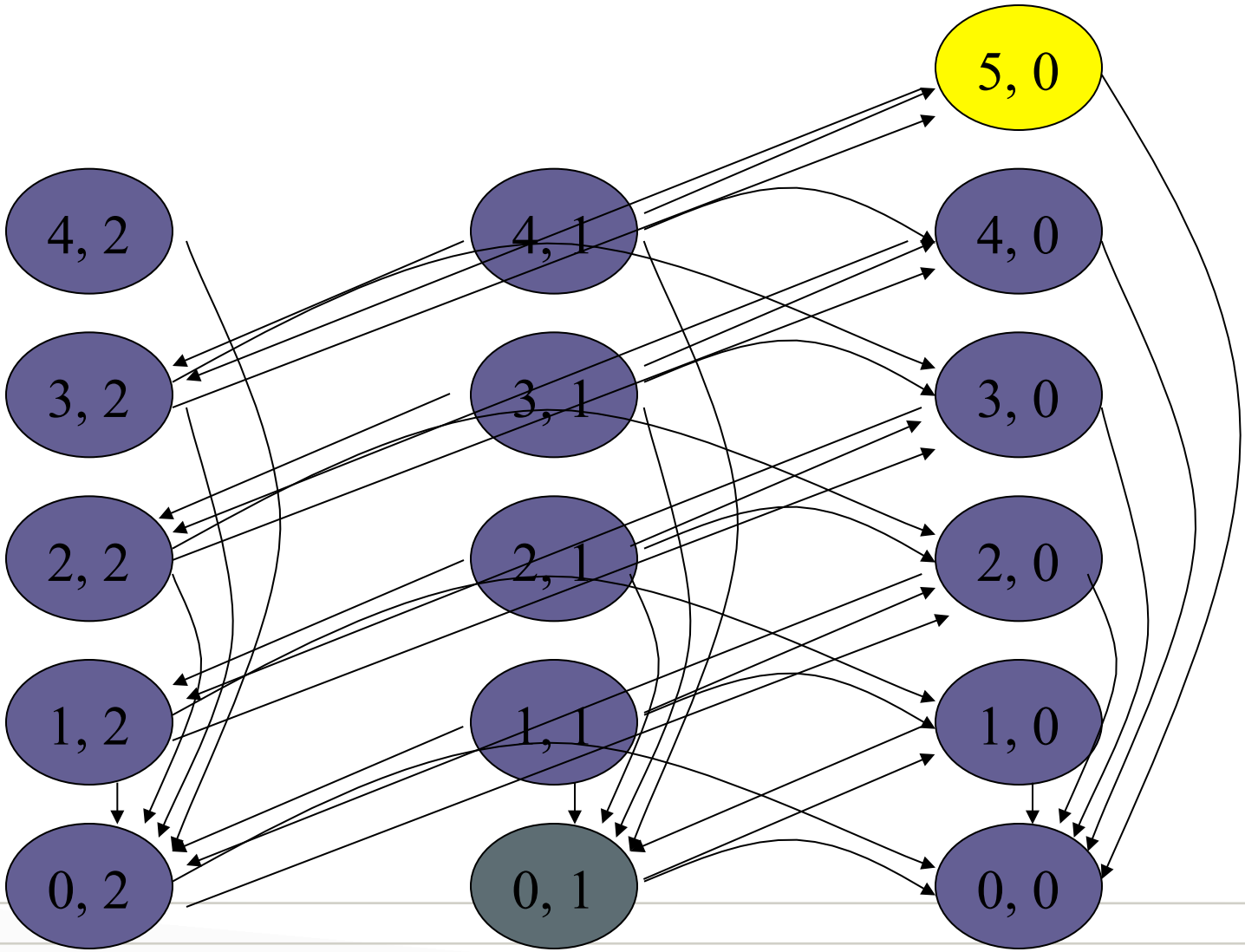
Operator table

Name	Cond.	Transition	Effect
Empty5	–	(x,y)→(0,y)	Empty 5-gal. jug
Empty2	–	(x,y)→(x,0)	Empty 2-gal. jug
2to5	$x \leq 3$	(x,2)→(x+2,0)	Pour 2-gal. into 5-gal.
5to2	$x \geq 2$	(x, 0)→(x-2,2)	Pour 5-gal. into 2-gal.
5to2part	$y < 2$	(1,y)→(0,y+1)	Pour partial 5-gal. into 2-gal.

Water jug state space



Water jug solution

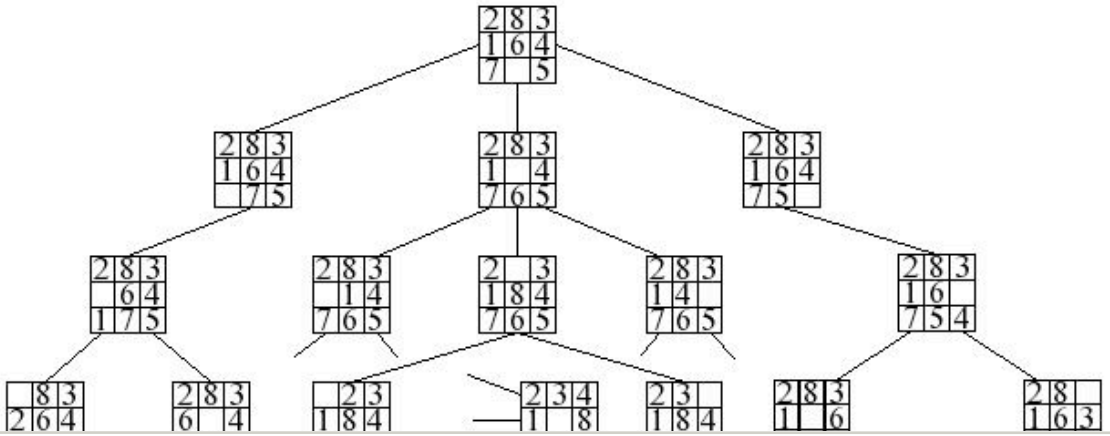


Formalizing Search III

- **State-space search** is the process of searching through a state space for a solution by **making explicit** a sufficient portion of an **implicit** state-space graph to find a goal node
  - Initially V={S}, where S is the start node
  - When S is expanded, its successors are generated; those nodes are added to V and the arcs are added to E
  - This process continues until a goal node is found
- It isn't usually practical to represent entire space

Formalizing search IV

- Each node implicitly or explicitly represents a partial **solution path** (and cost of the partial solution path) from the start node to the given node.
  - In general, from a node there are many possible paths (and therefore solutions) that have this partial path as a prefix



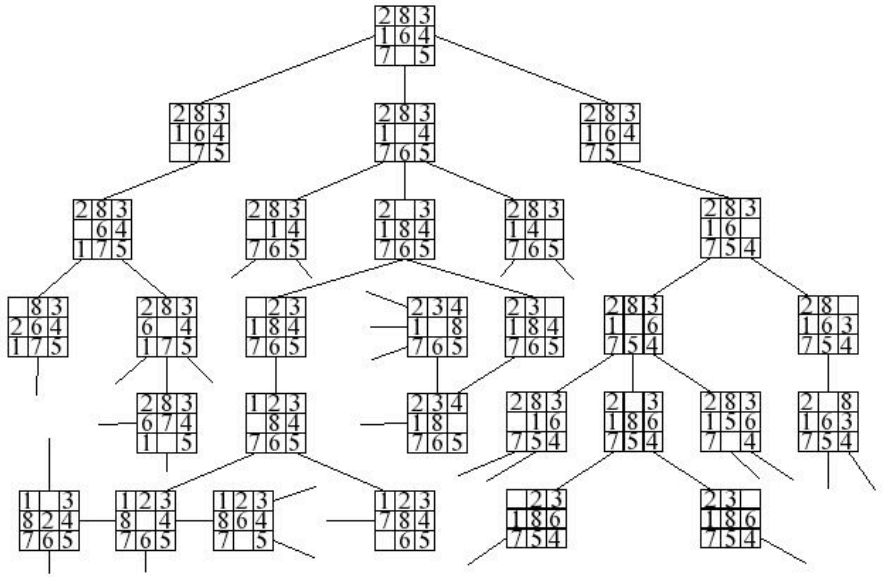
State-Space Search Algorithm

```
function general-search (problem, QUEUEING-FUNCTION)
;; problem describes start state, operators, goal test,
;;    and operator costs
;; queueing-function is a comparator function that
;;    ranks two states
;; general-search returns either a goal node or failure

nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
     if EMPTY(nodes) then return "failure"
     node = REMOVE-FRONT(nodes)
     if problem.GOAL-TEST(node.STATE) succeeds
        then return node
     nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
             problem.OPERATORS))
 end
     ;; Note: The goal test is NOT done when nodes are generated
     ;; Note: This algorithm does not detect loops
```

Key procedures to be defined

- `EXPAND`
  - Generate all successor nodes of a given node
- `GOAL-TEST`
  - Test if state satisfies goal conditions
- `QUEUEING-FUNCTION`
  - Used to maintain a ranked list of nodes that are candidates for expansion



Algorithm Bookkeeping

- Typical node data structure includes:
  - State at this node
  - Parent node (no loops!)
  - Operator applied to get to this node
  - Depth of this node (number of operator applications since initial state)
  - Cost of the path (sum of each operator application so far)

Some Issues

- Search process constructs a search tree, where:
  - **Root** is the initial state and
  - **Leaf nodes** are nodes that are either:
    - Not yet expanded (i.e., they are in the list "nodes") or
    - Have no successors (i.e., they're "dead ends", because no operators can be applied, but they are not goals)
- Search tree may be infinite
  - Even for small search space
  - How?

Some Issues

- Return a path or a node depending on problem
  - In 8-queens return a node; in 8-puzzle return a path
  - What about Missionaries & Cannibals?
- Changing definition of `Queueing-Function` ⇒ different search strategies
  - How do you choose what to expand next?

Evaluating Search Strategies

- Completeness:
  - Guarantees finding a solution if one exists
- Time complexity:
  - How long (worst or average case) does it take to find a solution?
  - Usually measured in number of states visited/nodes expanded
- Space complexity:
  - How much space is used by the algorithm?
  - Usually measured in maximum size of the "nodes" list during search
- Optimality/Admissibility
  - If a solution is found, is it guaranteed to be optimal (the solution with minimum cost)?