

CSC 422

State space search

Learning content

- 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 the difference between **expanding** a state and **generating** a state?
- What is the **frontier** (a.k.a. **open list**)?

Representing actions

- The number of actions / operators depends on the **representation** used in describing a state.
 - In the 8-puzzle, we could specify 4 possible moves for each of the 8 tiles, resulting in a total of **$4*8=32$ operators**.
 - On the other hand, we could specify four moves for the “blank” square and we would only need **4 operators**.
- Representational shift can greatly simplify a problem!

State Space Search - Example

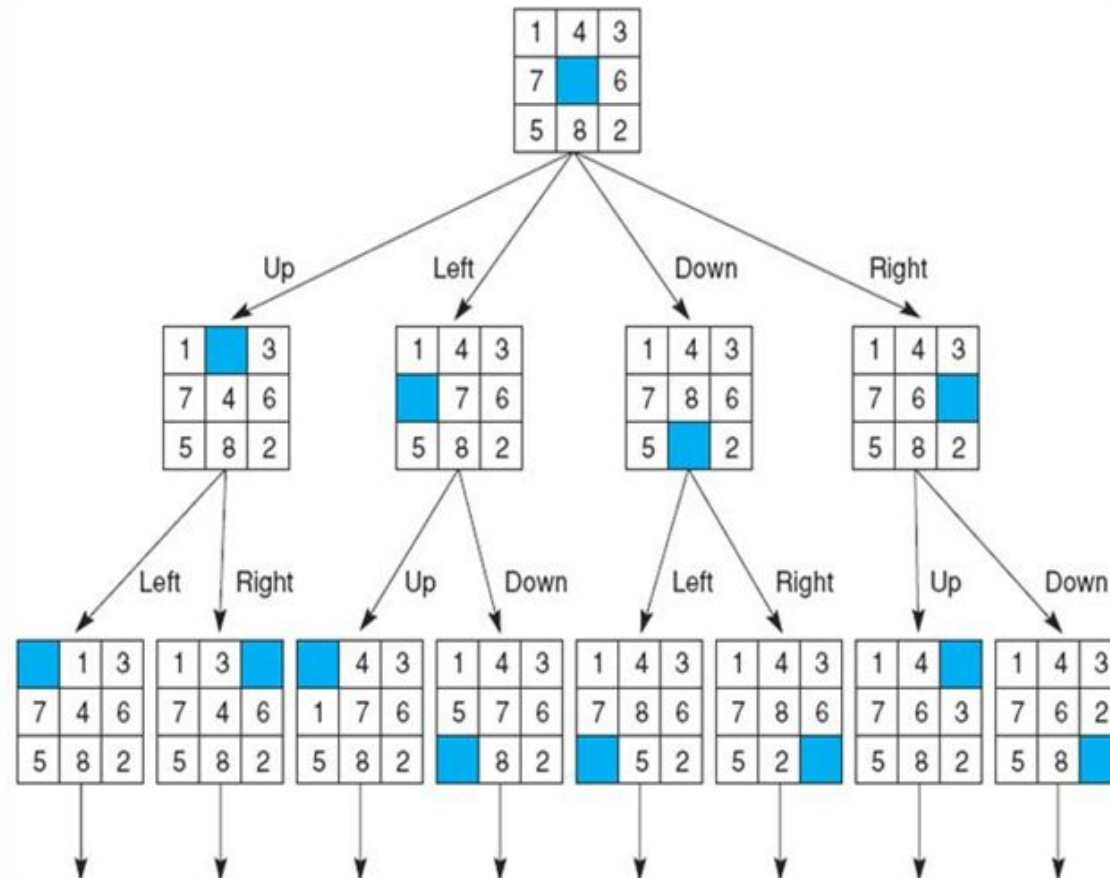
- generated by “move blank” operations

- \uparrow -- up

- \leftarrow -- left

- \downarrow -- down

- \rightarrow -- Right



Representing states

- What knowledge needs to be represented in a state description to adequately describe the current state or situation 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.**

Formalizing Search in a State Space

- A state space is a **directed graph**, (V, E) where V is a set of **nodes** and E is a set of **arcs**, where each arc is directed from a node to another node
- **node**: a **state**
 - state description
 - plus optionally other information related to the parent of the node, operation used to generate the node from that parent, and other bookkeeping data
- **arc**: an instance of an (applicable) action/operation.
 - the source and destination nodes are called as **parent** (**immediate predecessor**) and **child** (**immediate successor**) nodes with respect to each other
 - **ancestors** (predecessors) and **descendents** (successors)
 - each arc has a fixed, non-negative **cost** associated with it, corresponding to the cost of the action

Remember, state space is not a solution space

Formalizing Search in a State Space

- **State-space search** is the process of searching through a state space for a solution
- This is done by making explicit a sufficient portion of an **implicit** state-space graph to include a goal node.
 - Initially $V=\{S\}$, where S is the start node; when S is expanded, its successors are generated and those nodes are added to V and the associated arcs are added to E .
 - This process continues until a goal node is **generated** (included in V) and **identified** (by goal test)
- During search, a node can be in **one of the three categories**:
 - **Not generated yet** (has not been made explicit yet)
 - **OPEN**: generated but not expanded
 - **CLOSED**: expanded
- Search strategies differ mainly on **how to select an OPEN node** for expansion at each step of search

State Space Search Algorithm

STATE SPACE SEARCH

A *state space* is represented by a four-tuple [**N**,**A**,**S**,**GD**], where:

N is the set of nodes or states of the graph. These correspond to the states in a problem-solving process.

A is the set of arcs (or links) between nodes. These correspond to the steps in a problem-solving process.

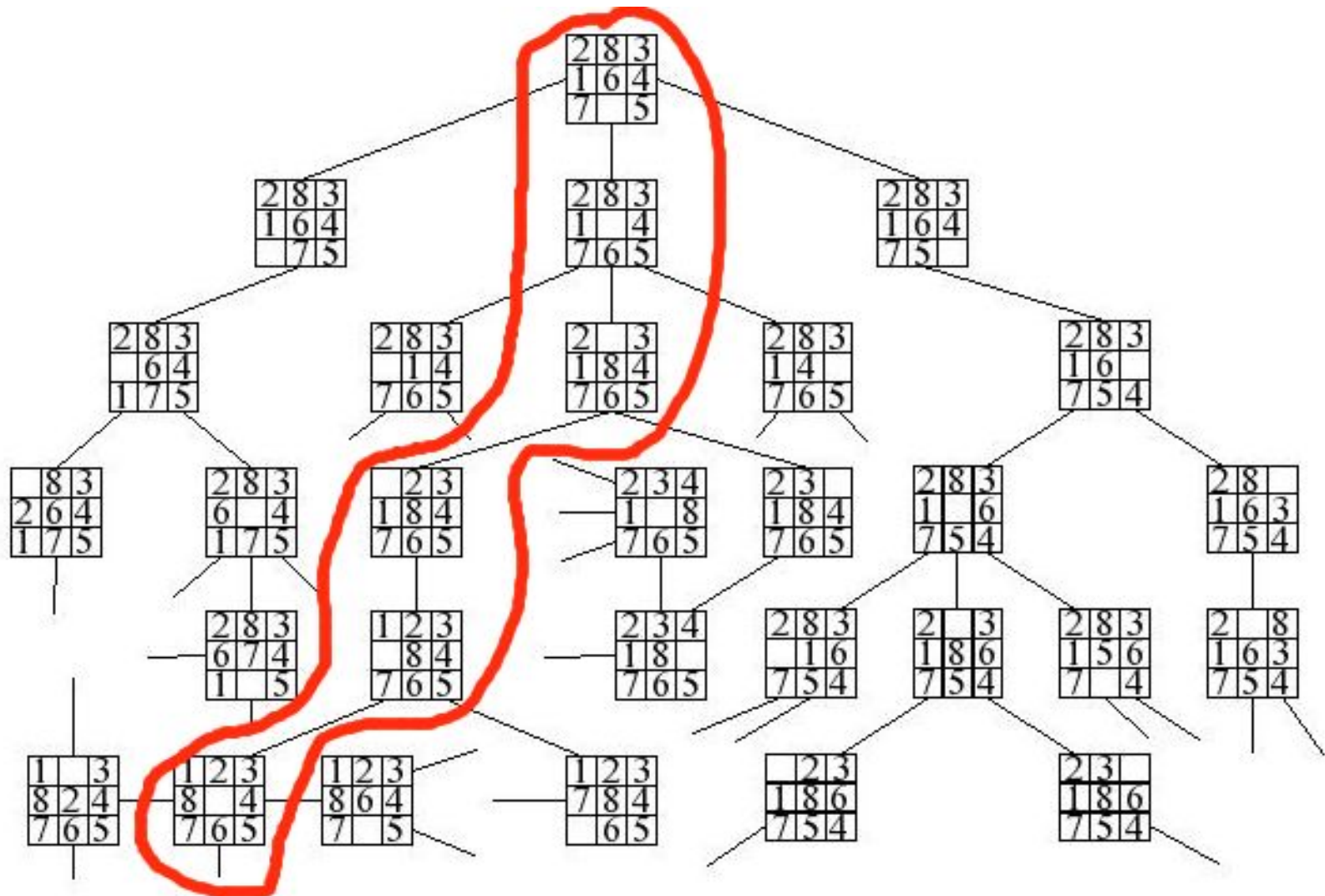
S, a nonempty subset of **N**, contains the start state(s) of the problem.

GD, a nonempty subset of **N**, contains the goal state(s) of the problem. The states in **GD** are described using either:

1. A measurable property of the states encountered in the search.
2. A property of the path developed in the search, for example, the transition costs for the arcs of the path.

A *solution path* is a path through this graph from a node in **S** to a node in **GD**.

Example of State Space Search Algorithm



Key procedures to be defined

- **EXPAND** -Generate all successor nodes (leaf nodes)
- **GOAL-TEST** - Test if state satisfies all goal conditions
- **QUEUEING-FUNCTION**
 - Used to maintain a ranked list of nodes that are candidates for expansion
- **Typical node data structure includes:**
 - State at this node
 - Parent node (root)
 - 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
 - not yet expanded (i.e., they are in the list “nodes”) or
 - having no successors (i.e., they’re “dead ends” because no operators were applicable and yet they are not goals)
- Search tree may be infinite because of loops even if state space is small
- Return a path or a node depending on problem.
 - 8-puzzle returns a path
- Changing definition of the QUEUEING-FUNCTION leads to different search strategies

Evaluating search strategies

- A **solution** is a sequence of operators that is associated with a path in a state space from a start node to a goal node.
- The **cost of a solution** is the sum of the arc costs on the solution path.
 - If all arcs have the same (unit) cost, then the solution cost is just the length of the solution (number of steps / state transitions)

Evaluating search strategies

- **Completeness**

- Guarantees finding a solution whenever one exists

- **Time complexity**

- How long (worst or average case) does it take to find a solution?
Usually measured in terms of the number of nodes expanded

- **Space complexity**

- How much space is used by the algorithm? Usually measured in terms of the maximum size of the “nodes” list during the search

- **Optimality/Admissibility**

- If a solution is found, is it guaranteed to be an optimal one? That is, is it the one with minimum cost?

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	—	$(x,y) \rightarrow (0,y)$	Empty 5-gal. jug
Empty2	—	$(x,y) \rightarrow (x,0)$	Empty 2-gal. jug
2to5	$x \leq 3$	$(x,2) \rightarrow (x+2,0)$	Pour 2-gal. into 5-gal.
5to2	$x \geq 2$	$(x,0) \rightarrow (x-2,2)$	Pour 5-gal. into 2-gal.
5to2part	$y < 2$	$(1,y) \rightarrow (0,y+1)$	Pour partial 5-gal. into 2-gal.

Water Jug Problem (cont.)

- To solve this we have to make some assumptions not mentioned in the problem. They are
- 1. We can fill a jug from the pump.
- 2. we can pour water out of a jug to the ground.
- 3. We can pour water from one jug to another.
- 4. There is no measuring device available.

This is one of the solution

0-3

3-0

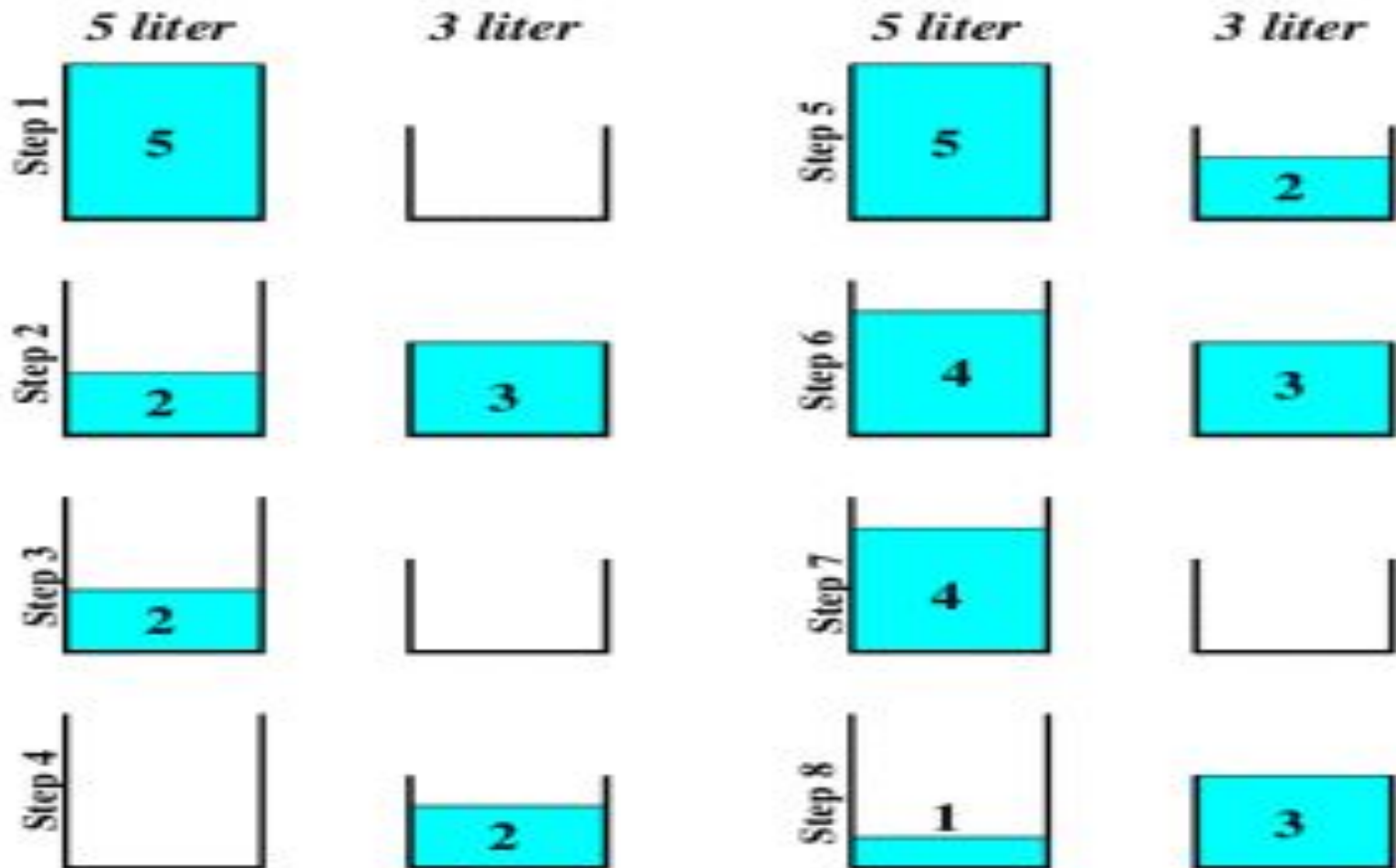
3-3

4-2

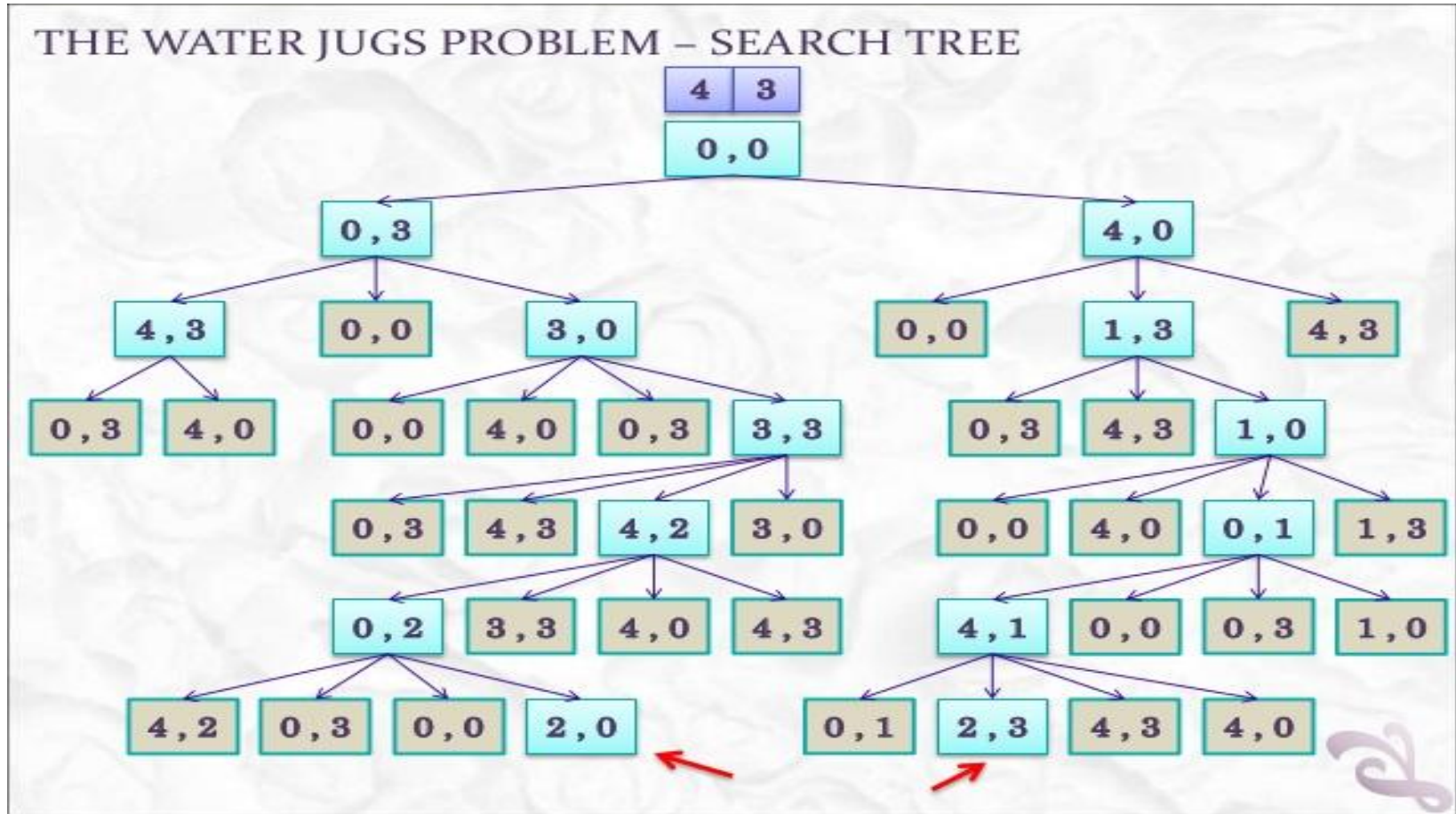
0-2

2-0

Water Jug Problem (cont.)



Water Jug Problem - Example



State Rules/Condition

1.	$(X, Y) \text{ if } X < 4 \rightarrow (4, Y)$	Fill the 4-gallon jug
2.	$(X, Y) \text{ if } Y < 3 \rightarrow (X, 3)$	Fill the 3-gallon jug
3.	$(X, Y) \text{ if } X = d \ \& \ d > 0 \rightarrow (X-d, Y)$	Pour some water out of the 4-gallon jug
4.	$(X, Y) \text{ if } Y = d \ \& \ d > 0 \rightarrow (X, Y-d)$	Pour some water out of 3-gallon jug
5.	$(X, Y) \text{ if } X > 0 \rightarrow (0, Y)$	Empty the 4-gallon jug on the ground
6.	$(X, Y) \text{ if } Y > 0 \rightarrow (X, 0)$	Empty the 3-gallon jug on the ground
7.	$(X, Y) \text{ if } X + Y \leq 4 \text{ and } Y > 0 \rightarrow 4, (Y - (4 - X))$	Pour water from the 3-gallon jug into the 4-gallon jug until the gallon jug is full.
8.	$(X, Y) \text{ if } X + Y \geq 3 \text{ and } X > 0 \rightarrow (X - (3 - Y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.
9.	$(X, Y) \text{ if } X + Y \leq 4 \text{ and } Y > 0 \rightarrow (X + Y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10.	$(X, Y) \text{ if } X + Y \leq 3 \text{ and } X > 0 \rightarrow (0, X + Y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11.	$(0, 2) \rightarrow (2, 0)$	Pour the 2-gallons water from 3-gallon jug into the 4;gallon jug
12.	$(2, Y) \rightarrow (0, Y)$	Empty the 2-gallons in the 4-gallon jug on the ground.

Fig. 2.3. Production rules (operators) for the water jug problem.

Water Jug Problem - Example

Water in 4-gallon jug (X)	Water in 3-gallon jug (Y)	Rule applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

Fig. 2.4 (a). A solution to water jug problem.

X	Y	Rule applied (Control strategy)
0	0	
4	0	1-
1	3	8
1	0	6
0	1	10
4	1	1
2	3	8

Fig. 2.4 (b). 2nd solution to water jug problem.

CLASS EXERCISE

- Representing a Sudoku puzzle as a search space
 - What are the states?
 - What are the operators?
 - What are the constraints (on operator application)?
 - What is the description of the goal state?

- Let's try it!

	3		
			1
3			
		2	