NTNU

TDT4136 - Introduction to AI

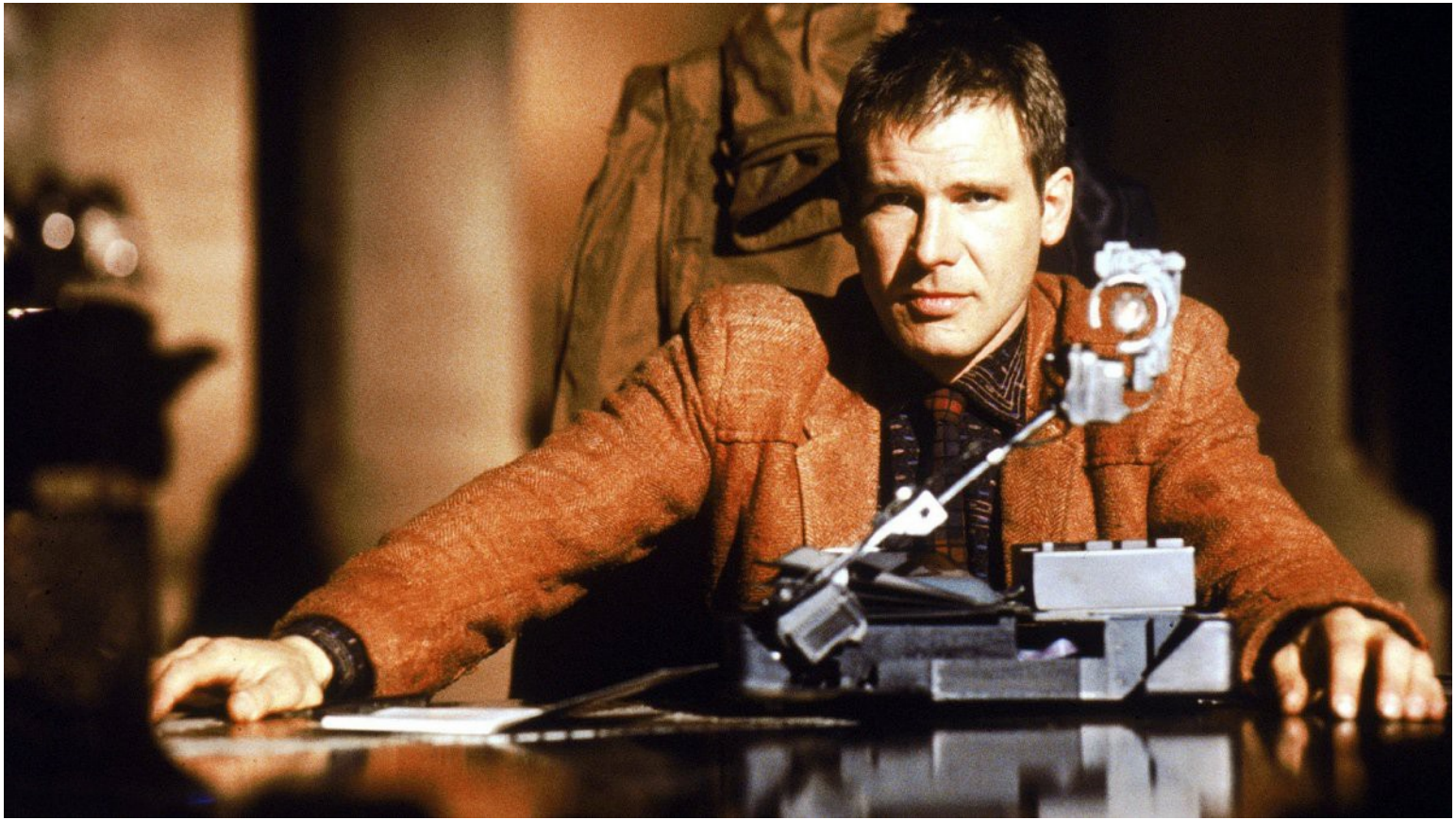
Odd Erik Gundersen

Adjunct Associate Professor

Chief AI Officer, TrønderEnergi

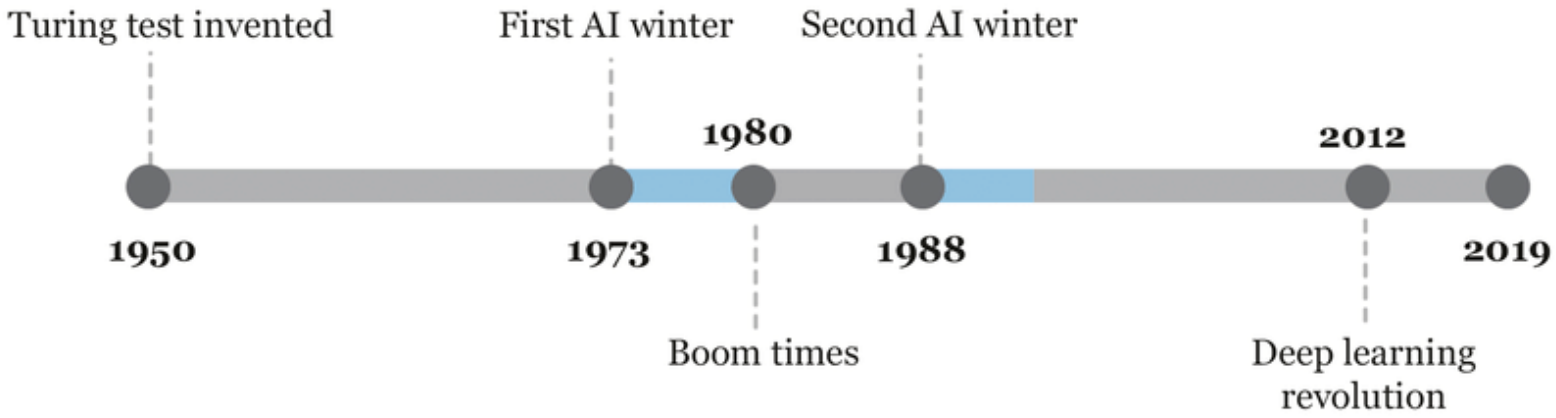
What is AI?

Turing Test

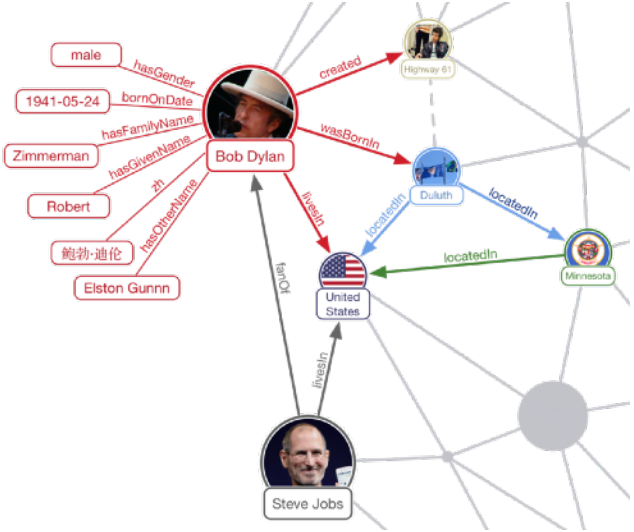


Thinking rationally

The AI seasons



Representing Knowledge?



VS

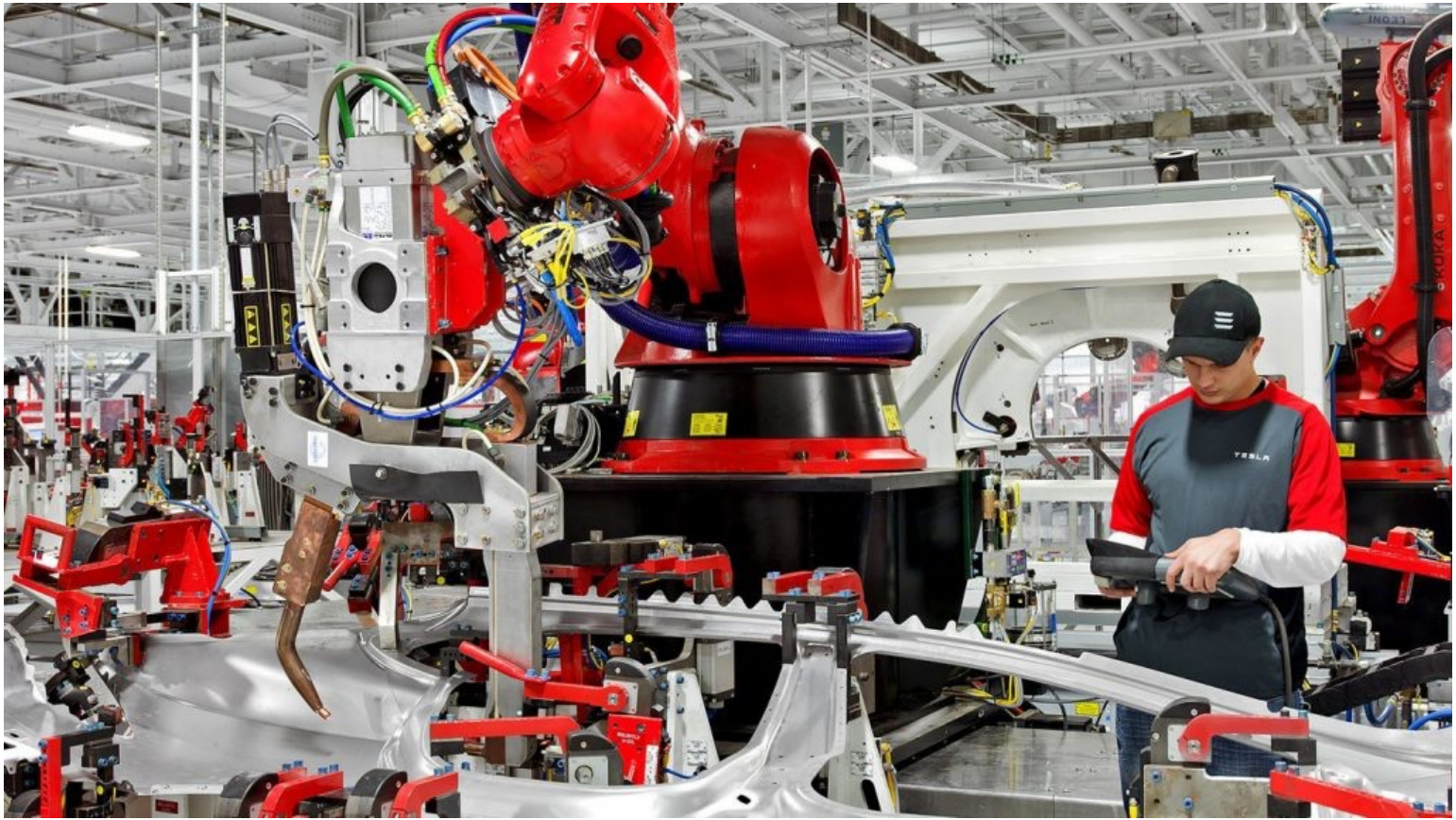
Big data


mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
elif _operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
#selection at the end -add back the deselected mirror
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob is the active ob
#mirror_ob.select = 0

We do not need representations!?

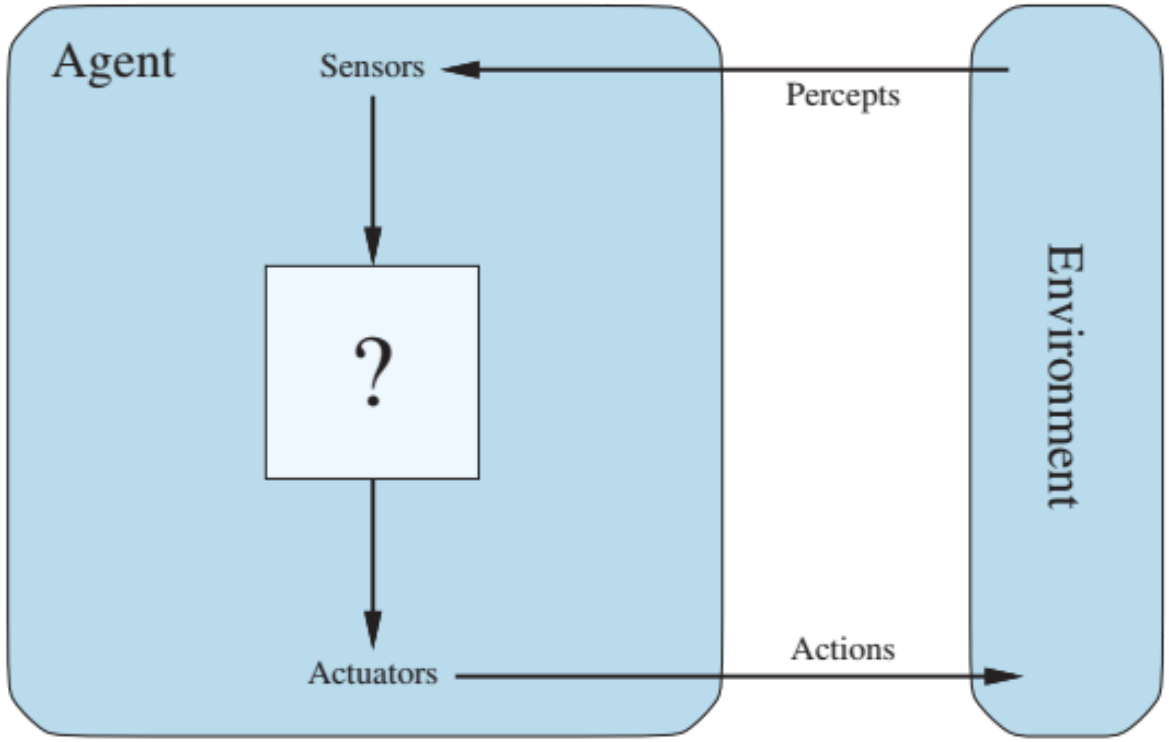
We do not need representations!?

Interacting with AI



Lecture 2 - Intelligent Agents

Gleb Sizov
Norwegian University of Science and Technology



Agent

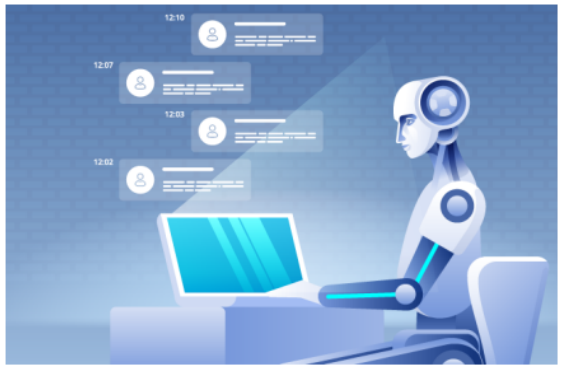
Human agent

hardware agent

software agent

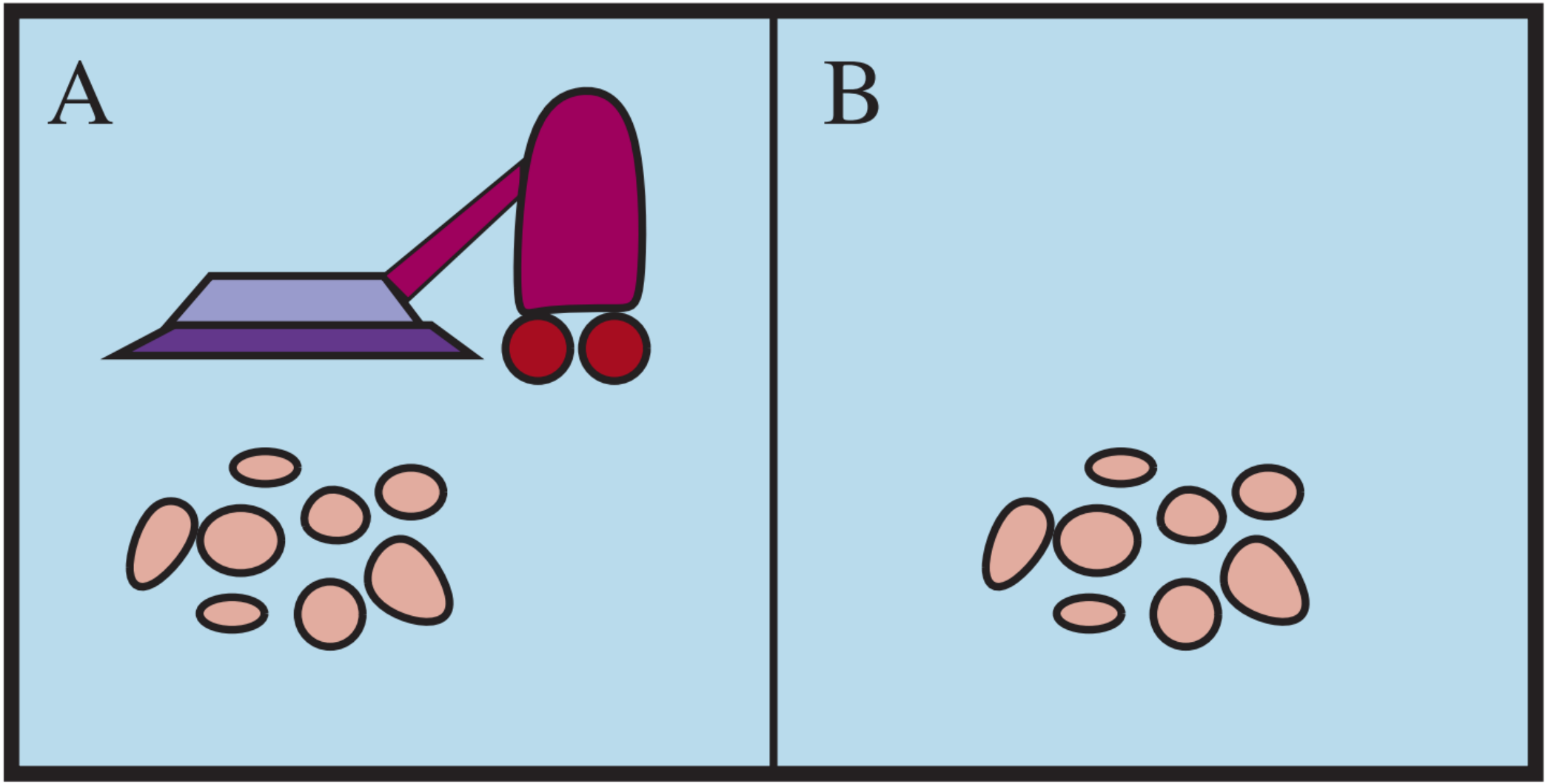
(a human driver)

(a robot driver)



(a chatbot. www.callcentrehelper.com)

Example: Vacuum-cleaner world



Rational agent maximizes its expected performance

Depends on:

- Performance measure
- The agent’s prior knowledge of the environment.
- The actions that the agent can perform.
- The agent’s percept sequence to date.

Rationality vs omniscient vs perfect

Rational $\neq$ *omniscient* agent knows the actual outcome of its actions

Rational $\neq$ *perfect* agent maximizes actual performace

Rationality requires information gathering and exploration

Doing actions in order to modify future percepts.

Rationality requires learning and autonomy

Learning agent modifies knowledge of the environment from experience.

Autonomous agents learns what it can to compensate for partial or incorrect prior knowledge.

Task environment

PEAS (**P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors)

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users	Roads, other traffic, police, pedestrians, customers, weather	Steering, accelerator, brake, signal, horn, display, speech	Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen

Properties of the environment - Smart building

1. Fully vs. partially observable?
2. Single agent vs. multiagent?
3. Deterministic vs. nondeterministic?
4. Episodic vs. sequential?
5. Static vs. dynamic?
6. Discrete vs. continuous?
7. Known vs unknown?

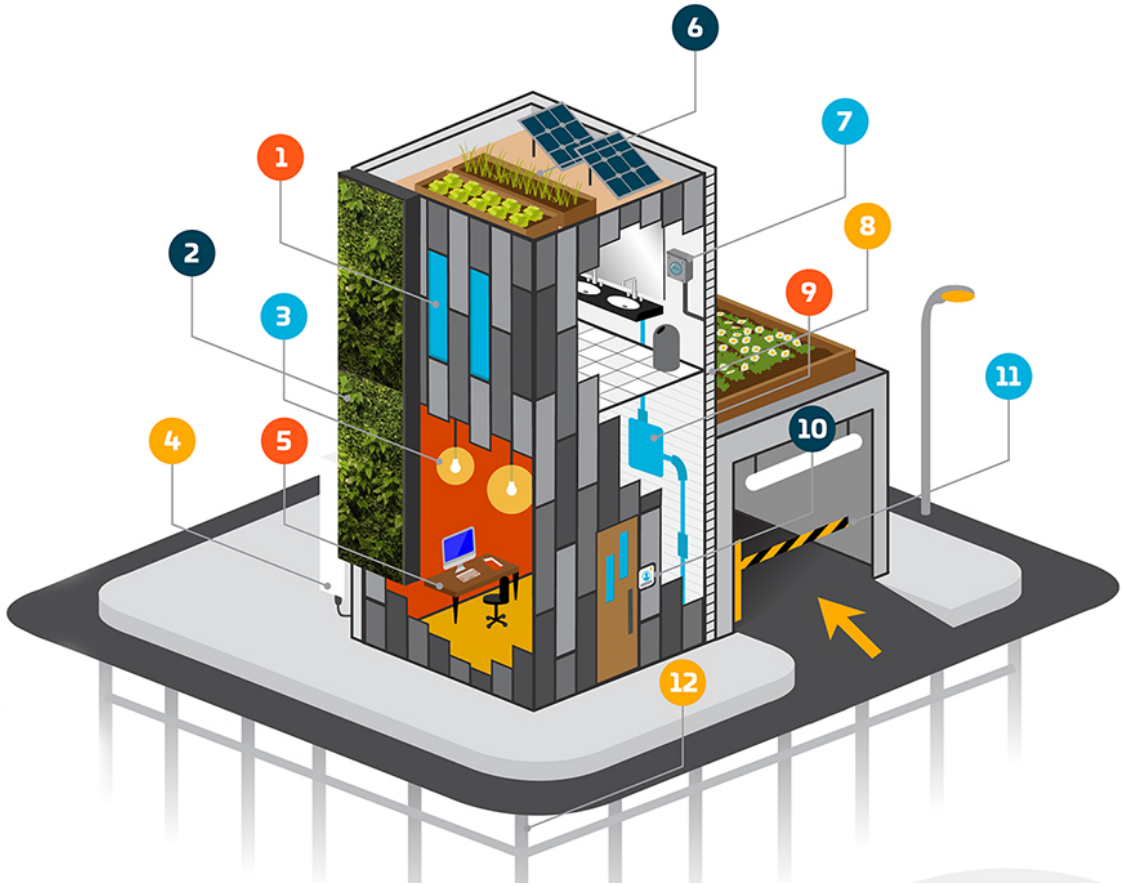


Table-driven agent

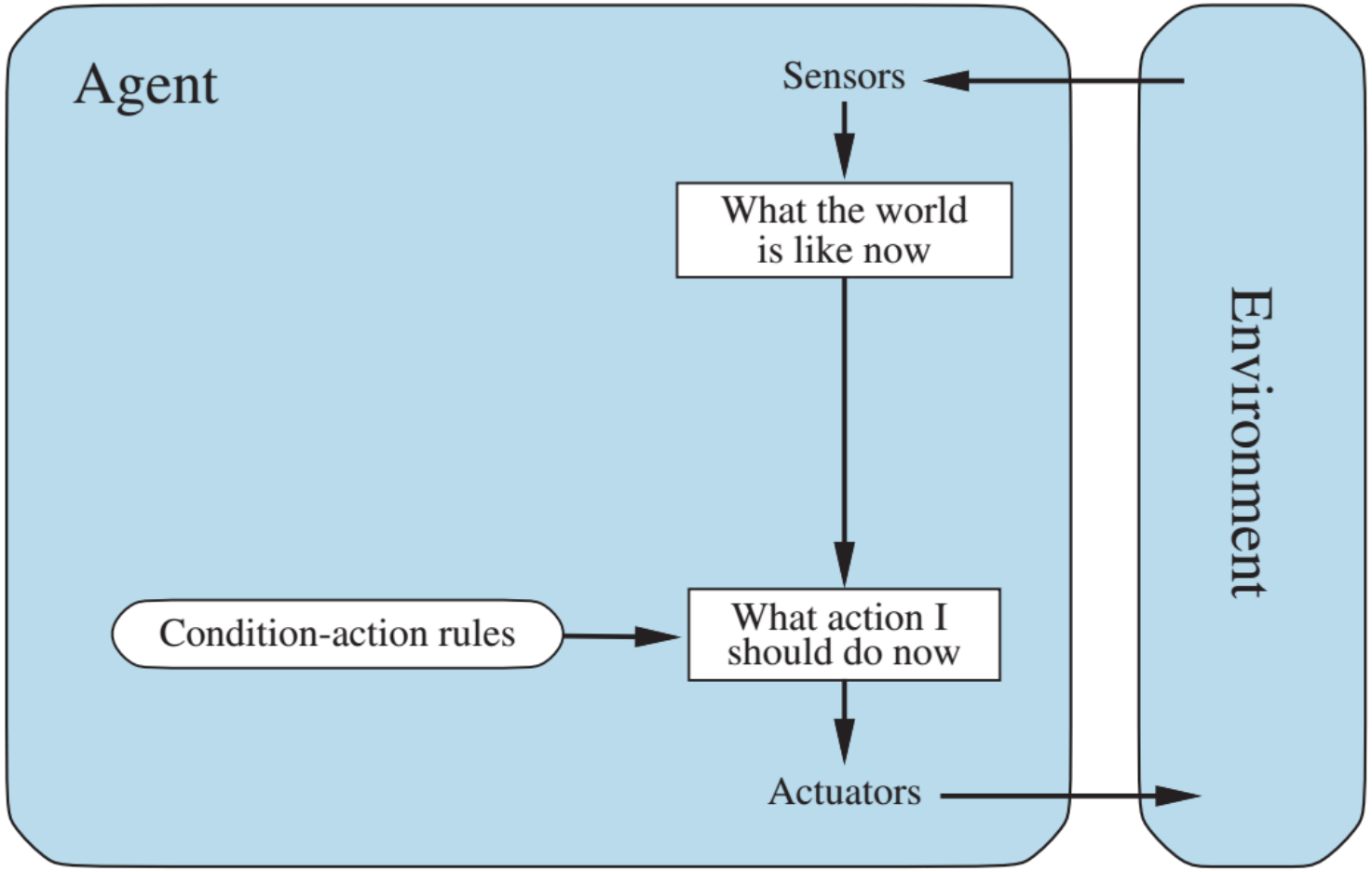
Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Clean], [A, Dirty]$	$Suck$
⋮	⋮

Simple reflex agent

function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action

if *status* = *Dirty* **then return** *Suck*
else if *location* = *A* **then return** *Right*
else if *location* = *B* **then return** *Left*

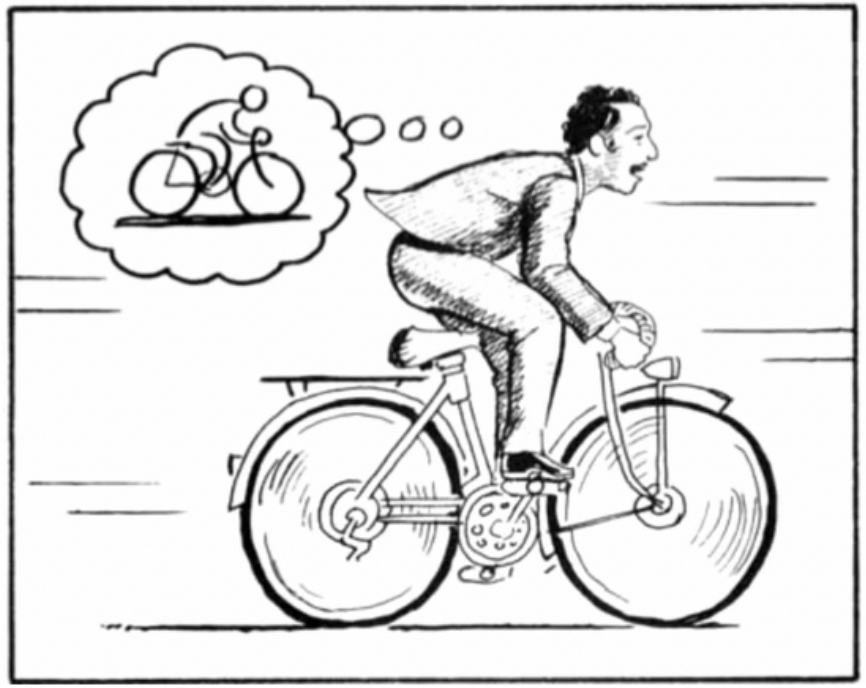
Simple reflex agent



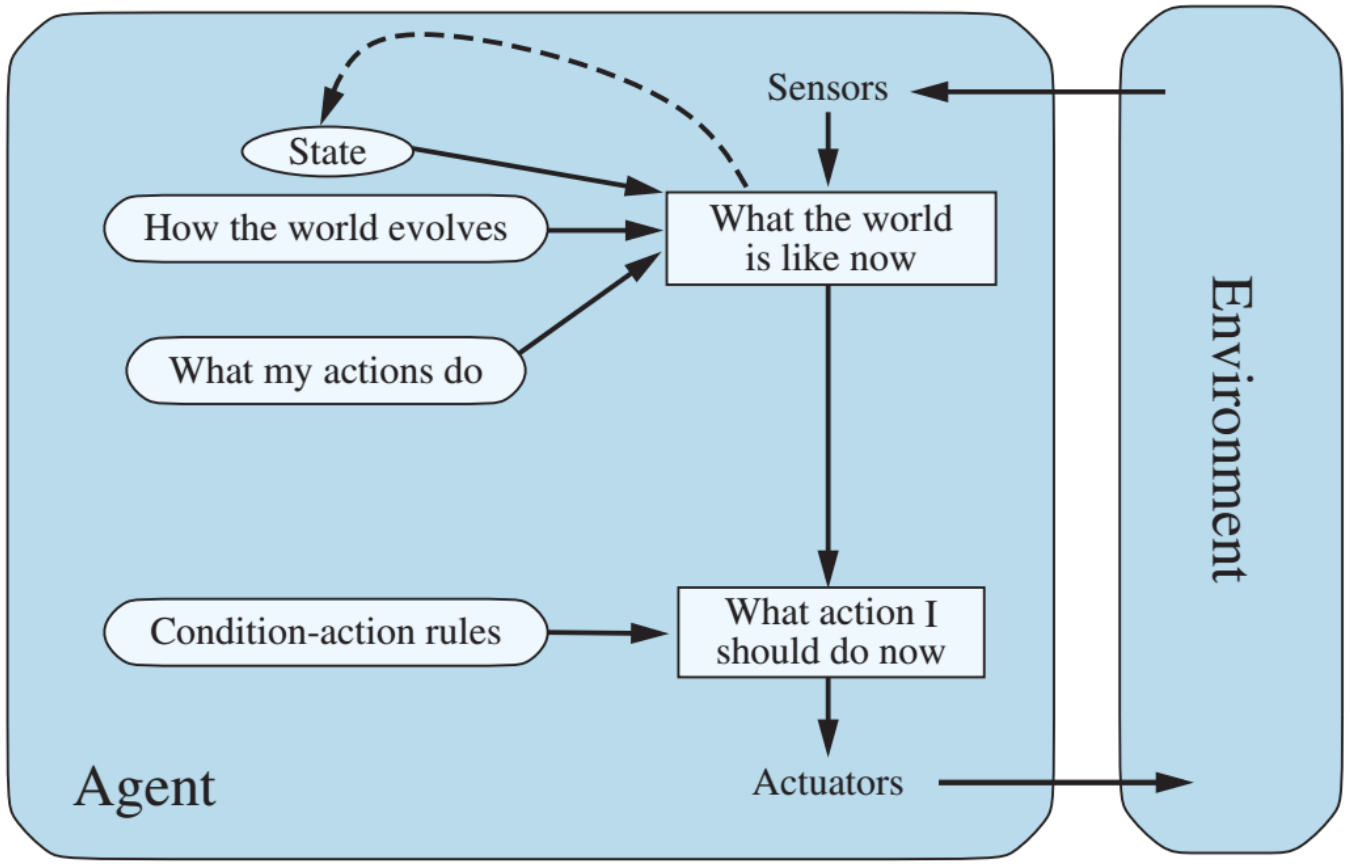
Will it handle partially observable environment?

Model-based reflex agent

- Maintains internal state
- Has a model of how the world works: transition and sensor models
- Handles partially observable environments



Model-based reflex agent

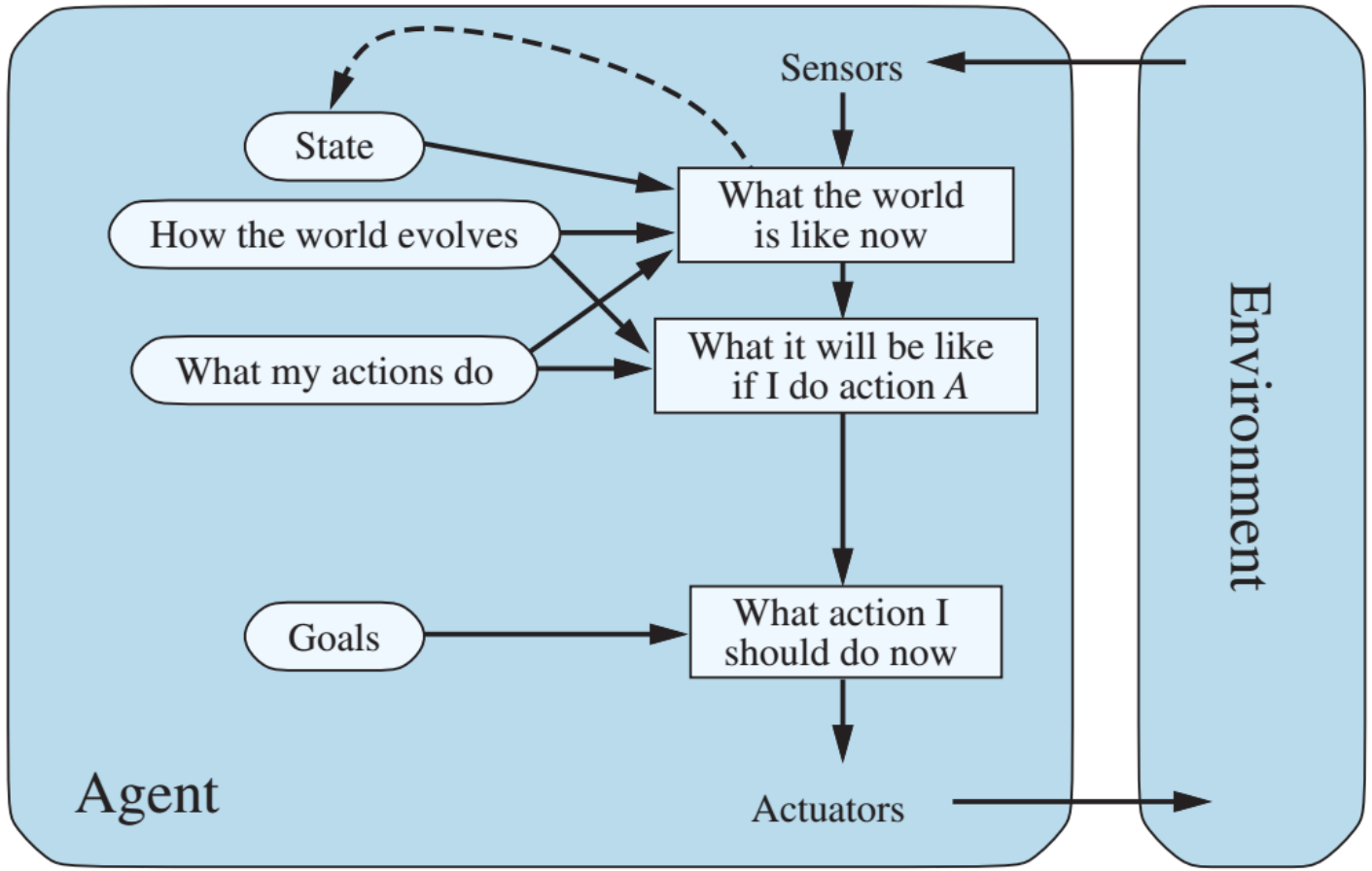


Goal-based agent

- Goal identifies desirable states.
- What will happen if I do *X*? Will *X* make me happy?
- More flexible, general than reflex agent
- Handles sequential environments, e.g. taxi driver, chess

Goal-based agent

Chooses an action that will (eventually) lead to the achievement of its goals

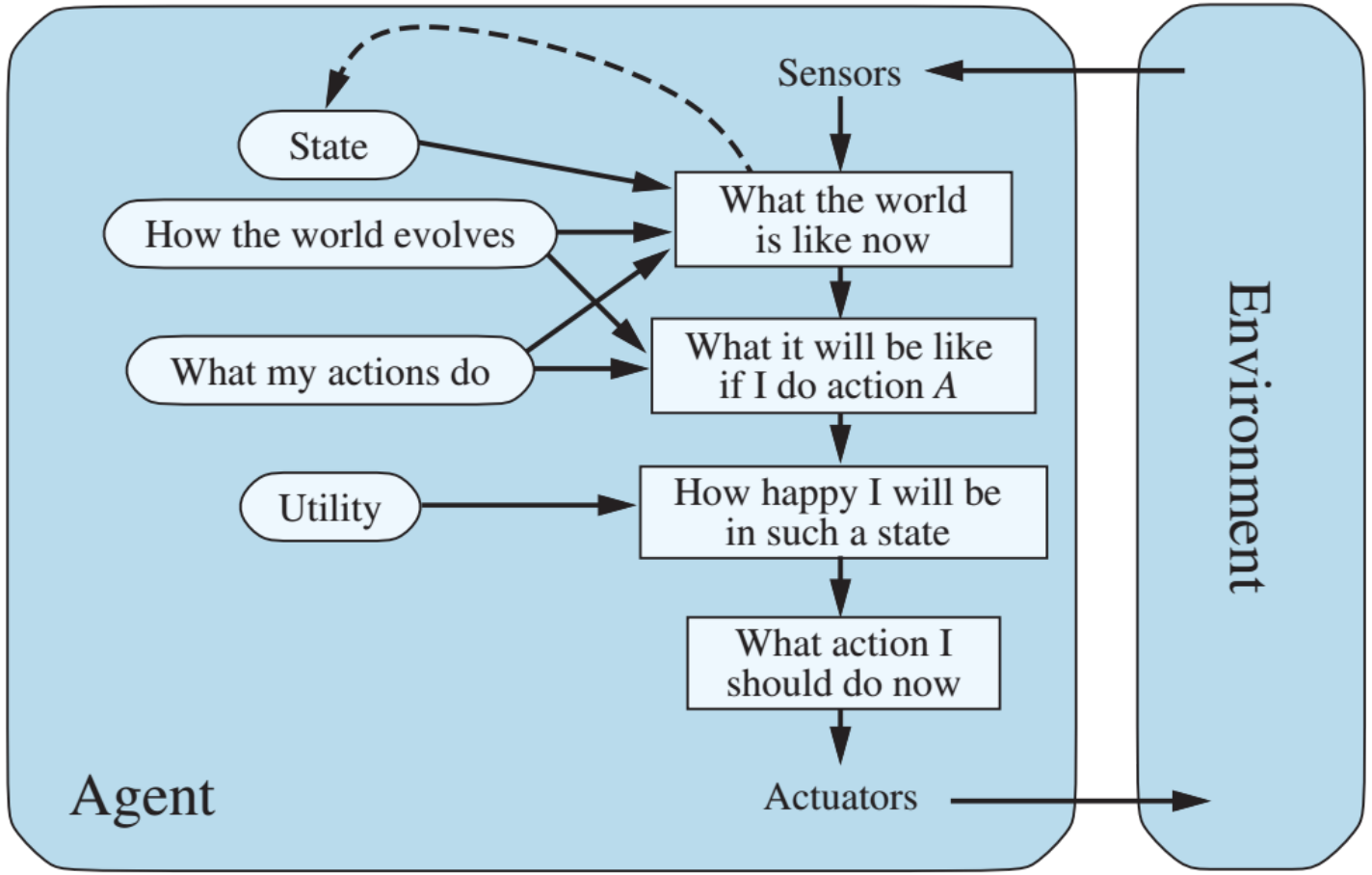


Utility function

- Utility provides hapiness level for (sequence of) states.
- Same goal may be achieved in more than one way, with different utility values.
- **Utility function** - internalization of the performance measure
- **Expected utility** - average utility weighted by state probabilities.
- Handles nondeterministic and partially observable environments, e.g. poker

Utility-based agent

Chooses an action to maximize the expected utility

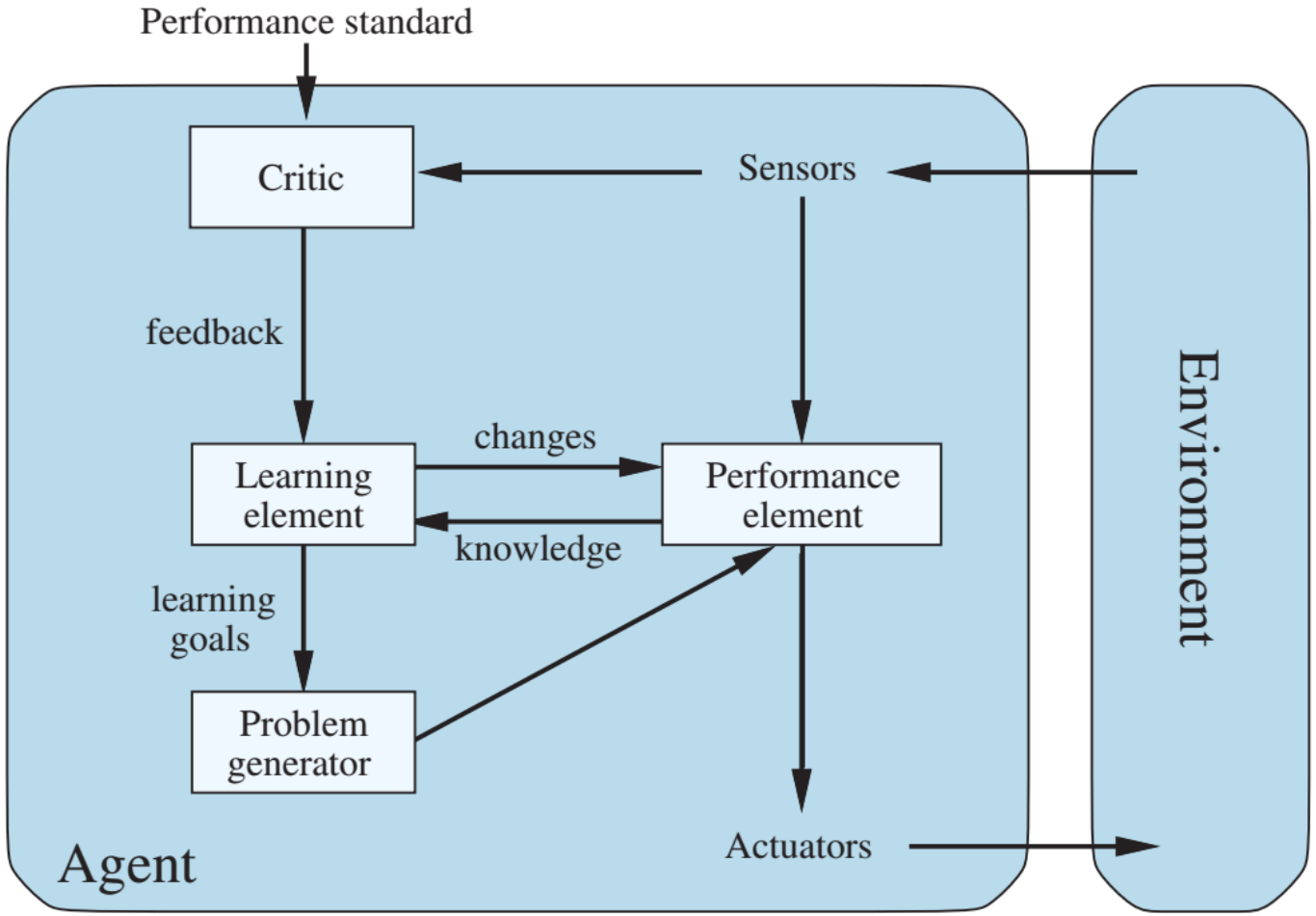


Learning agent

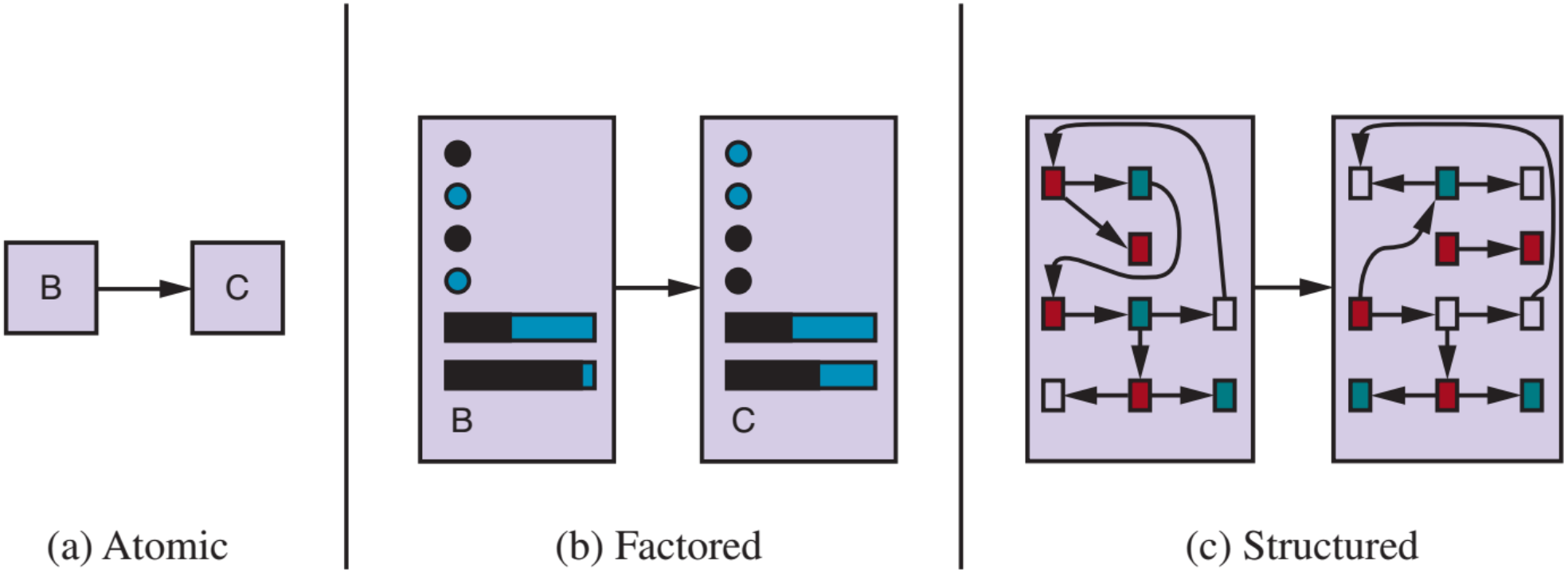
- Becomes more competent than its initial knowledge allows.
- Modifies its components to bring them into closer agreement with the feedback.
- Does some suboptimal actions to explore.
- Handles well unknown and dynamic environemnts.

Learning agent

Chooses the best action given what it knows and/or leads to new and informative experiences.

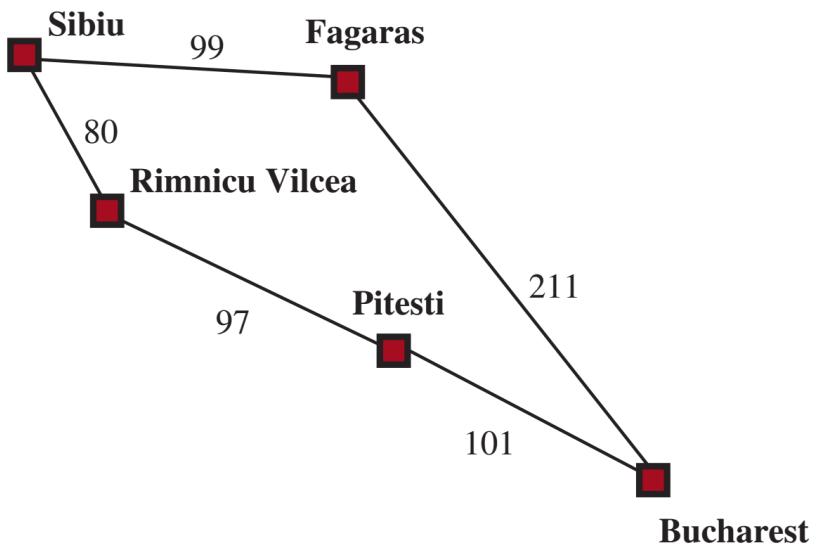


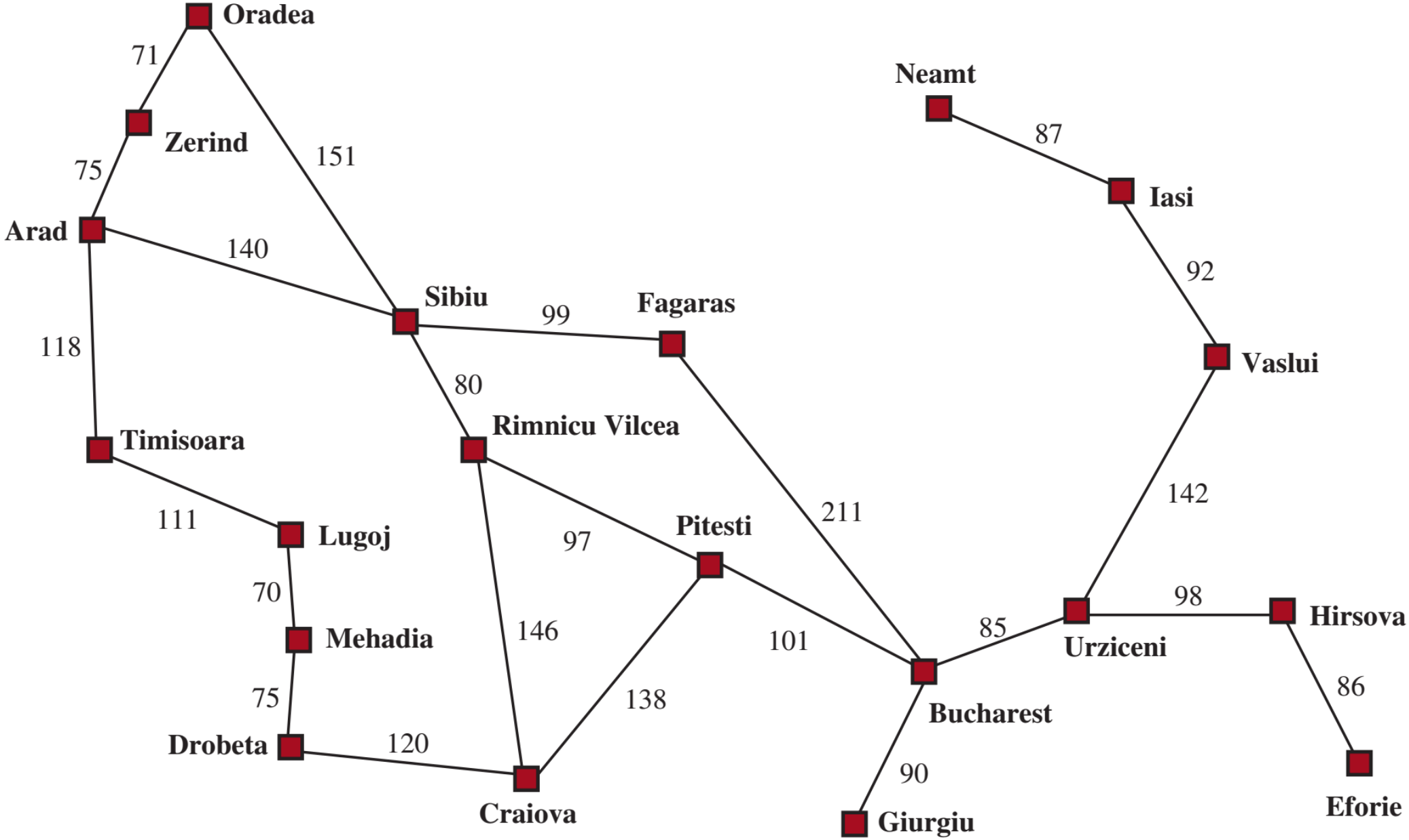
Representations of states and transitions



Lecture 3
Solving Problems by Searching

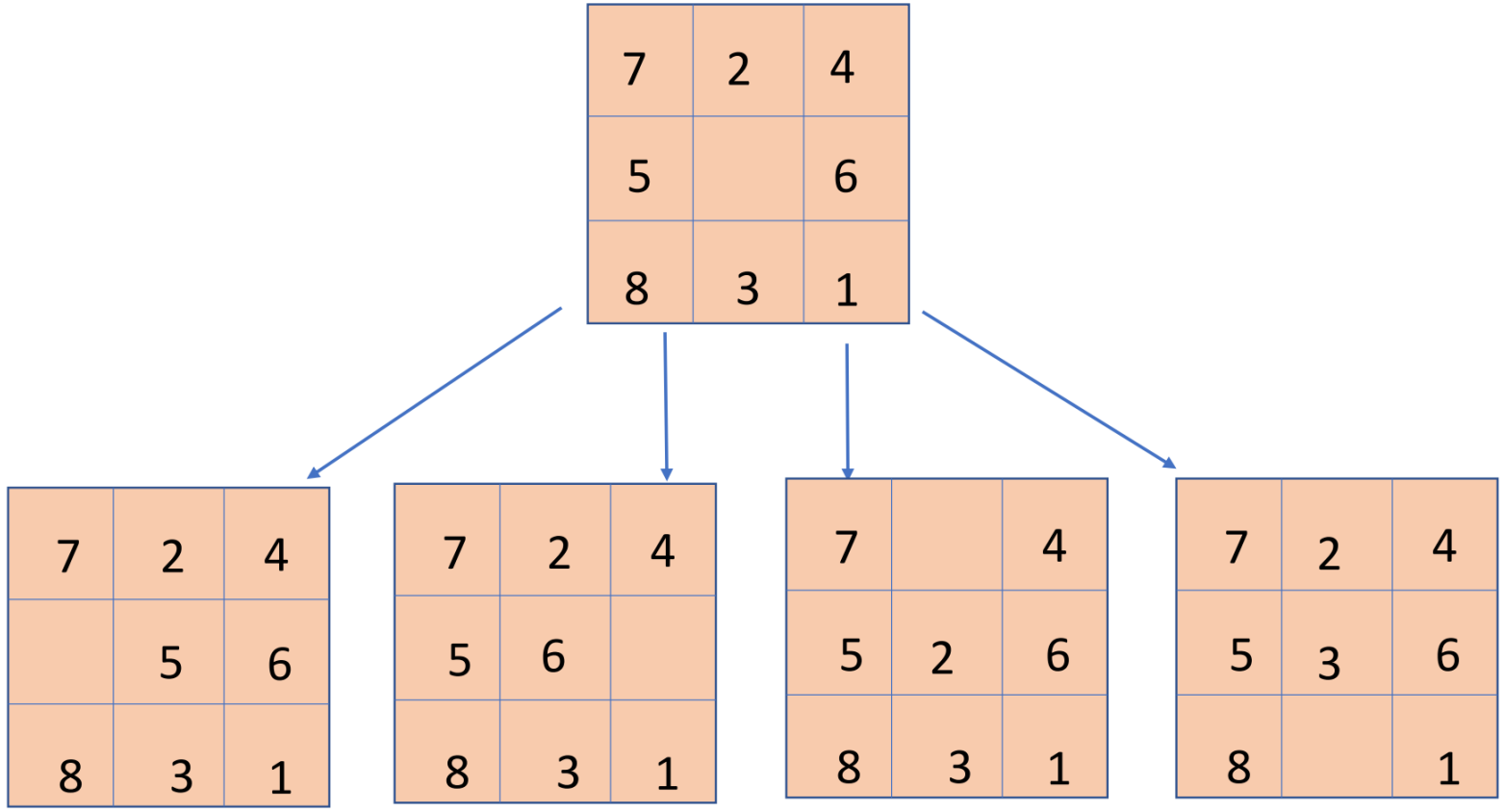
Gleb Sizov
Norwegian University of Science and Technology





A simplified road map of part of Romania, with road distances in miles.

8-puzzle



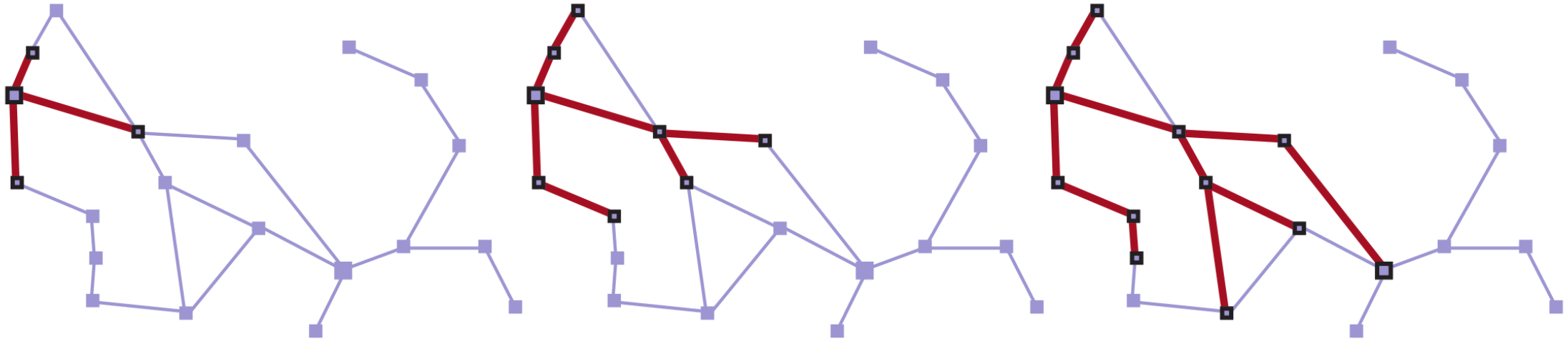
YouTube - Infinite Mario AI

Problem formulation - Romania example

1. State-space, e.g. locations on the map
2. Initial state, e.g. Arad
3. Goal states: $IsGoal(Bucharest) = True$
4. Actions: $Actions(Arad) = ToSibiu, ToTimisoara, ToZerind$
5. Transition model: $Result(Arad, ToZerind) = Zerind$
6. Action cost function: $ActionCost(s, a, s')$

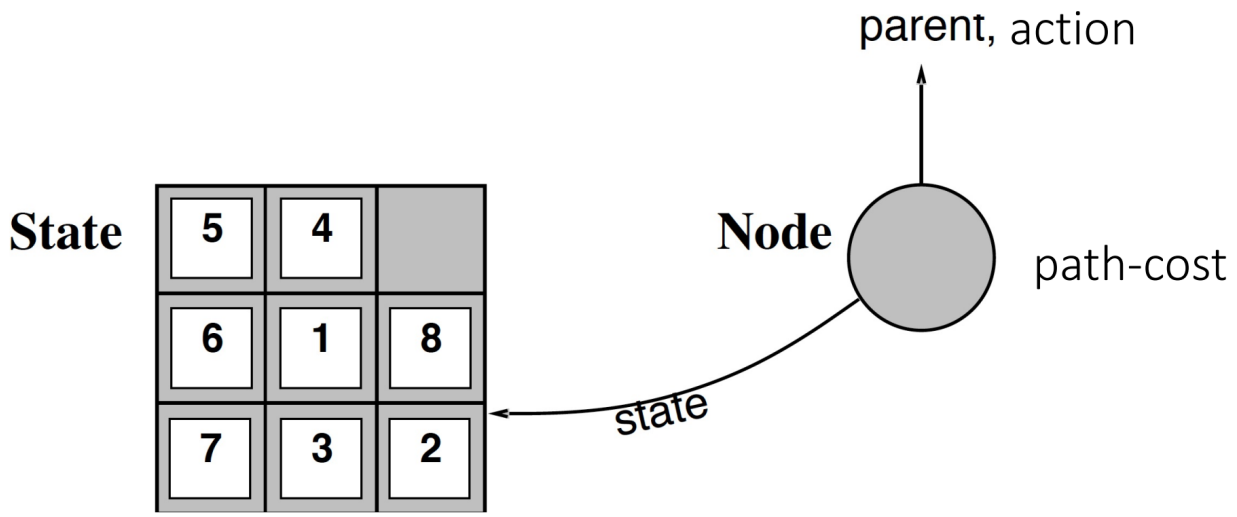
> A good problem formulation has the right level of abstraction.

Search trees



Node data structure

- node.STATE
- node.PARENT
- node.ACTION
- node.PATH-COST



Frontier data structures

- Priority queue - best-first search
- FIFO queue - breadth-first search
- LIFO queue (stack) - depth first search

Dealing with redundant paths

Cycle (loop) - special case of redundant path

1. Remember previously reached states, e.g. best-first search
2. Don't worry about redundant path, e.g. assembly problem
3. Check for cycles but not for other redundant paths

Graph search vs tree-like search.

Measuring problem-solving performance

Completeness - is algorithm guaranteed to find a solution (or failure)?

Cost optimality - does it find a solution with the lowest path cost?

Time complexity - how long it takes to find a solution (in seconds or states and actions considered)?

Space complexity - how much memmory needed, e.g. $frontier$, $reached$?

How to measure complexity

For explicit graph:

$|V| + |E|$ - size of state-space graph

For implicit graph defined by initial state, actions and transition model:

- d - depth - number of actions in optimal path
- b - branching factor - number of successors of node to consider
- m - maximum depth of the state space (may be infinite)

Uninformed vs informed search

Uninformed search - no clue about how close a state is to the goal(s).

Informed search - have some estimate of how close a state is to the goal(s).

S1

7	5	2
	4	3
8	1	6

S2

1	2	3
4		6
7	5	8

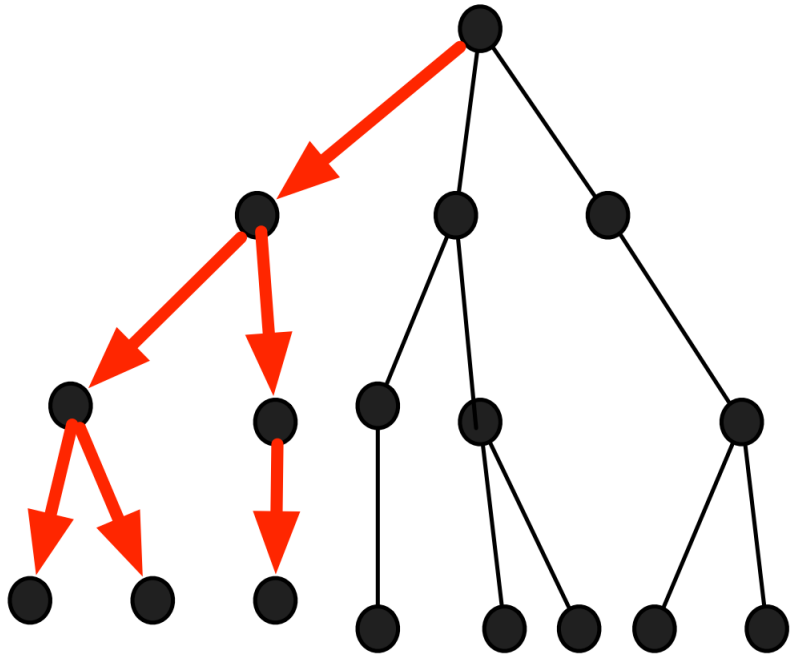
1	2	3
4	5	6
7	8	

Goal state

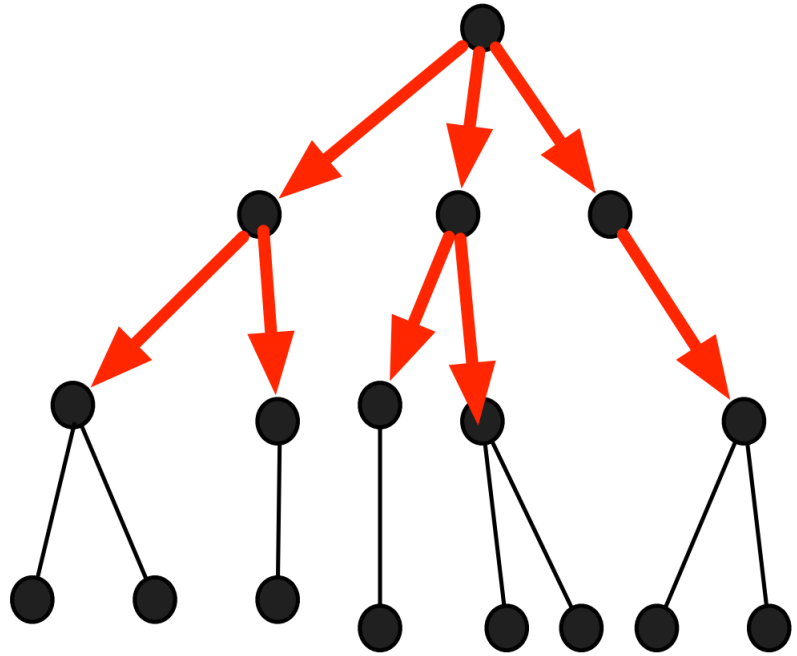
Uninformed search strategies

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Depth-first vs Breadth-fist



Depth-First

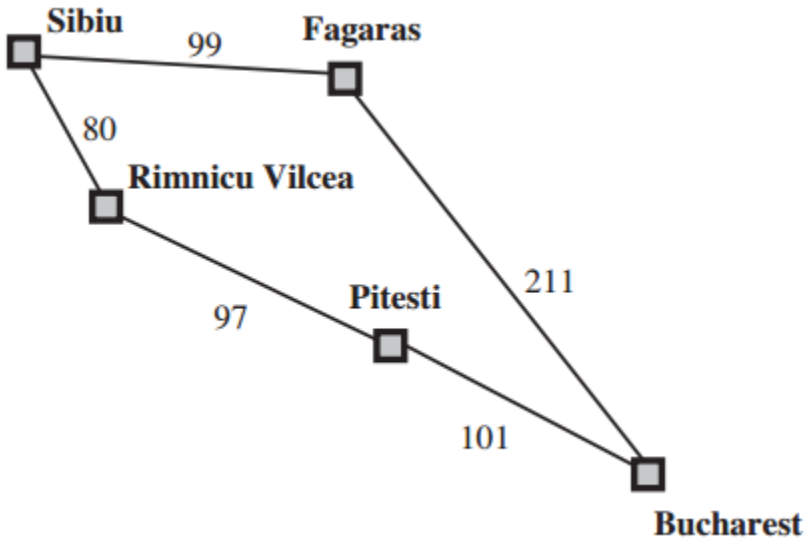


Breadth-First

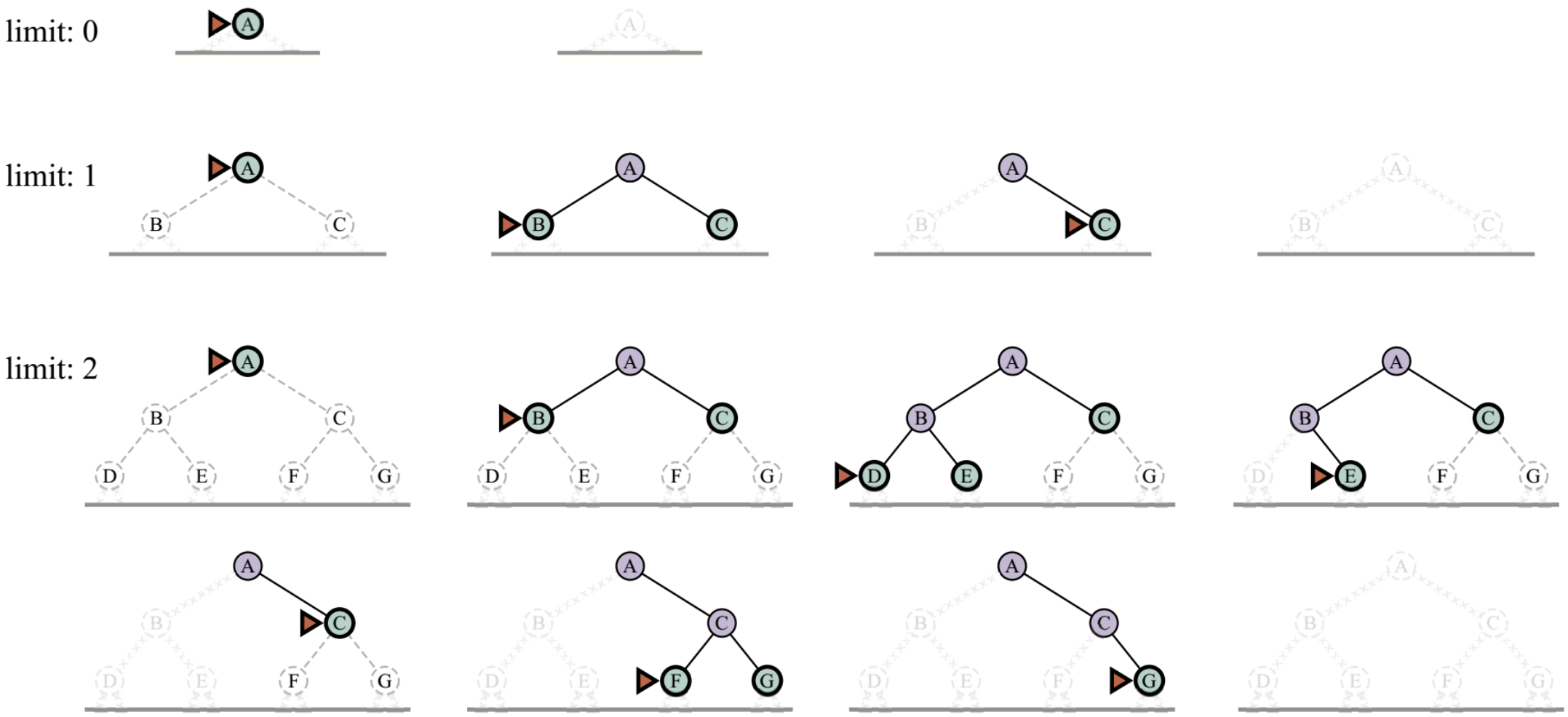
Uniform-cost search

- Handles varying positive step cost
- Uses priority queue sorted by path cost
- Expands nodes with least path cost first.

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution node, or *failure*
  return BEST-FIRST-SEARCH(*problem*, PATH-COST)



Depth-limited and iterative deepening search



Uninformed strategies - Summary of the properties

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

b - branching factor

d - depth of the shallowest solution

m - maximum depth

l - depth limit

Lecture 4
Informed Search
Search in Complex Environments

Gleb Sizov
Norwegian University of Science and Technology

Informed (Heuristic) search

> Use domain-specific hints about the location of goals.

Heuristic function:
$h(n)$ = estimated cost of the cheapest path from the state at node n to a goal state

Greedy best-first search

> Expand first the node with the lowest h(n)

Example heuristic:

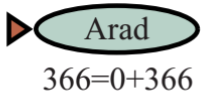
h_{SLD}—straight-line distances to Bucharest.

A* search

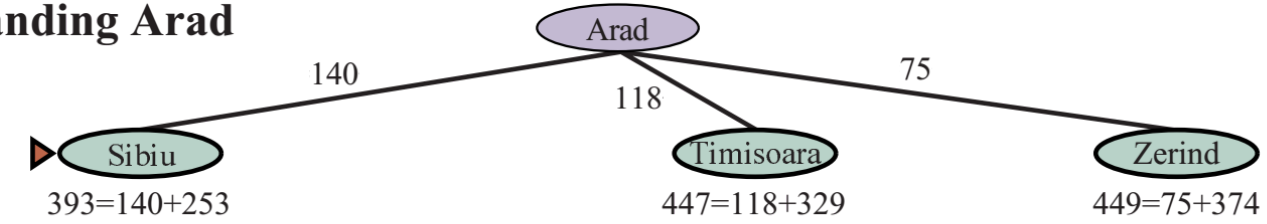
Best-first search with $f(n) = g(n) + h(n)$

- $g(n)$ - path cost from initial node to node n
- $h(n)$ - estimated cost of the shortest path from n to goal state

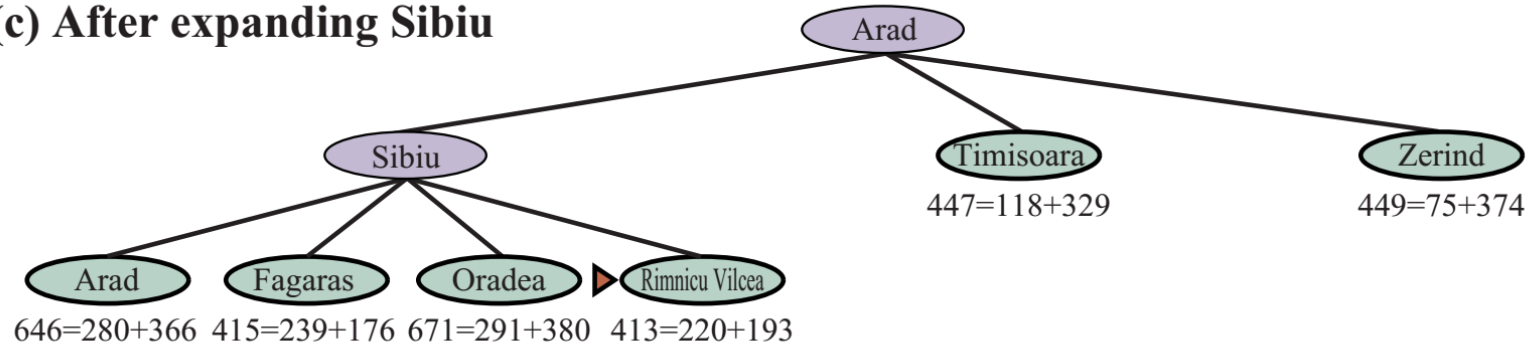
(a) The initial state



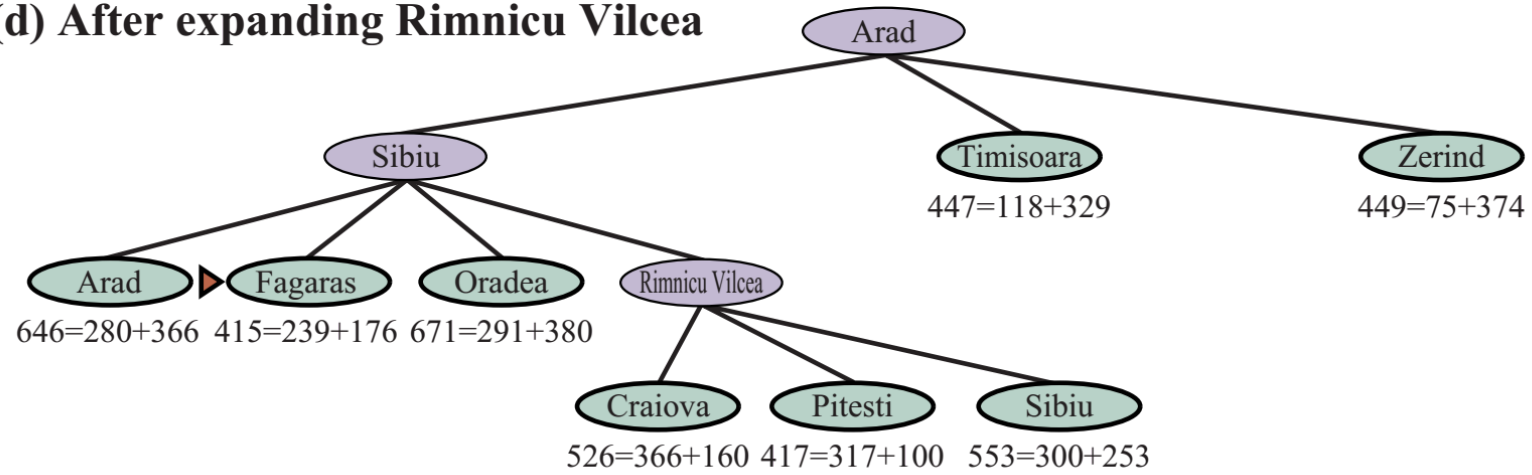
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Rimnicu Vilcea



Heuristic admissibility

Admissibility - an admissible heuristic never overestimates the cost to reach a goal.

It is optimistic!

S1

7	5	2
	4	3
8	1	6

S2

1	2	3
4		6
7	5	8

1	2	3
4	5	6
7	8	

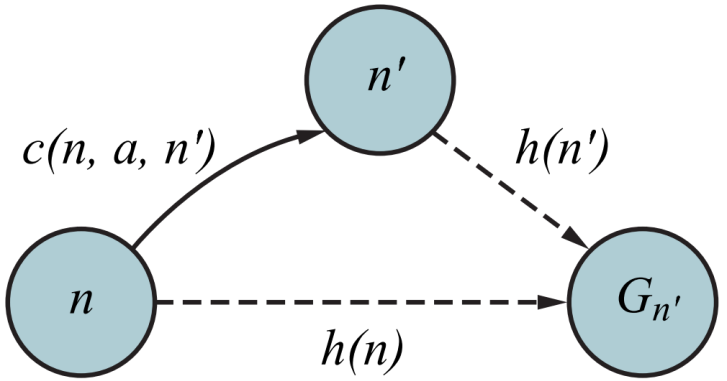
Goal state

Heuristic consistency

Consistency - $h(n) \leq c(n, a, n') + h(n')$
for every node n and every successor n' of n generated by an action a

- $c(n, a, n')$ - action cost

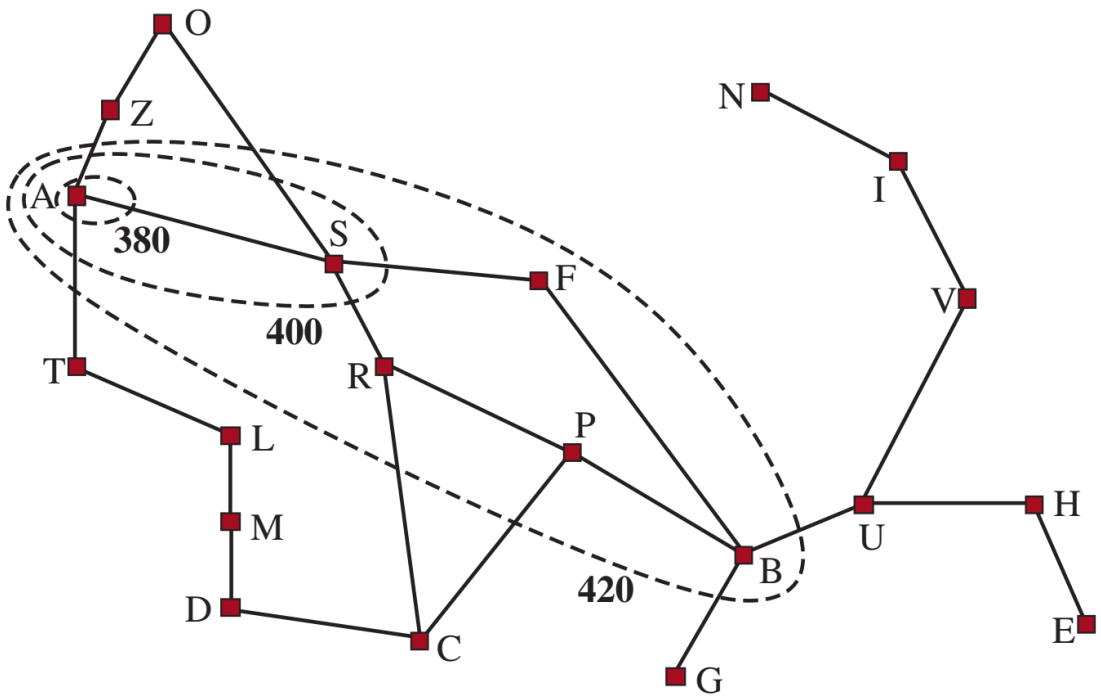
> Every consistent heuristic is admissible, but not vice versa.



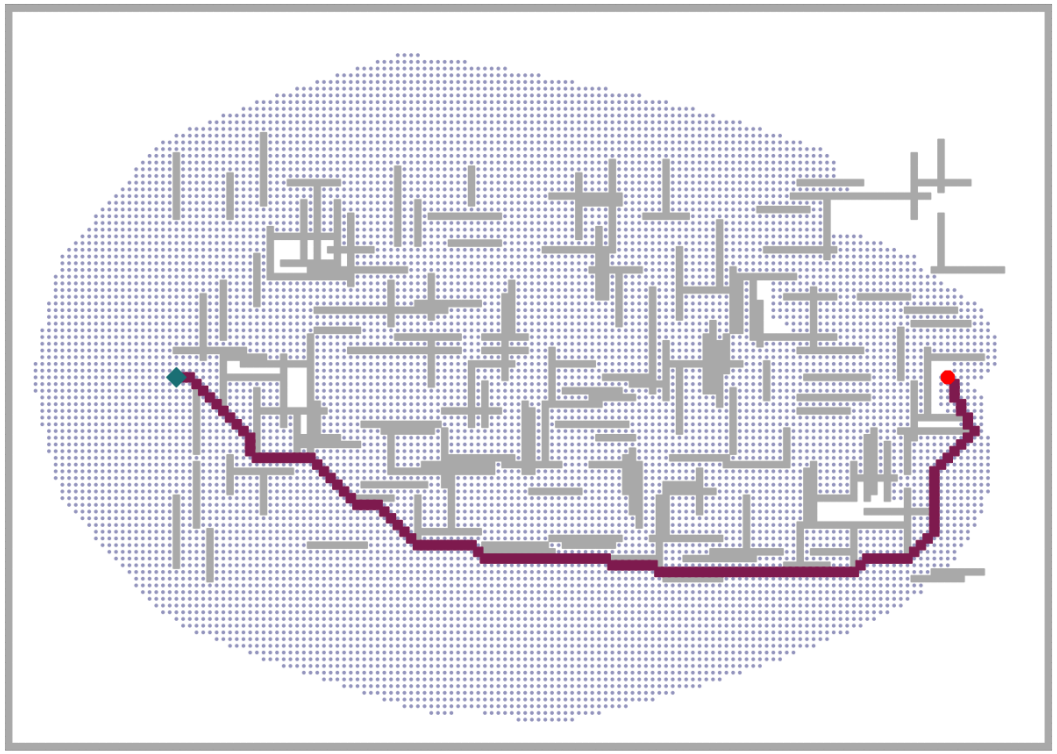
f - contours

A* expands nodes in order of increasing f.

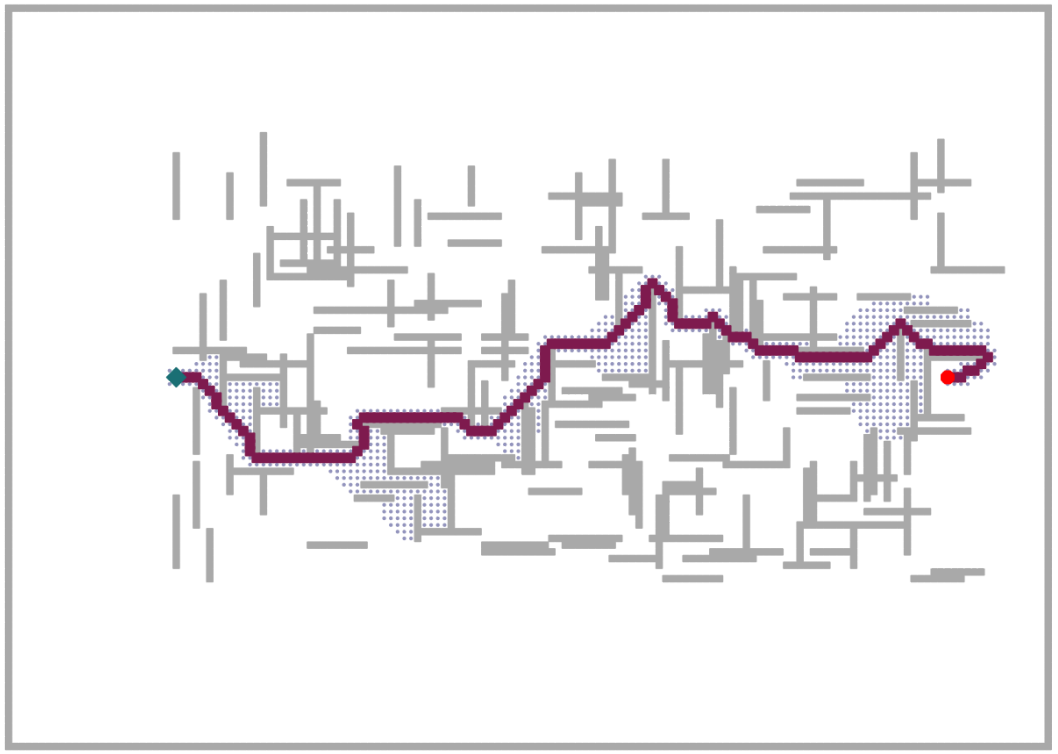
- Surely expanded nodes
- Optimally efficient
- Pruning



Satisficing good enough solutions



(a)



(b)

(a) A* (b) Weighted A* with $W = 2$

Memory-bounded search

Main issue of A* is the use of memory - keeps all nodes in memory (*reached*)

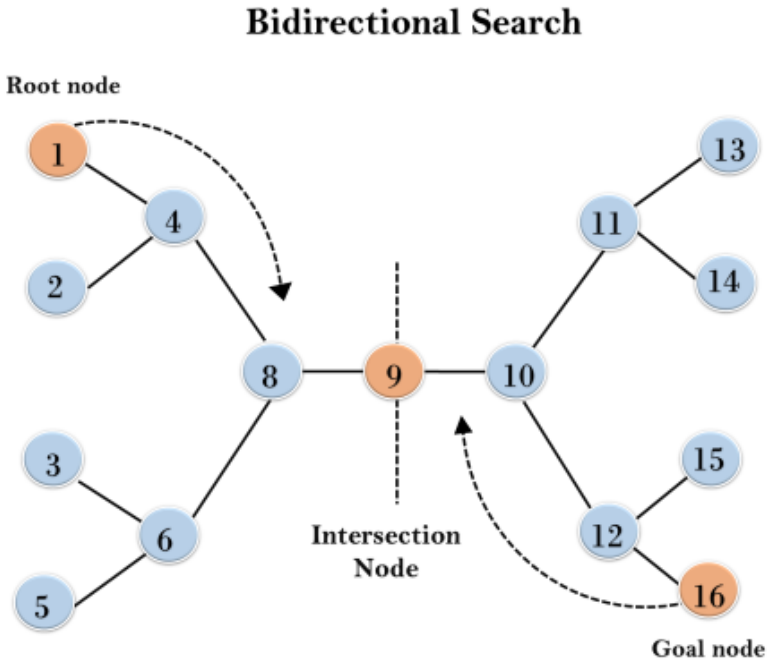
How to reduce memory use:

1. Reference count - remove a state when there are no more ways to reach it
2. Beam search - limit size of frontier to k best candidates
3. Iterative-deepening A* - gradually increase f-cost cutoff
4. Recursive best-first search (RBFS)
5. Memory-bounded A* - expand until memory is full, then drop the worst

Bidirectional search - Uninformed

Search forwards from initial state and backwards ffrom the goal, hoping that searches will meet.

Time and space: $O(b^{d/2}) + O(b^{d/2}) = O(b^{d/2})$, e.g. 50,000 times less than $O(b^d)$ for $b = d = 10$)



Heuristic functions

8-puzzle - $9!/2 = 181000$ states

15-puzzle - $16!/2 = 10000000000000$ states

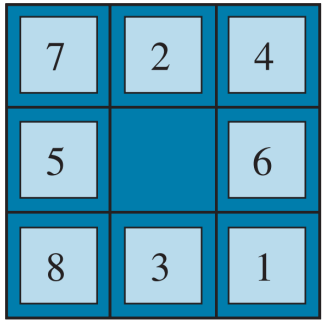
$h_1(n)$ = number of misplaced tiles,

e.g. 8

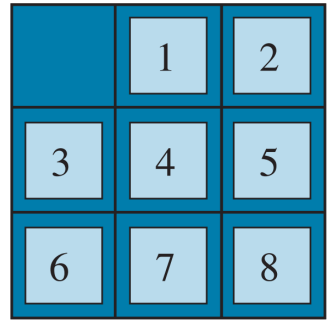
$h_2(n)$ = **Manhattan distance** - number of squares from desired location of each tile,

e.g. 3 + 1 + 2 + 2 + 3 + 2 + 2 + 3 = 18

Shortest solution cost is 26 actions.



Start State

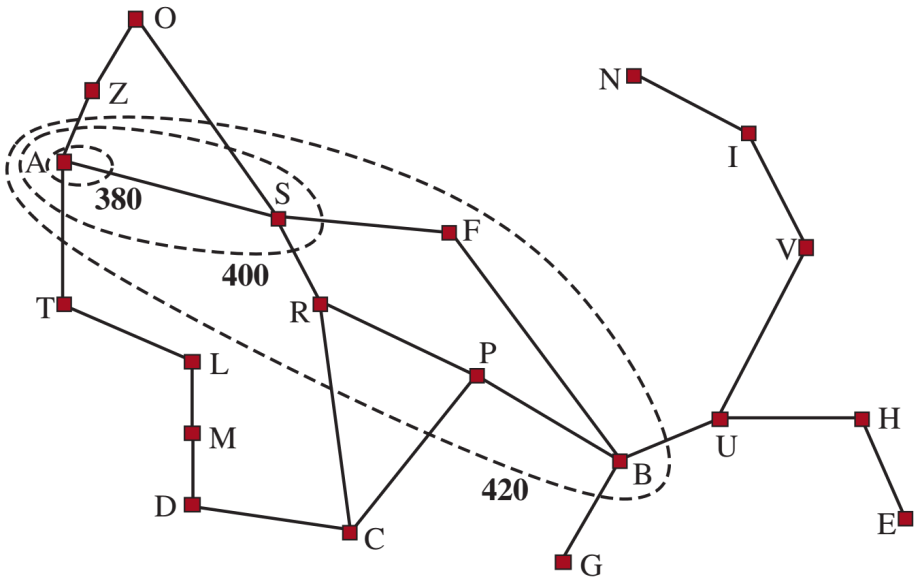


Goal State

Effective branching factor for h

Braching factor that a uniform tree of depth d would have to contain $N + 1$ nodes.

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)d$$



	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	A*(h_1)	A*(h_2)	BFS	A*(h_1)	A*(h_2)
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

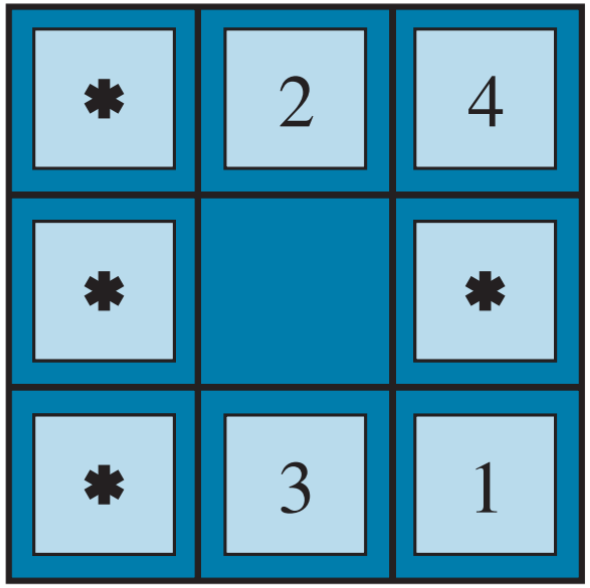
Generating heuristics from relaxed problems

Relaxed problem - simplified version of the problem, fewer restrictions, with shortcuts.

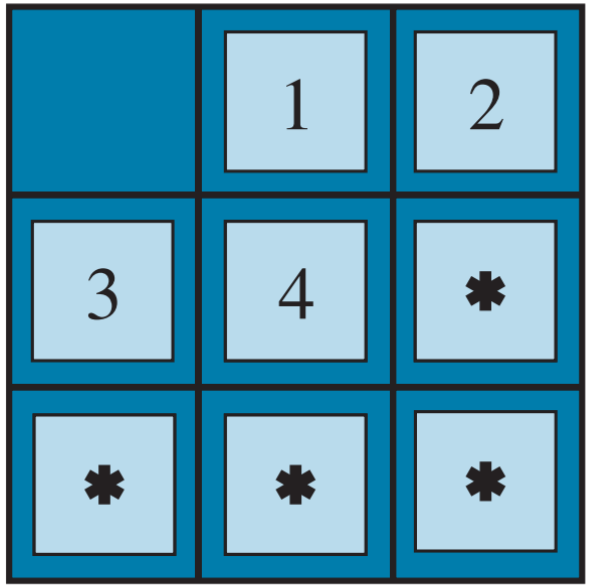
Examples for 8-puzzle:

a. A tile can move from square X to square Y if X is adjacent to Y.
b. A tile can move from square X to square Y if Y is blank.
c. A tile can move from square X to square Y.

Generating heuristics from subproblems
Pattern databases



Start State



Goal State

9 × 8 × 7 × 6 × 5 = 15,120 patterns in the database

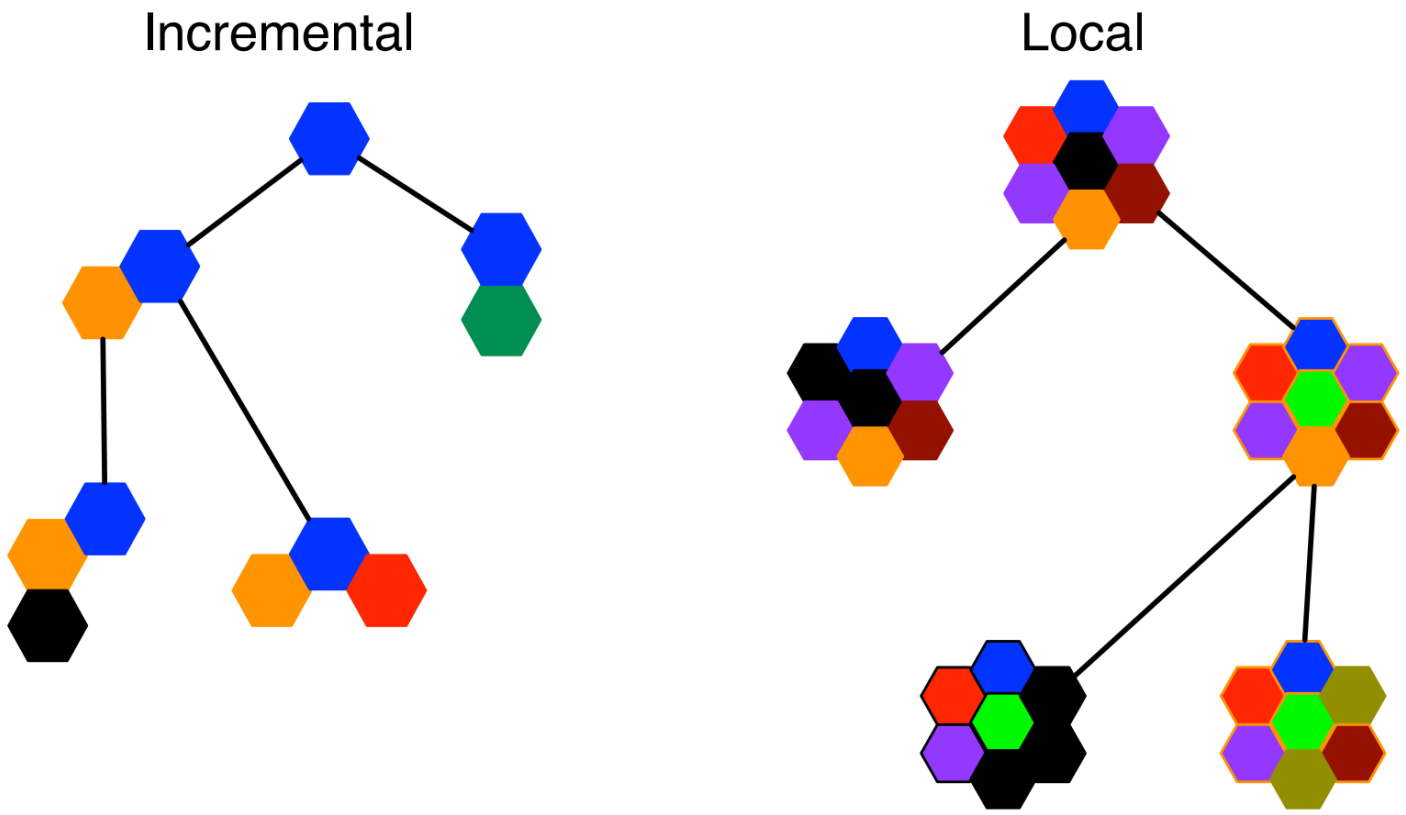
Local search

Path is irrelevant, **goal state** is the solution.

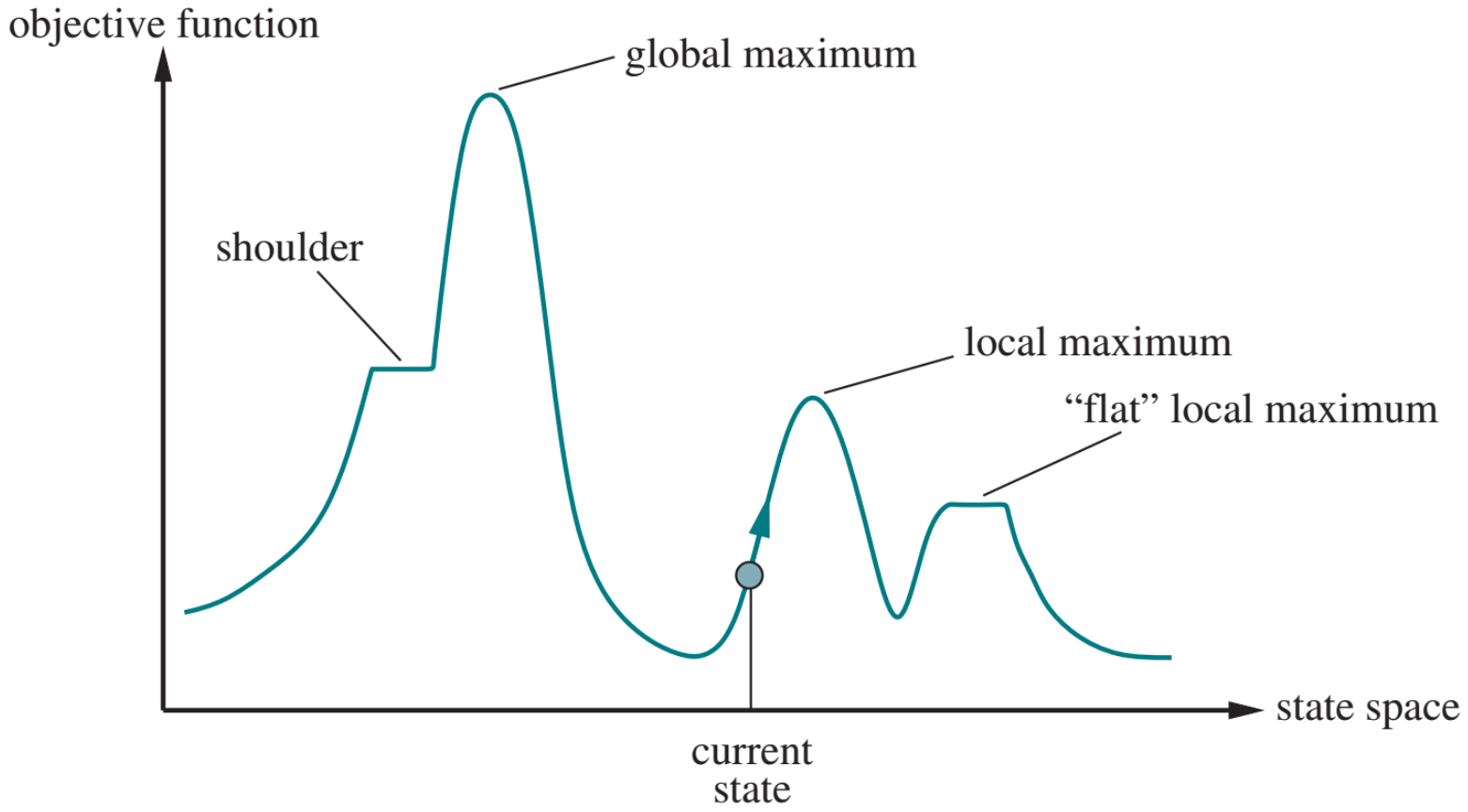
State space = set of "complete" configurations.
e.g. find optimal layout or timetable that satisifes constraints

Idea: Search from current state(s) to neighbour states to find a better one.

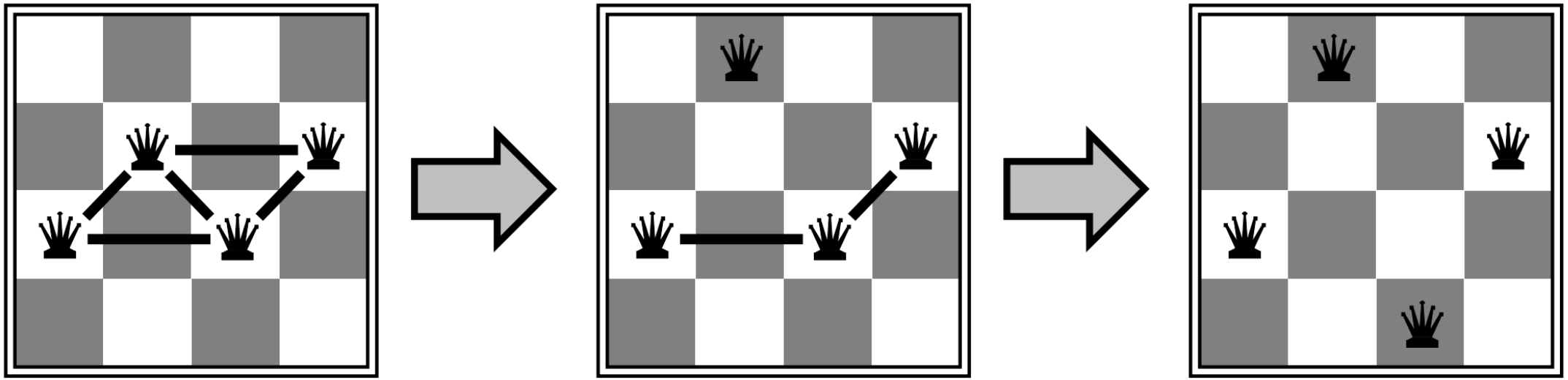
Incremental vs local search



State-space landscape



Example: n-queens

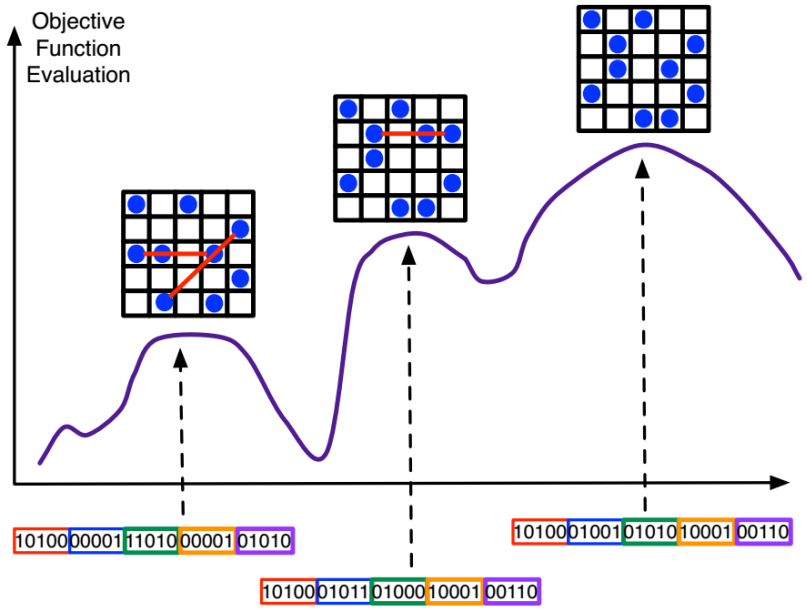


Properties of local search

- Uses very **little memory** - only keeps one (or a set) of current states, not paths or reached states.
- Often finds **reasonable solutions** in large or infinite state space
- Can solve **optimization problems**, finding the best state according to an **objective function**

Hill-climbing search

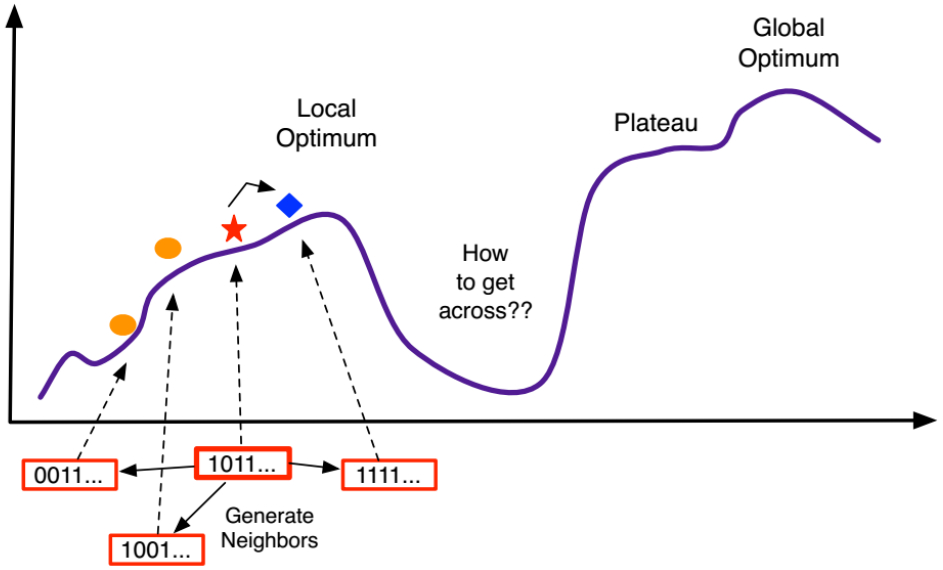
function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  current ← *problem*.INITIAL
  while *true* **do**
    neighbor ← a highest-valued successor state of *current*
    if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
    current ← *neighbor*



Hill-climbing gets stuck

- Local maxima
- Ridges
- Plateaus

"Like climbing Everest
in thick fog with amnesia"



Hot to not get stuck

- Sideways move
- Sochastic hill climbing
- First-choice hill climbing
- Random-restart hill climbing

Simulated annealing

Idea: Escape local minima by allowing some "bad" moves.

Gradually decrease their size and frequency.

function SIMULATED-ANNEALING($problem, schedule$) **returns** a solution state
  $current \leftarrow problem.$INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** $current$
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow$ VALUE($current$) – VALUE($next$)
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$

Local beam search

Idea: Keep k states instead of 1; choose top k of all their successors.

- Not the same as k searches run in parallel.
- Searches that find good states "recruit" other searches to join them.

Problem: Ofte all k states end up on same local hill.

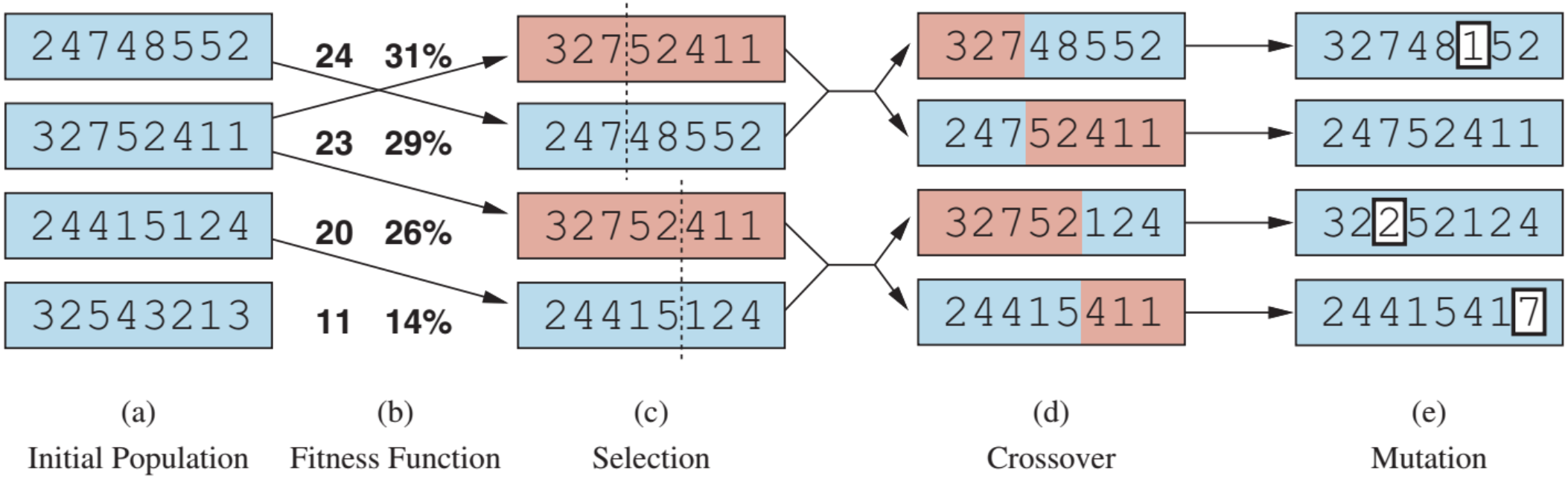
Idea: Choose k successors randomly, biased towards good ones.

Analogy to natural selection!

Genetic algorithms

1. Start with a **population** of *k* randomly generated states
2. Randomly choose two **parent** states weighted by their **fitness**
3. Generate a **child** state by combining **parent** states randomly.
4. **Mutate** the **child** state by introducing random changes.
5. Add **child** to the **population**.
6. Repeat from 2 until **child** is fit enough or time has elapsed.

Genetic algorithms

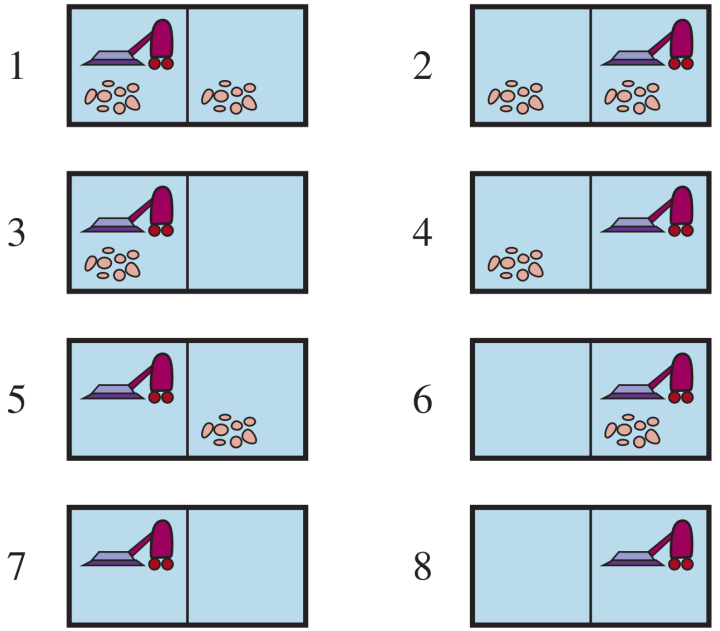


Search with nondeterministic actions

Example: Erratic vacuum world

Non-deterministic suck action:

- When applied to a dirty square, it cleans the square and sometimes cleans and adjacent square too.
  e.g. **Results(1, Suck) = {5,7}**
- When applied to a clean square, it may deposit dirt on the square.
  e.g. **Results(7, Suck) = {7,3}**



Solution in state 1:

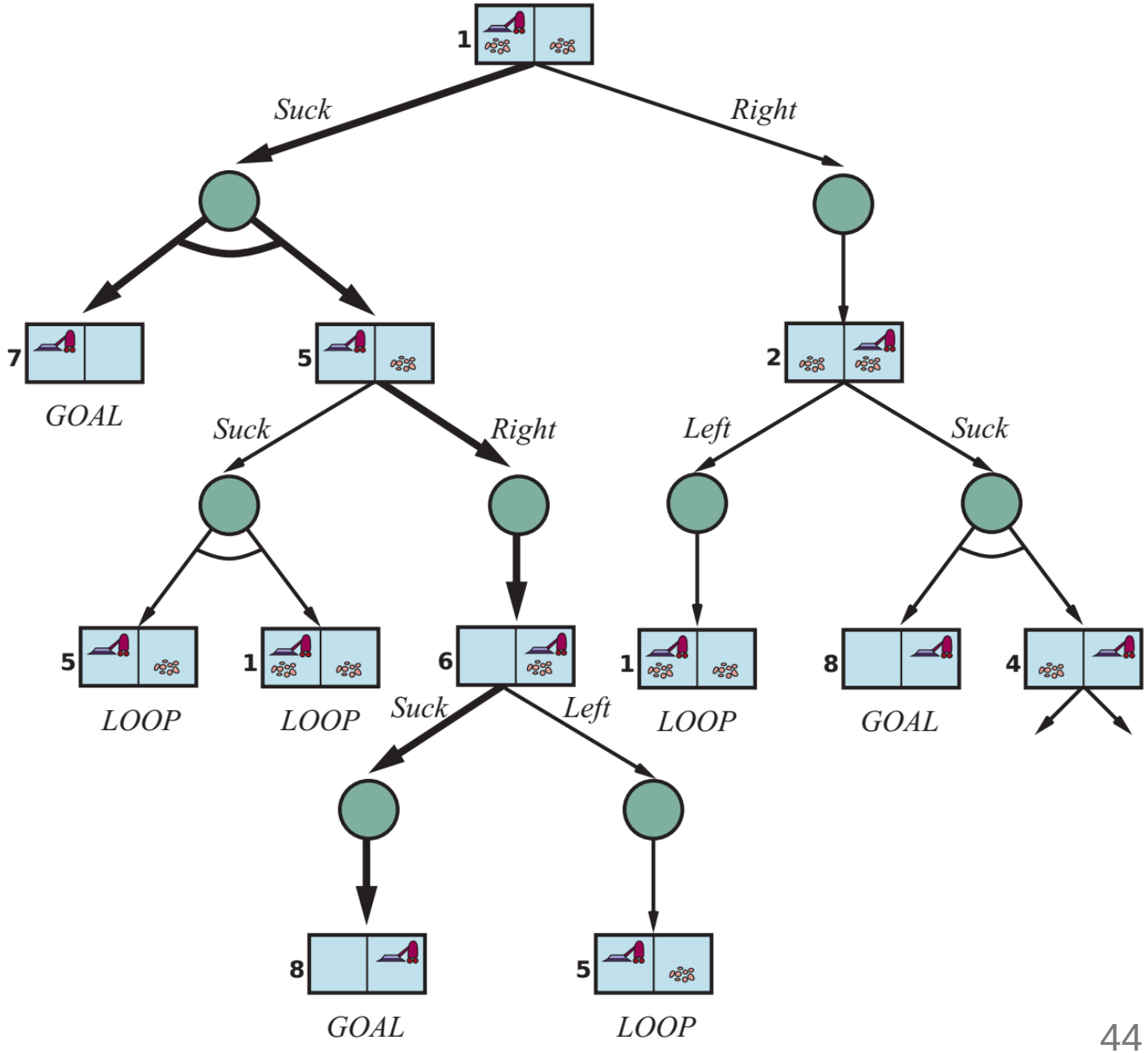
[Suck, if State = 5 then [Right, Suck] else []]

AND-OR search trees

OR nodes - actions

AND nodes - outcomes

(**belief states**)

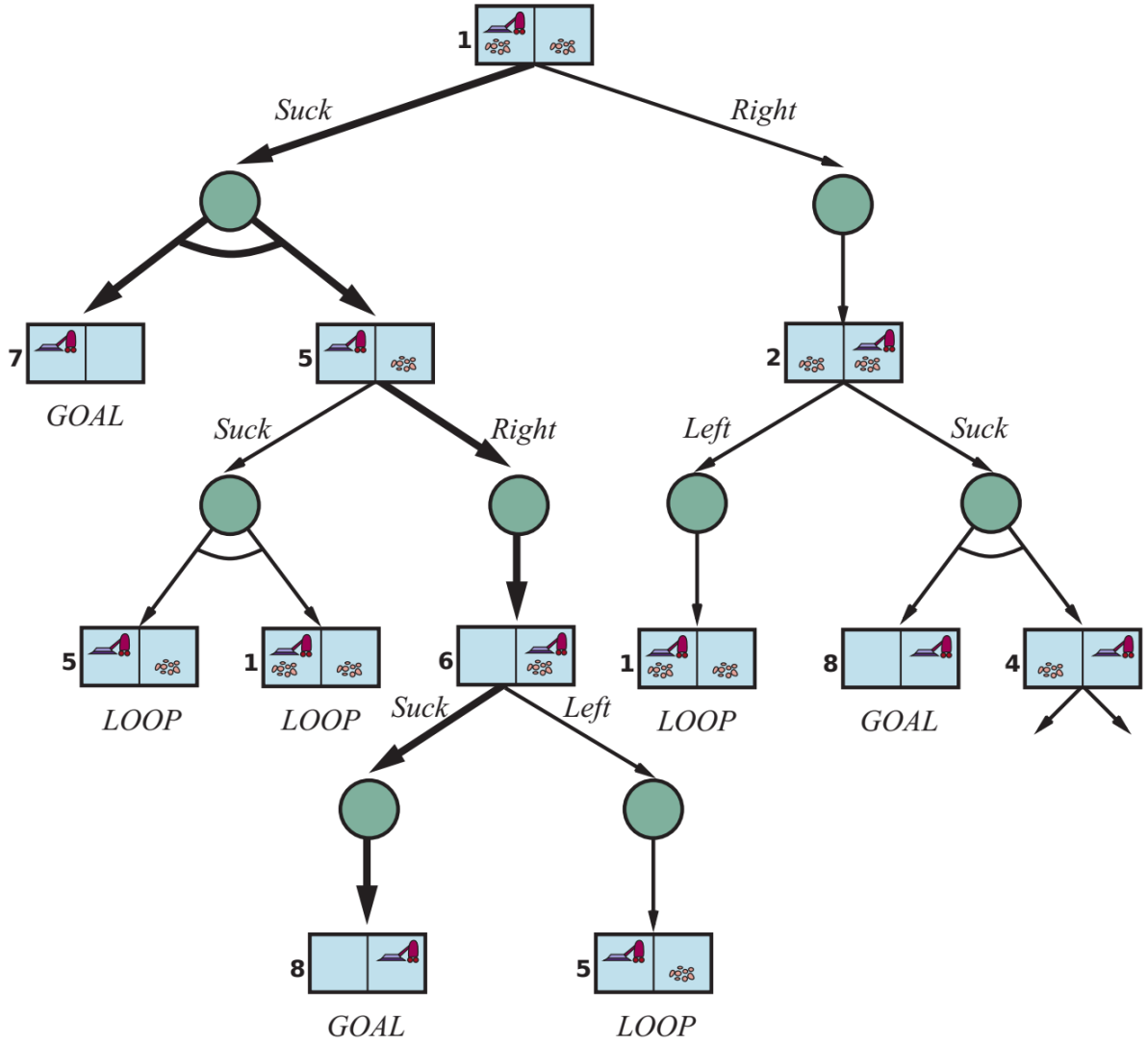


Search in partially observable environments

Partial or noisy observations are not enough to pin down the exact state.

Sensorless problem - no information from percepts at all.

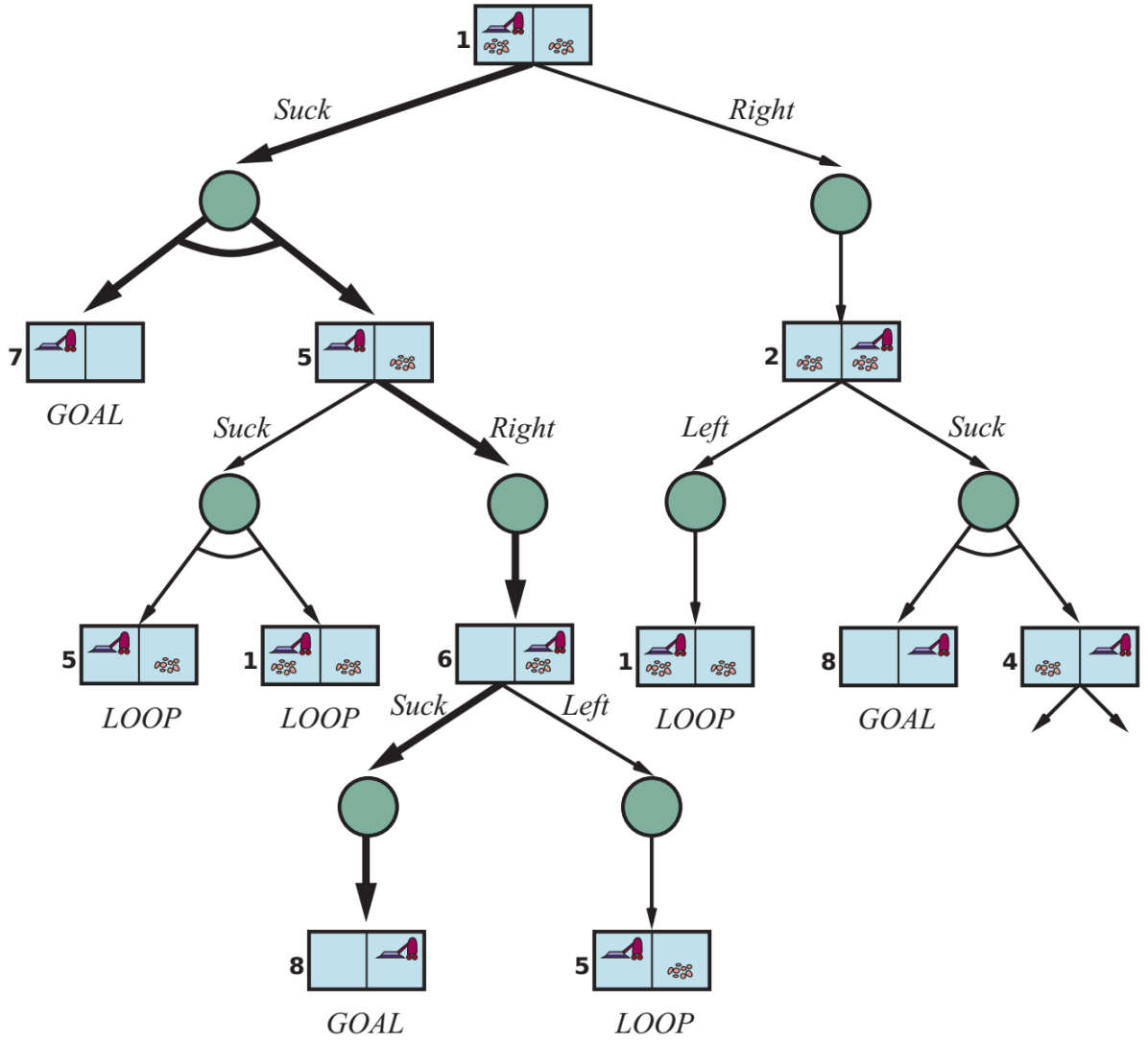
How to solve?



Sensorless (deterministic) vacuum world

Initial belief state: {1,2,3,4,5,6,7,8}

1. Result({1,2,3,4,5,6,7,8}, Right) = {2,4,6,8}
2. Result({2,4,6,8}, Suck) = {4,8}
3. Result({4,8}, Left) = {1,7}
4. Result({1,7}, Suck) = {7}



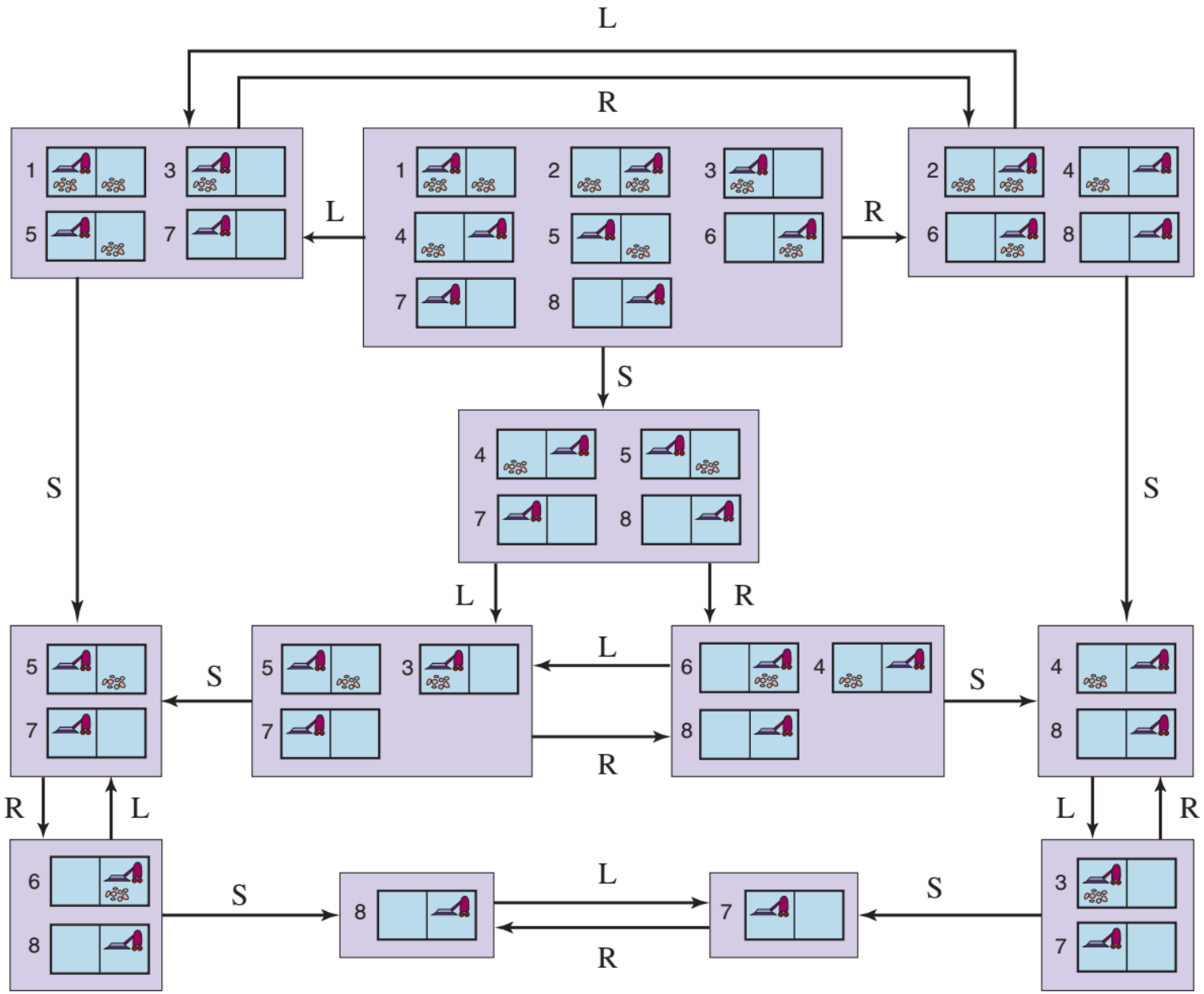
Sensorless belief-state space

Deterministic vacuum world

Possibly vs necessarily achieve goal

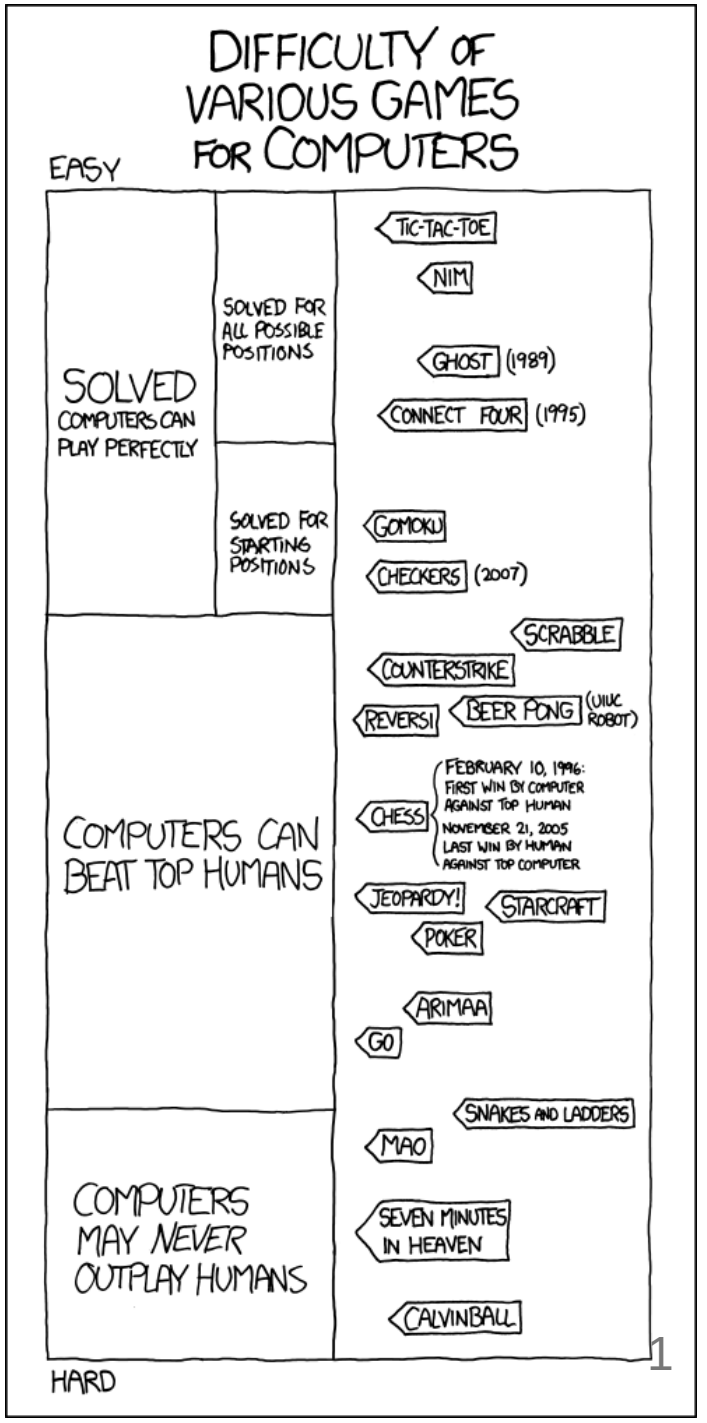
Prune supersets of reached belief state.

Problem: 2^N search space



Lecture 5
Adversarial Search and Games

Gleb Sizov
Norwegian University of Science and Technology



Two-player zero-sum games

Deterministic, two-player, turn-taking, perfect information, **zero-sum**,
e.g. chess, go

- S_0 - initial state
- $ToMove(s)$ - player whose turn it is to move
- $Actions(s)$ - set of legal moves
- $Result(s, a)$ - **transition model**, result state from taking action
- $IsTerminal(s)$ - game over
- $Utility(s, p)$ - objective function, value for player p, e.g. 1 - win, 0 - lose, 0.5 - draw

Game tree

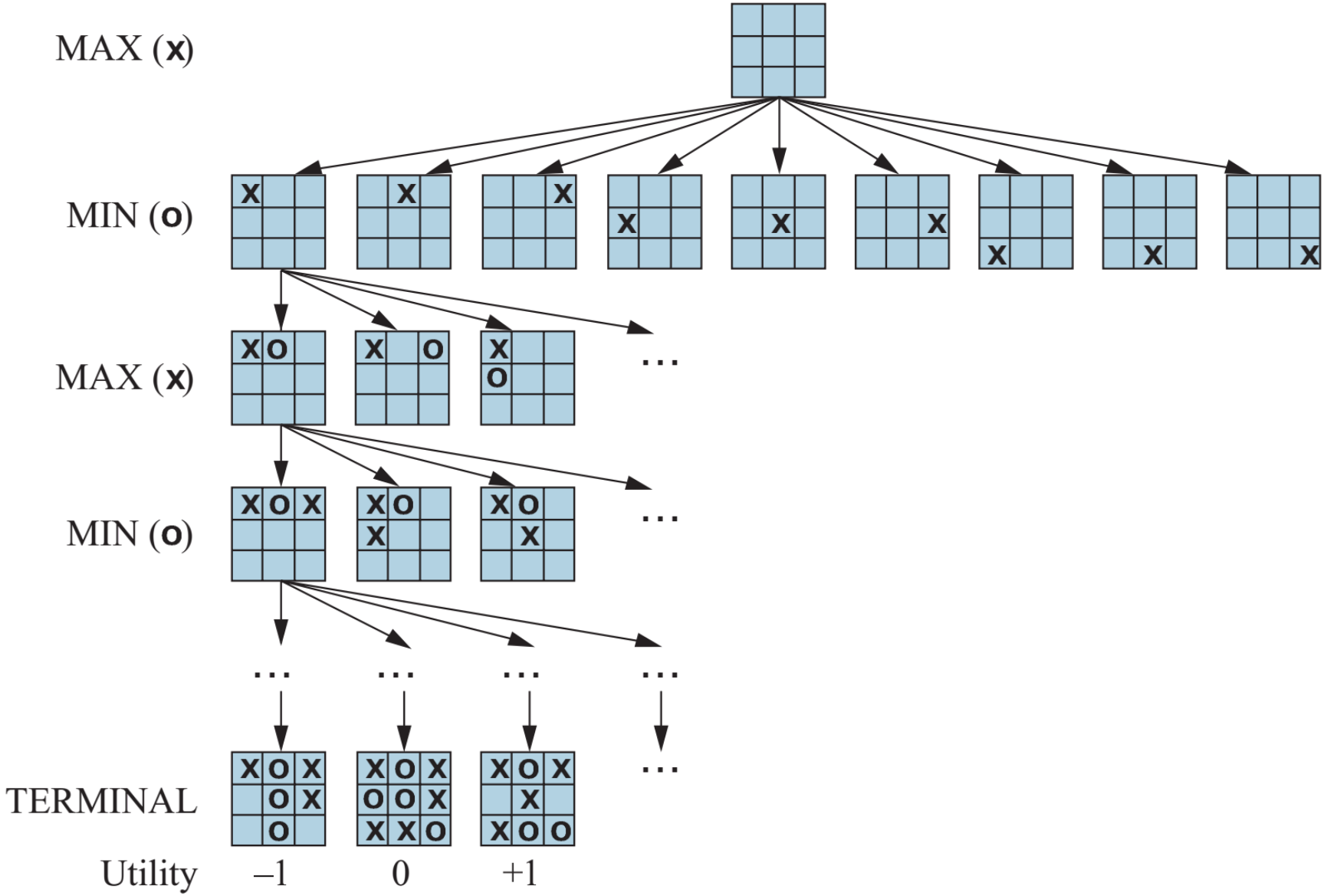
Tic-tac-toe:

363 880 nodes

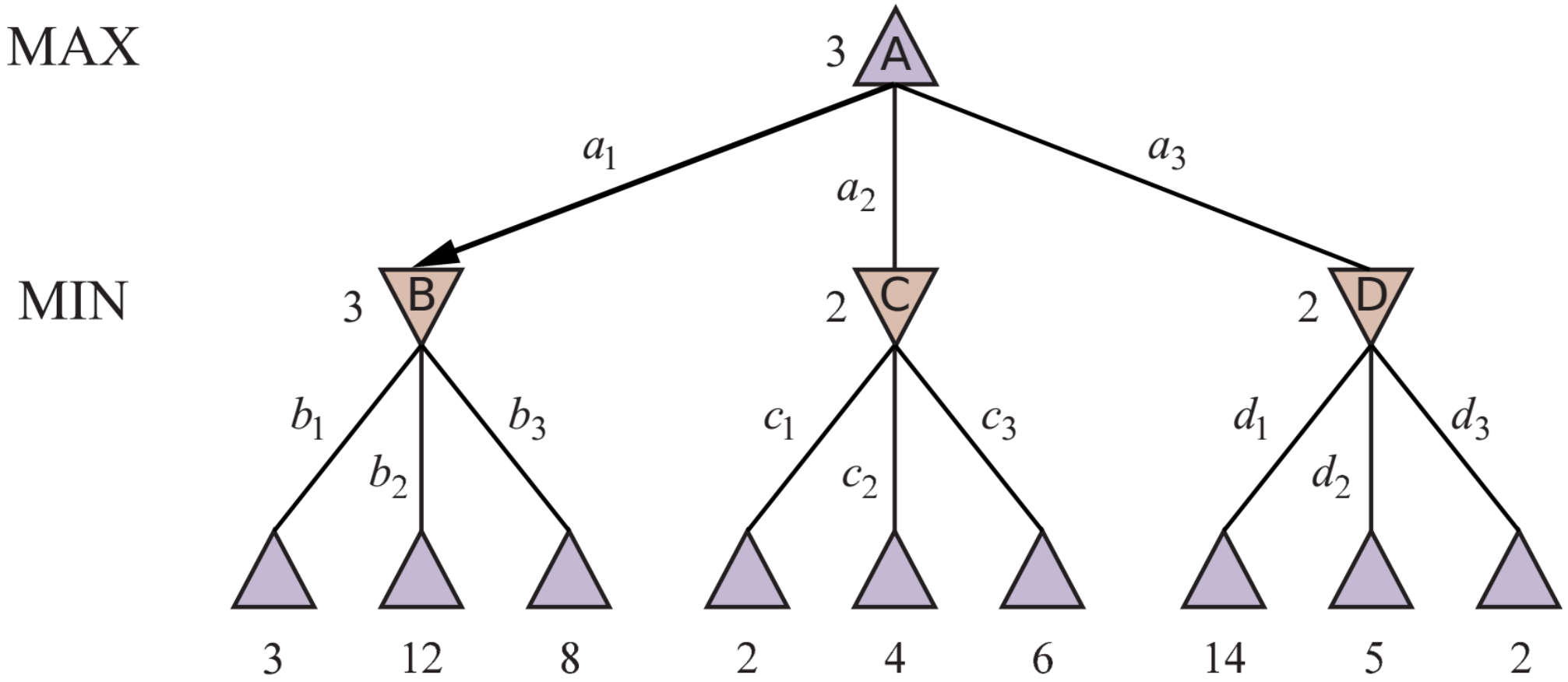
Chess:

10^{40} nodes

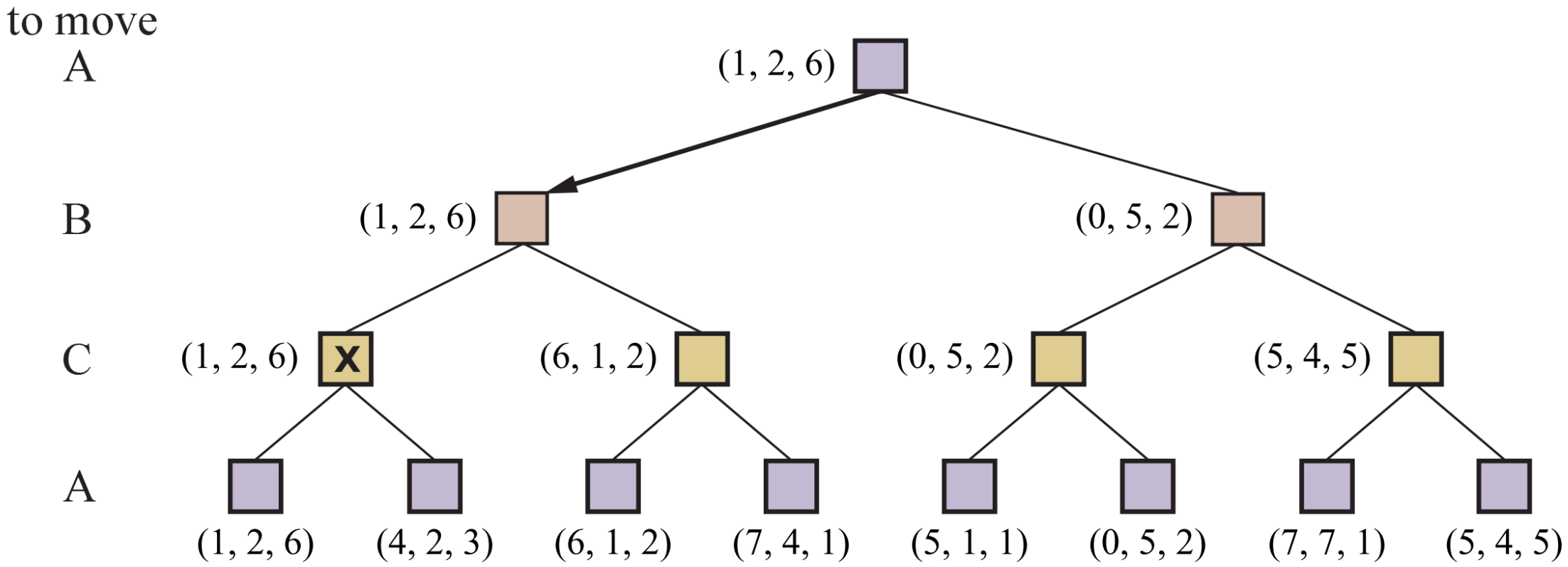
Theoretical construct



Minimax search

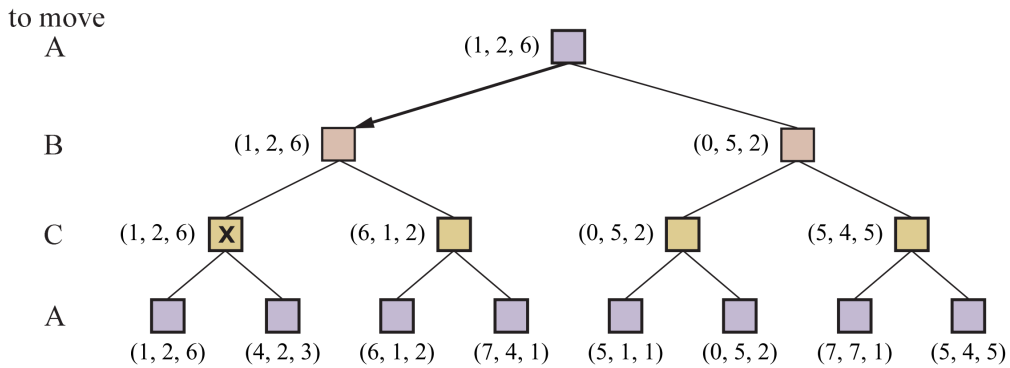


Multiplayer game tree

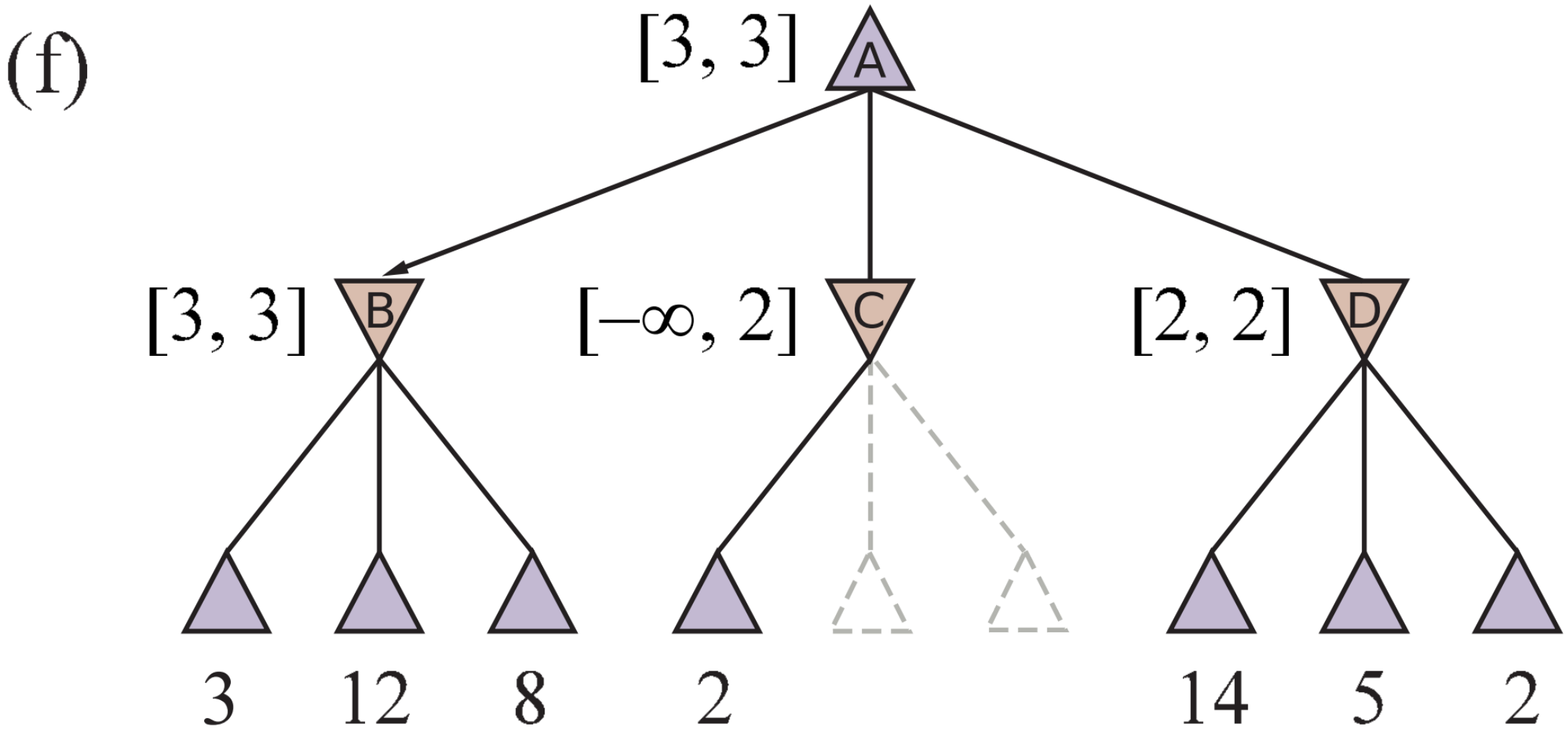


Alliances in multiplayer games

- Alliances - attack the strongest player
- Collaboration from selfish behaviour
- Breaking alliance - trust issues
- Collaboration



Alpha-beta pruning: Example - Stage 6



Type A vs Type B strategy

> Claude Shannon (1950), Programming a Computer for Playing Chess

Type A strategy - wide but shallow - considers all moves to a certain depth, then uses a heuristic function to estimate their utility.

Type B strategy - deep but narrow - ignores moves that look bad, and follows promising lines as deep as possible.

Historically, Type A for *chess* (b = 35), Type B for *Go* (b = 250)
Now, Type B for *chess* as well.

Heuristic minimax value

$$\text{H-MINIMAX}(s, d) = \begin{cases} \text{EVAL}(s, \text{MAX}) & \text{if IS-CUTOFF}(s, d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,\ a), d+1) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,\ a), d+1) & \text{if TO-MOVE}(s) = \text{MIN}. \end{cases}$$

$Eval(s, MAX)$ - heuristic **evaluation function**

$IsCutoff(s, d)$ - **cutoff test**

Forward pruning

> Remove moves that appear to be poor moves, but might possibly be good ones

Approaches:

- Beam search - consider n best moves.
- ProbCut - combines alpha-beta search with statistics from prior experiences.
- Late move reduction - uses move ordering to reduce the depth for later nodes

Endgame lookup tables

Generate a table for endgames with seven or few pieces.
Table size: 400 trillion position

Retrograde analysis - start from from final positions (check mate, steal mate) and play backwards to mark all nodes in the tree as win, loose or draw.

Bring it all together: Chess example

Compute: 60 million nodes/min

1. Combine **minimax** with **evaluation function**, **cuttoff** and **quiescence search**
   35^5 = 50 million, depth = 5
   Average human player, depth = 6, 8
2. Add **alpha-beta prunning**, **transposition table**
   Expert level, depth = 14
3. Add **8 GPUs**, **better evaluation function** + **lookup table**
   Sockfish, depth = 30
   Beats any human player

Weaknesses of heuristic alpha-beta tree search

Example: Go

1. High branching factor = 361, can only reach depth = 4, 5
2. Hard to define goode evaluation function:
    i. Material value is not a strong indicator
    ii. Most positions are influx until the endgame.

Monte Carlo tree search: Main idea

Estimate value of a *state* as the average utility over a a number of simulations of complete games starting from this *state*.

Simulation = **playout** = rollout

For win/loss games: average utility = win percentage

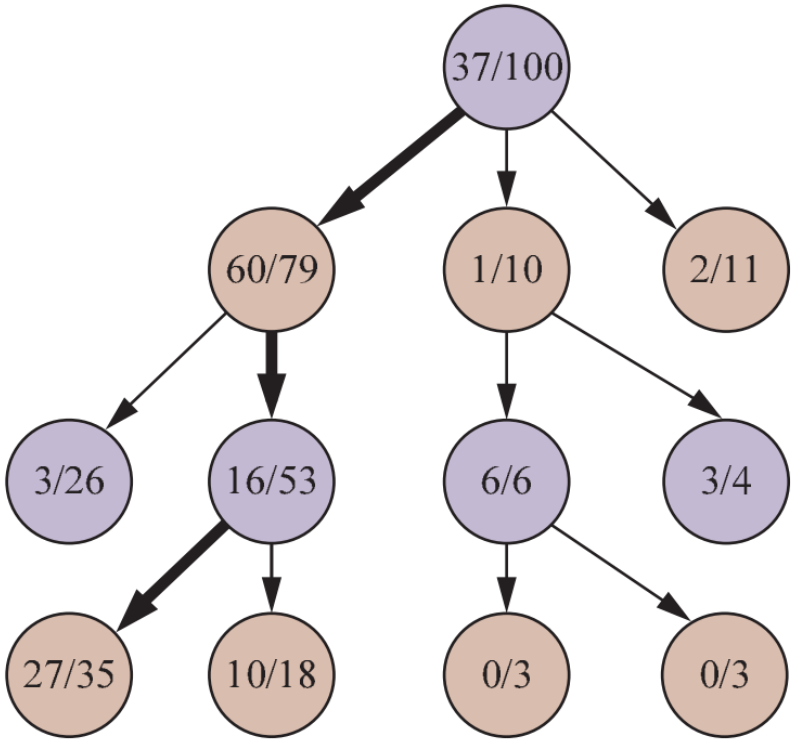
Monte Carlo tree search: Selection policy

Focus computational resources on the imporatant parts of the game tree.

Balance exploration vs exploitation:

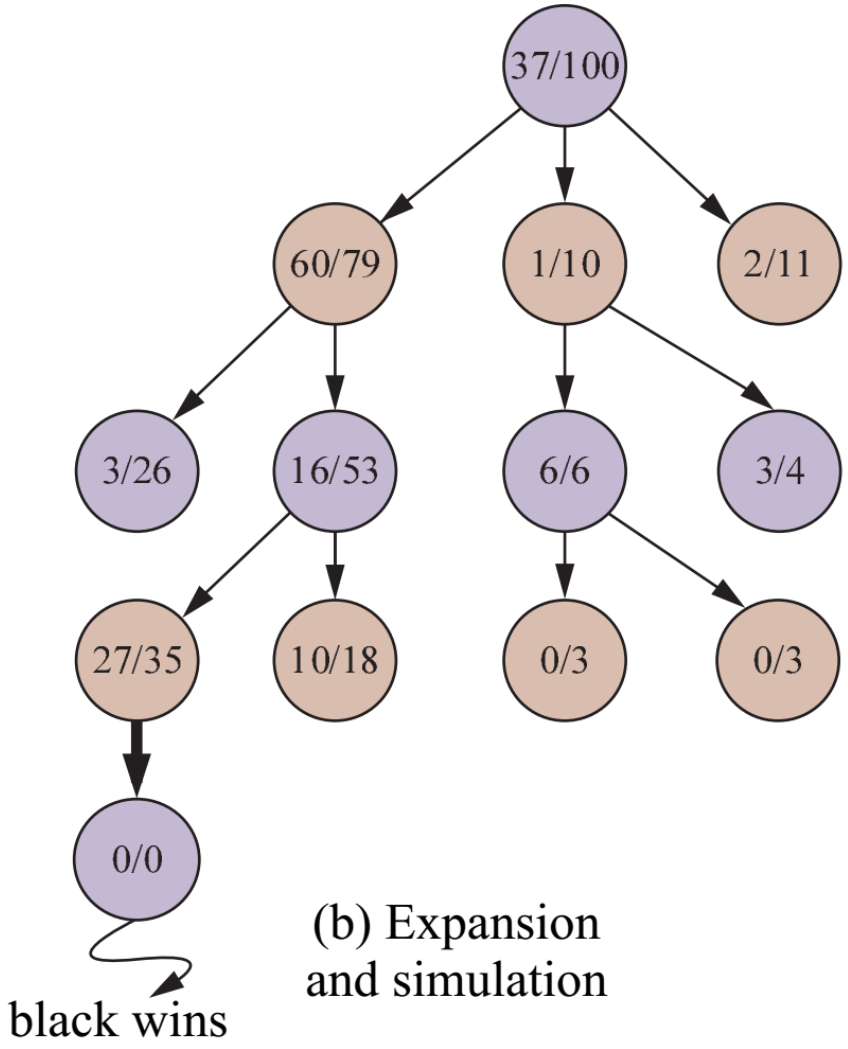
- **Exploration** - states that have had few playouts
- **Exploitation** - states that have done well in the past playouts

Monte Carlo tree search: Selection step



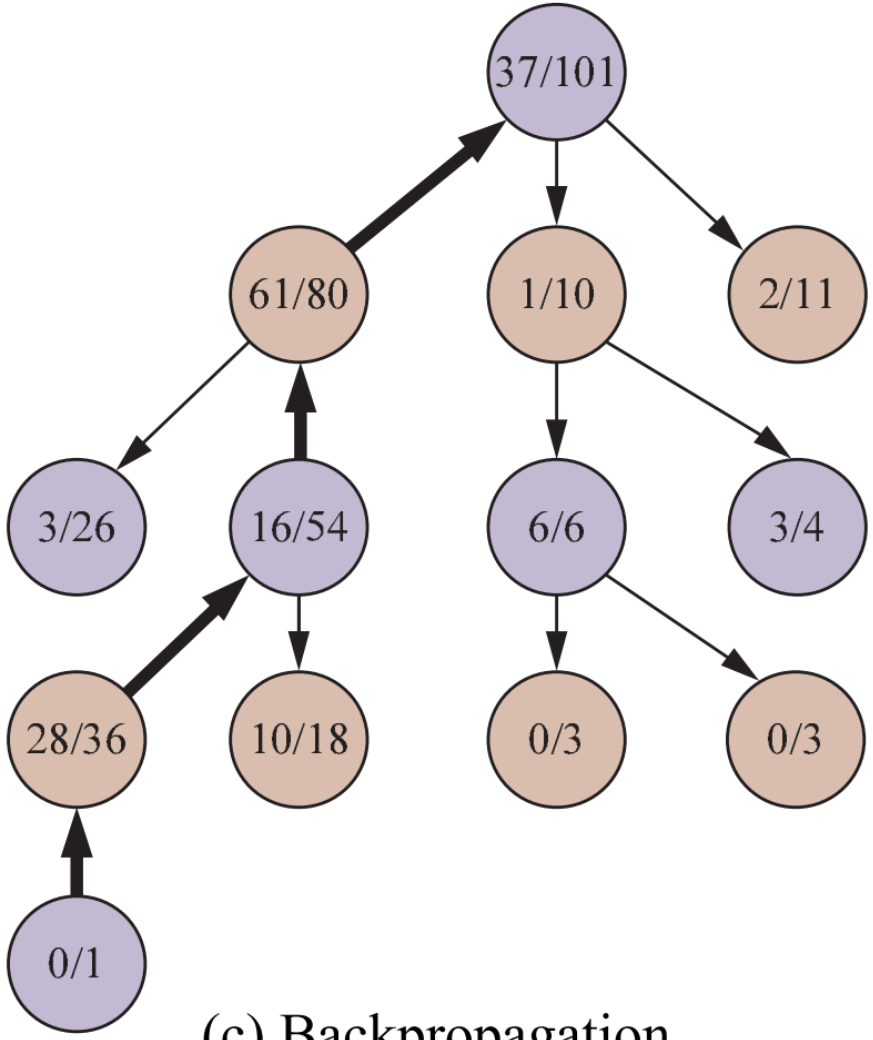
(a) Selection

Monte Carlo tree search: Expansion and simulation steps



(b) Expansion and simulation

Monte Carlo tree search: Backpropagation



(c) Backpropagation

Monte Carlo tree search pros and cons

Pros:

- Better for high branching factor than alpha-beta search
- Less sensistive to evaluation function errors than alpha-beta search
- Can be applied to new games without evaluation function
- Easily parallelized

Cons:

- Fails for games where a single move can often change the course of the game.

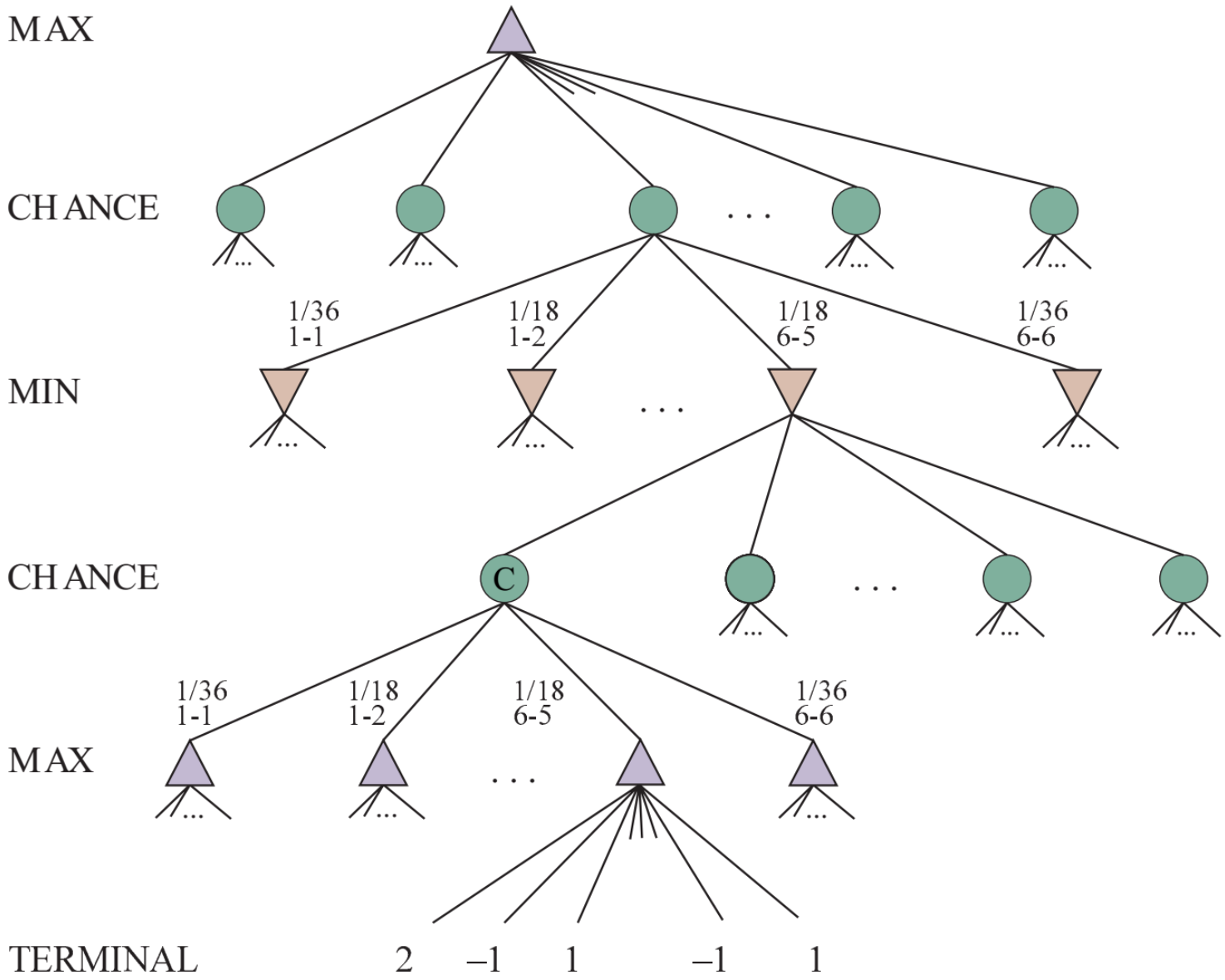
Variations:

- Monte Carlo tree search + evaluation function

Stochastic game tree

Nodes:

- Max
- Min
- Chance nodes (probability)



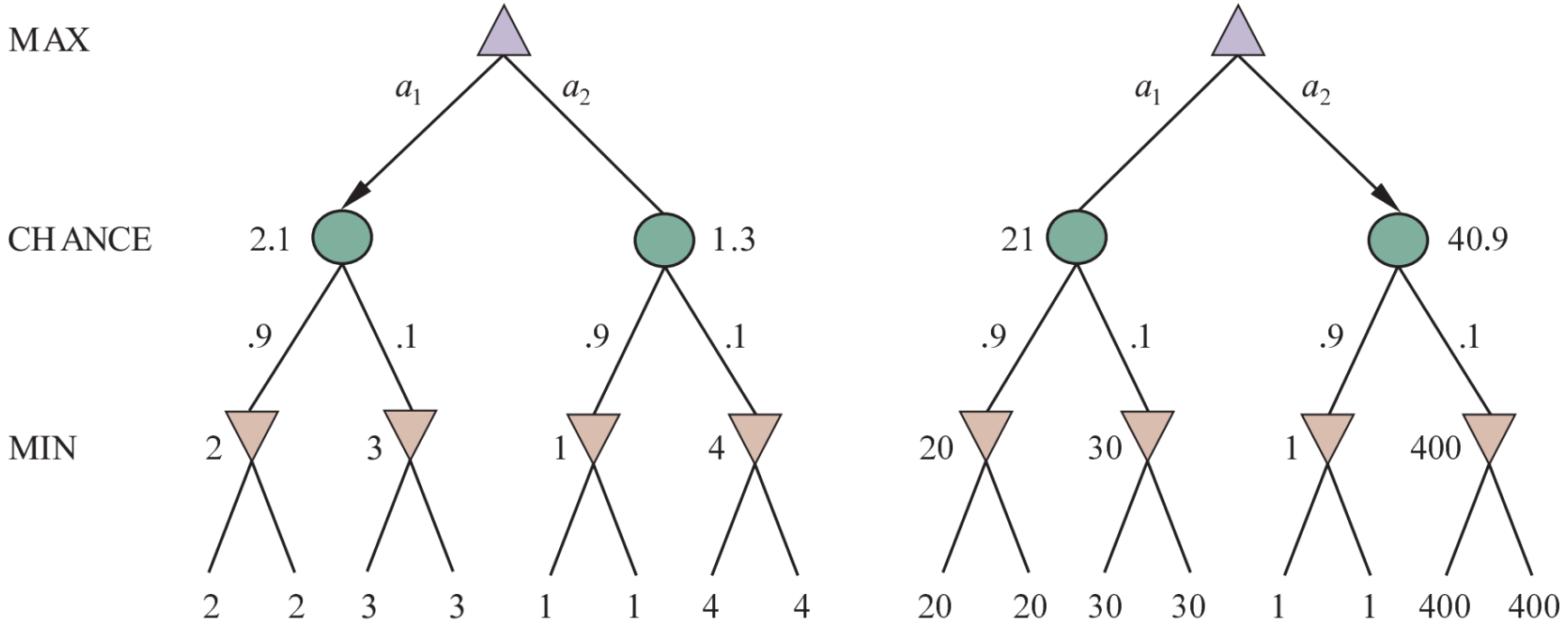
Expected value: expectiminimax value

$$\textsc{Expectiminimax}(s) = \begin{cases} \textsc{Utility}(s,\textsc{max}) & \text{if } \textsc{Is-Terminal}(s) \\ \max_a \text{if } \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{To-Move}(s) = \textsc{max} \\ \min_a \text{if } \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{To-Move}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}\Big(\textsc{Result}\Big(s,r\Big)\Big) & \text{if } \textsc{To-Move}(s) = \textsc{chance} \end{cases}$$

Stochastic games: evaluation functions

if *game*.IS-CUTOFF(*state*, *depth*) **then return** *game*.EVAL(*state*, *player*), *null*

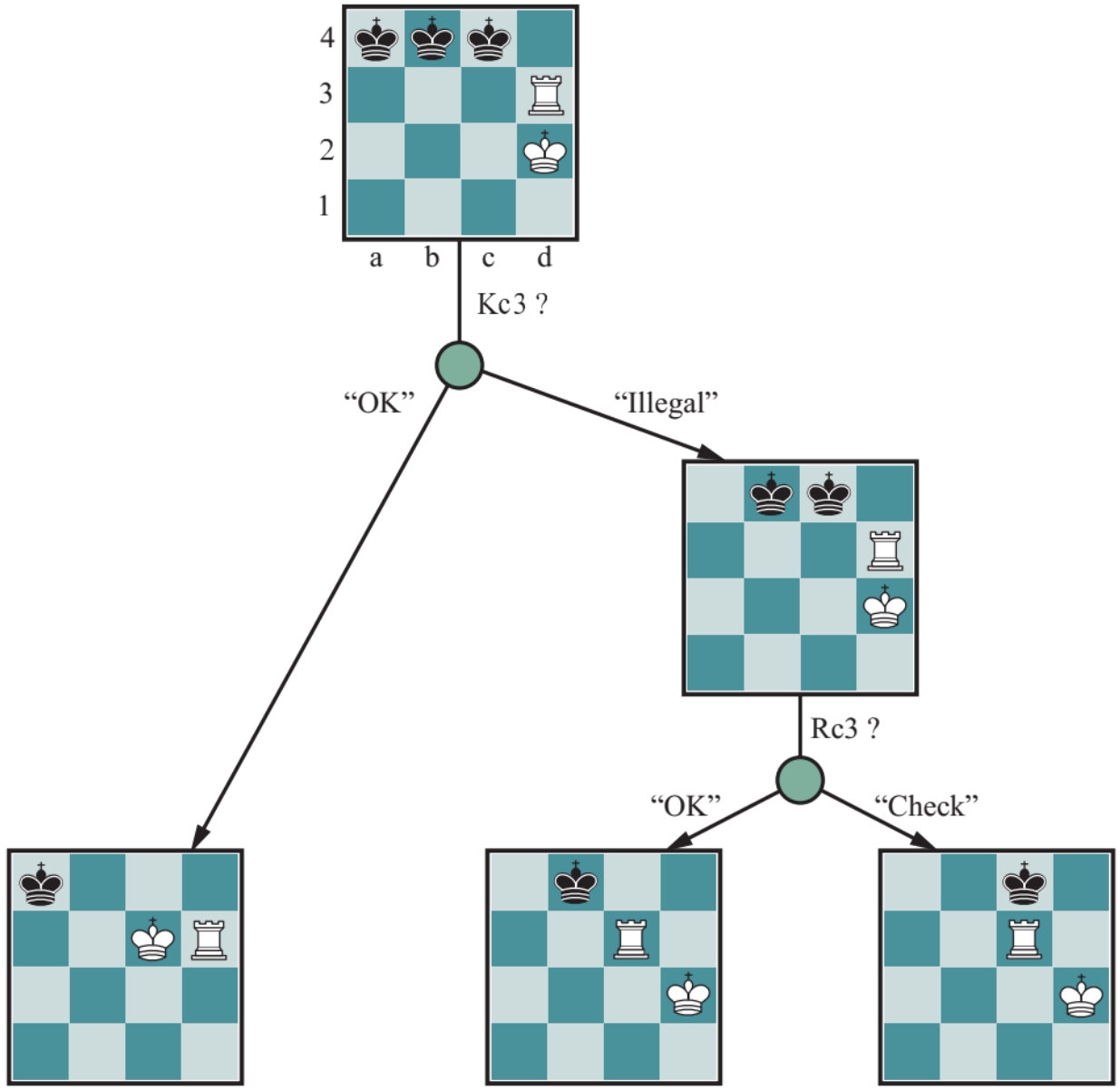
Problem: "Higher values to better postions" is not enough.



Fix: Evaluation function should be positive linear transformation of the expected utility.

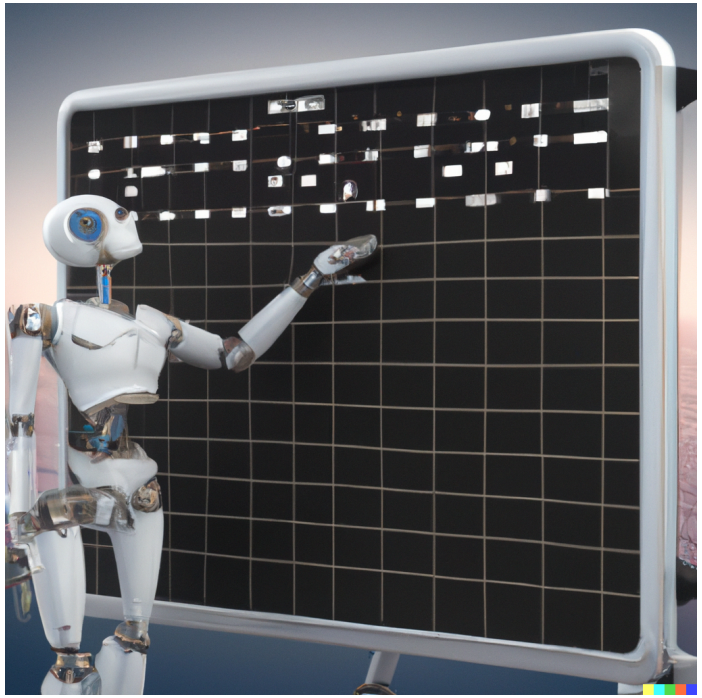
AND-OR search for partially observable games

- Belief states
- Guaranteed checkmate
- Probabalistic checkmate
- Accidental checkmate
- Confusing random moves



Lecture 6 - Constraint Satisfaction Problems

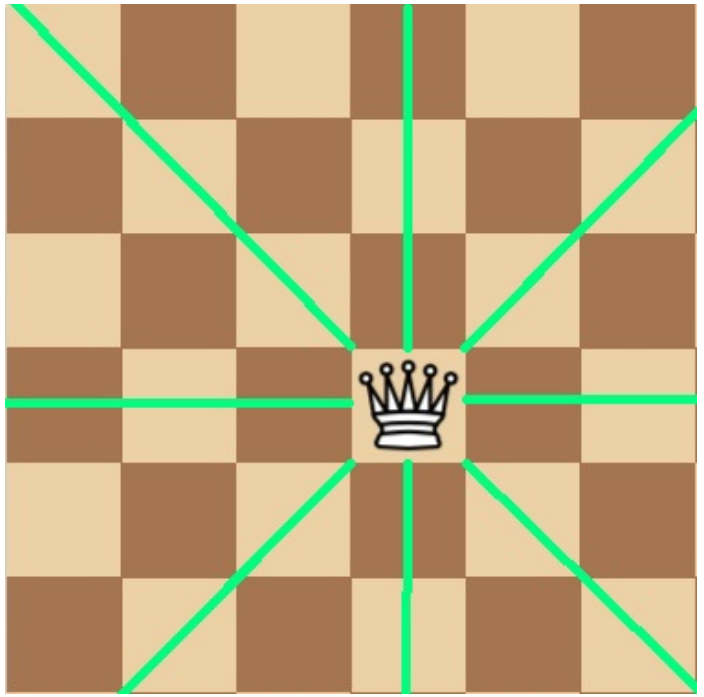
Gleb Sizov
Norwegian University of Science and Technology



What is Constraint Satisfaction Problem (CSP)?

Example: N-Queens problem

Place N queens on an NxN chess board with the **constraint** that they don’t attack each other.



CSP definition example: Map-coloring

Task: Colour this map with 3 colours so that adjacent regions have different colours

Variables: $\{WA, NT, Q, NSW, V, SA, T\}$

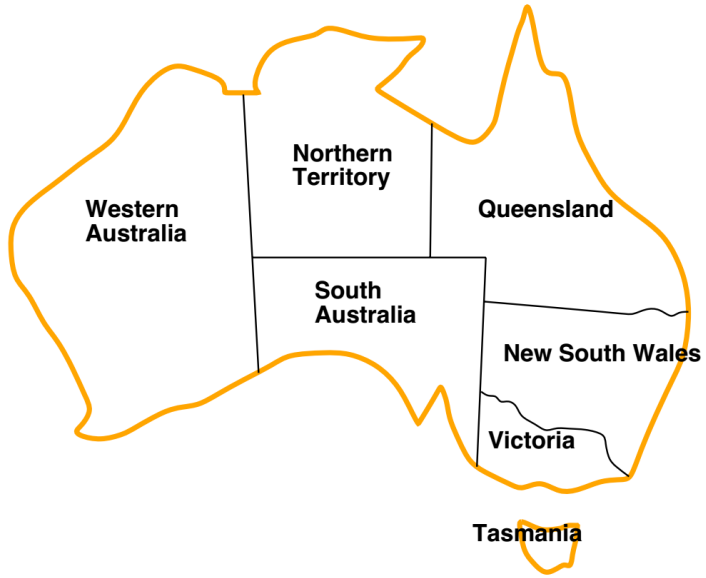
Domain: $\{red, green, blue\}$

Constraints: adjacent regions must have different colors

e.g. $WA \neq NT$

or $(WA, NT) \in$

$\{(red, green), (red, blue), (green, red), (green, blue)\}$



Constraint types

Unary constraints involve a single variable,

e.g., $SA \neq green$

Binary constraints involve pairs of variables,

e.g., $SA \neq WA$

Global constraints involve 3 or more variables,

e.g., Sudoku, $Alldiff$

Preferences (soft constraints),

e.g., red is better than green

Often includes cost for variable assignment

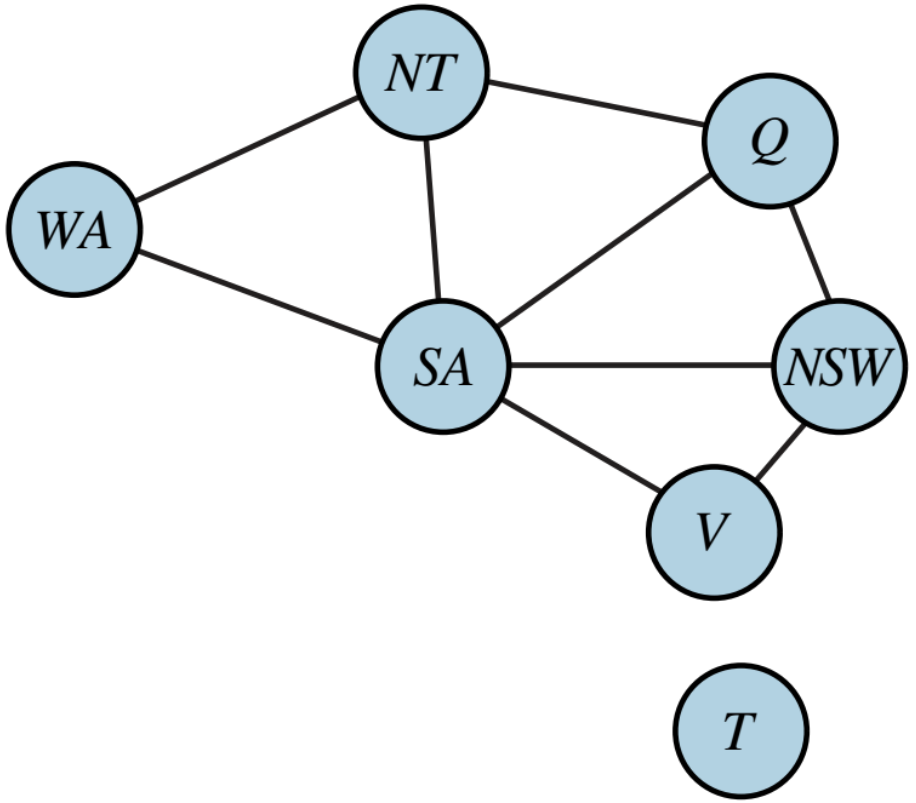
$\rightarrow$ constrained optimization problems

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

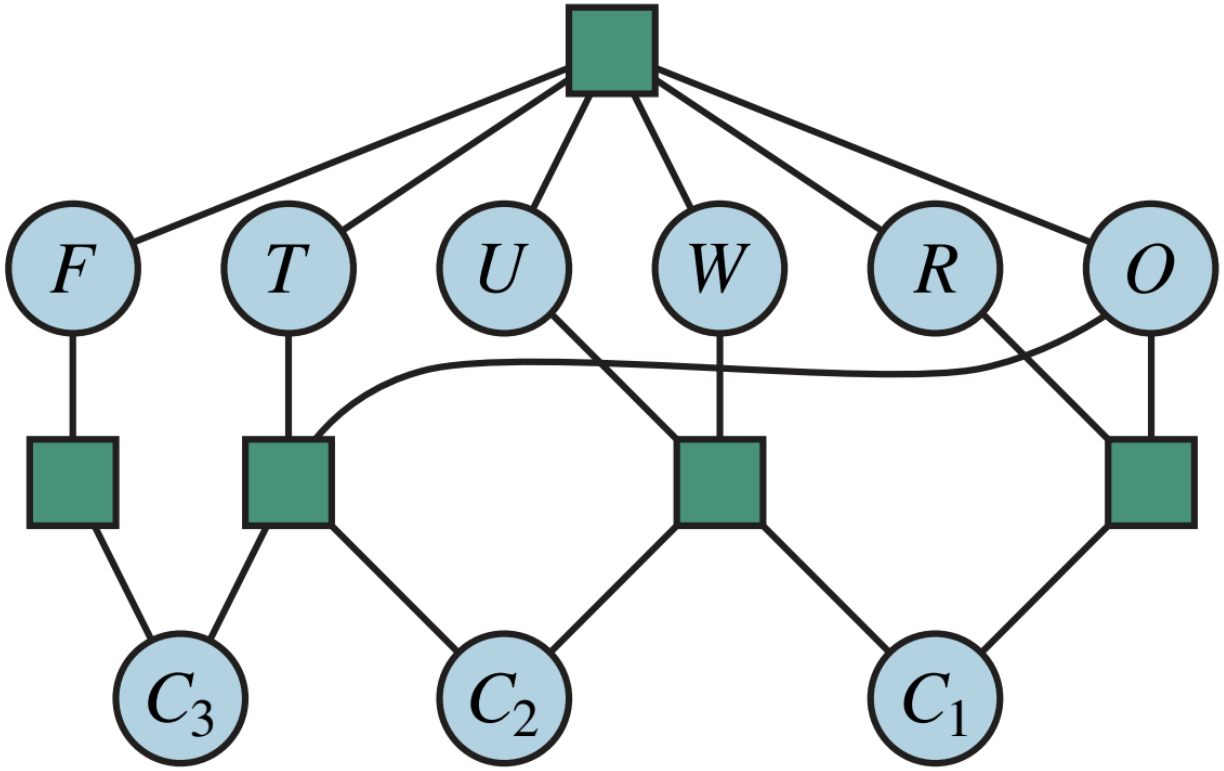
Constraint graph

Nodes - variables, arcs - constraints



Constraint hypergraph

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$



Approaches to solving CSPs

1. Inference - Constraint propagation
2. Incremental search - Backtracking
3. Search with inference, e.g., Backtracking with Forward Checking
4. Local Search

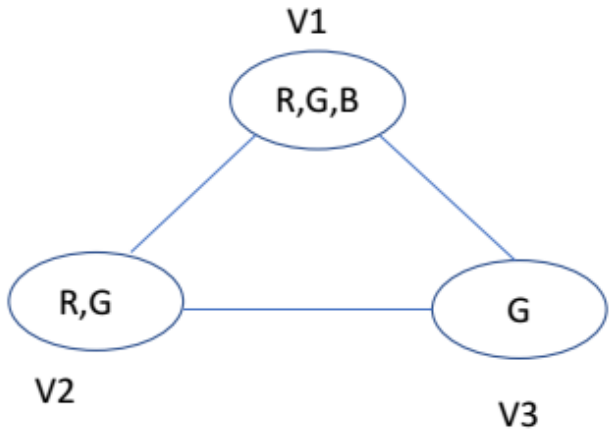
Arc consistency

A variable is **arc-consistent** if every value in its **domain** satisfies the variable's **binary constraints**.

An arc $X \to Y$ is consistent if
For every value in D_x , there exist a value in D_y that satisfies the binary constraint of the arc (X, Y).

Example:
Domain: $D(X) = D(Y) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Constraint: $Y = X^2$
Reduced domains: $R(X) = \{0, 1, 2, 3\}$, $R(Y) = \{0, 1, 4, 9\}$

AC-3 example - Step 5



Constraints:

DIFF-color(V1,V2)

DIFF-color(V1,V3)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted	
V1-V2	none	(V1={R,G,B})
V2-V1	none	(V2={R,G})
V1-V3	G	(V1={R,B})
V3-V1	none	(V3={G})
V2-V3	G	(V2={R})
V3-V2	none	
V2-V1	none	
V1-V2	R	(V1={B})
V2-V1	none	
V3-V1	none	

Path-consistency

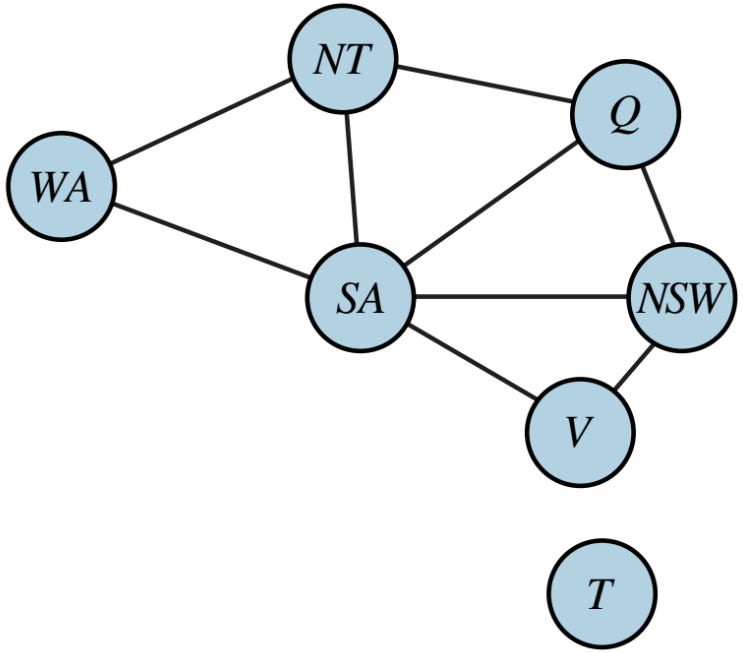
Australia coloring with two colors - arc consistency doesn't help.

Make $\{WA, SA\}$ **path-consistent** with respect to NT:

1. Enumerate assignments for WA and SA: $\{WA = red, SA = blue\}, \{WA = blue, SA = red\}$.
2. Remove assignments that are in conflict with NT.

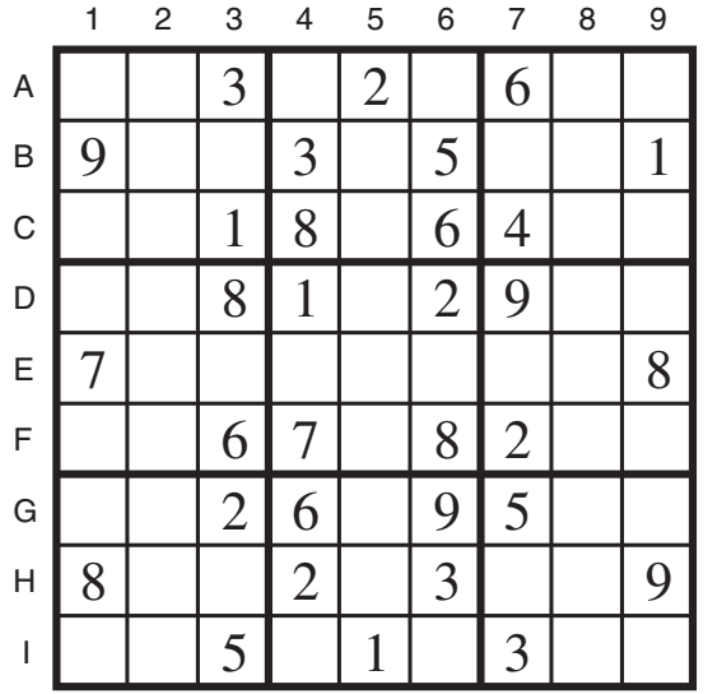
No vald assignments!

Higher-order consistency - **k-consistency**.



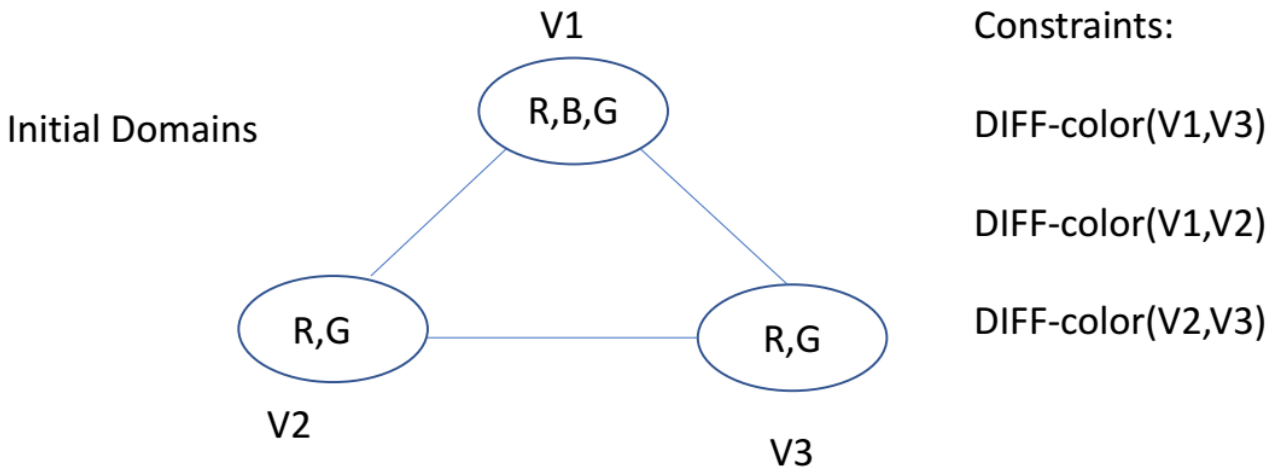
Global constraints

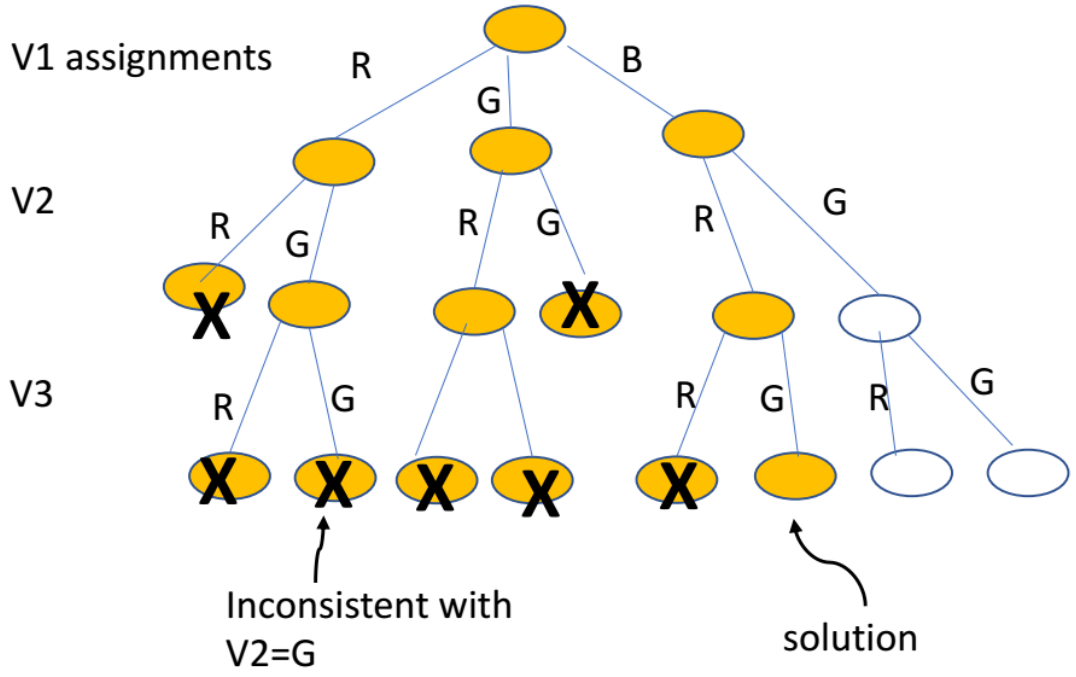
- $Alldiff$
- Resource constraint: $Atmost(10, P_1, P_2)$
- Bounds propagation - $Between(10, 40, P_1, P_2)$



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Backtracking example - Step 3

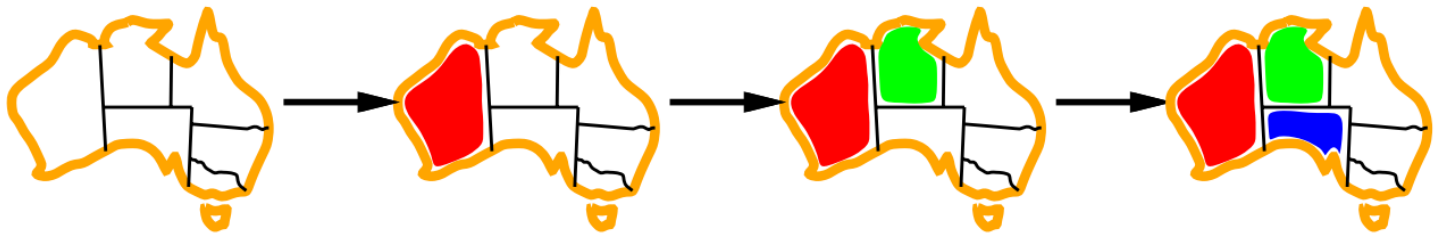




Minimum-remaining-values (MRV) heuristic

> Which variable should be assigned next?

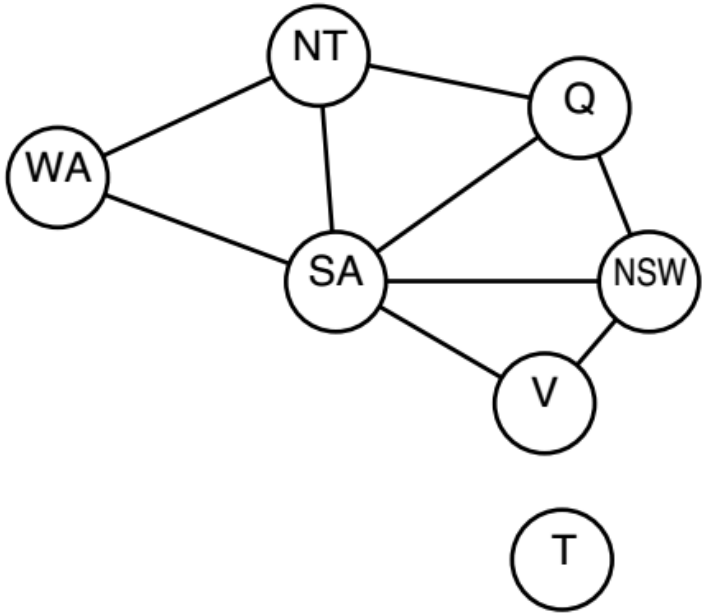
- Choose the variable with the fewest legal values.
- Also called "most constrained variable" or "fail first" heuristic.
- Objective: Detect immediately if variable has no legal values(left) or most likely to cause a failure soon.

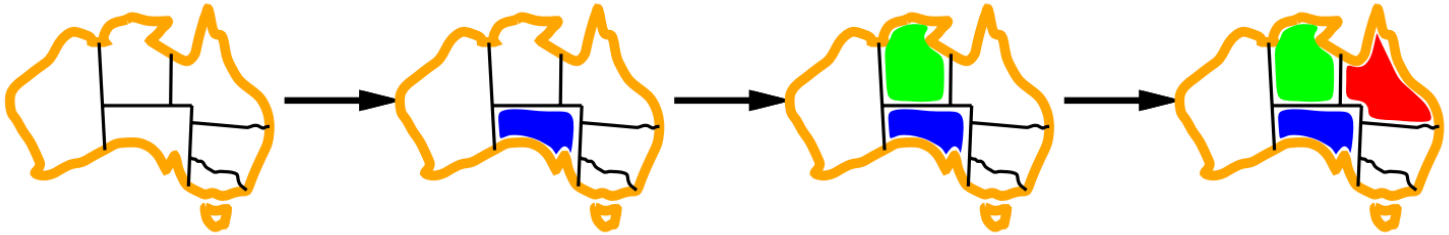


Degree heuristic

> Which variable should be assigned next?

- Heuristic: Choose the variable with the most constraints on remaining variables
- Objective: Reduce the future branching on future choices.
- Helps in the beginning, tie-breaker among MRV variables.

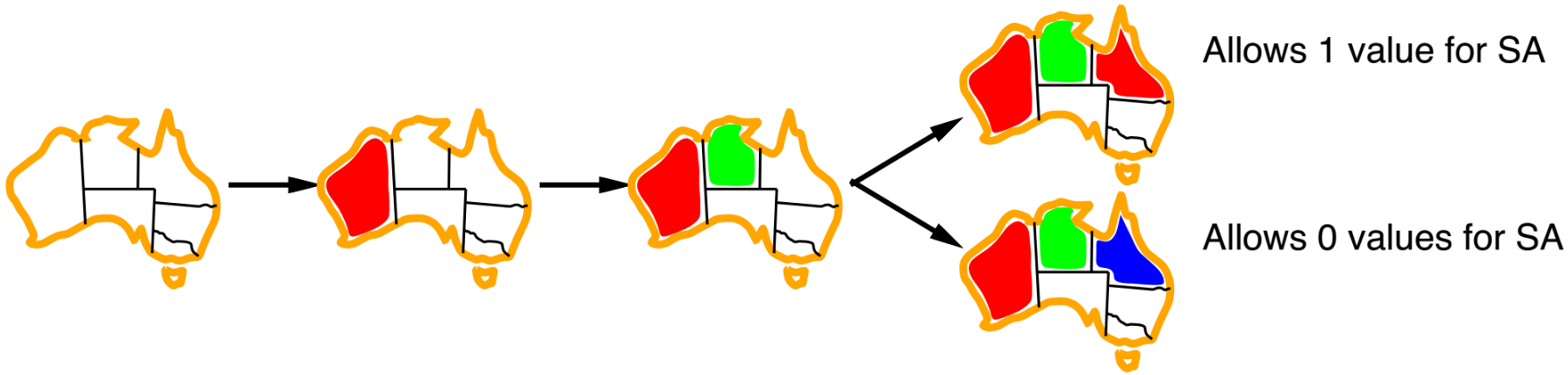




Least-constrained-value heuristic

> In what order should its values be tried?

- Heuristic: Given a variable, choose the value that rules out the fewest values in the remaining variables.
- Objective: Leave the maximum flexibility for subsequent variable assignments.



- Backtracking + MRV + degree + least-constrained-value can solve 1000-queens problem vs 25-queens without heuristics.

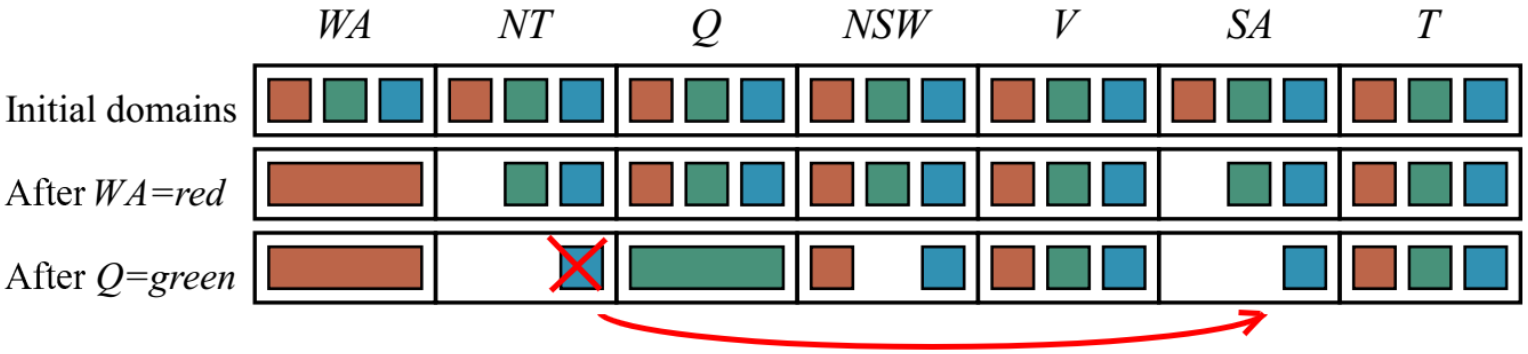
Improving backtracking using inference

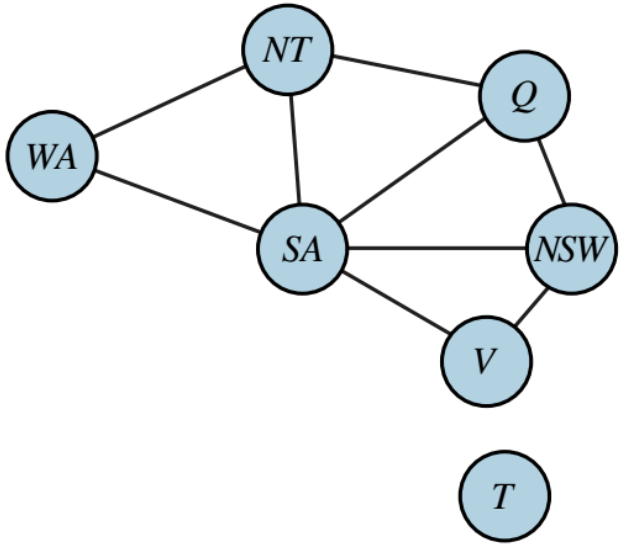
> Can we detect unpromissing nodes early?

- Use AC-3 before or during search - time consuming
- **Forward checking**
    i. Arc consistency between unassigned neighbour nodes and current node
    ii. Keep track of remaining legal values of unassigned variables.
    iii. Terminate when any variable has no legal values

Maintaining arc consistency (MAC)

- Inference with AC-3 starting with arcs to unassigned neighbour nodes.
- Similar to forward checking but with propagation.





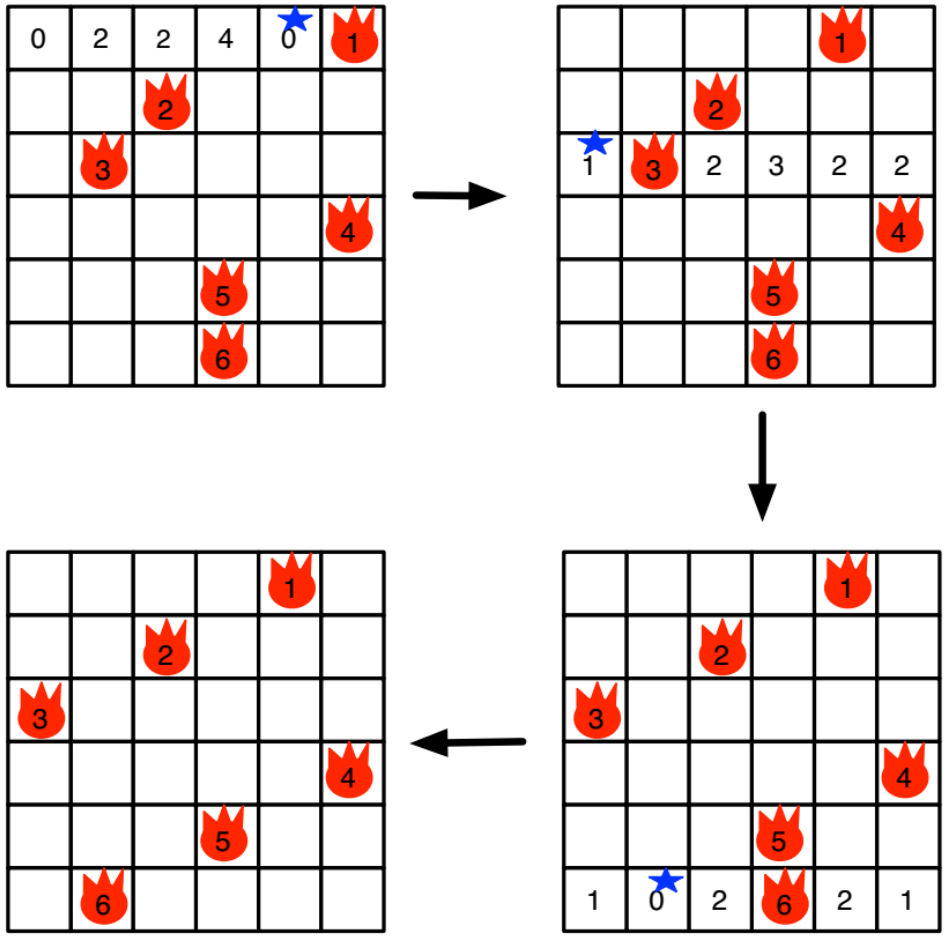
Local search for CSP

Steps:

1. Start with complete assignment, typically violating several constraints.
2. Randomly choose a conflicted variable.
3. Change its value so it brings us closer to a solution,

Min-conflicts heuristic - select a value with the minimum number of conflicts.

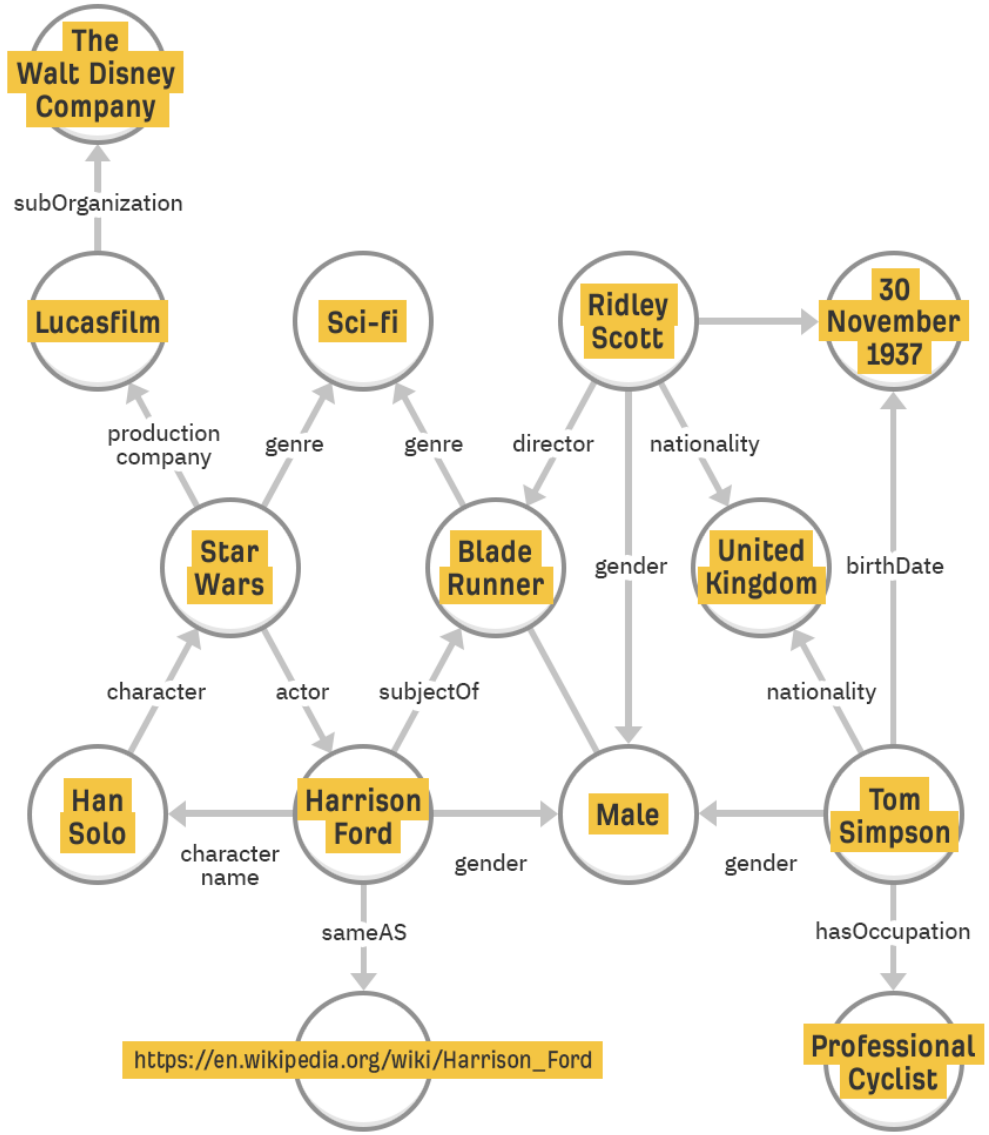
Animation

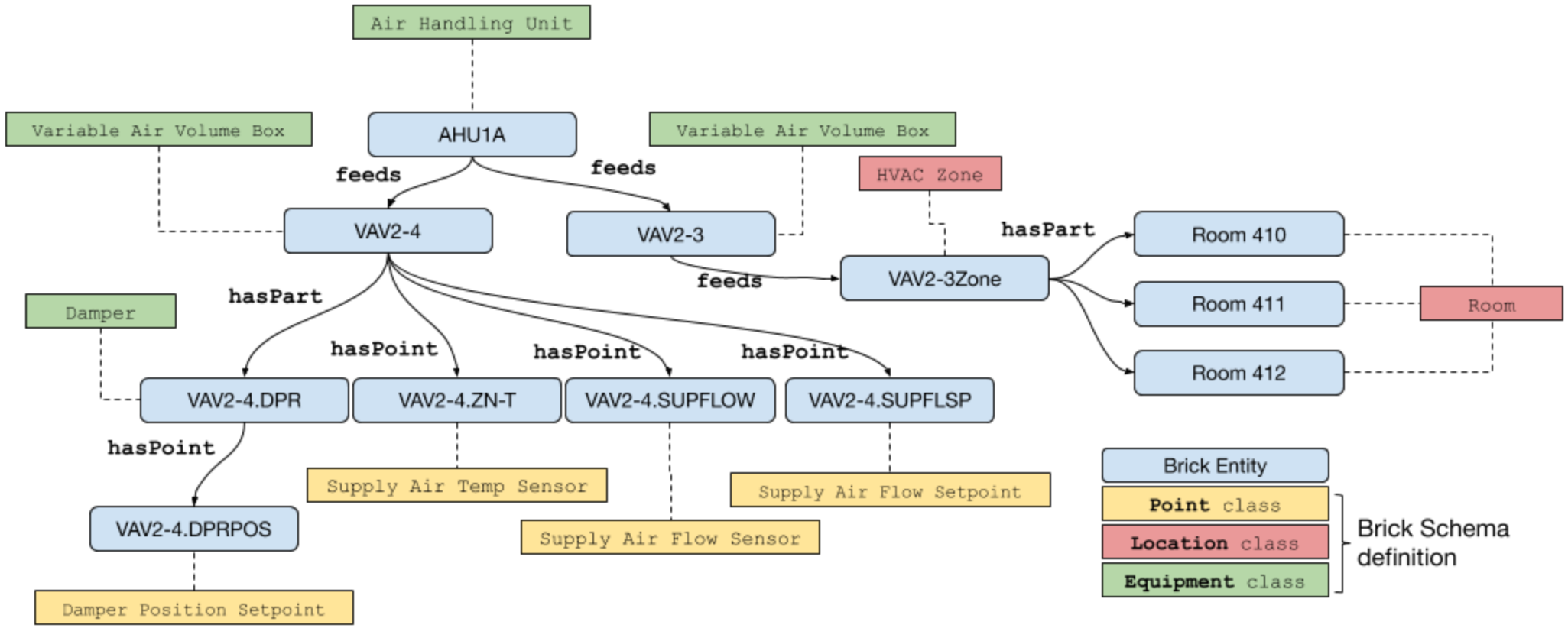


Lecture 10.1 - Knowledge Representation

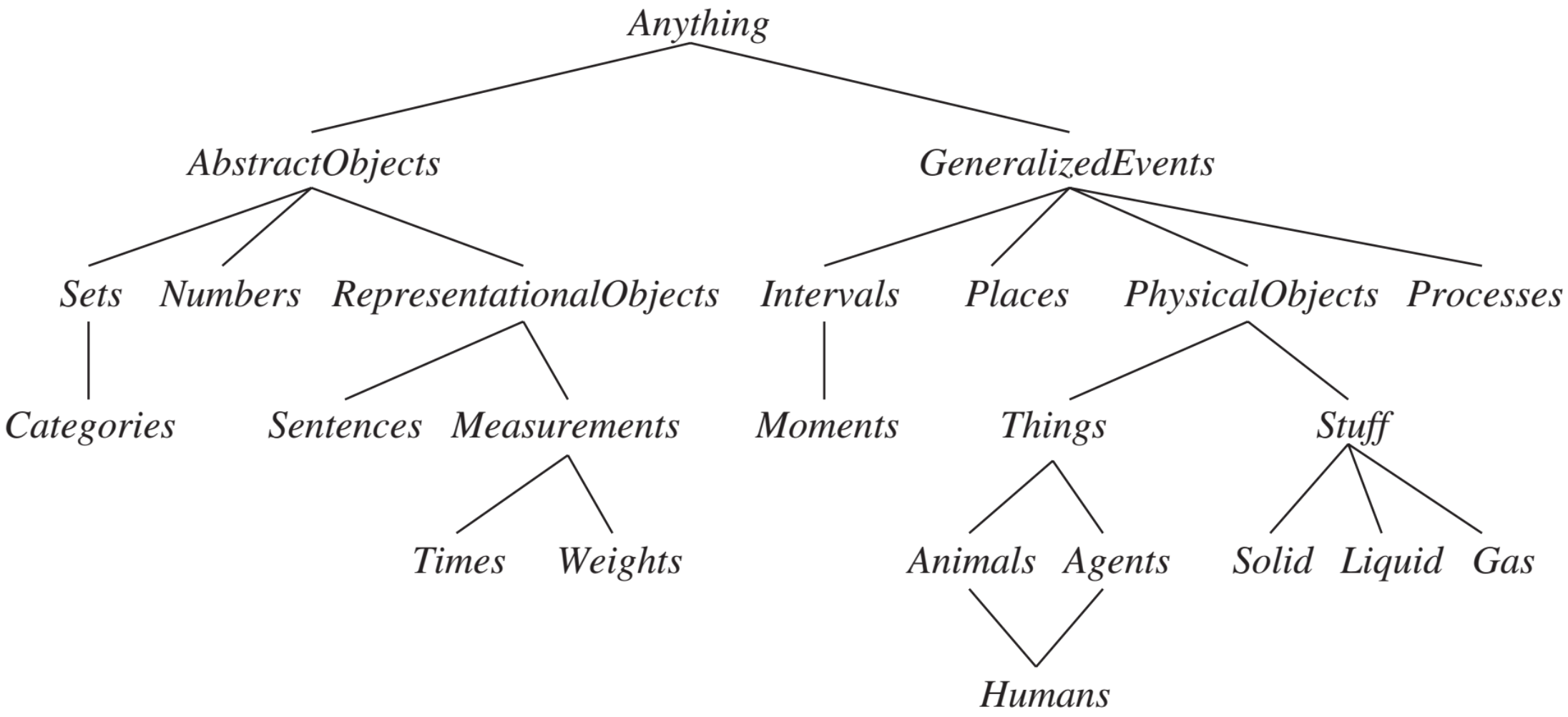
> How to represent diverse facts about the real world in a form that can be used to reason and solve problems?

Gleb Sizov
Norwegian University of Science and Technology




Air Handling Unit
Variable Air Volume Box
AHU1A
Variable Air Volume Box
feeds
feeds
HVAC Zone
VAV2-4
VAV2-3
hasPart
Room 410
feeds
VAV2-3Zone
Damper
hasPart
Room 411
Room
hasPoint
hasPoint
hasPoint
Room 412
VAV2-4.DPR
VAV2-4.ZN-T
VAV2-4.SUPFLOW
VAV2-4.SUPFLSP
hasPoint
Supply Air Temp Sensor
Supply Air Flow Setpoint
VAV2-4.DPRPOS
Supply Air Flow Sensor
Damper Position Setpoint
Brick Entity
Point class
Location class
Equipment class
Brick Schema definition

Upper ontology - general framework fo concepts



First order logic and categories

An object is a member of a category:

$BB_{12} \in Basketballs$

$Member(BB12, Basketballs)$

A category is a subclass of another category:

$Basketballs \subset Balls$

$Subcategory(Basketballs, Balls)$

All members of a category have some properties:

$(x \in Basketballs) \Rightarrow Spherical(x)$

All members of a category can be recognized by some properties:

$Orange(x) \wedge Round(x) \wedge Diameter(x) = 9.5" \wedge x \in Balls \Rightarrow x \in Basketballs$

Natural kinds

Some categories can be defined exactly:

$x \in Bachelors \Leftrightarrow Unmarried(x) \wedge x \in Adults \wedge x \in Males$

Natural kind categories have no exact defintion.

$x \in Typical(Tomatoes) \Rightarrow Red(x) \wedge Spherical(x)$

Composition

$PartOf(Bucharest, Romania)$

$PartOf(Romania, Europe)$

$PartOf(Europe, Earth)$

Transitive:

$PartOf(x, y) \wedge PartOf(y, z) \Rightarrow PartOf(x, z)$

Reflexive:

$PartOf(x, x)$

Event calculus

Fluents:

$$At(Shankar, Berkeley)$$

Events:

$$E_1 \in Flyings \wedge Flyer(E_1, Shankar) \wedge Origin(E_1, SF) \wedge Destination(E_1, DC)$$

Time points:

$$T(At(Shankar, Berkeley), t_1, t_2)$$

Example - effect of flying:

$$E = Flyings(a, here, there) \wedge Happens(E, t_1, t_2) \Rightarrow Terminates(E, At(a, here), t_1) \wedge Initiates(E, At(a, there), t_2)$$

Time

$$Interval\,(i) \Rightarrow Duration\,(i) = (Time\,(End\,(i)) - Time\,(Begin\,(i)))\,,$$
$$Time\,(Begin\,(AD1900)) = Seconds\,(0)\,.$$
$$Time\,(Begin\,(AD2001)) = Seconds\,(3187324800)\,.$$
$$Time\,(End\,(AD2001)) = Seconds\,(3218860800)\,.$$
$$Duration\,(AD2001) = Seconds\,(31536000)\,.$$

Time relations

$$
\begin{aligned}
Meet\,(i,j) &\Leftrightarrow End\,(i) = Begin\,(j)\\
Before\,(i,j) &\Leftrightarrow End\,(i) < Begin\,(j)\\
After\,(j,i) &\Leftrightarrow Before\,(i,j)\\
During\,(i,j) &\Leftrightarrow Begin\,(j) < Begin\,(i) < End\,(i) < End\,(j)\\
Overlap\,(i,j) &\Leftrightarrow Begin\,(i) < Begin\,(j) < End\,(i) < End\,(j)\\
Starts\,(i,j) &\Leftrightarrow Begin\,(i) = Begin\,(j)\\
Finishes\,(i,j) &\Leftrightarrow End\,(i) = End\,(j)\\
Equals\,(i,j) &\Leftrightarrow Begin\,(i) = Begin\,(j) \wedge End\,(i) = End\,(j)
\end{aligned}
$$

Mental Objects

Knowledge *about* beliefs and deduction.

Useful for controlling inference.

Q: Is the prime minister sitting down right now?

Propositional attitudes:

$$Knows(Lois, CanFly(Superman))$$

Referential transparency - terms used don't matter:

$$(Superman = Clark) \wedge Knows(Lois, CanFly(Superman)) \models Knows(Lois, CanFly(Clark))$$

We need **referential opacity** - terms used do matter.

Modal logic

Modal operators:

$\boldsymbol{K}_A P$, means A knows P

$\boldsymbol{B}_A P$, means A has a belief P

Lois knows that Clark knows whether Superman's secret identity is Clark:

$\boldsymbol{K}_{Lois}[\boldsymbol{K}_{Clark} Identity(Superman, Clark) \vee$
$\boldsymbol{K}_{Clark} \neg Identity(Superman, Clark)]$

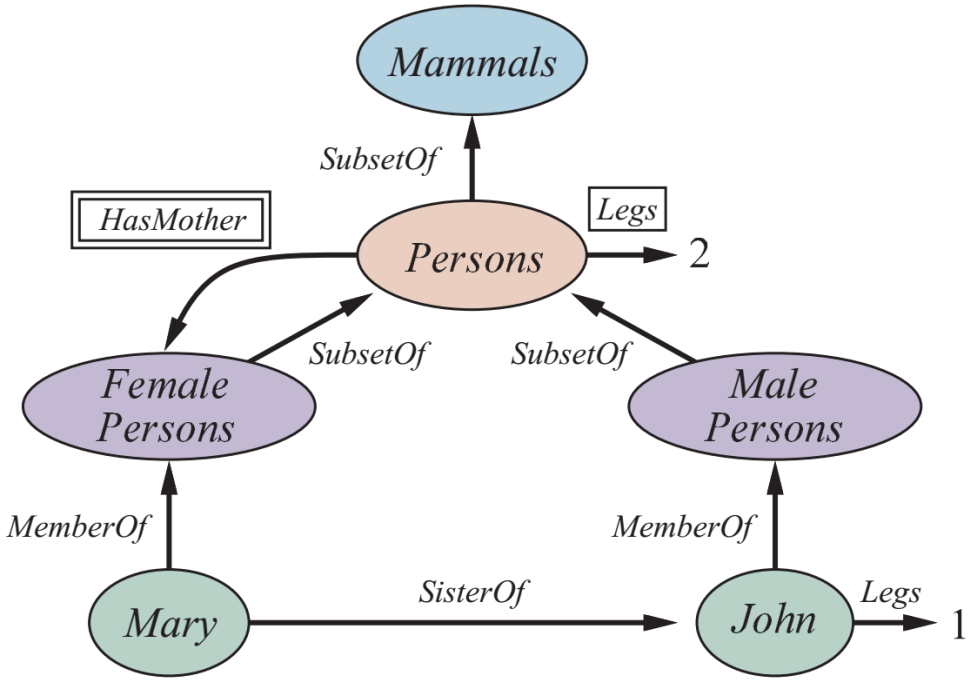
Some axioms:

1. $(\boldsymbol{K}_a P \wedge \boldsymbol{K}_a(P \Rightarrow Q)) \Rightarrow \boldsymbol{K}_a Q$
2. $\boldsymbol{K}_a P \Rightarrow P$
3. $\boldsymbol{K}_a P \Rightarrow \boldsymbol{K}_a(\boldsymbol{K}_a P)$

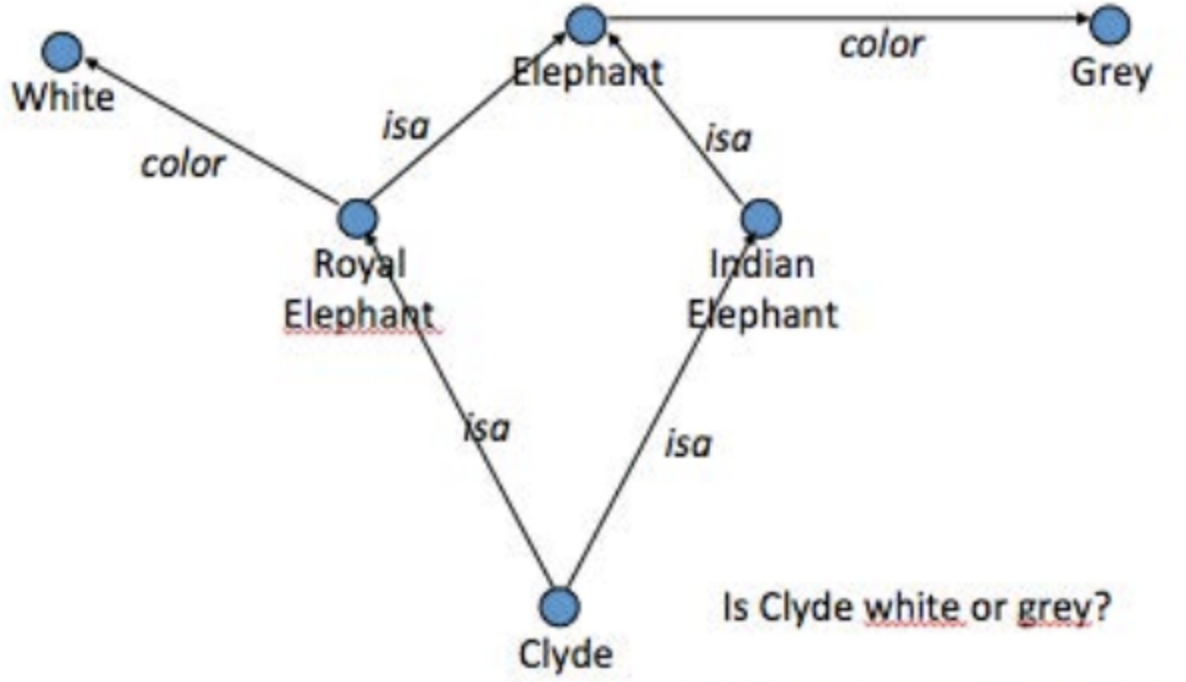
Issue: **Logical omniscience** - knowing all consequences.

Semantic networks

1. Semantic network is a type of logic.
2. Focus on connections between pieces of knowledge.
3. Efficient inference algorithm through inheritance.
4. Ability to represent **default values** for categories and allow **overrides**



Semantic networks - Multiple inheritance



Description logics (DL)

Main aspects:

1. Make it easier to describe definitions and properties of categories
2. Add formal (logic-based) semantics to frames and semantic networks.
3. Tractability of inference.

Example:
Men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments.

$$And(Man, AtLeast(3, Son), AtMost(2, Daughter),$$
$$All(Son, And(Unemployed, Married, All(Spouse, Doctor))),$$
$$All(Daughter, And(Professor, Fills(Department, Physics, Math)))).$$

Reasoning with default information

FOL is **monotonic**:
If $KB \models \alpha$ then $KB \wedge \beta \models \alpha$

Commonsense reasoning is **nonmonotonic**, e.g.

1. Inheritance in semantic networks with default values
2. Closed-world assumption: if α not in KB then $KB \models \neg\alpha$, but $KB \wedge \alpha \models \alpha$

Circumscription

> More powerful and precise version of the closed-world assumption.

$Bird(x) \land \neg Abnormal_1(x) \Rightarrow Flies(x)$

$Abnormal_1$ is to be **circumscribed**
Assume $\neg Abnormal_1(x)$ unless $Abnormal_1(x)$

Example:
$Tweety$ flies until we asser $Abnormal_1(Tweety)$.

Default logic

Use **default rules** for nonmonotonic conclusions.

Example:

$Bird(x) : Flies(x)/Flies(x)$

Meaning: if $Bird(x)$ is true and $Flies(x)$ is consistent with KB then $Flies(x)$.

Default rule components: prerequisite (P), justification (J), conclusion (C)

$P : J_1 ....., J_n/C$

If P and $J_1 ....., J_n$ cannot be proven false, then the conclusion can be drawn.

Truth maintainance systems (TMS)

1. KB contains P
2. $Tell(KB, \neg P)$
3. To avoid contradiction we must $Retract(KB, P)$

What about Q inferred from P, e.g. $P \Rightarrow Q$?
Q might also have other justifications, e.g. $R \Rightarrow Q$

TMS revises beliefs to avoid contradictions.

Lecture 10.2 - The Ethics of AI

Gleb Sizov
Norwegian University of Science and Technology

> Given that AI is a powerful technology, we have a moral obligation to use it well, to promote the positive aspects and avoid or mitigate the negative ones

Positive side of AI

1. AI in crop management and food production help feed the world
2. AI in drug helps to cure diseases
3. Drive assistance helps make driving safer
4. Optimization of business processes increases wealth
5. Automation for tedious and dangerous tasks
6. AI-based assistance for people with disabilities
7. Smart grid and buildings save energy and infrastructure
8. Machine translation helps people to communicate

   ...

Principles of Robotics

Ensure safety	Establish accountability
Ensure fairness	Uphold human rights and values
Respect privacy	Reflect diversity/inclusion
Promote collaboration	Avoid concentration of power
Provide transparency	Acknowledge legal/policy implications
Limit harmful uses of AI	Contemplate implications for employment

UK’s Engineering and Physical Sciences Research Council, 2010

Lethal autonomous weapons - Issues

1. Discriminating between combatants and non-combatants
2. Judgment of attack necessity
3. Assesment of potential collateral damage
4. Reliability
5. Cybersecurity
6. Scalable weapons of mass destruction
7. Attacker advantage

Lethal autonomous weapons - Regulations, treaties

- 2014 - Convention on Certain Conventional Weapons (CCW)
  Ban of lethal autonomous weapons
  Supported by 30 nations, opposed by Israel, Russia, US
- 2015 - **Compaign to Stop Killer Robots**
  140 NGOs in over 60 countries
  4000 AI researchers and 22000 others

> AI is a **dual use** technology

Surveillance

1976 - Joseph Weizenbaum warned that automated speech recognition could lead to widespread wiretapping, and hence to a loss of civil liberties

2018 - 350 million surveillance cameras in China and 70 million in US.

2019 - China's social credit system puts emphasis on use of big data and AI

Privacy

1. Data collectors have a moral and legal responsibility to be good stewards of the data they hold.
2. Balance between privancy rights and sociaety gains from sharing data, e.g. stop terrorists and cure diseases.

Regulations:

US: Privacy of medical and student records:

- Health Insurance Portability and Accountability Act (HIPAA)
- Family Educational Rights and Privacy Act (FERPA)

EU: General Data Protection Regulation (GDPR)

Privacy protection methods

1. De-identification, e.g. remove field, can be re-identified
2. Generalizing fields, e.g. 20-30 years old
3. K-anonimity - indistinguishable form at least k - 1 records
4. Aggregate quering - rules for minimum number of items
5. Differential privacy - add noise to query results
6. Federated learning - share model parameters, not data

Security

Market for ML cybersecurity $100 billion by 2021

- *Attackers* use AI to automate **cyberscurity** attacks.
- *Defenders* use AI to detect anomlaous network traffic and fraud transactions

Designing secure AI systems:

1. High compelxity, autnonomy, dependance on data
2. Not a well established field

Fairness and bias

> Designers of machine learning systems have a moral responsibility to ensure that their systems are in fact fair.

Avoid discrimination by algorithms, bias in data.

Domains:

- Loan approaval
- Education
- Employment
- Housing

Fairness

1. Individual fairness - similar individuals are treated similarly
2. Group fairness - two classes are treated similarly
3. Fainess through unawareness - e.g. remove gender and race fields
4. Equal outcome - each demographic class get the same result
5. Equal opportunity - individuals are treated according to their true ability, regardless of the class.
6. Equal impact - individuals with similar ability should have the same expected utility, regardless of the class

Fairness and bias issues

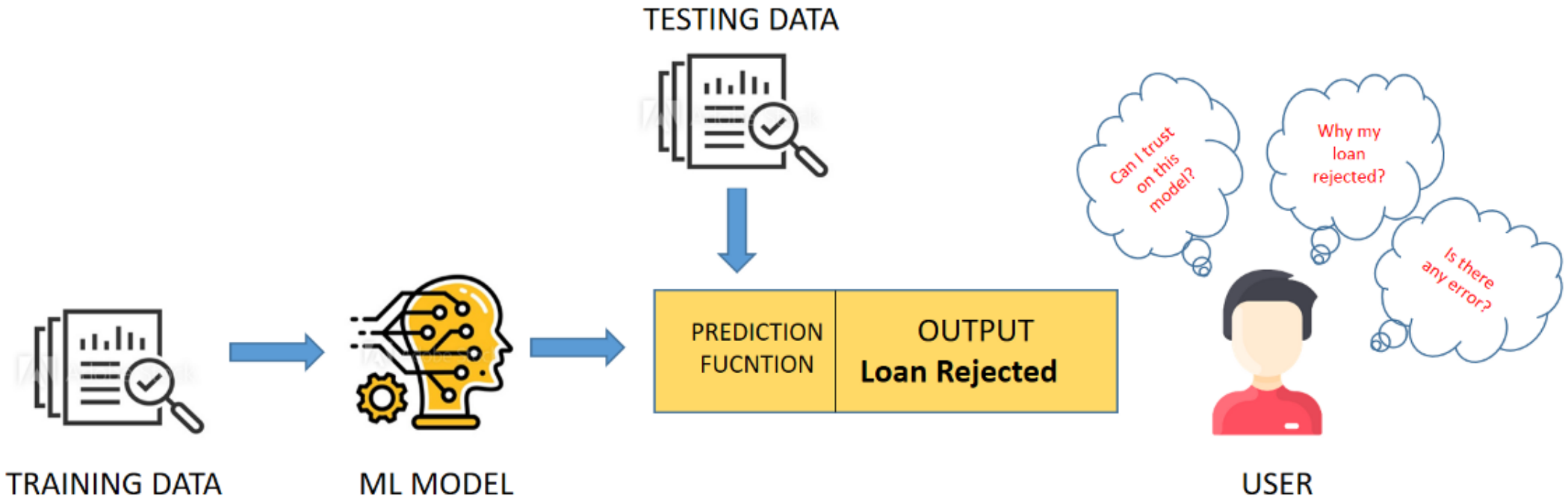
1. Biased data generated by biased sociatal process.
2. AI is used to *justify* bias.
3. Finding the right (fair) objective, e.g. exisitng skills vs learning on the job.
4. Decide which classes to protect, e.g. race, color, religion, sec, ddisability, familyt status etc.
5. Samples size disparity
6. Bias in software development process

Defences against bias

1. Understand the limits of your data, provide model data sheets
2. Diversity among engineers
3. De-bias your data, e.g. over-sampling.
4. ML algorithms that more resistant to bias.
5. Monitor metrics for different groups and fix bias issues.

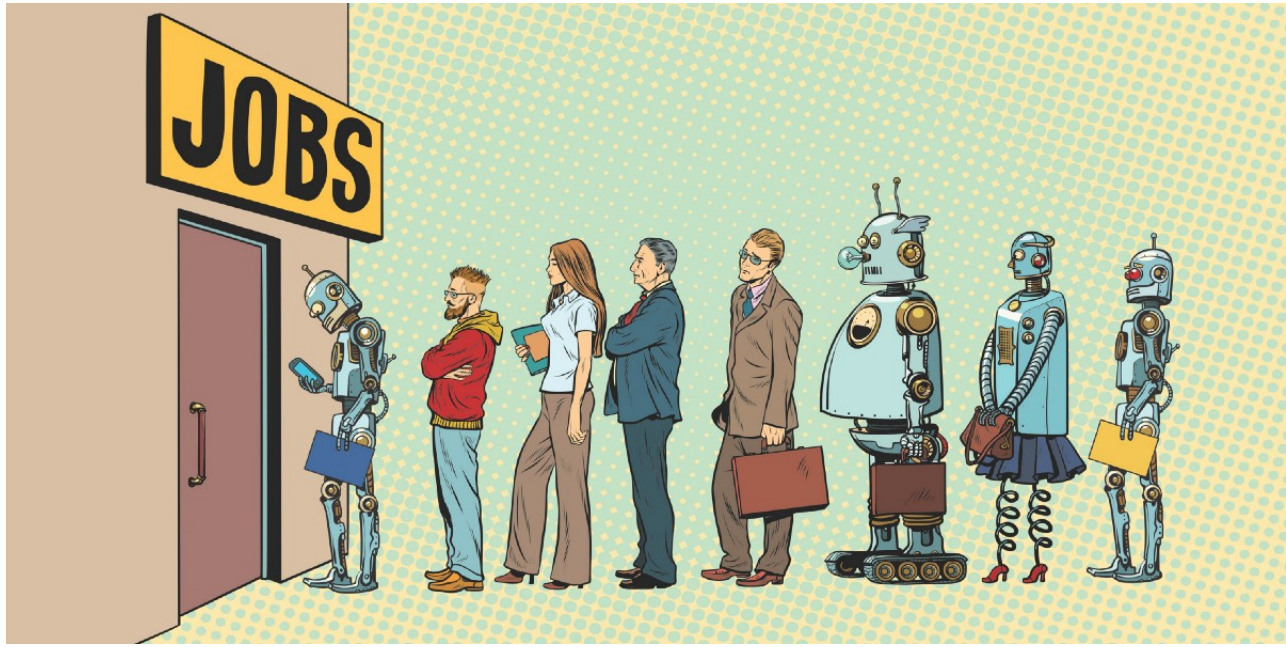
Trust and transperancy

1. Verification and validation methodology
2. Certification of AI systems
3. Transparency - explainable AI



The future of work

1. Technological unemployment
2. Buisness process automation
3. Income inequality

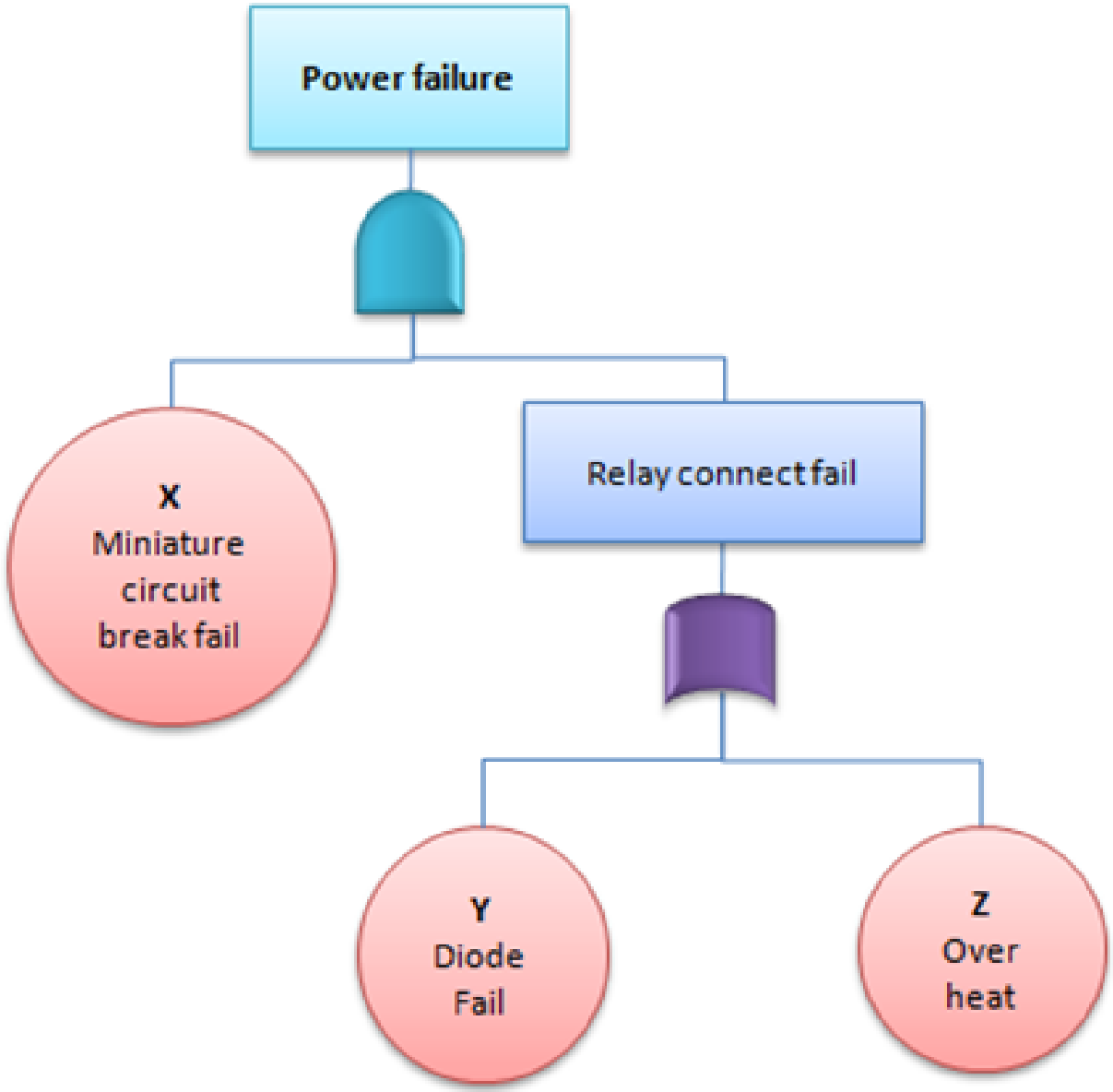


AI Safety

Safety-critical applications:
Driving, robotics, construction, medicine

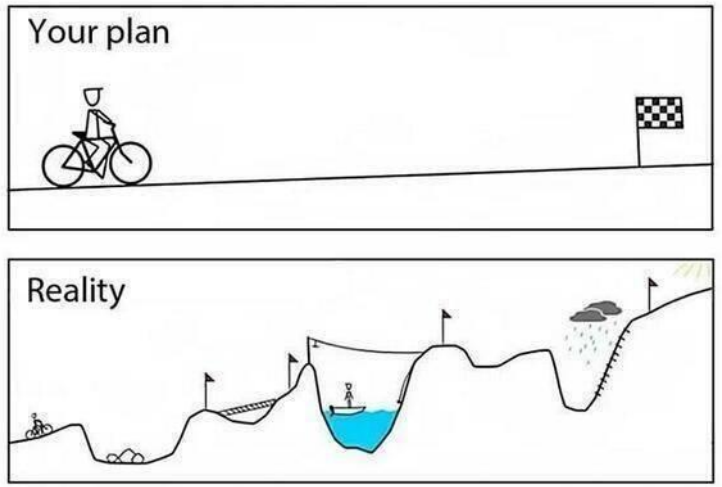
Safety engineering

1. Failure modes (FMEA) - analyze what can go wrong
2. Falut tree analysis
3. Software engineering practices
4. Utility function analysis - side effects, value alignment



Lecture 11 - Automated Planning

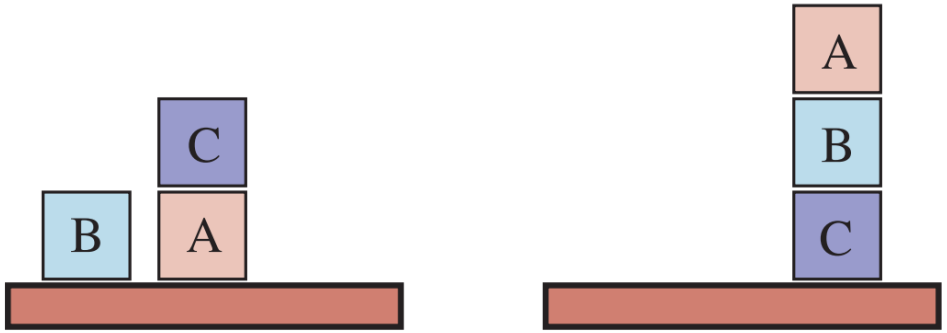
Gleb Sizov
Norwegian University of Science and Technology



PDDL Example - Blocks world

PDDL description:

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table))$
$Goal(On(A, B) \land On(B, C))$
$Action(Move(b, x, y),$
PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$(b \neq x) \land (b \neq y) \land (x \neq y),$
EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$
$Action(MoveToTable(b, x),$
PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land Block(x),$
EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$



Start State Goal State

Solution:

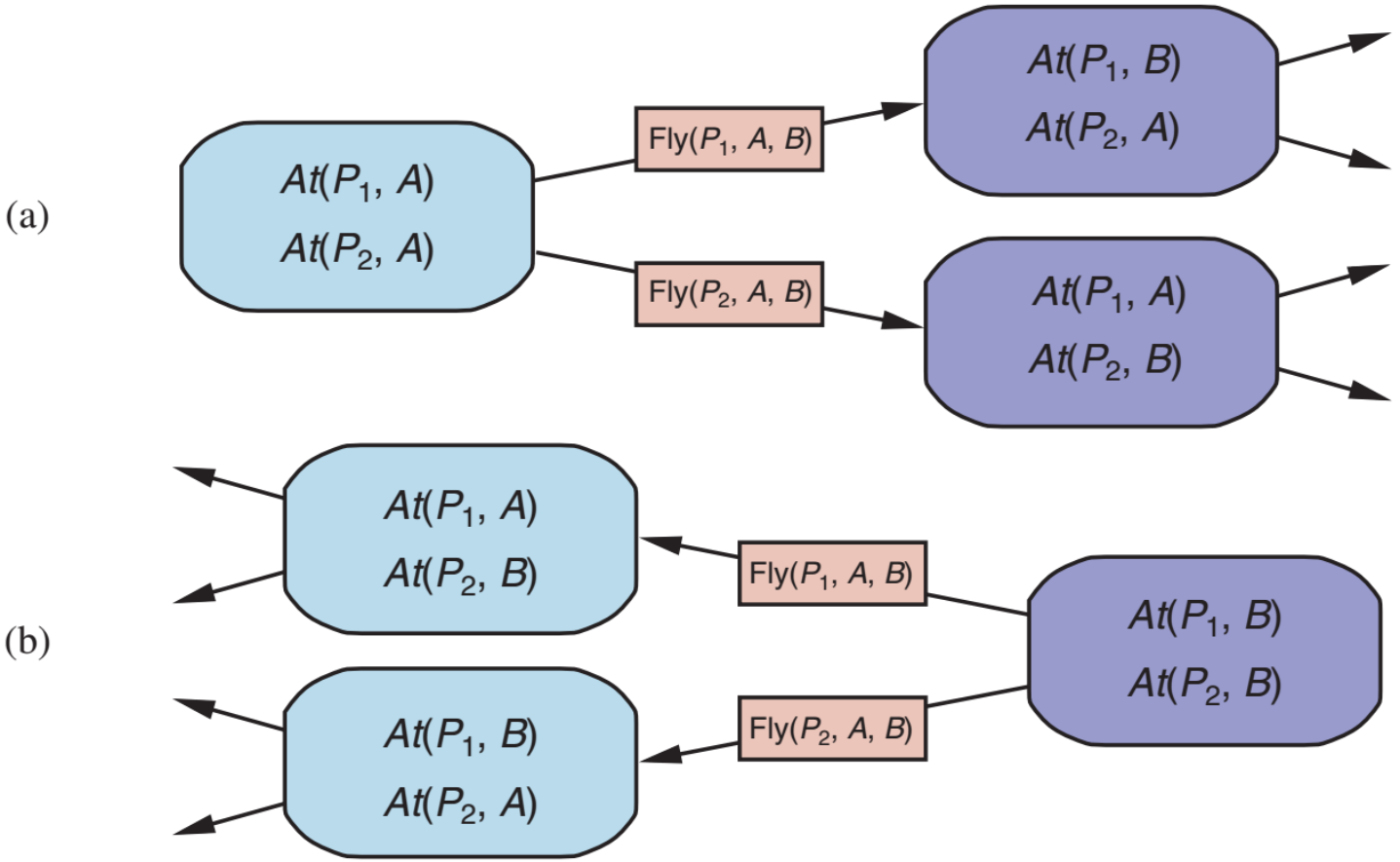
$[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$

Algorithms for Classical Planning

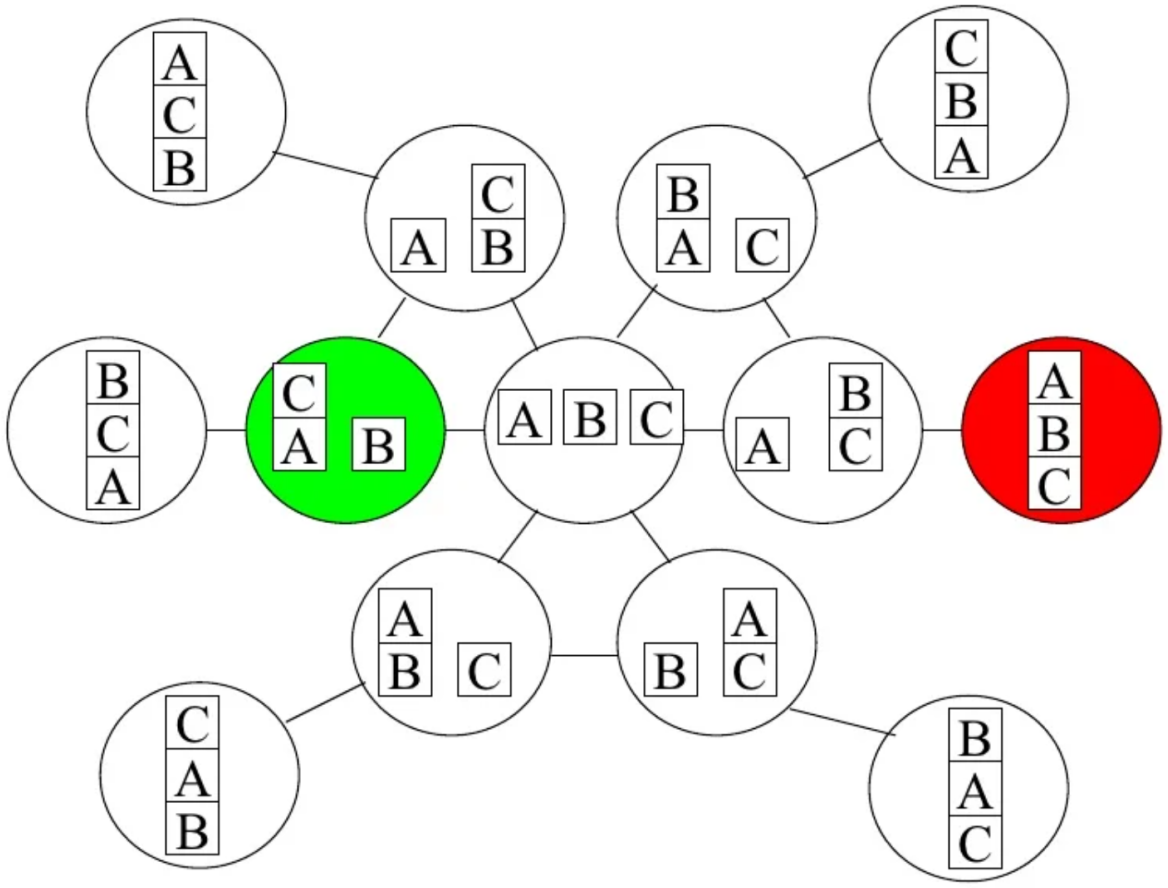
a. Forward (progression) search

b. Backward (regression) search

c. Logical inference



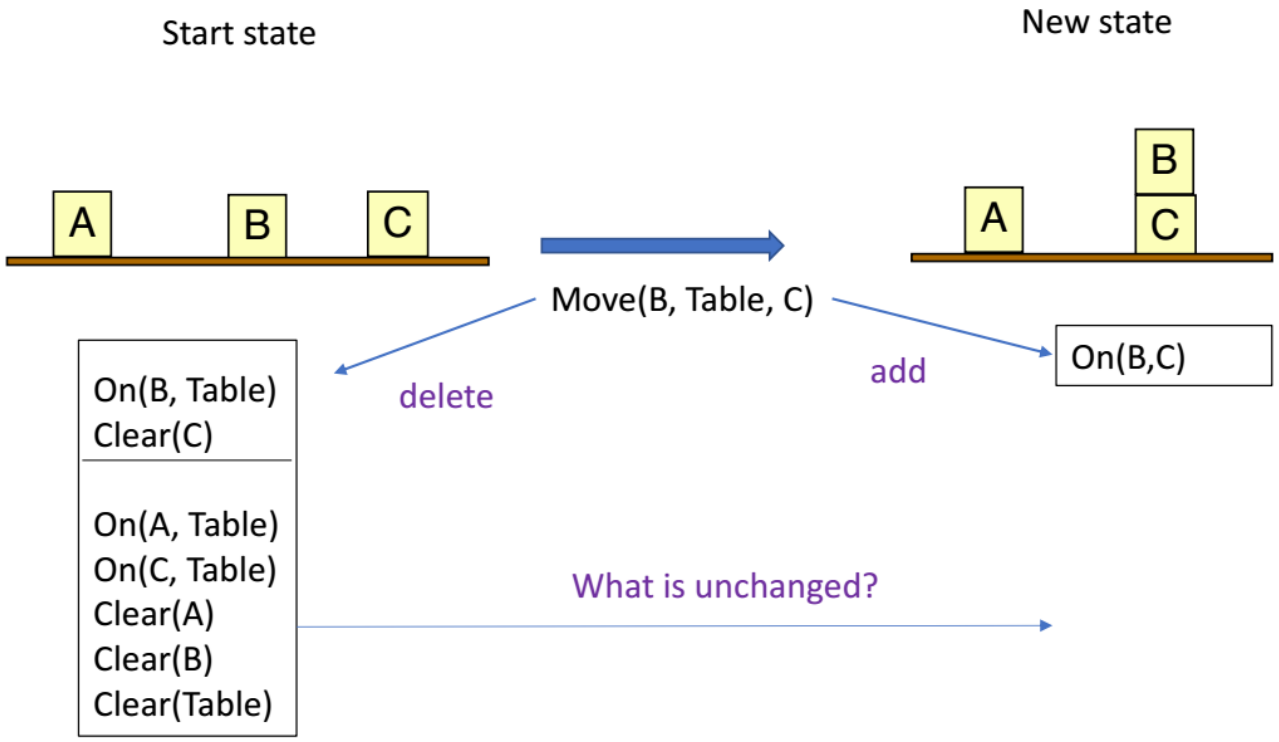
Planning search space



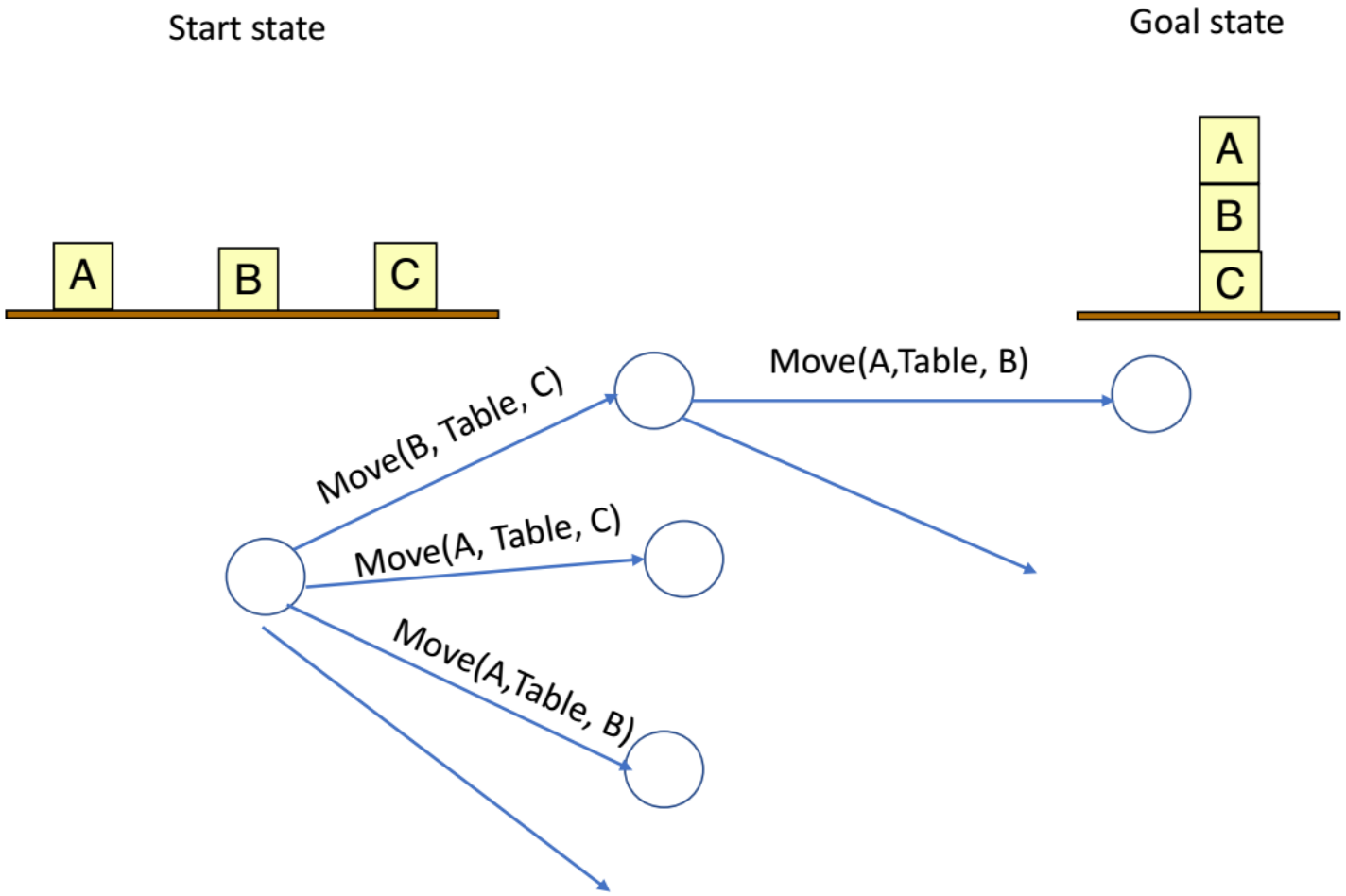
(Michael Moll)

Forward search example - Step 1

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)

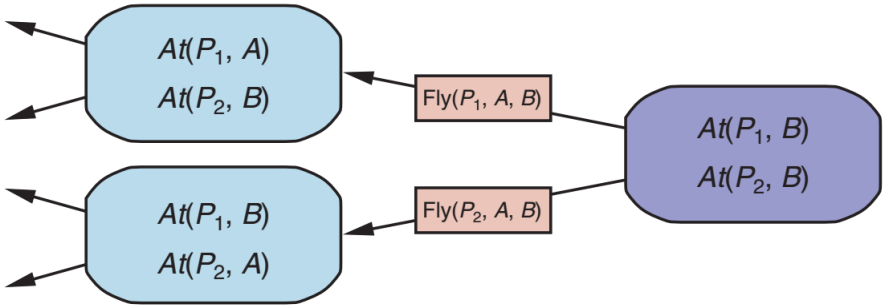


Forward search example - Branching



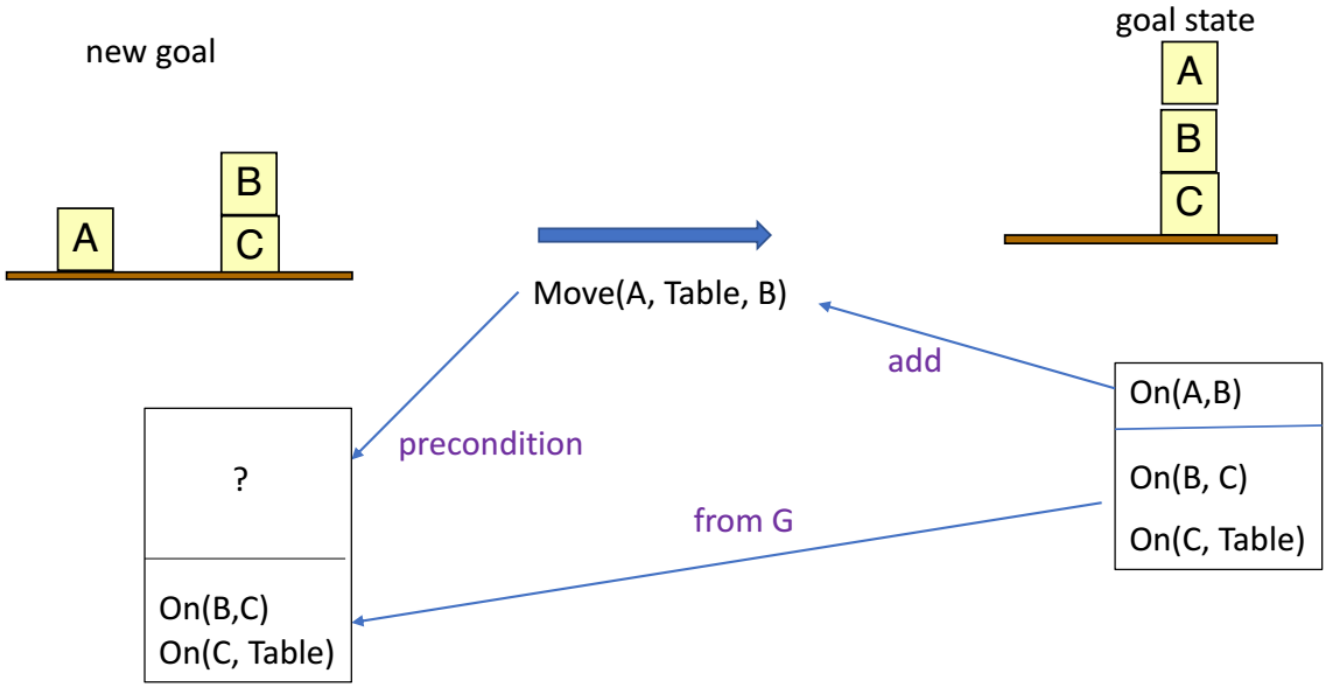
Backward (regression) search algorithm

1. Pick a **relevant action** that satisfies (some of) goal propositions.
2. Make a new goal by applying an **action backwards**:
    - Removing the goal conditions satisfied by the action
    - Adding the preconditions of this action
    - Keeping any unsolved goal propositions
3. Repeat until the goal set is satisfied by the start state

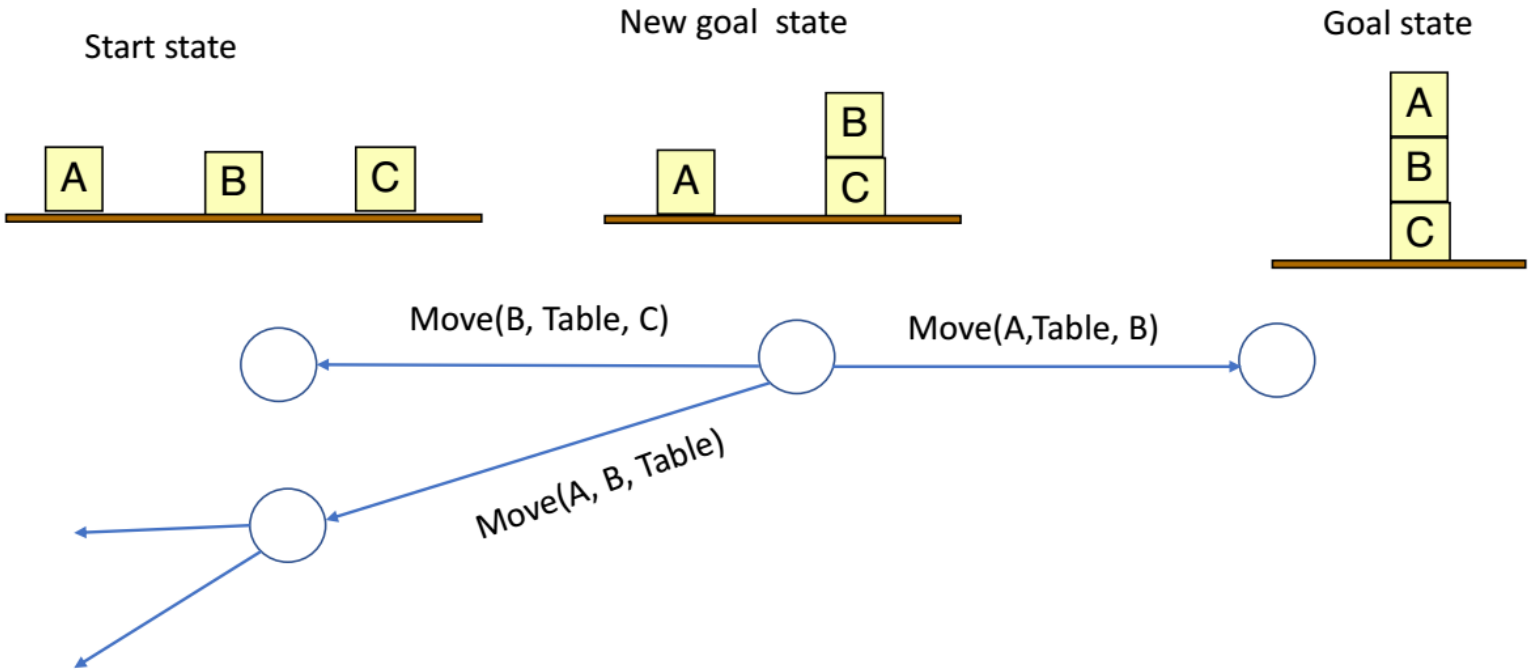


Backward search example - Step 2

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)



Backward search example - Branching

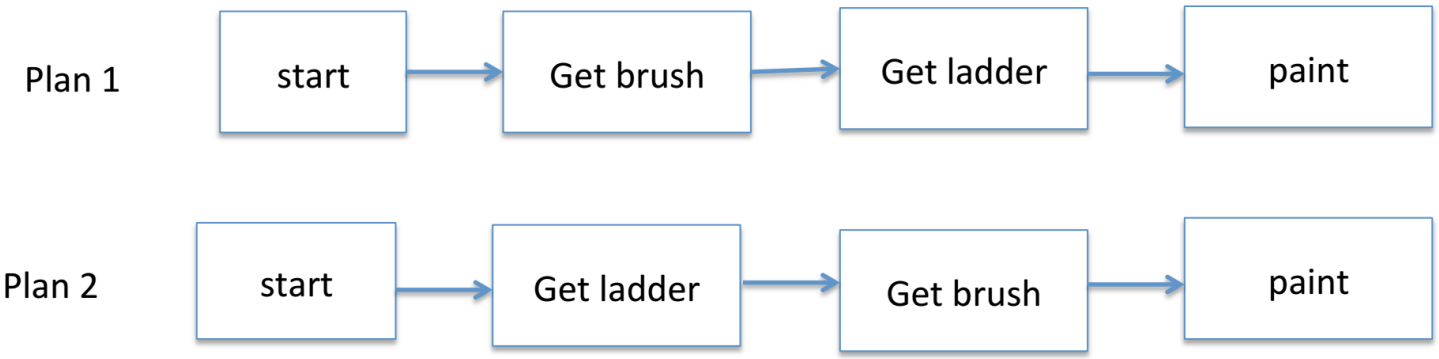


Other classical planning approaches

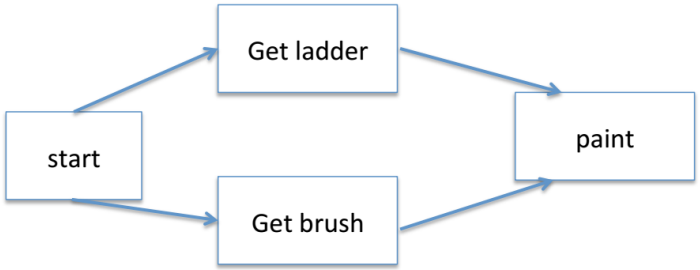
1. Graphplan - encodes precondition and effect constraints in a **planning graph**
2. Situation calculus - uses first-order logic and first-order solvers
3. Constrain sastifaction problem - similar to SAT but more compact

Total vs partial order plan

Total order: Plan is always a strict sequence of actions

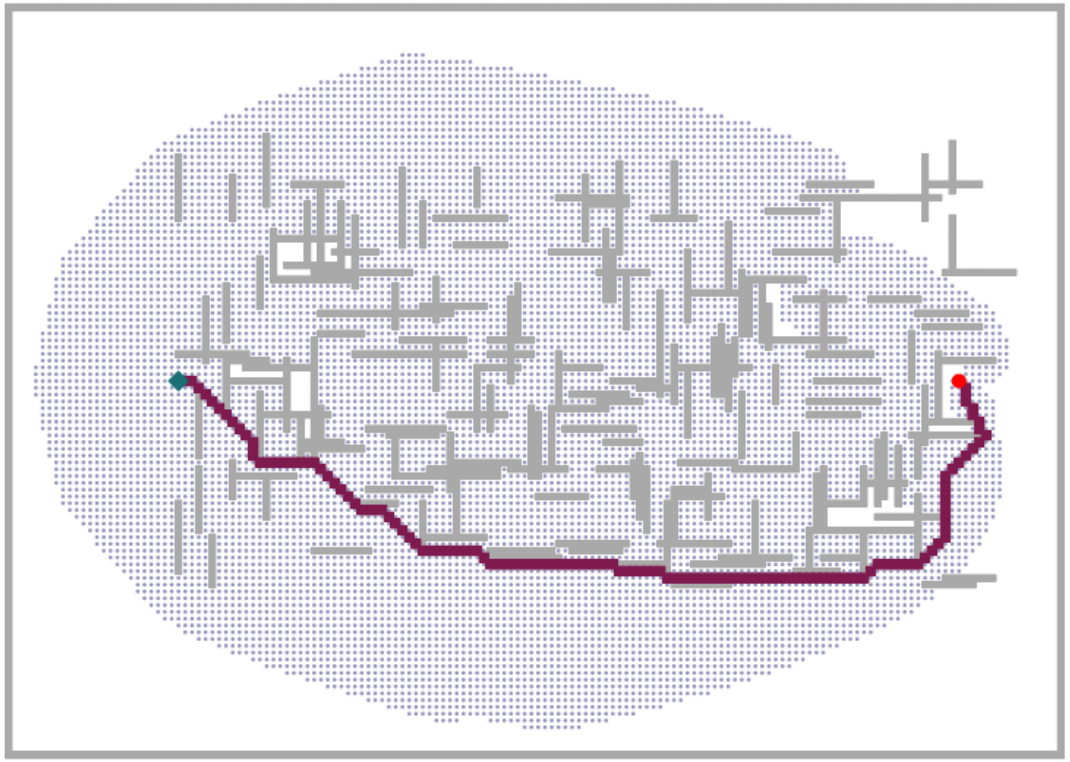


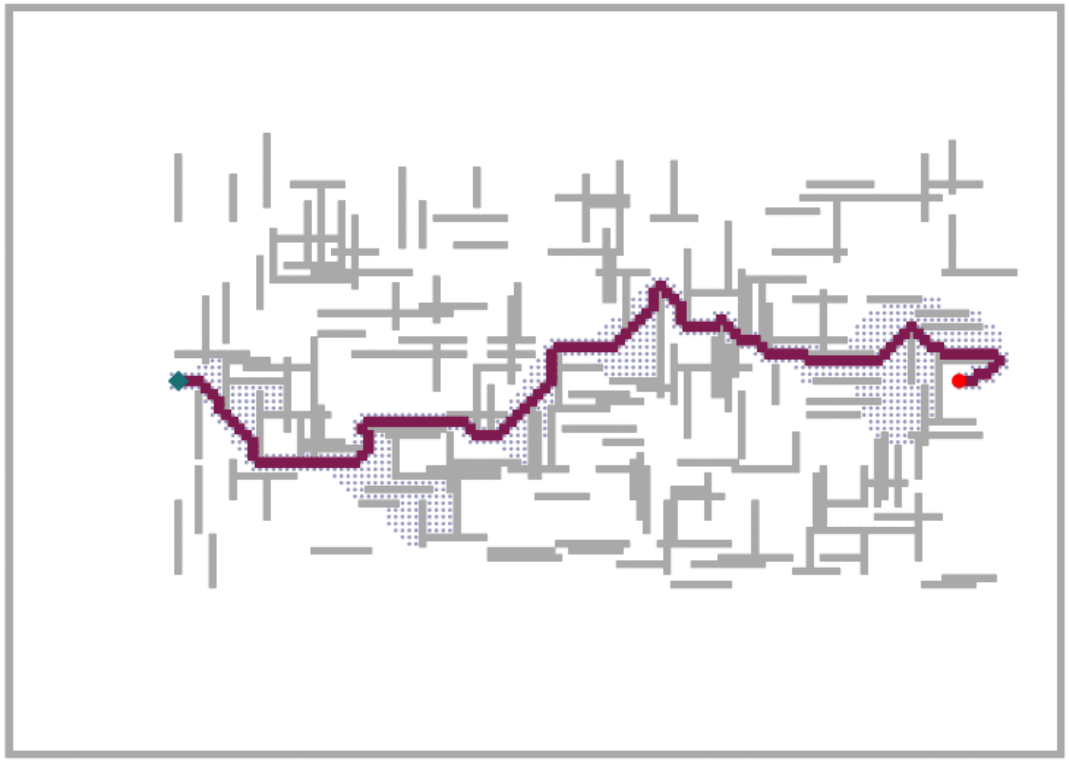
Partial order: Plan steps may be unordered



Heuristics for Planning

Forward and backward search are inefficient without a good heuristic $h(s)$.





Ignore selected preconditions

$Action(Slide(t, s1, s2)$
$Precond : On(t, s1) \wedge Tile(t) \wedge Blank(s2) \wedge Adjacent(s1, s2)$
$Effect : On(t, s2) \wedge Tile(t) \wedge Blank(s1) \wedge \neg On(t, s1) \wedge \neg Blank(s2)$

Nr-of-misplaced tiles - remove $Blank(s2) \wedge Adjacent(s1, s2)$

Manhattan distance - remove $Blank(s2)$

Ignore-delete list heuristic

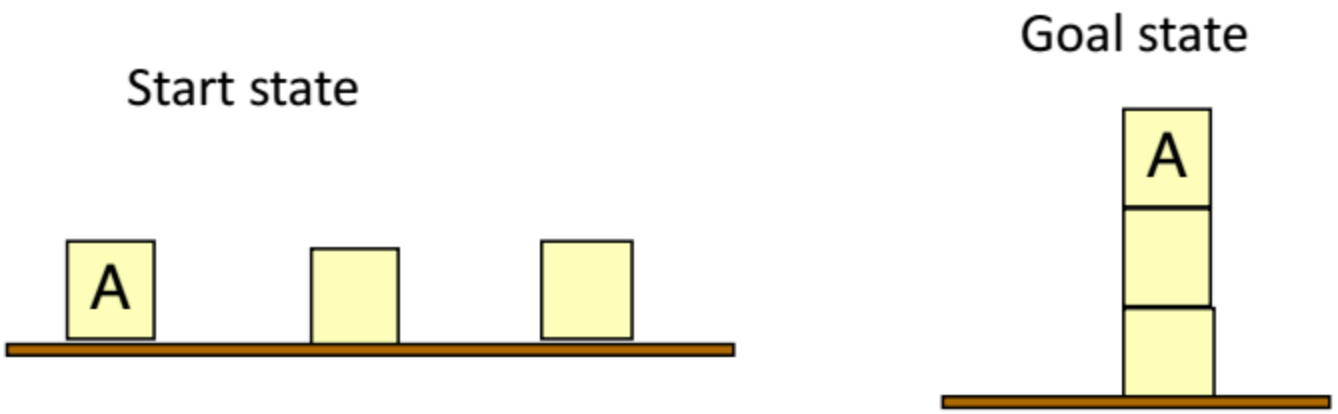
Relaxed version:

- Remove negative literals from preconditions and goals.
- Remove delete lists from all actions.

Properties:

- No undos by other actions - monotonic progress.
- NP-hard but can be approximated with hill-climbing

Domain-independent pruning - Symmetry reduction



Domain-independent pruning - Preferred action

Focus on promising branches. How?

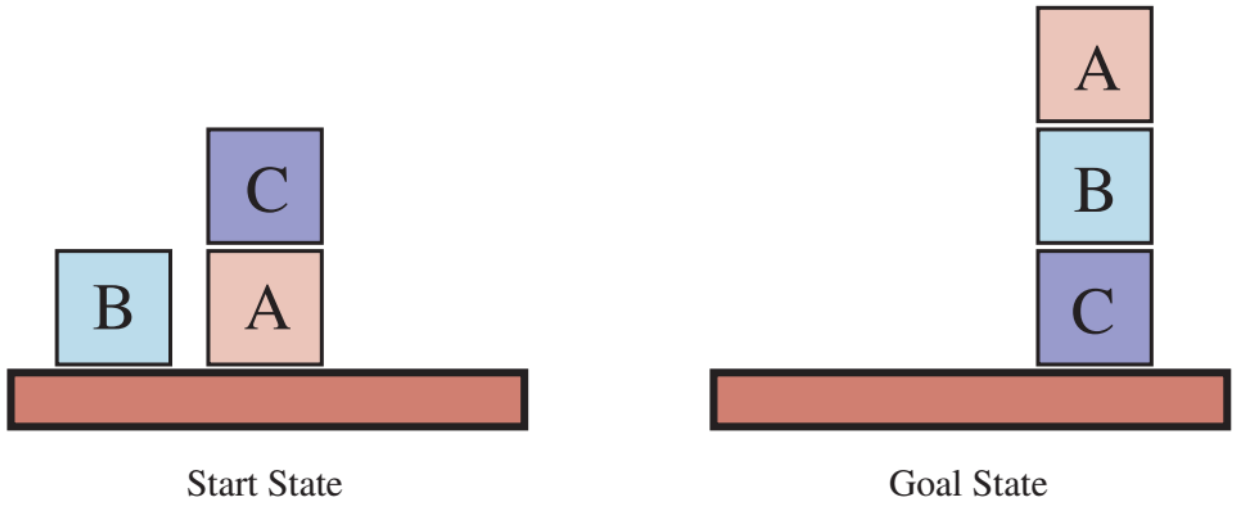
1. Define and solve a relaxed version to get **relaxed plan**
2. Choose preferred actions - steps of the relaxed plan or achieve a precondition of the relaxed plan

Domain-independent pruning - Serializable subgoals

Order of subgoals that can be achieved without undoing previously achived subgoals.

No backtracking!

Example 1: Bottom-to-top



Example 2: No dependency - Arbitrary switching on lights.

State abstraction

Many-to-one mapping of states to abstract representation.

Methods:

1. Identifying and achieving similar subgoals as one,
   e.g. many packages with the same src and destination.
2. Creating heuristic from decomposed subgoals,
   e.g. $Cost(P_i) + Cost(P_j)$
3. Pattern databases for subgoals

Planning and acting in non-classical environment

"Non-classical" environment:

- Partially observable and/or nondeterministic environment
- Incorrect information (differences between world and model)
- The agent’s future actions will depend on future percepts
- The future percepts cannot be determined in advance

Use percepts:

- Perceive the changes in the world
- Act accordingly
- Adapt plan when necessary

Types of non-classical planning

Sensorless planning for environments with no observations.

Contingency planning for partially observable and nondeterministic environments.

Online planning and **replanning** for unknown environments.

Percept schema - Part 2

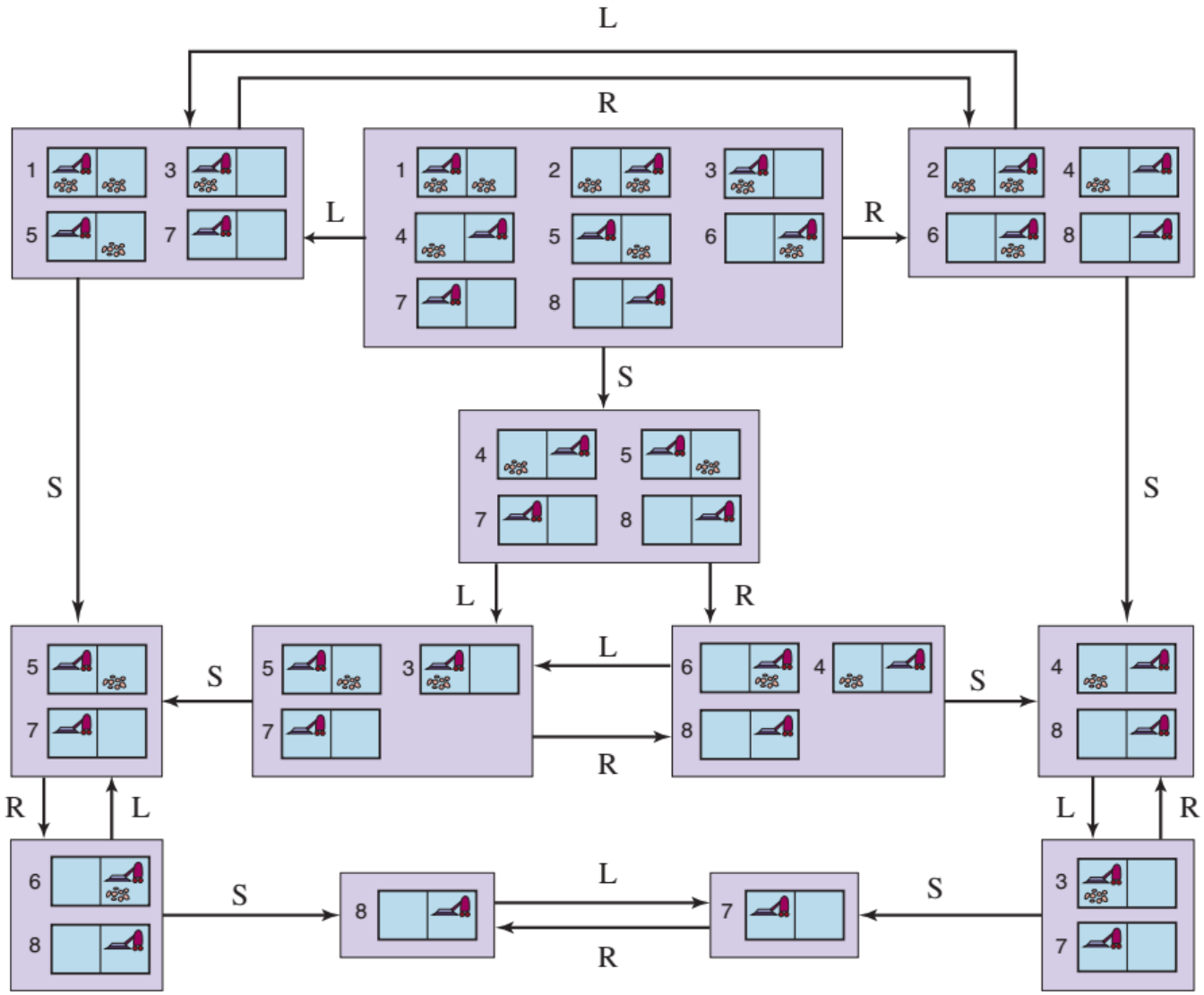
$$
\begin{aligned}
&Percept(Color\,(x, c)\,,\\
&\quad \textsc{Precond}{:}Object\,(x) \wedge InView\,(x)\\
&Percept(Color\,(can, c)\,,\\
&\quad \textsc{Precond}{:}Can\,(can) \wedge InView\,(can) \wedge Open\,(can)
\end{aligned}
$$

$$
\begin{aligned}
&Action(LookAt\,(x)\,,\\
&\quad \textsc{Precond}{:}InView\,(y) \wedge (x \neq y)\\
&\quad \textsc{Effect}{:}InView\,(x) \wedge \neg InView\,(y))
\end{aligned}
$$

Who will make a better plan?

Sensorless agent, contingency planning agent, online planning agent

Recall: Belief states in state-space search



Sensorless planning - Keep belief state simple

$$[Right, Suck, Left, Suck]$$

$$b_0 = True$$
$$b_1 = AtR$$
$$b_2 = AtR \wedge CleanR$$
$$b_3 = AtL \wedge CleanR$$
$$b_4 = AtL \wedge CleanR \wedge CleanL$$

Real-life: We always try to eliminate uncertainty.

Sensorless planning
Use subset of belief state as heuristic

Admissible, solving subset of a belief state is easier:

$$\text{if } b_1 \subseteq b_2 \text{ then } h^*(b_1) \leq h^*(b_2).$$

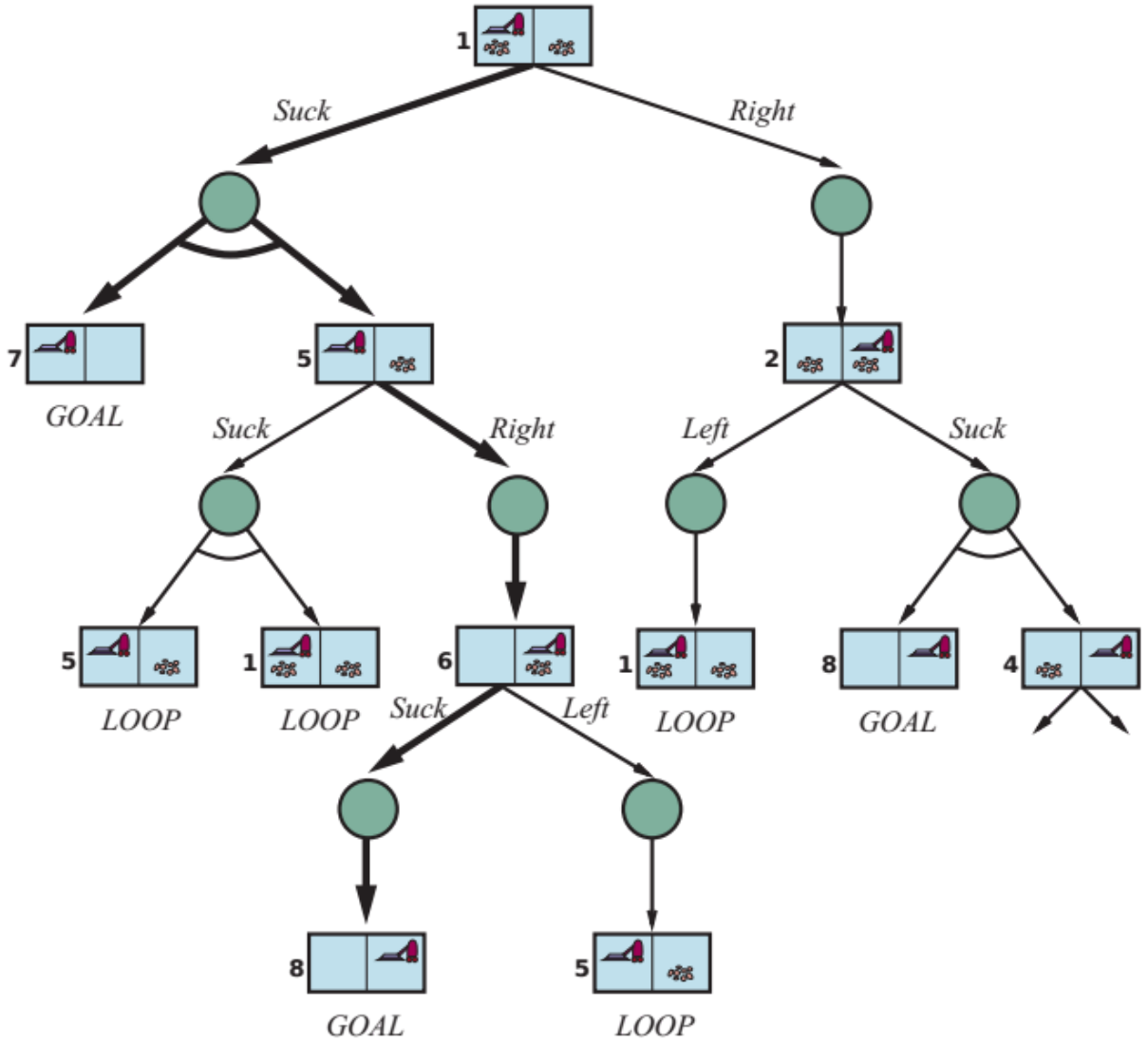
Combine heuristic values of individual states in belif state:

$$H(b) = \max\{h(s_1), \ldots, h(s_N)\}$$

Contingent planning

For partial observable and/or non-deterministic environments.

AND-OR search trees.



Contingent planning - Example

Given a chair and a table, the goal is to have them of the same colour. In the initial state we have two cans of paint, but the colours of the paint and the chair are unknown.

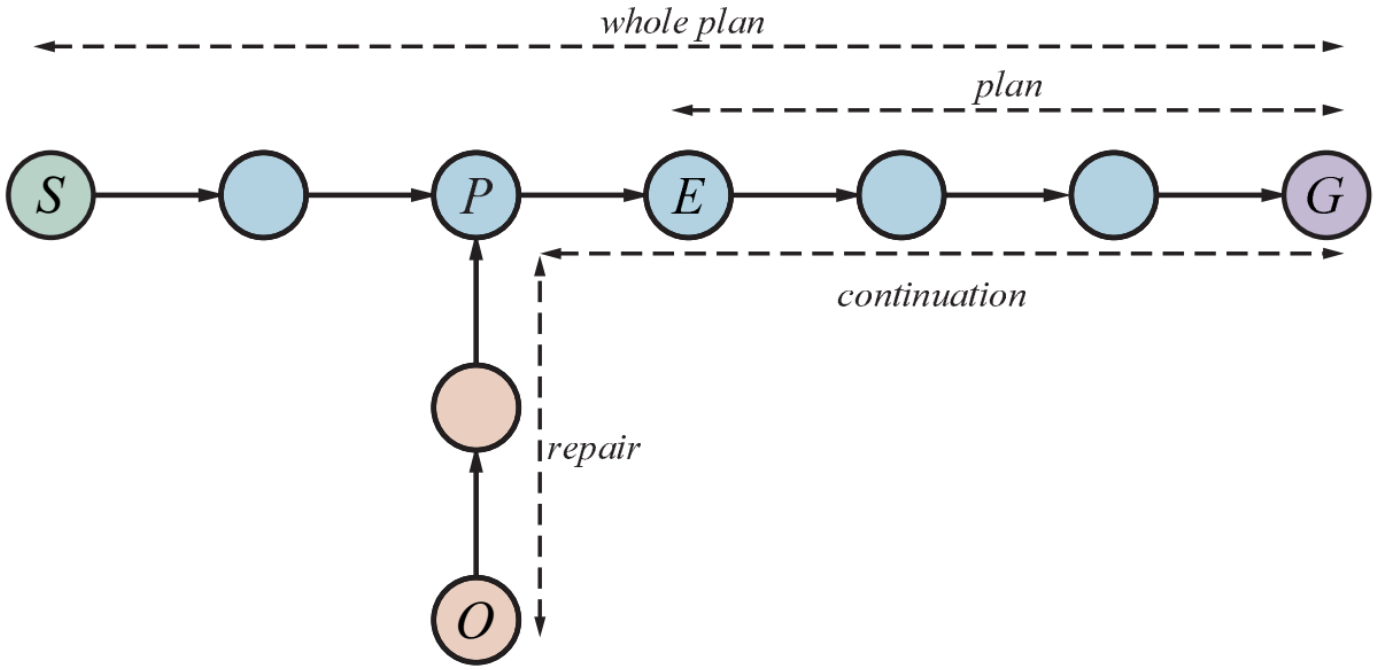
$$
\begin{aligned}
&[LookAt\,(Table)\,, LookAt\,(Chair)\,,\\
&\ \textbf{if}\ Color\,(Table, c) \wedge Color\,(Chair, c)\ \textbf{then}\ NoOp\\
&\quad \textbf{else}\,[RemovedLid\,(Can_1)\,, LookAt\,(Can_1)\,, RemovedLid\,(Can_2)\,, LookAt\,(Can_2)\,,\\
&\qquad \textbf{if}\ Color\,(Table, c) \wedge Color\,(can, c)\ \textbf{then}\ Paint\,(Chair, can)\\
&\qquad \textbf{else if}\ Color\,(Chair, c) \wedge Color\,(can, c)\ \textbf{then}\ Paint\,(Table, can)\\
&\qquad \textbf{else}\ [Paint\,(Chair, Can_1)\,, Paint\,(Table, Can_1)]]]
\end{aligned}
$$

Online planning - monitoring approaches

Action monitorin - check that all preconditions hold

Plan monitoring - check the plan holds

Goal monitoring - check if there is a better goal



Lecture 12: Multiagent-Systems and Game Theory

NTNU

Pinar Øzturk

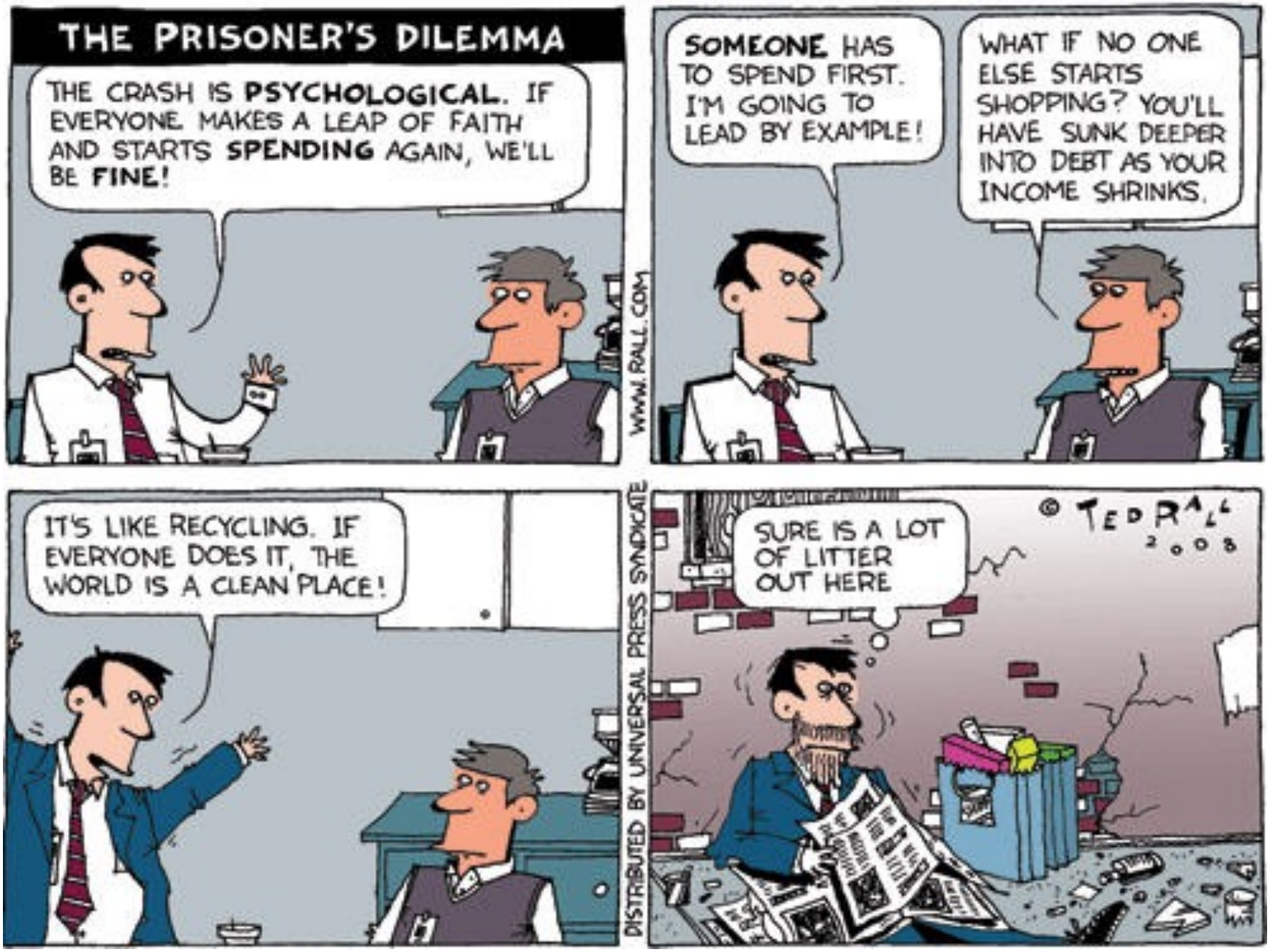
2021

Game theory

- Studies the decisions of agents in a multiagent environment where
  - Actions of each agent have an effect in the environment
  - So, outcome of an action of one agent may depend on other agents' actions
  - Hence, "strategical decision" making

Strategical Reasoning

- An agent's decision depends on what it reasons/thinks the other agent(s) will do.
  - The agent is **uncertain** about the decisions (i.e, actions) of the other participating agents
  - Tries to predict other's next decisions
  - Computes how these affect him/her/them
  - Makes own decisions accordingly


THE PRISONER'S DILEMMA
THE CRASH IS PSYCHOLOGICAL. IF EVERYONE MAKES A LEAP OF FAITH AND STARTS SPENDING AGAIN, WE'LL BE FINE!
SOMEONE HAS TO SPEND FIRST. I'M GOING TO LEAD BY EXAMPLE!
WHAT IF NO ONE ELSE STARTS SHOPPING? YOU'LL HAVE SUNK DEEPER INTO DEBT AS YOUR INCOME SHRINKS.
WWW.RALL.COM
IT'S LIKE RECYCLING. IF EVERYONE DOES IT, THE WORLD IS A CLEAN PLACE!
SURE IS A LOT OF LITTER OUT HERE
© TED RALL 2008
DISTRIBUTED BY UNIVERSAL PRESS SYNDICATE

Defining Strategical environments

- Main components
  1. Agents
  2. Strategies/actions
  3. Outcomes
  4. Payoffs (utilities)

Strategic Normal Games

A game in strategic normal form is defined by:

1. Agents: (N>1)
2. Actions:
   - Each agent *i* chooses an action a_i from its own action set A_i
   - The vector a=$(a_1, ..., a_n)$ of individual actions is called a **joint action** (or **action profile** or strategy profile). The set *A* is the set of all joint actions.

		agent j: action1	agent j: action2
agent i	action 1	1 1	4 1
agent i	action 2	1 4	4 4

Right now don't think about the numbers in the "squares"

Outcomes

3. Outcomes: Suppose we have the case where two agents have Action={a,b}, i.e., each agent has just two possible actions that it can perform. Four possible different outcomes can be produced by the system:

$\tau(a,a) = \omega_1$ $\tau(a,b) = \omega_2$

$\tau(b,a) = \omega_3$ $\tau(b,b) = \omega_4$

Payoff Matrix

- Assume agent i has Ai={a,b} and agent j has Aj={c,d}
- Agent's utilities are shown as a payoff matrix:

agent i

	agent j: action c	agent j: action d
action a	u_2 u_1	u_4 u_3
action b	u_6 u_5	u_8 u_7

OR

a_i \ a_j	action c	action d
action a	u_1, u_2	u_3, u_4
action b	u_5, u_6	u_7, u_8

- The payoff u_1 is the utility for agent *i* when agent *i* chooses action *a* and agent *j* chooses action *c*.
- The payoff u_2 is the utility for agent *j* when agent *i* chooses action *a* and agent *j* chooses action *c*.

PD - II

Confess = defect
Stay silent= cooperate

a_i \ a_j	cooperate	defect
cooperate	3, 3	1, 4
defect	4, 1	2, 2

- Note that the numbers of the payoff matrix do not refer to years in prison but to how good an outcome is for the agent, the shorter time in jail the better.
- Top left: Reward for mutual cooperation (1 year each).
- Top right: If *i* cooperates and *j* defects, *i* gets sucker's payoff of 1, while *j* gets 4 (3 vs zero years)
- Bottom left: If *j* cooperates and *i* defects, *j* gets sucker's payoff of 1, while *i* gets 4 (e.g., 3 vs zero years).
- Bottom right: If both defect, then both get punishment for mutual defection (e.g., 2 years each).

Solution Concepts

- A solution to a game is a prediction of the outcome of the game using the assumption that all agents are rational and strategic.

- How does the game theory predict?
  1. Strictly(Strongly) *dominant* equilibriums.
  2. Weakly *dominant* equilibriums.
  3. Iterated elimination of **dominated** actions.
  4. Nash equilibrium.

Strictly Dominant Strategy equilibrium - example

		agent j: defect	agent j: cooperate
agent i	defect	2, 2	4, 1
	cooperate	1, 4	3, 3

- Preferences:
  - i's: **D**,D > C,D and **D**,C > C,C (compare rows)
  - j's: D,**D** > D,C and C,**D** > C,C (compare columns)

- "Defect" is the **strictly dominant strategy** for both agents.

- (D,D) is the strictly (also called **strongly)** dominant strategy equilibrium.

WDS equilibrium

		agent j: action a	agent j: action b
agent i	action a	2, 1	0, 2
	action b	2, 3	4, 3

- Neither agent has a strongly dominant strategy.
- Action *b* is a weakly dominant strategy for both agents *i* and *j*.
- Therefore *(b,b)* is a weakly dominant strategy equilibrium.
- Rational agents will usually play this strategy.

Example: No dominant strategy at all

agent j

agent i	L	M	R
U	1,0	1,2	0,1
D	0,3	0,1	2,0

- Are there/What are the dominant strategies?
- The game has no dominant strategy equilibrium.

Iterated elimination of strictly dominated actions - Example

agent j

agent i	L	M	R
U	1,0	1,2	0,1
D	0,3	0,1	2,0

agent j

agent i	L	M	R
U	1,0	1,2	0,1
D	0,3	0,1	2,0

agent j

agent i	L	M	R
U	1,0	1,2	0,1
D	0,3	0,1	2,0

- R is strictly dominated for agent *j* (by action M).
- We eliminate R because agent *i*, being rational, knows that agent *j* will not play R.
- Now agent *j* notices that D is strictly dominated for agent *i*. Agent *j* eliminated in his head the action for agent *i*.
- Eliminate L for agent *j*.
- The order of elimination does not matter in IESD

Example on pareto optimality

		A	B
Agent1	A	9,9	7, 10
	B	10, 7	8, 8

- Which joint action(s) is/are pareto optimal?
  - (A,A) : optimal because no other outcome makes an agent better without making the other worse
  - (A,B): optimal, because payoff "7" of agent1 can be improved but then agent2 is worse off.
  - (B,A): optimal
  - (B,B): **not** optimal. Both agents are better off when they shift to (A,A)

Social Welfare – defn.

- Social welfare value (of a solution)
  - Sum of utilities of all agents for this strategy profile.
  - If a solution maximizes social welfare, i.e., social optimum, then the available utilities are not wasted.

- if a solution is a social optimum, then it is Pareto efficient.
  - The converse does not hold

Solution concept: Nash Equilibrium - Definition

- **Nash equilibrium** is a strategy profile (i.e., a collection of strategies, one for each player) such that each strategy is a **best response** (maximizes payoff) to all the other agents' strategies

- **Best response:** The best response of agent *i* is given by $BR_i(a_{-i}) = \{a_i \in A_i : u_i(a_i, a_{-i}) \geq u_i(b_i, a_{-i}) \text{ for all } b_i \in A_i\}$.

Example: Prisoner's Dilemma

agent j

agent i	defect	cooperate
defect	2, 2	4, 1
cooperate	1, 4	3, 3

- The strategy profile (defect, defect) is a Nash Equilibrium.
  - If agent *i* changes its strategy(defect) to cooperate - the new situation is (cooperate, defect) - agent i will be worse off (payoff from 2 to 1).
  - If *j* changes its strategy (defect→ cooperate) it will be worse off (payoff, again, from 2 to 1).

- So defection is the best response to all possible strategies.

Multiple Nash equilibria

- Not every interaction scenario has a single Nash equilibrium

		agent j: l	agent j: r
agent i	l	1, 1	0, 0
	r	0, 0	2, 2

- Consider two agents each of whom chooses either l (left) or r (right). If their choices do not match, they receive a payoff of zero. See the other payoff values in the figure.
- There are two Nash equilibria, (l,l) and (r,r).
- The two Nash equilibria are not equal. Would the agents possibly ”agree” (coordinate) upon one of the solutions?
  - (r,r) is *social optimum*.
- This type of game is known as “pure coordination game”

Mixed Strategies

agent j

agent i	H	T
H	-1 +1	+ 1, -1
T	+1, -1	-1, +1

- No Nash: Because in Nash, the players don't change their strategies even when they know what the other agent will play
- Here for every strategy pair, there is an agent that may want to change their strategy to get more payoff.
- Hence, no **pure** strategy equilibrium?
- What now? Randomized behaviour : Mixed strategies

Why inefficient outcomes becomes solution?

- Need for new theories

- Need for modification in the modelled environment?
  - E.g., mafia norms, “nobody talks to authorities”
  - Legislations/Regulations?
  - Incentives/subsidization

Iterated Prisoner's Dilemma

- How to evolve cooperation?
  - One answer: play the game more than once.
- If you know you will be meeting your opponent again, then the incentive to defect appears to evaporate.

Strategies in Axelrod's Tournament

- ALLD:
  - Always defect - the hawk strategy.
- RANDOM
  - Selects either cooperate or defect on random.
- TIT-FOR-TAT
  - On round t = 0, cooperate
  - On round t > 0, do what your opponent did on round t-1.
- TESTER
  - On 1st round defect. If the opponent ever retaliated with defection, then play TIT-FOR-TAT. Otherwise play a repeated sequence of cooperating for two rounds, then defecting.
- JOSS
  - As TIT-FOR-TAT, except periodically defect.

Recipes for Success in Axelrod's Tournament

Axelrod suggested the following rules for succeeding in his tournament:

- Don't be envious:

  Don't Play as if it were zero sum!

- Be nice:

  Start by cooperating, and reciprocate cooperation.

- Retaliate appropriately:

  Always punish defection immediately, but use "measured" force - don't overdo it.

- Don't hold grudges:

  Always reciprocate cooperation immediately.