Artificial Intelligence

Chapter 1

Outline

- ♢ What is AI?
- ♢ A brief history
- ♢ The state of the art

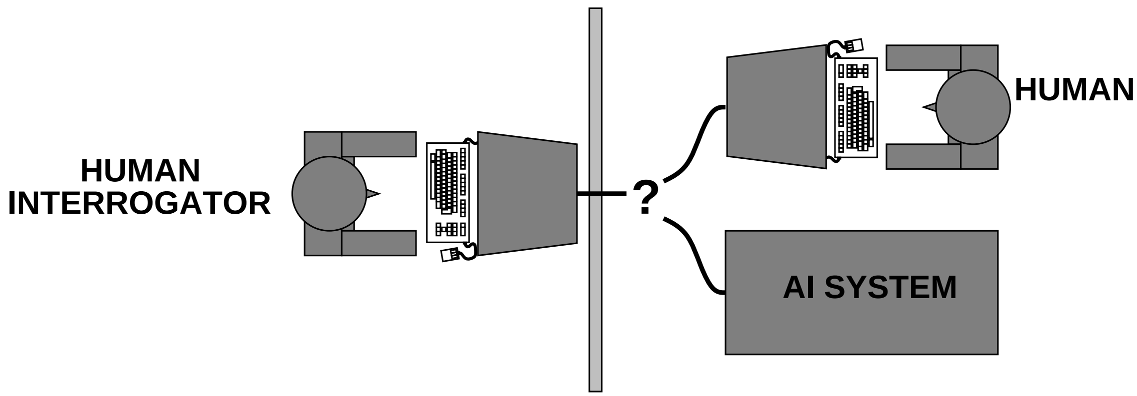
What is AI?

Systems that think like humans	**Systems that think rationally**
Systems that act like humans	**Systems that act rationally**

Acting humanly: The Turing test

Turing (1950) "Computing machinery and intelligence":

- ♢ "Can machines think?" ⟶ "Can machines behave intelligently?"
- ♢ Operational test for intelligent behavior: the Imitation Game



- ♢ Predicted that by 2000, a machine might have a 30% chance of fooling a lay person for 5 minutes
- ♢ Anticipated all major arguments against AI in following 50 years
- ♢ Suggested major components of AI: knowledge, reasoning, language understanding, learning

Problem: Turing test is not **reproducible**, **constructive**, or amenable to **mathematical analysis**

Thinking humanly: Cognitive Science

1960s "cognitive revolution": information-processing psychology replaced prevailing orthodoxy of behaviorism

Requires scientific theories of internal activities of the brain
- What level of abstraction? "Knowledge" or "circuits"?
- How to validate? Requires
  1) Predicting and testing behavior of human subjects (top-down)
  or 2) Direct identification from neurological data (bottom-up)

Both approaches (roughly, Cognitive Science and Cognitive Neuroscience) are now distinct from AI

Both share with AI the following characteristic:
the available theories do not explain (or engender) anything resembling human-level general intelligence

Hence, all three fields share one principal direction!

Thinking rationally: Laws of Thought

Normative (or prescriptive) rather than descriptive

Aristotle: what are correct arguments/thought processes?

Several Greek schools developed various forms of logic:
notation and **rules of derivation** for thoughts;
may or may not have proceeded to the idea of mechanization

Direct line through mathematics and philosophy to modern AI

Problems:
1) Not all intelligent behavior is mediated by logical deliberation
2) What is the purpose of thinking? What thoughts **should** I have
out of all the thoughts (logical or otherwise) that I **could** have?

Acting rationally

Rational behavior: doing the right thing

The right thing: that which is expected to maximize goal achievement, given the available information

Doesn't necessarily involve thinking—e.g., blinking reflex—but thinking should be in the service of rational action

Aristotle (Nicomachean Ethics):

> **Every art and every inquiry, and similarly every action and pursuit, is thought to aim at some good**

Rational agents

An agent is an entity that perceives and acts

This course is about designing rational agents

Abstractly, an agent is a function from percept histories to actions:

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

For any given class of environments and tasks, we seek the
agent (or class of agents) with the best performance

Caveat: **computational limitations make perfect rationality unachievable**
$\rightarrow$ design best program for given machine resources

AI prehistory

Philosophy	logic, methods of reasoning
	mind as physical system
	foundations of learning, language, rationality
Mathematics	formal representation and proof
	algorithms, computation, (un)decidability, (in)tractability
	probability
Psychology	adaptation
	phenomena of perception and motor control
	experimental techniques (psychophysics, etc.)
Economics	formal theory of rational decisions
Linguistics	knowledge representation
	grammar
Neuroscience	plastic physical substrate for mental activity
Control theory	homeostatic systems, stability
	simple optimal agent designs

Potted history of AI

1943	McCulloch & Pitts: Boolean circuit model of brain
1950	Turing's "Computing Machinery and Intelligence"
1952–69	Look, Ma, no hands!
1950s	Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
1956	Dartmouth meeting: "Artificial Intelligence" adopted
1965	Robinson's complete algorithm for logical reasoning
1966–74	AI discovers computational complexity Neural network research almost disappears
1969–79	Early development of knowledge-based systems
1980–88	Expert systems industry booms
1988–93	Expert systems industry busts: "AI Winter"
1985–95	Neural networks return to popularity
1988–	Resurgence of probability; general increase in technical depth "Nouvelle AI": ALife, GAs, soft computing
1995–	Agents, agents, everywhere . . .
2003–	Human-level AI back on the agenda

State of the art

Which of the following can be done at present?

◇ Play a decent game of table tennis

State of the art

Which of the following can be done at present?

◇ Play a decent game of table tennis
◇ Drive safely along a curving mountain road

State of the art

Which of the following can be done at present?

◊ Play a decent game of table tennis
◊ Drive safely along a curving mountain road
◊ Drive safely along Telegraph Avenue

State of the art

Which of the following can be done at present?

◊ Play a decent game of table tennis
◊ Drive safely along a curving mountain road
◊ Drive safely along Telegraph Avenue
◊ Buy a week's worth of groceries on the web

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl
- ♢ Play a decent game of bridge

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl
- ♢ Play a decent game of bridge
- ♢ Discover and prove a new mathematical theorem

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl
- ♢ Play a decent game of bridge
- ♢ Discover and prove a new mathematical theorem
- ♢ Design and execute a research program in molecular biology

State of the art

Which of the following can be done at present?

- ◇ Play a decent game of table tennis
- ◇ Drive safely along a curving mountain road
- ◇ Drive safely along Telegraph Avenue
- ◇ Buy a week's worth of groceries on the web
- ◇ Buy a week's worth of groceries at Berkeley Bowl
- ◇ Play a decent game of bridge
- ◇ Discover and prove a new mathematical theorem
- ◇ Design and execute a research program in molecular biology
- ◇ Write an intentionally funny story

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl
- ♢ Play a decent game of bridge
- ♢ Discover and prove a new mathematical theorem
- ♢ Design and execute a research program in molecular biology
- ♢ Write an intentionally funny story
- ♢ Give competent legal advice in a specialized area of law

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl
- ♢ Play a decent game of bridge
- ♢ Discover and prove a new mathematical theorem
- ♢ Design and execute a research program in molecular biology
- ♢ Write an intentionally funny story
- ♢ Give competent legal advice in a specialized area of law
- ♢ Translate spoken English into spoken Swedish in real time

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl
- ♢ Play a decent game of bridge
- ♢ Discover and prove a new mathematical theorem
- ♢ Design and execute a research program in molecular biology
- ♢ Write an intentionally funny story
- ♢ Give competent legal advice in a specialized area of law
- ♢ Translate spoken English into spoken Swedish in real time
- ♢ Converse successfully with another person for an hour

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl
- ♢ Play a decent game of bridge
- ♢ Discover and prove a new mathematical theorem
- ♢ Design and execute a research program in molecular biology
- ♢ Write an intentionally funny story
- ♢ Give competent legal advice in a specialized area of law
- ♢ Translate spoken English into spoken Swedish in real time
- ♢ Converse successfully with another person for an hour
- ♢ Perform a complex surgical operation

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl
- ♢ Play a decent game of bridge
- ♢ Discover and prove a new mathematical theorem
- ♢ Design and execute a research program in molecular biology
- ♢ Write an intentionally funny story
- ♢ Give competent legal advice in a specialized area of law
- ♢ Translate spoken English into spoken Swedish in real time
- ♢ Converse successfully with another person for an hour
- ♢ Perform a complex surgical operation
- ♢ Unload any dishwasher and put everything away

State of the art

Which of the following can be done at present?

- ♢ Play a decent game of table tennis
- ♢ Drive safely along a curving mountain road
- ♢ Drive safely along Telegraph Avenue
- ♢ Buy a week's worth of groceries on the web
- ♢ Buy a week's worth of groceries at Berkeley Bowl
- ♢ Play a decent game of bridge
- ♢ Discover and prove a new mathematical theorem
- ♢ Design and execute a research program in molecular biology
- ♢ Write an intentionally funny story
- ♢ Give competent legal advice in a specialized area of law
- ♢ Translate spoken English into spoken Swedish in real time
- ♢ Converse successfully with another person for an hour
- ♢ Perform a complex surgical operation
- ♢ Unload any dishwasher and put everything away

Unintentionally funny stories

One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe threatened to hit Irving if he didn't tell him where some honey was. The End.

Henry Squirrel was thirsty. He walked over to the river bank where his good friend Bill Bird was sitting. Henry slipped and fell in the river. Gravity drowned. The End.

Once upon a time there was a dishonest fox and a vain crow. One day the crow was sitting in his tree, holding a piece of cheese in his mouth. He noticed that he was holding the piece of cheese. He became hungry, and swallowed the cheese. The fox walked over to the crow. The End.

Unintentionally funny stories

Joe Bear was hungry. He asked Irving Bird where some honey was. Irving refused to tell him, so Joe offered to bring him a worm if he'd tell him where some honey was. Irving agreed. But Joe didn't know where any worms were, so he asked Irving, who refused to say. So Joe offered to bring him a worm if he'd tell him where a worm was. Irving agreed. But Joe didn't know where any worms were, so he asked Irving, who refused to say. So Joe offered to bring him a worm if he'd tell him where a worm was . . .

Intelligent Agents

Chapter 2

Reminders

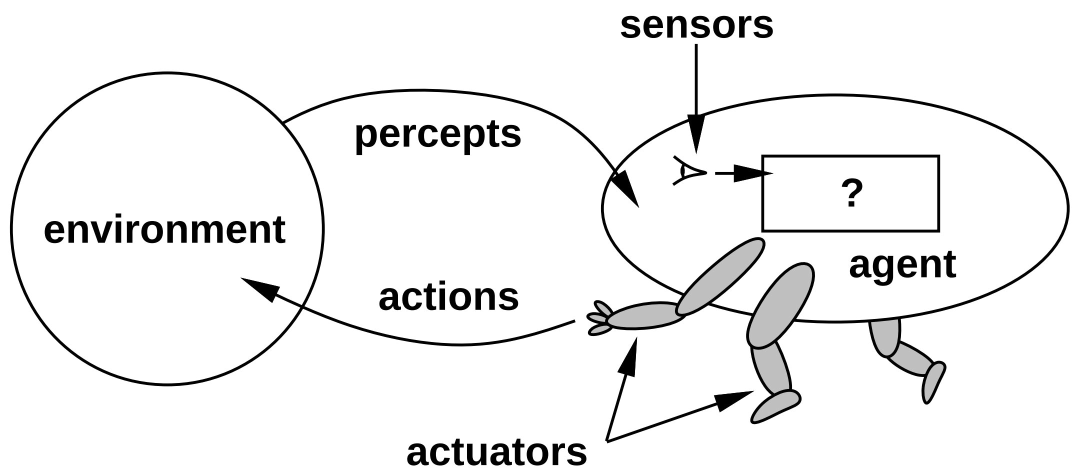
Assignment 0 (lisp refresher) due 1/28

Lisp/emacs/AIMA tutorial: 11-1 today and Monday, 271 Soda

Outline

- ♢ Agents and environments
- ♢ Rationality
- ♢ PEAS (Performance measure, Environment, Actuators, Sensors)
- ♢ Environment types
- ♢ Agent types

Agents and environments



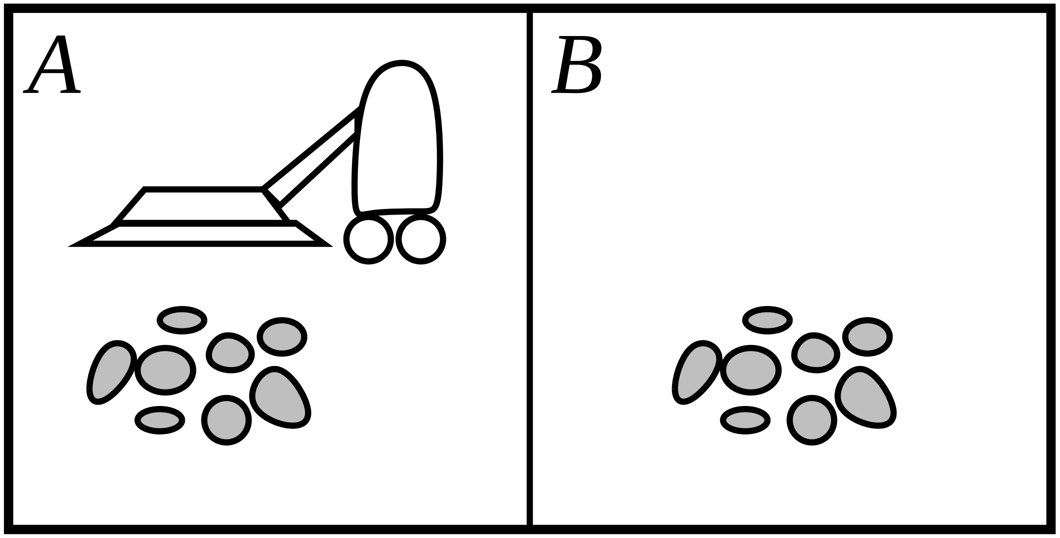
Agents include humans, robots, softbots, thermostats, etc.

The agent function maps from percept histories to actions:

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

The agent program runs on the physical architecture to produce f

Vacuum-cleaner world



Percepts: location and contents, e.g., $[A, Dirty]$

Actions: $Left$, $Right$, $Suck$, $NoOp$

A vacuum-cleaner agent

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

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*

What is the **right** function?
Can it be implemented in a small agent program?

Rationality

Fixed performance measure evaluates the environment sequence

- one point per square cleaned up in time T?
- one point per clean square per time step, minus one per move?
- penalize for $> k$ dirty squares?

A rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date

Rational $\neq$ omniscient

- percepts may not supply all relevant information

Rational $\neq$ clairvoyant

- action outcomes may not be as expected

Hence, rational $\neq$ successful

Rational $\Rightarrow$ exploration, learning, autonomy

PEAS

To design a rational agent, we must specify the task environment

Consider, e.g., the task of designing an automated taxi:

Performance measure??

Environment??

Actuators??

Sensors??

PEAS

To design a rational agent, we must specify the task environment

Consider, e.g., the task of designing an automated taxi:

Performance measure?? safety, destination, profits, legality, comfort, ...

Environment?? US streets/freeways, traffic, pedestrians, weather, ...

Actuators?? steering, accelerator, brake, horn, speaker/display, ...

Sensors?? video, accelerometers, gauges, engine sensors, keyboard, GPS, ...

Internet shopping agent

Performance measure??

Environment??

Actuators??

Sensors??

Internet shopping agent

Performance measure?? price, quality, appropriateness, efficiency

Environment?? current and future WWW sites, vendors, shippers

Actuators?? display to user, follow URL, fill in form

Sensors?? HTML pages (text, graphics, scripts)

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??				
Deterministic??				
Episodic??				
Static??				
Discrete??				
Single-agent??				

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??				
Episodic??				
Static??				
Discrete??				
Single-agent??				

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??				
Static??				
Discrete??				
Single-agent??				

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??				
Discrete??				
Single-agent??				

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Semi	Semi	No
Discrete??				
Single-agent??				

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Semi	Semi	No
Discrete??	Yes	Yes	Yes	No
Single-agent??				

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Semi	Semi	No
Discrete??	Yes	Yes	Yes	No
Single-agent??	Yes	No	Yes (except auctions)	No

The environment type largely determines the agent design

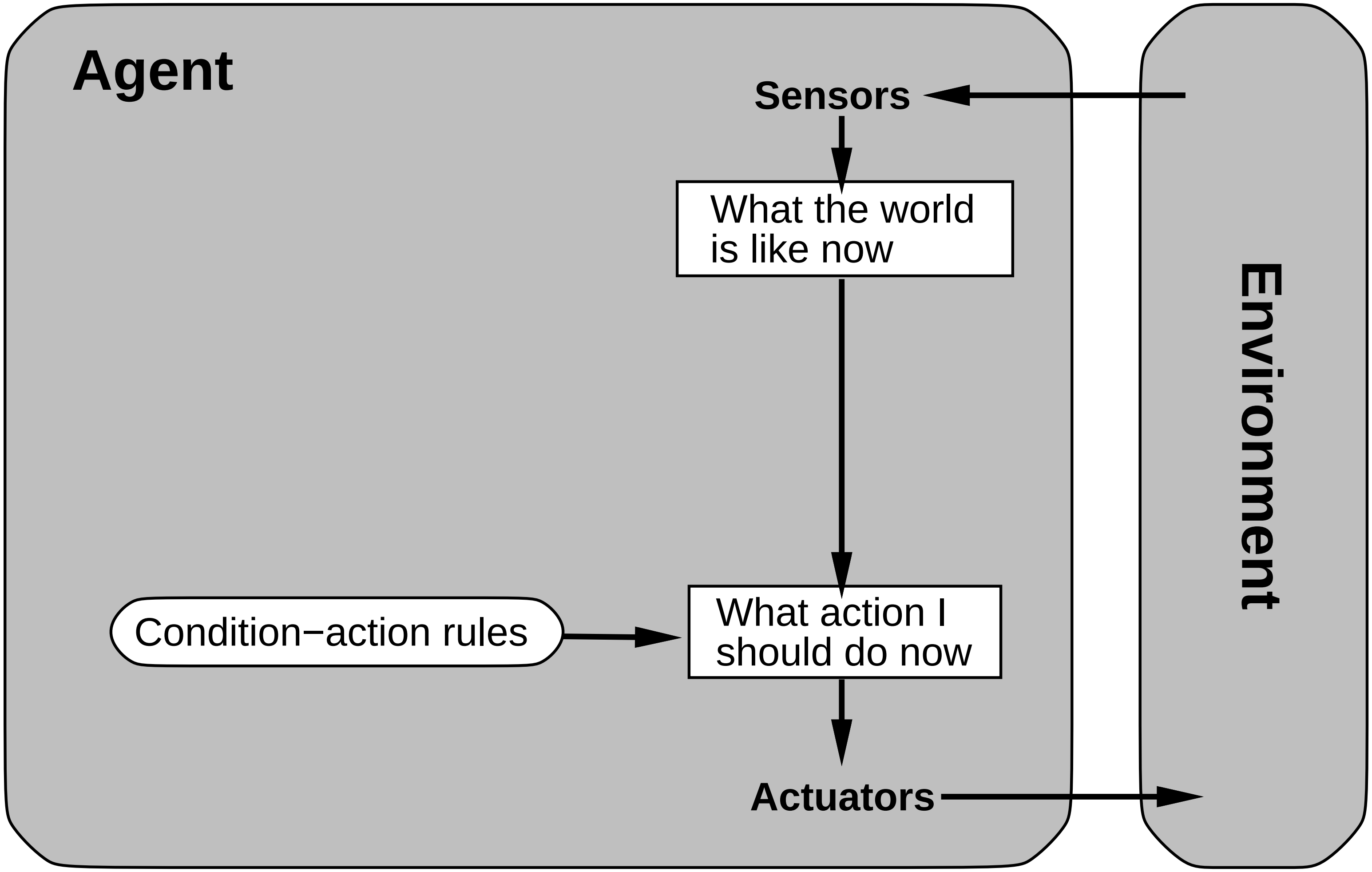
The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

Agent types

Four basic types in order of increasing generality:
- simple reflex agents
- reflex agents with state
- goal-based agents
- utility-based agents

All these can be turned into learning agents

Simple reflex agents



Example

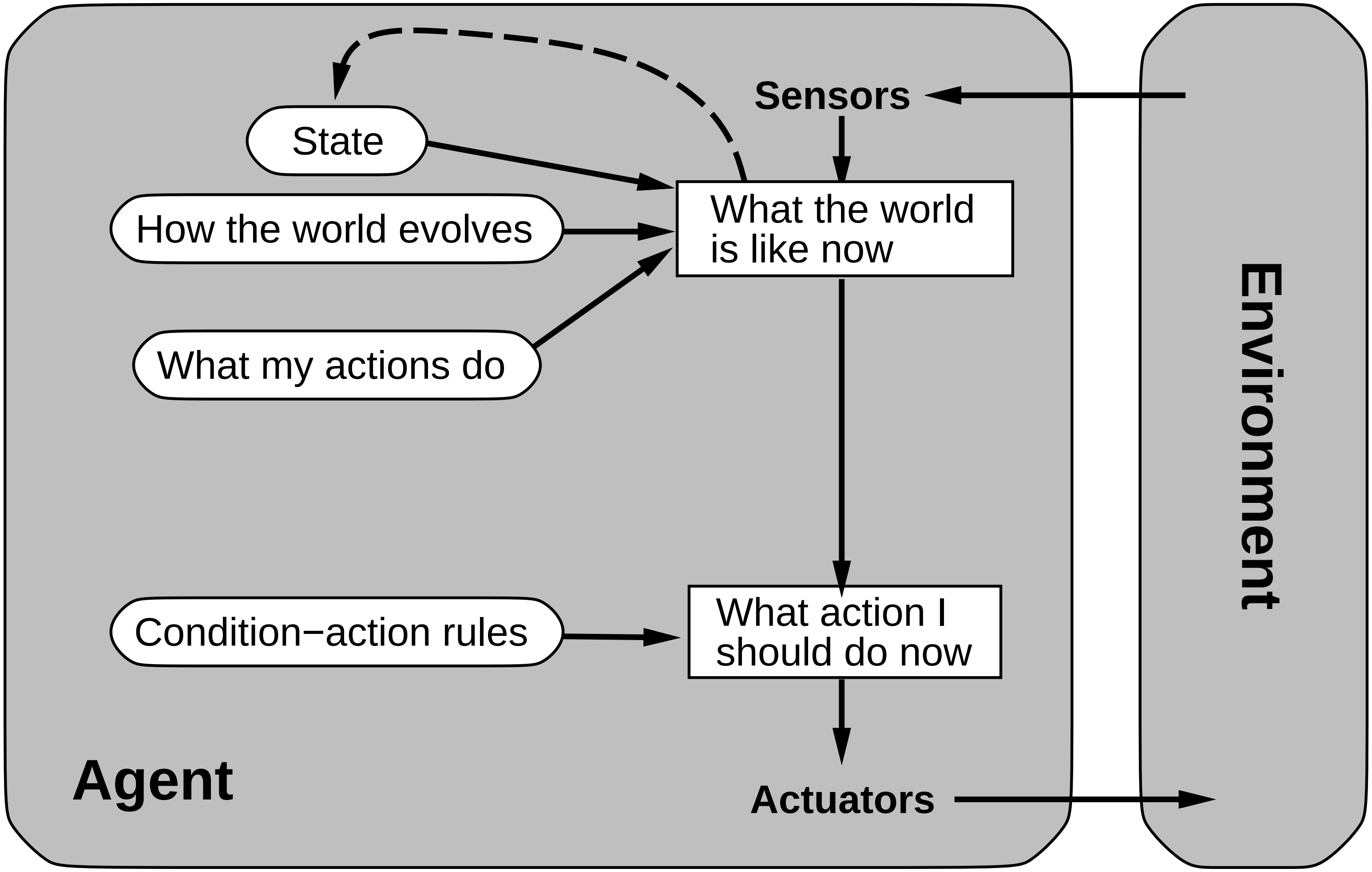
function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action

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

```
(setq joe (make-agent :name 'joe :body (make-agent-body)
                      :program (make-reflex-vacuum-agent-program)))

(defun make-reflex-vacuum-agent-program ()
  #'(lambda (percept)
      (let ((location (first percept)) (status (second percept)))
        (cond ((eq status 'dirty) 'Suck)
              ((eq location 'A) 'Right)
              ((eq location 'B) 'Left)))))
```

Reflex agents with state



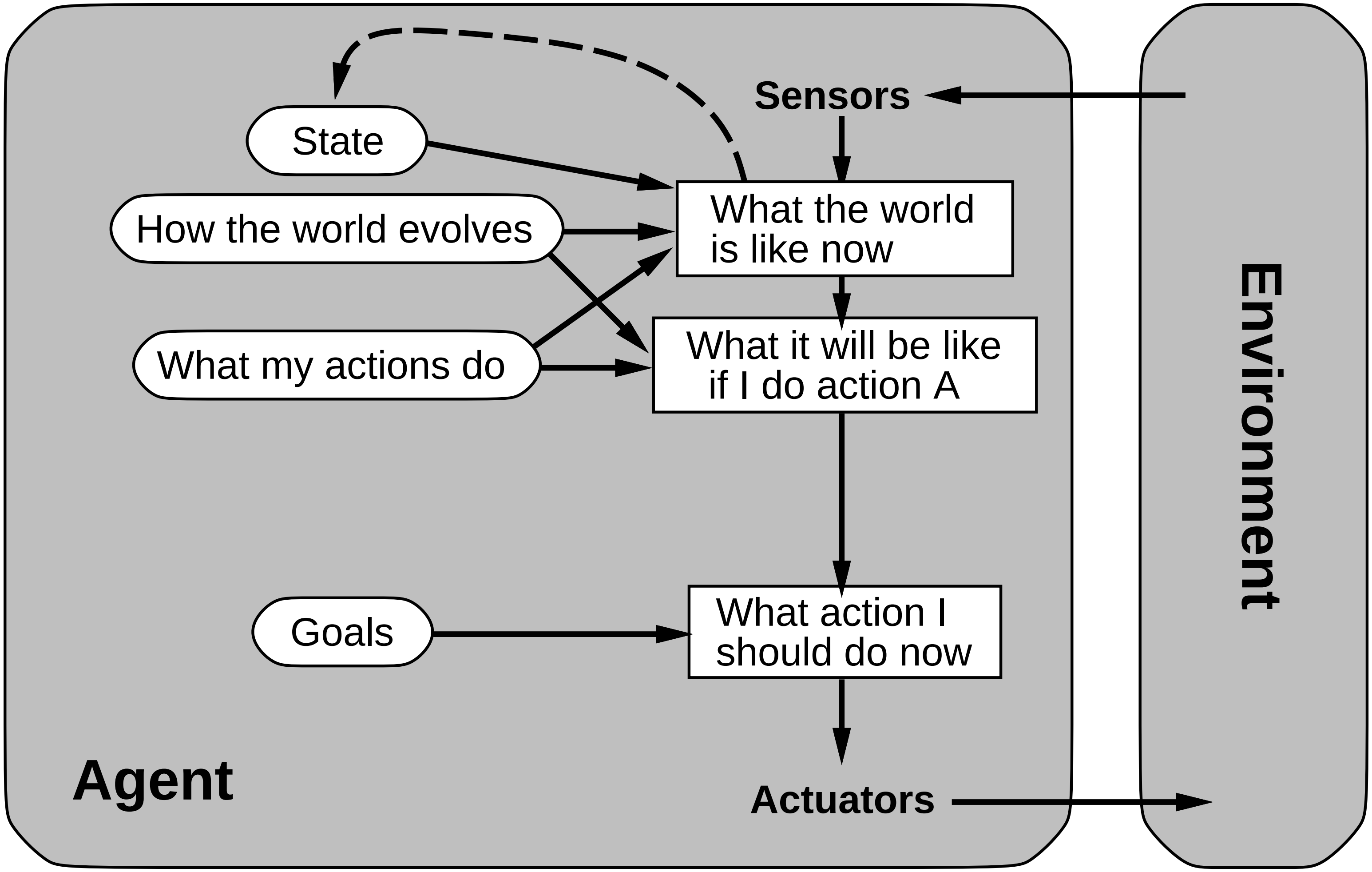
Example

function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action
static: $last\_A,\ last\_B$, numbers, initially ∞

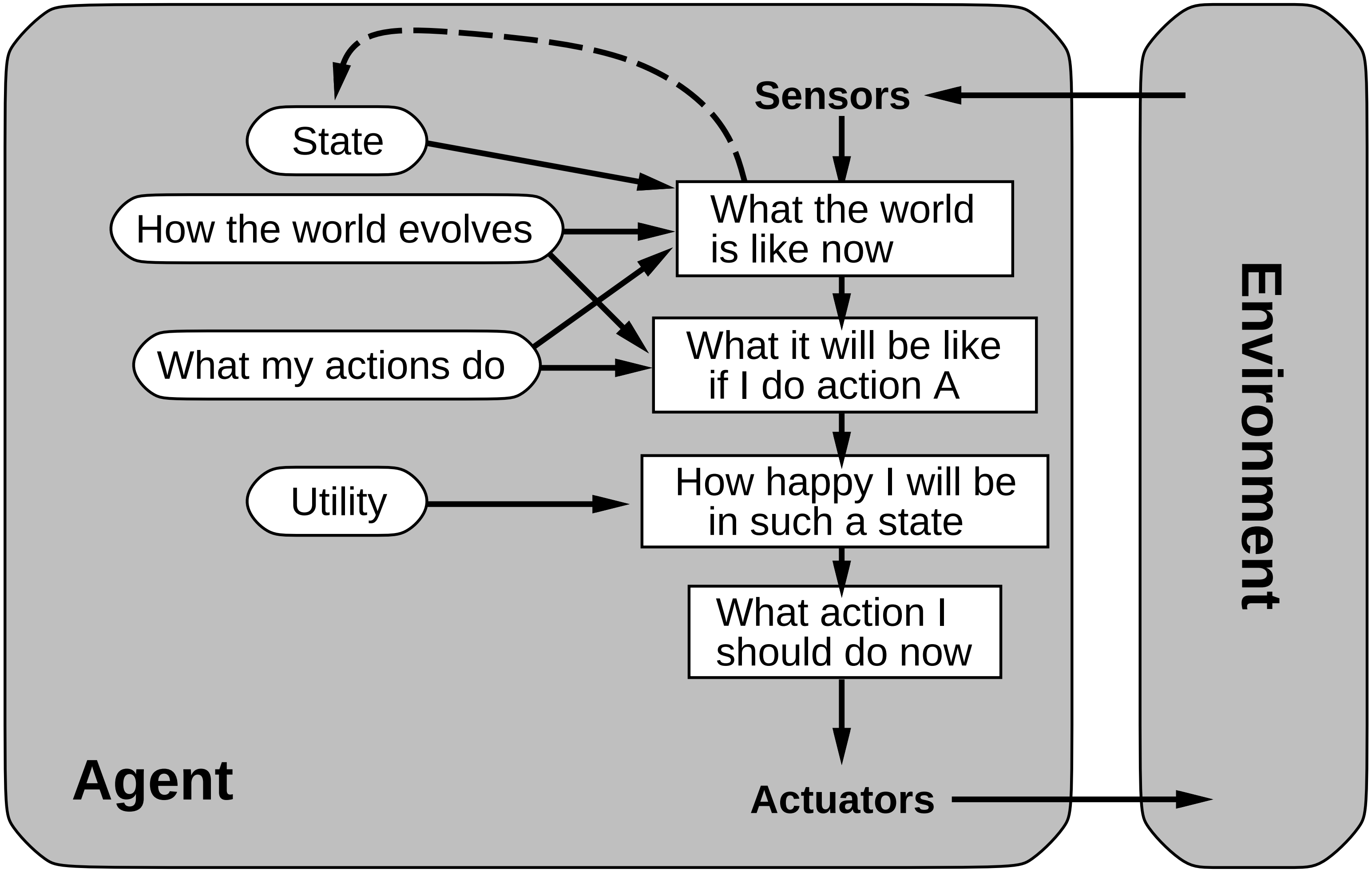
  if $status = Dirty$ **then** ...

```
(defun make-reflex-vacuum-agent-with-state-program ()
  (let ((last-A infinity) (last-B infinity))
  #'(lambda (percept)
      (let ((location (first percept)) (status (second percept)))
        (incf last-A) (incf last-B)
        (cond
         ((eq status 'dirty)
          (if (eq location 'A) (setq last-A 0) (setq last-B 0))
          'Suck)
         ((eq location 'A) (if (> last-B 3) 'Right 'NoOp))
         ((eq location 'B) (if (> last-A 3) 'Left 'NoOp)))))))
```

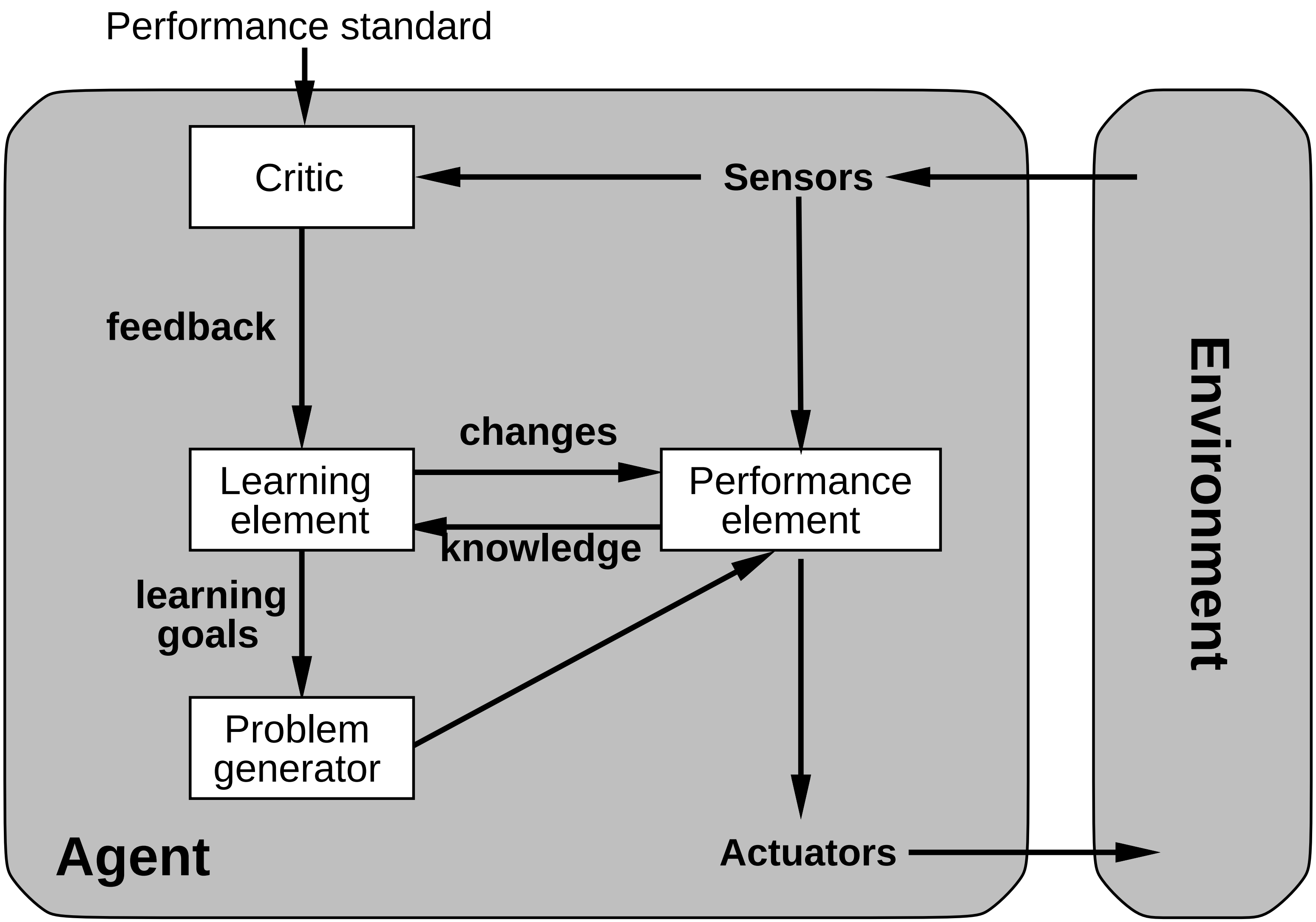
Goal-based agents



Utility-based agents



Learning agents



Summary

Agents interact with environments through actuators and sensors

The agent function describes what the agent does in all circumstances

The performance measure evaluates the environment sequence

A perfectly rational agent maximizes expected performance

Agent programs implement (some) agent functions

PEAS descriptions define task environments

Environments are categorized along several dimensions:
observable? deterministic? episodic? static? discrete? single-agent?

Several basic agent architectures exist:
reflex, reflex with state, goal-based, utility-based

Problem solving and search

Chapter 3

Reminders

Assignment 0 due 5pm today

Assignment 1 posted, due 2/9

Section 105 will move to 9-10am starting next week

Outline

- ◊ Problem-solving agents
- ◊ Problem types
- ◊ Problem formulation
- ◊ Example problems
- ◊ Basic search algorithms

Problem-solving agents

Restricted form of general agent:

```
function SIMPLE-PROBLEM-SOLVING-AGENT( percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH( problem)
    action ← RECOMMENDATION(seq, state)
    seq ← REMAINDER(seq, state)
    return action
```

Note: this is offline problem solving; solution executed "eyes closed."
Online problem solving involves acting without complete knowledge.

Example: Romania

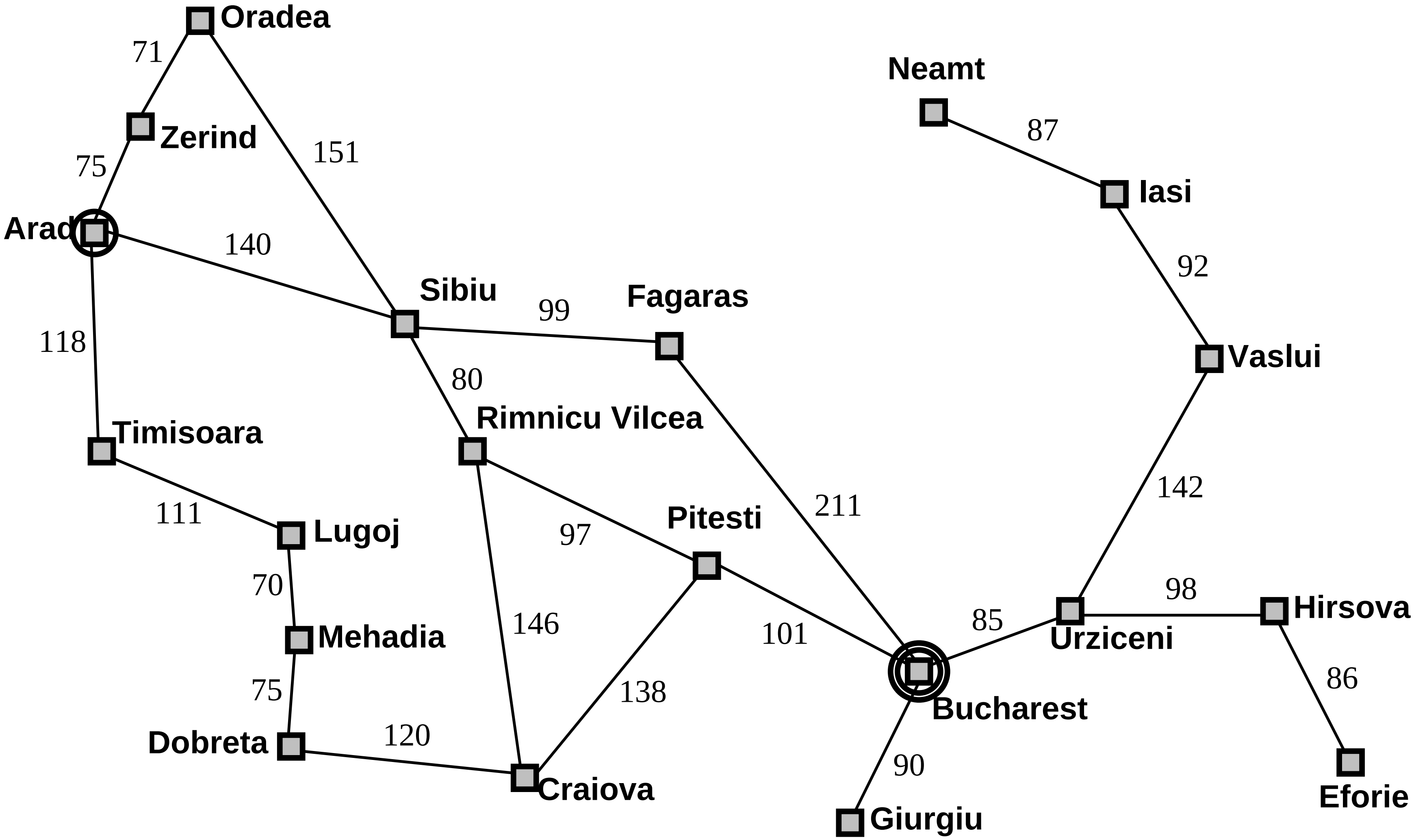
On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest

Formulate goal:
be in Bucharest

Formulate problem:
states: various cities
actions: drive between cities

Find solution:
sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Example: Romania



Problem types

Deterministic, fully observable $\Longrightarrow$ single-state problem

Agent knows exactly which state it will be in; solution is a sequence

Non-observable $\Longrightarrow$ conformant problem

Agent may have no idea where it is; solution (if any) is a sequence

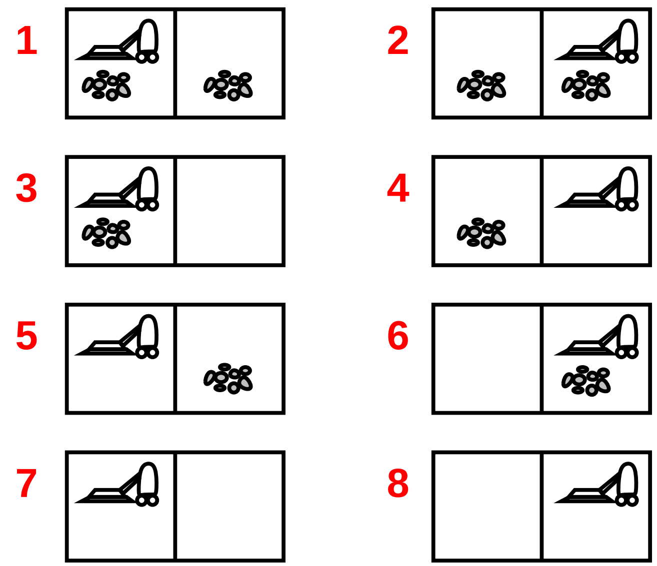
Nondeterministic and/or partially observable $\Longrightarrow$ contingency problem

percepts provide **new** information about current state
solution is a contingent plan or a policy
often **interleave** search, execution

Unknown state space $\Longrightarrow$ exploration problem ("online")

Example: vacuum world

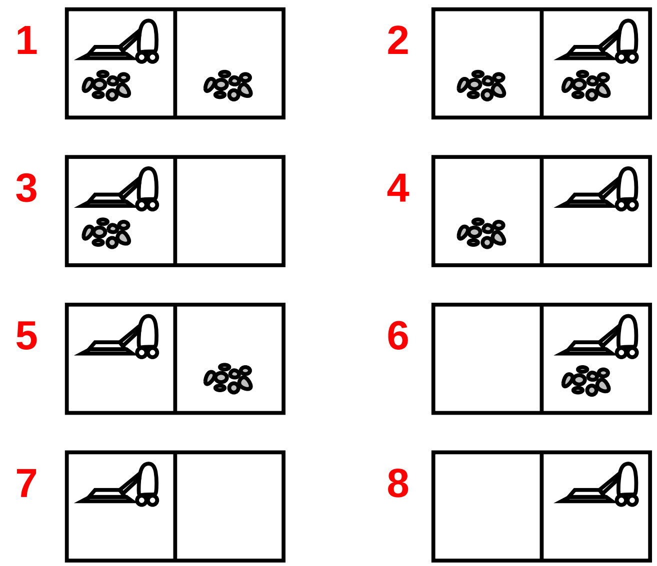
Single-state, start in #5. Solution??



Example: vacuum world

Single-state, start in #5. Solution??
$[Right, Suck]$

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution??

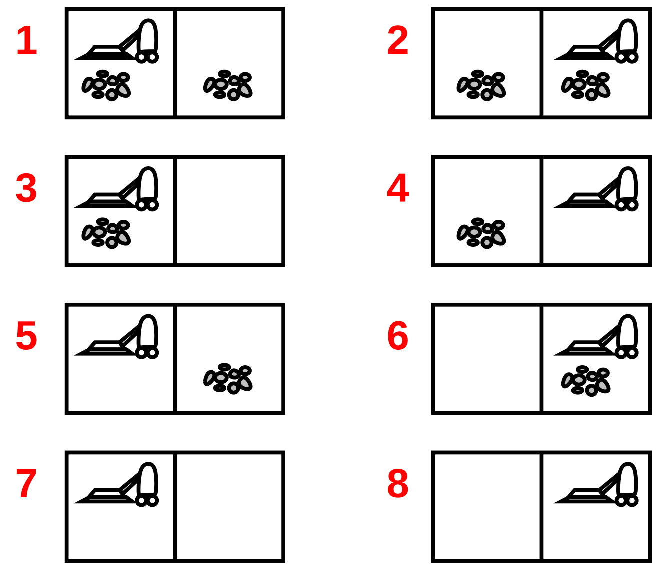


Example: vacuum world

Single-state, start in $\#5$. Solution??
$[Right, Suck]$

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution??
$[Right, Suck, Left, Suck]$

Contingency, start in $\#5$
Murphy's Law: $Suck$ can dirty a clean carpet
Local sensing: dirt, location only.
Solution??

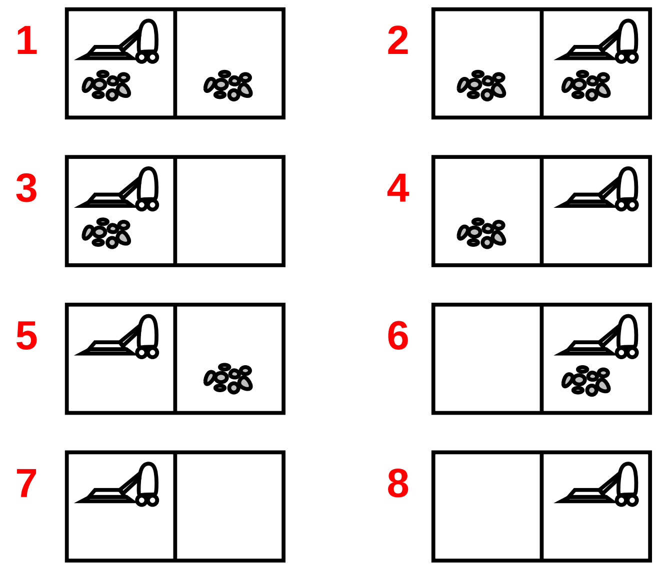


Example: vacuum world

Single-state, start in $\#5$. Solution??
$[Right, Suck]$

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution??
$[Right, Suck, Left, Suck]$

Contingency, start in $\#5$
Murphy's Law: $Suck$ can dirty a clean carpet
Local sensing: dirt, location only.
Solution??
$[Right, \textbf{if } dirt \textbf{ then } Suck]$



Single-state problem formulation

A problem is defined by four items:

initial state e.g., "at Arad"

successor function $S(x)$ = set of action–state pairs
  e.g., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind\rangle, \ldots\}$

goal test, can be
  explicit, e.g., x = "at Bucharest"
  implicit, e.g., $NoDirt(x)$

path cost (additive)
  e.g., sum of distances, number of actions executed, etc.
  $c(x, a, y)$ is the step cost, assumed to be ≥ 0

A solution is a sequence of actions
leading from the initial state to a goal state

Selecting a state space

Real world is absurdly complex

$\Rightarrow$ state space must be **abstracted** for problem solving

(Abstract) state = set of real states

(Abstract) action = complex combination of real actions

e.g., "Arad $\rightarrow$ Zerind" represents a complex set

of possible routes, detours, rest stops, etc.

For guaranteed realizability, **any** real state "in Arad"

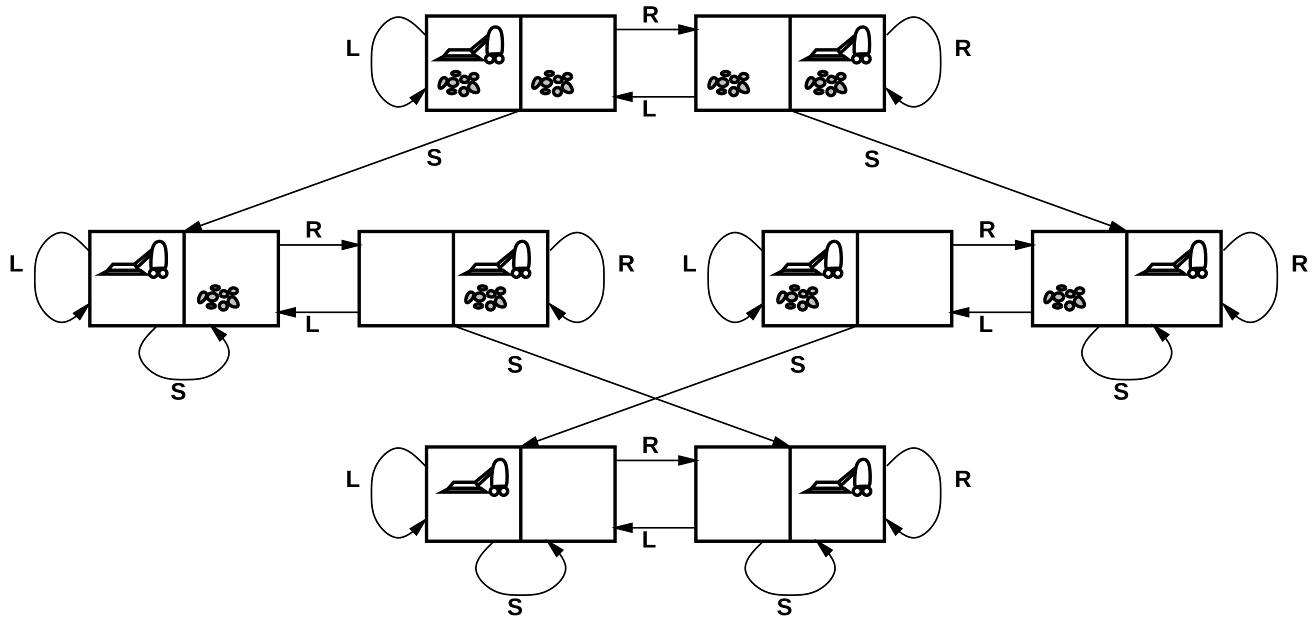
must get to some real state "in Zerind"

(Abstract) solution =

set of real paths that are solutions in the real world

Each abstract action should be "easier" than the original problem!

Example: vacuum world state space graph



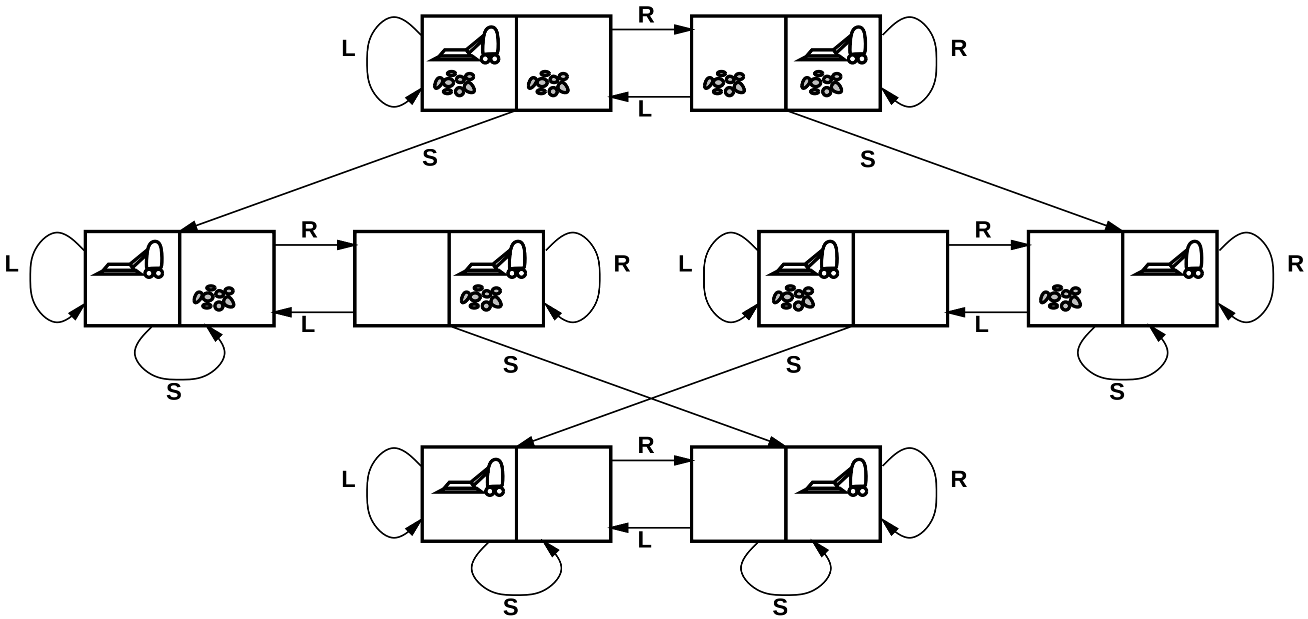
states??

actions??

goal test??

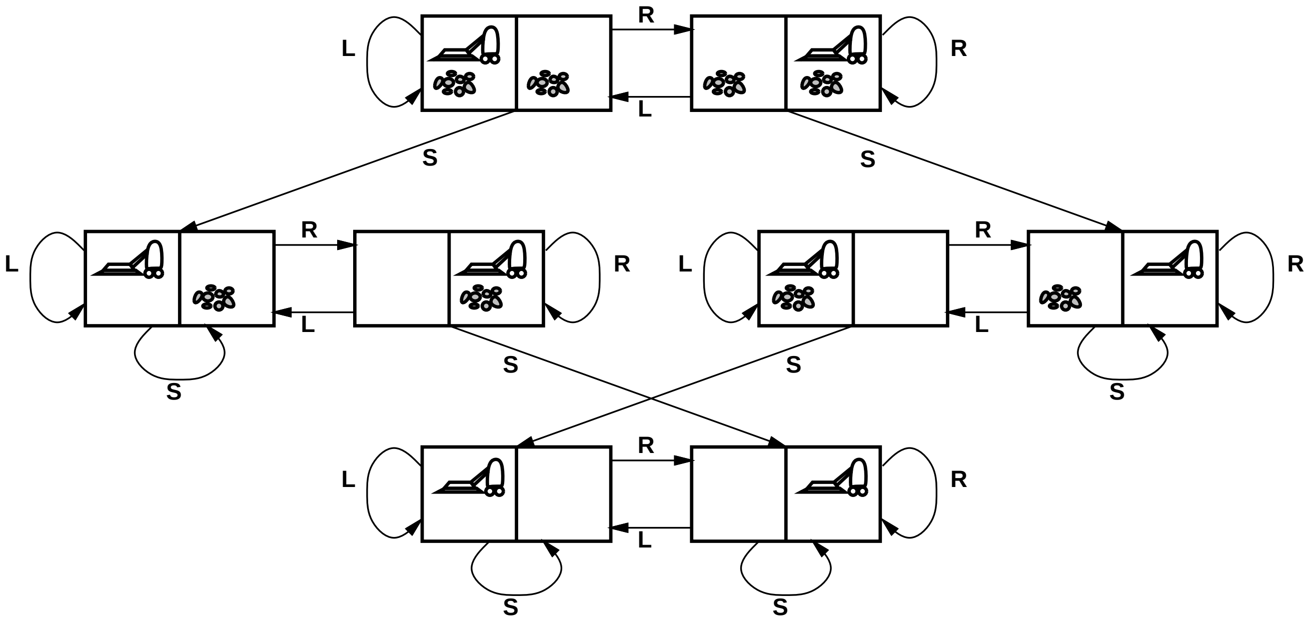
path cost??

Example: vacuum world state space graph



states??: integer dirt and robot locations (ignore dirt amounts etc.)
actions??
goal test??
path cost??

Example: vacuum world state space graph



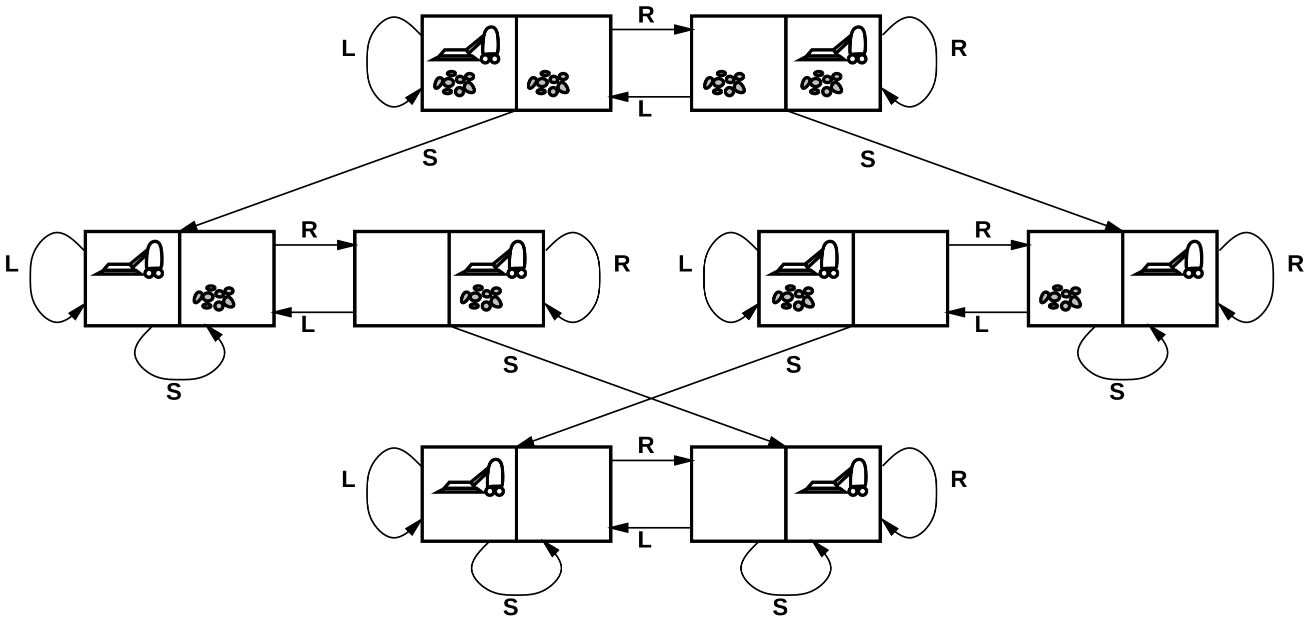
states??: integer dirt and robot locations (ignore dirt amounts etc.)

actions??: $Left$, $Right$, $Suck$, $NoOp$

goal test??

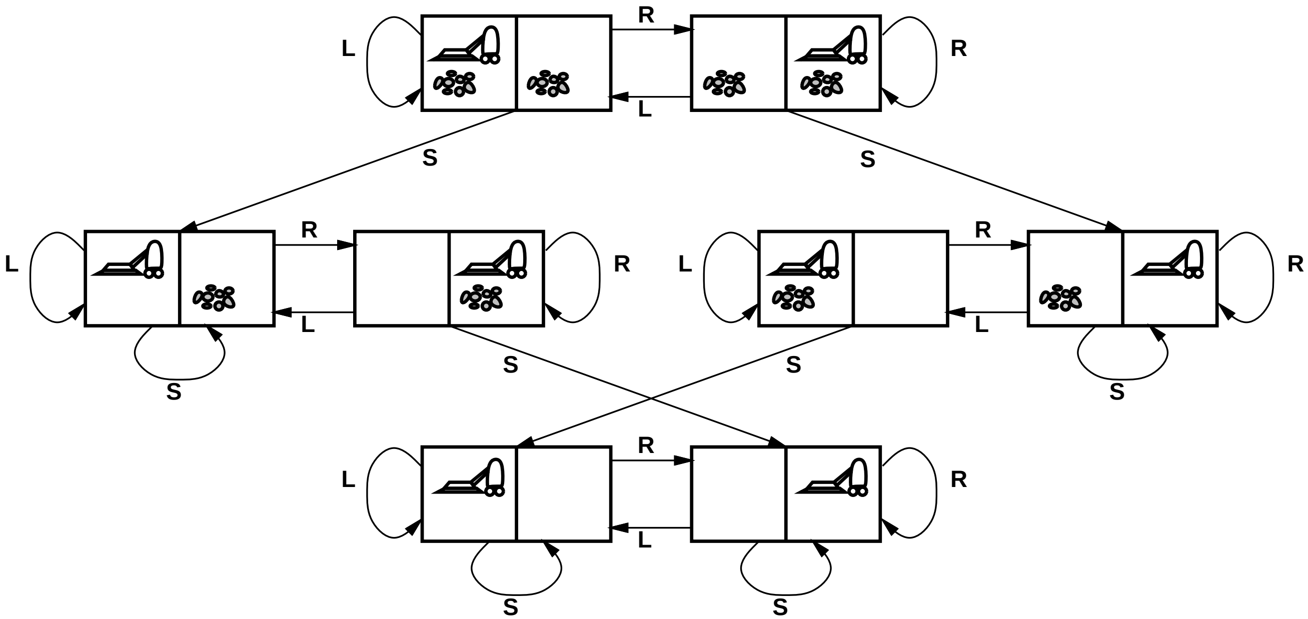
path cost??

Example: vacuum world state space graph



states??: integer dirt and robot locations (ignore dirt amounts etc.)
actions??: $Left$, $Right$, $Suck$, $NoOp$
goal test??: no dirt
path cost??

Example: vacuum world state space graph



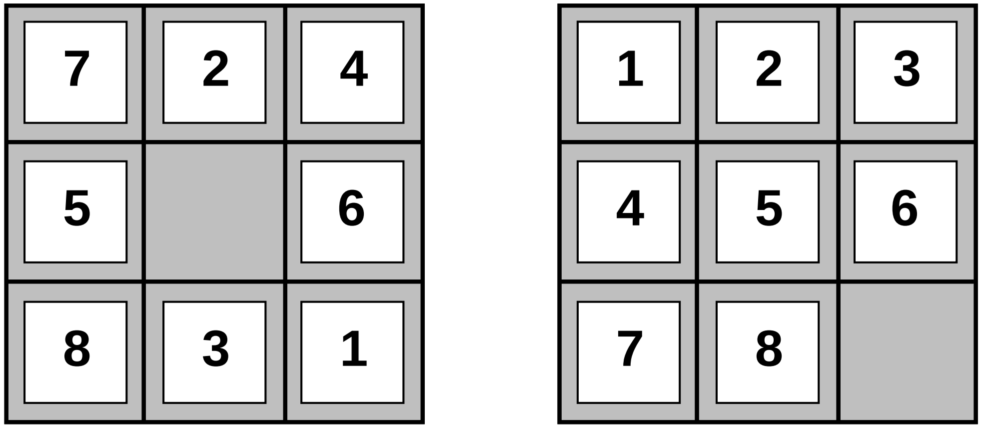
states??: integer dirt and robot locations (ignore dirt amounts etc.)

actions??: *Left*, *Right*, *Suck*, *NoOp*

goal test??: no dirt

path cost??: 1 per action (0 for *NoOp*)

Example: The 8-puzzle



Start State Goal State

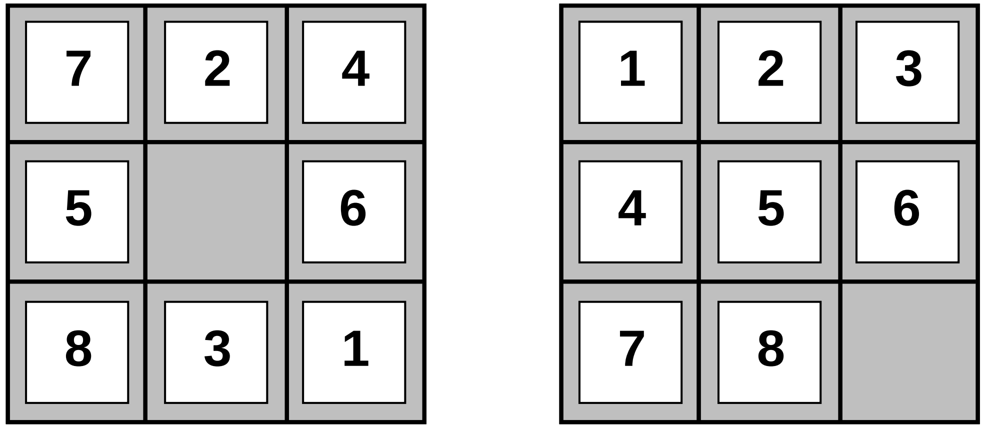
states??

actions??

goal test??

path cost??

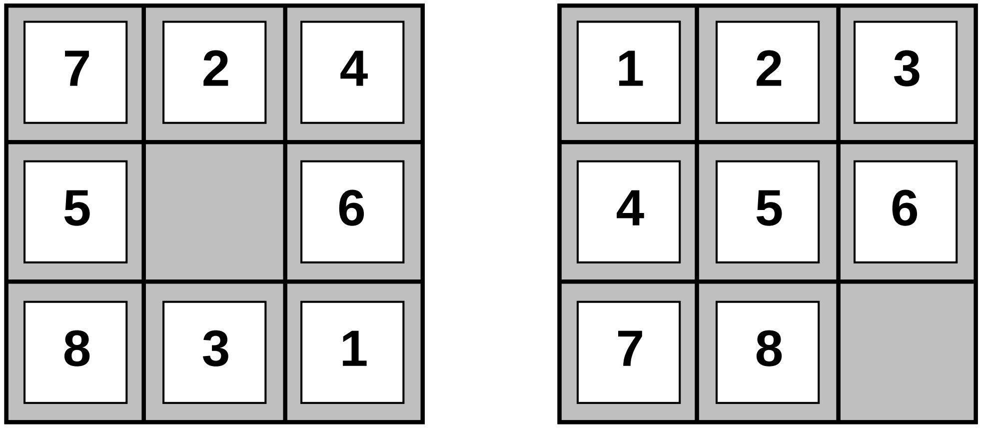
Example: The 8-puzzle



Start State Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??
goal test??
path cost??

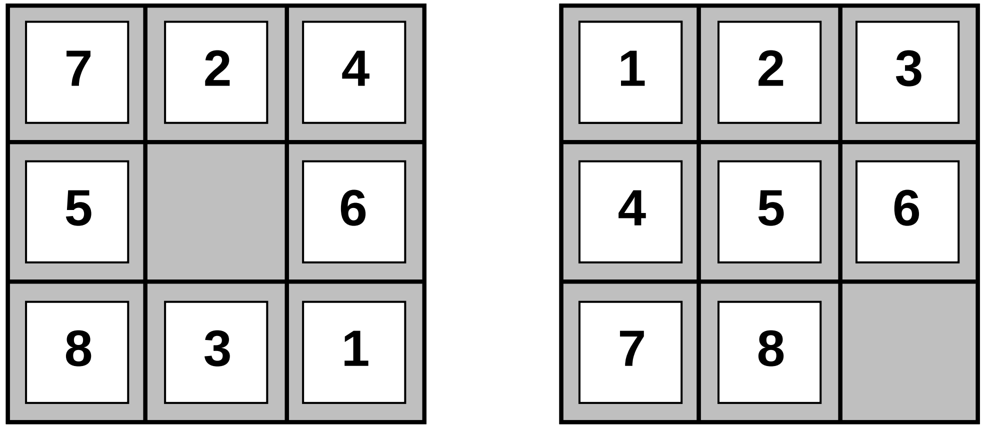
Example: The 8-puzzle



Start State **Goal State**

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??
path cost??

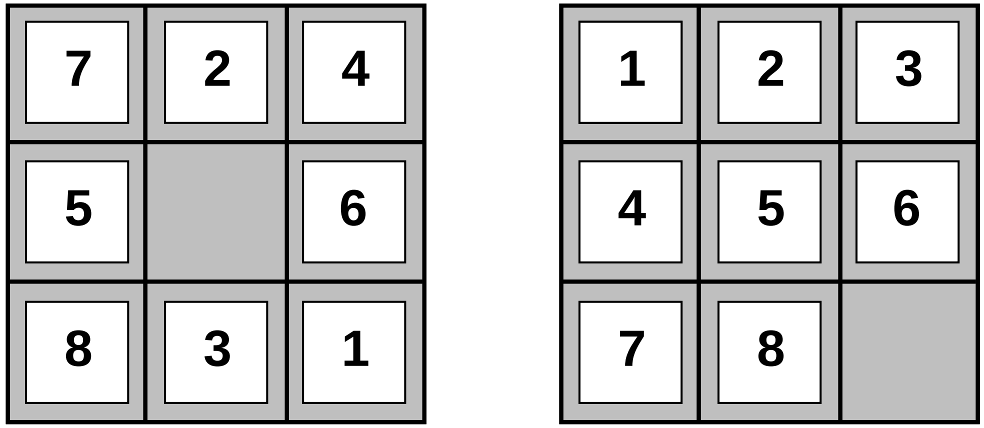
Example: The 8-puzzle



Start State Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??: = goal state (given)
path cost??

Example: The 8-puzzle

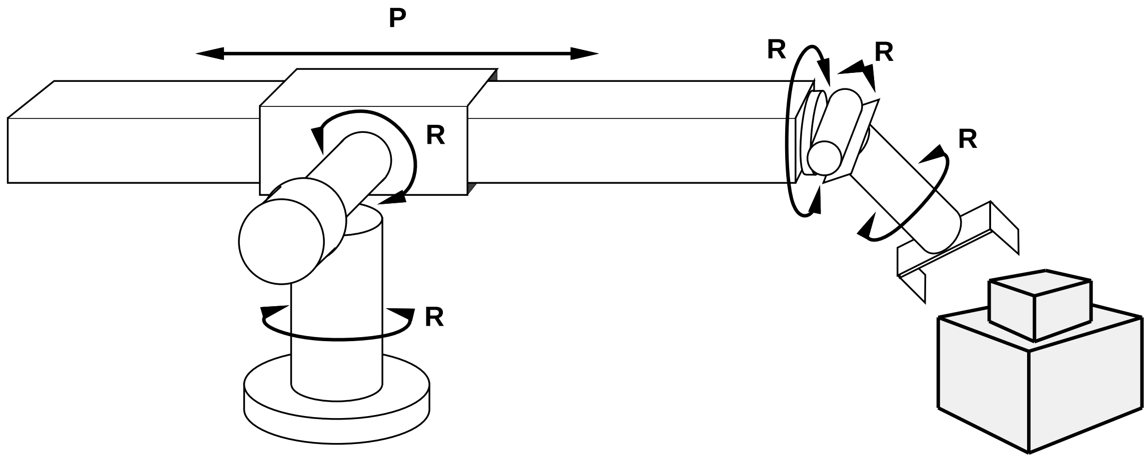


Start State **Goal State**

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??: $=$ goal state (given)
path cost??: 1 per move

[Note: optimal solution of n-Puzzle family is NP-hard]

Example: robotic assembly



states??: real-valued coordinates of robot joint angles
parts of the object to be assembled

actions??: continuous motions of robot joints

goal test??: complete assembly **with no robot included!**

path cost??: time to execute

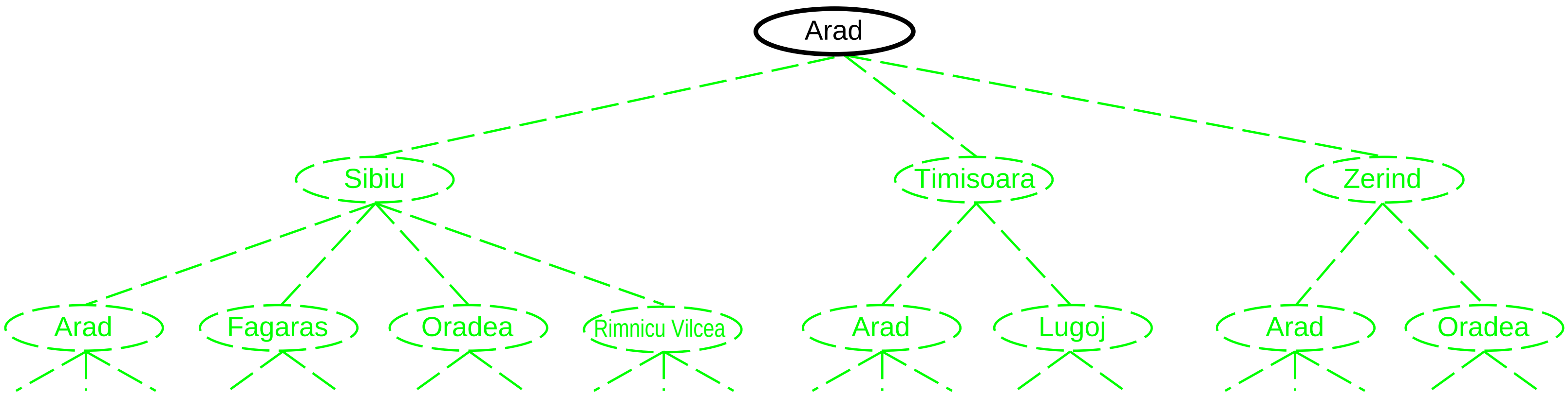
Tree search algorithms

Basic idea:

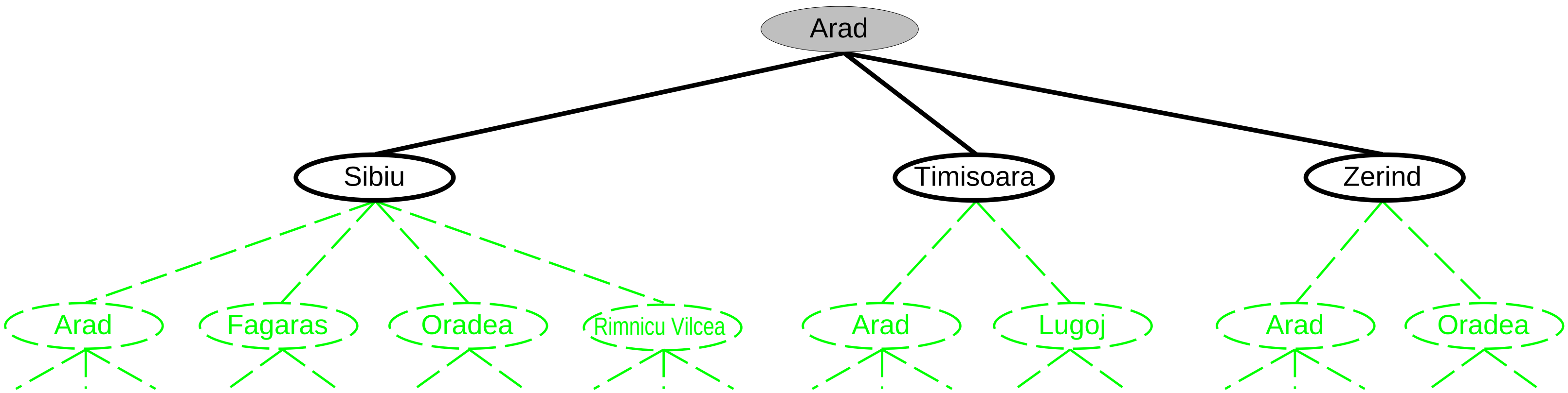
offline, simulated exploration of state space
by generating successors of already-explored states
(a.k.a. expanding states)

```
function Tree-Search( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

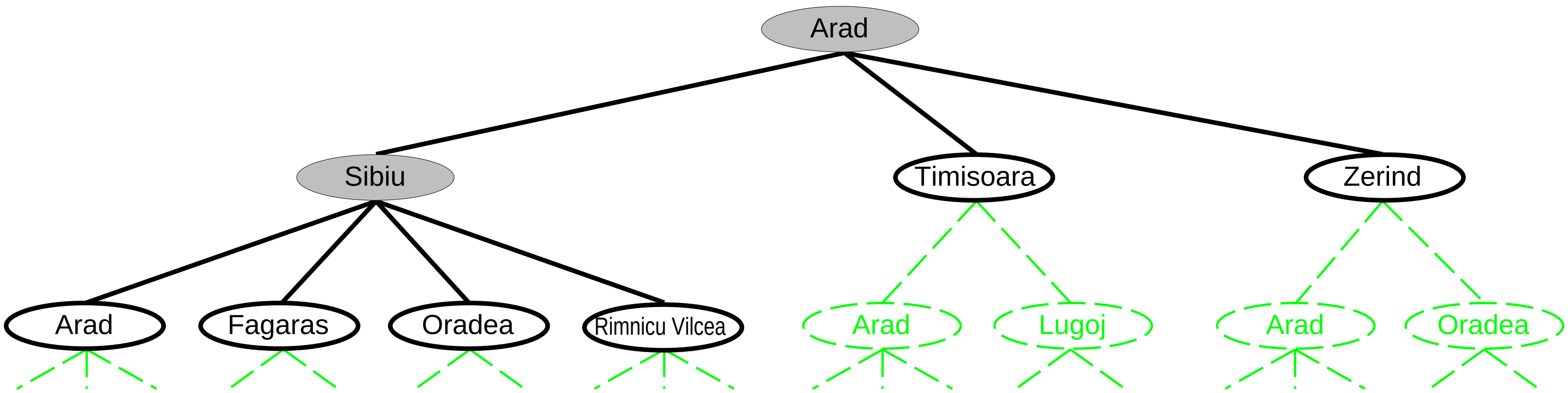
Tree search example



Tree search example



Tree search example



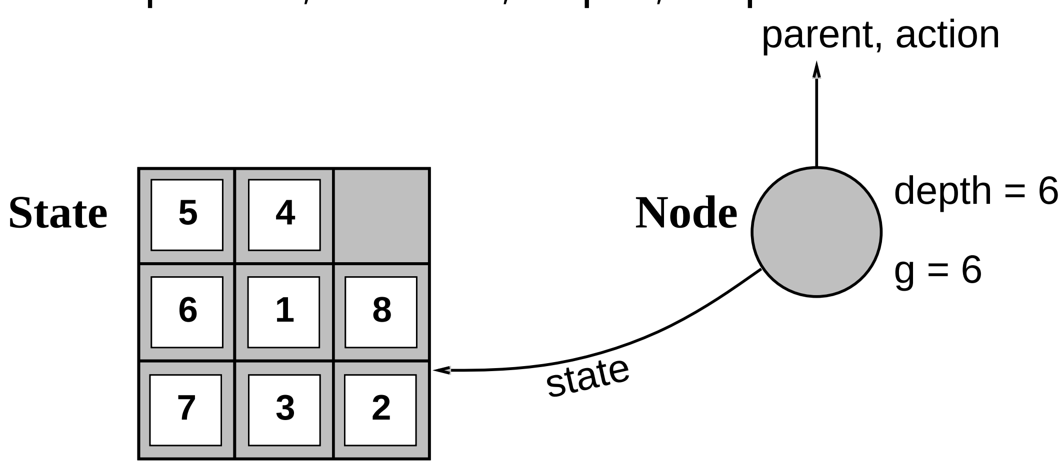
Implementation: states vs. nodes

A state is a (representation of) a physical configuration

A node is a data structure constituting part of a search tree

includes parent, children, depth, path cost $g(x)$

States do not have parents, children, depth, or path cost!



The EXPAND function creates new nodes, filling in the various fields and using the SUCCESSORFN of the problem to create the corresponding states.

Implementation: general tree search

```
function Tree-Search( problem, fringe) returns a solution, or failure
    fringe ← Insert(Make-Node(Initial-State[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← Remove-Front(fringe)
        if Goal-Test(problem, State(node)) then return node
        fringe ← InsertAll(Expand(node, problem), fringe)
```

```
function Expand( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in Successor-Fn(problem, State[node]) do
        s ← a new Node
        Parent-Node[s] ← node;  Action[s] ← action;  State[s] ← result
        Path-Cost[s] ← Path-Cost[node] + Step-Cost(node, action, s)
        Depth[s] ← Depth[node] + 1
        add s to successors
    return successors
```

Search strategies

A strategy is defined by picking the **order of node expansion**

Strategies are evaluated along the following dimensions:

- completeness—does it always find a solution if one exists?
- time complexity—number of nodes generated/expanded
- space complexity—maximum number of nodes in memory
- optimality—does it always find a least-cost solution?

Time and space complexity are measured in terms of

- b—maximum branching factor of the search tree
- d—depth of the least-cost solution
- m—maximum depth of the state space (may be ∞)

Uninformed search strategies

Uninformed strategies use only the information available
in the problem definition

Breadth-first search

Uniform-cost search

Depth-first search

Depth-limited search

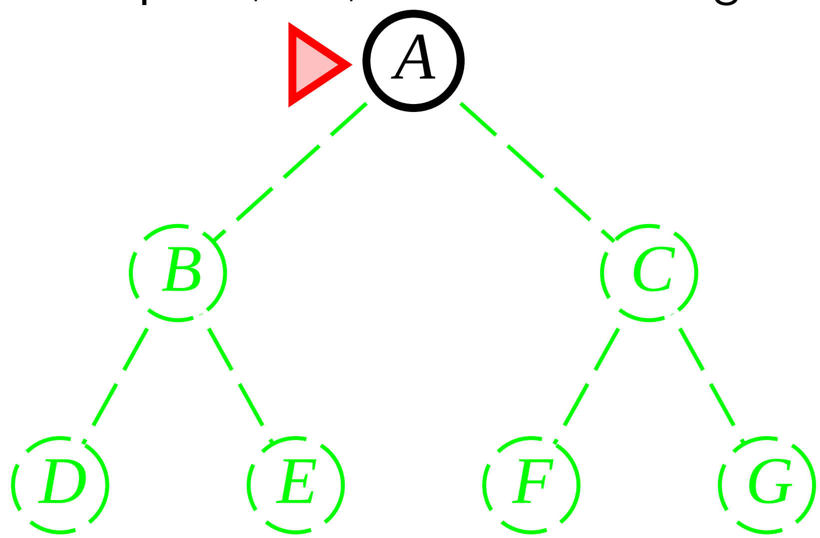
Iterative deepening search

Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

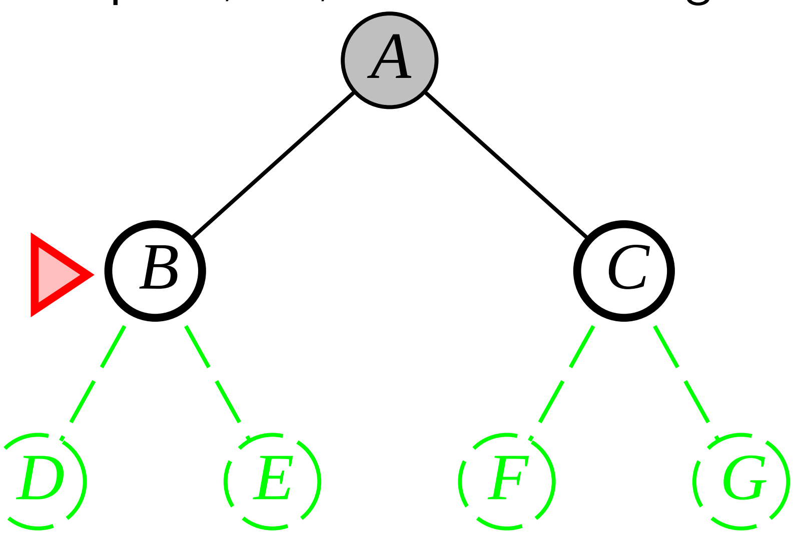


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

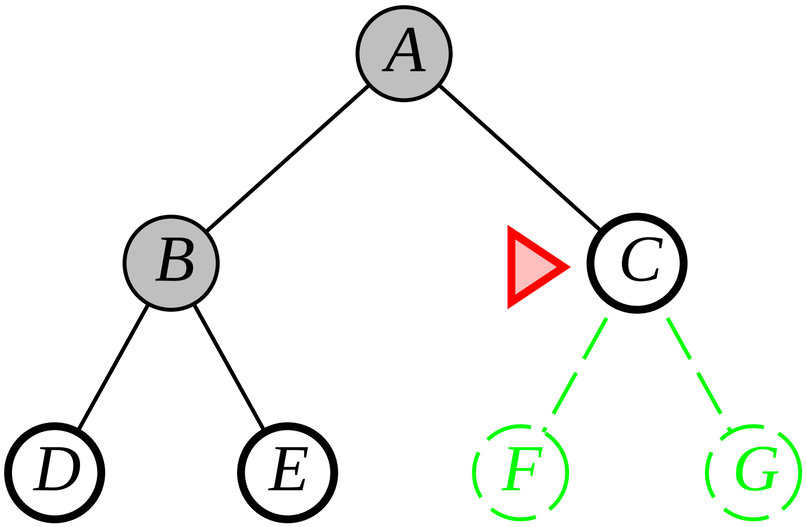


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

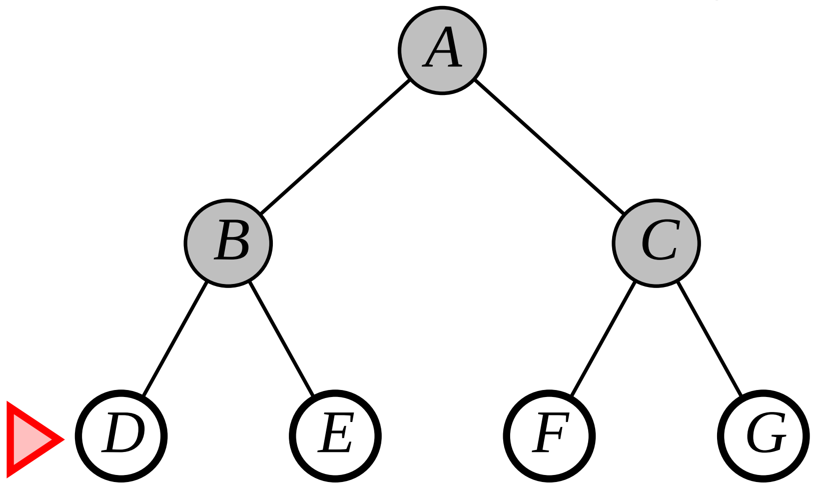


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end



Properties of breadth-first search

Complete??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal?? Yes (if cost $= 1$ per step); not optimal in general

Space is the big problem; can easily generate nodes at 100MB/sec
so 24hrs $=$ 8640GB.

Uniform-cost search

Expand least-cost unexpanded node

Implementation:
$fringe =$ queue ordered by path cost, lowest first

Equivalent to breadth-first if step costs all equal

Complete?? Yes, if step cost $\geq \epsilon$

Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

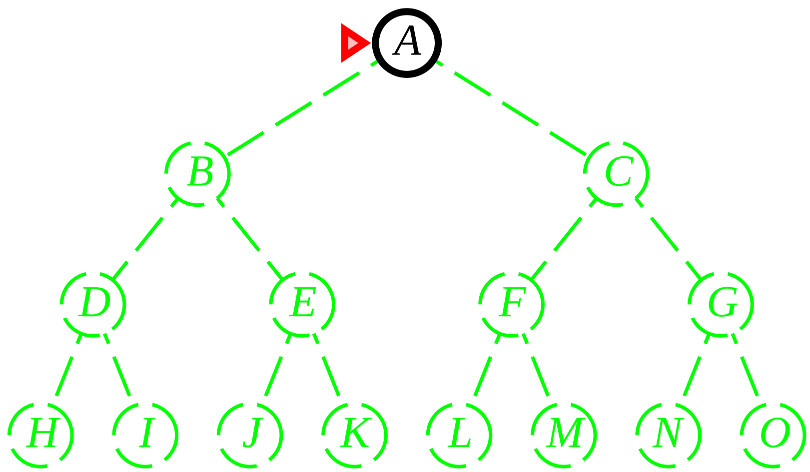
Optimal?? Yes—nodes expanded in increasing order of $g(n)$

Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

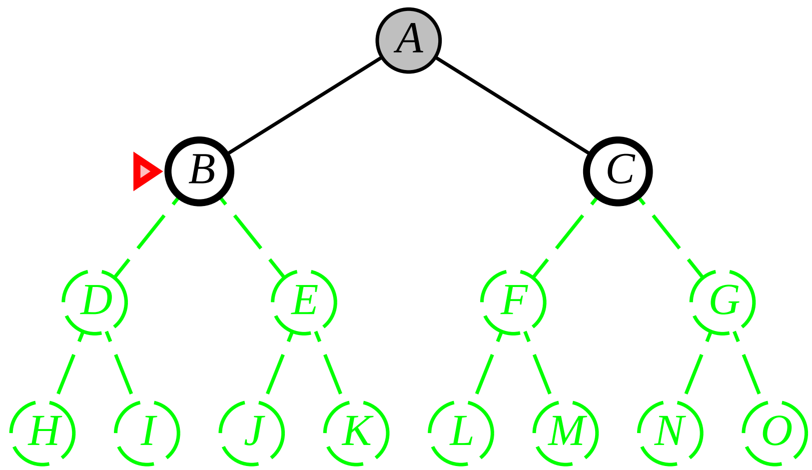


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

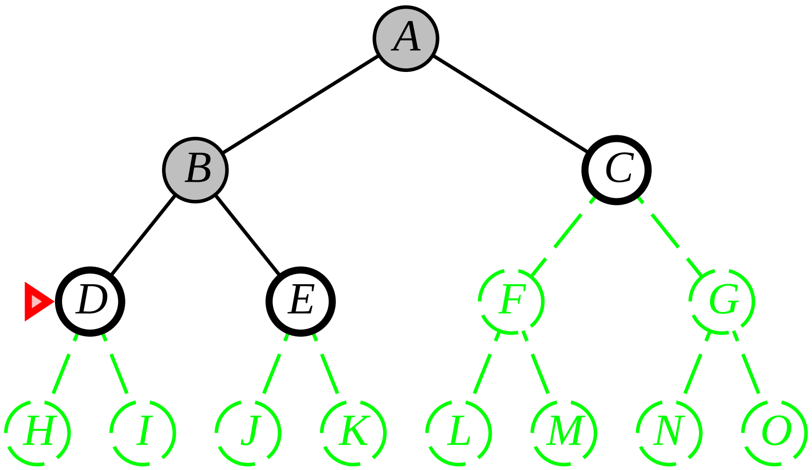


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

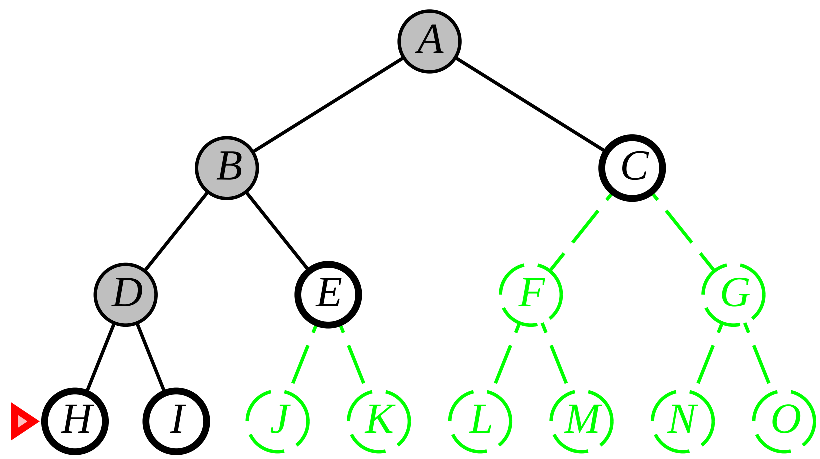


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe =$ LIFO queue, i.e., put successors at front

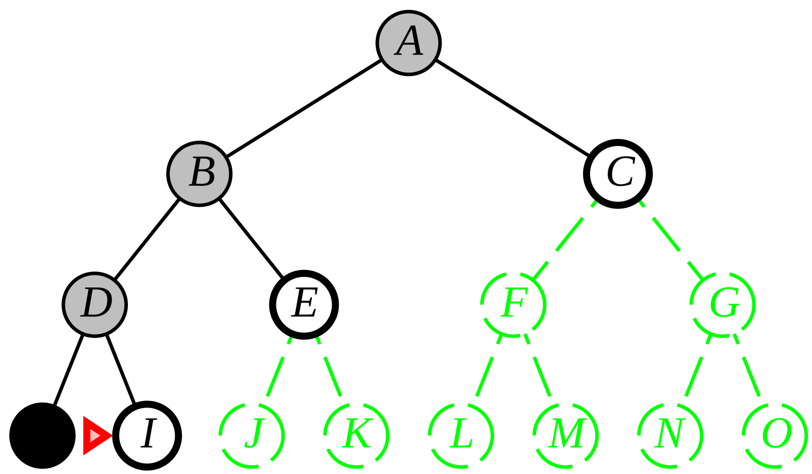


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

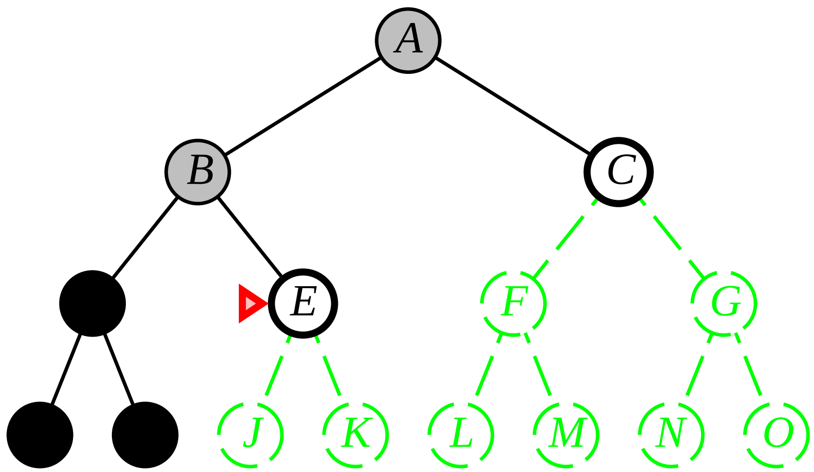


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

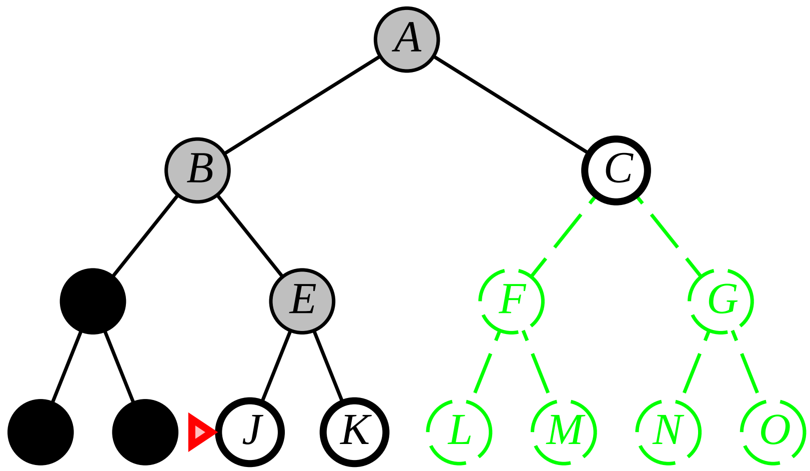


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

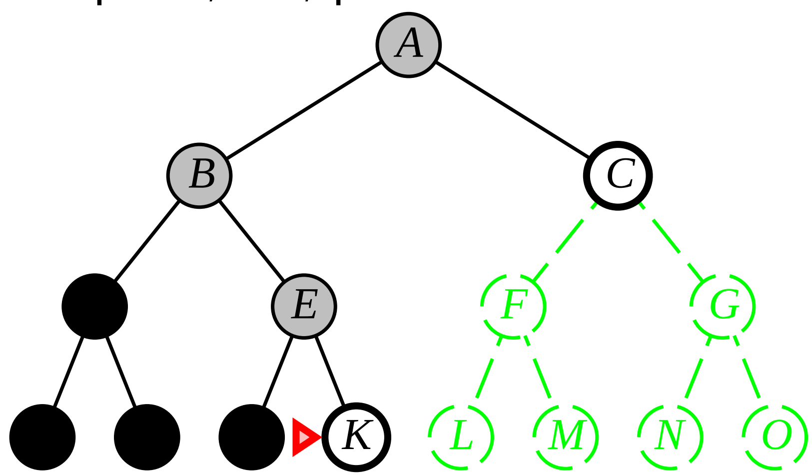


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

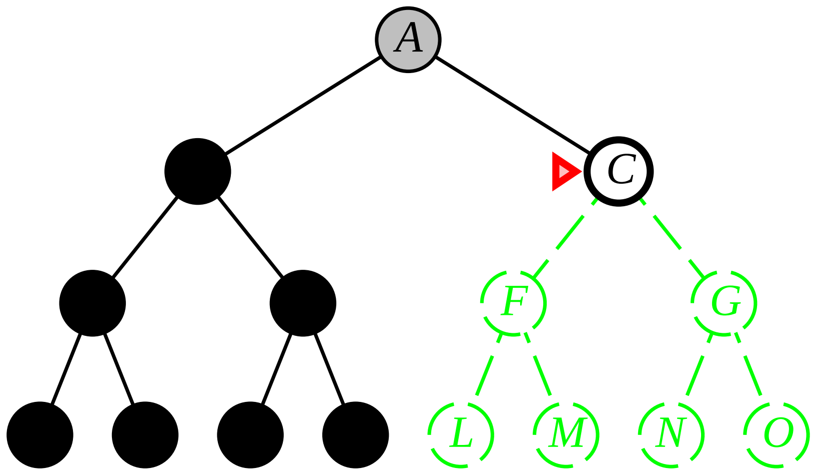


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

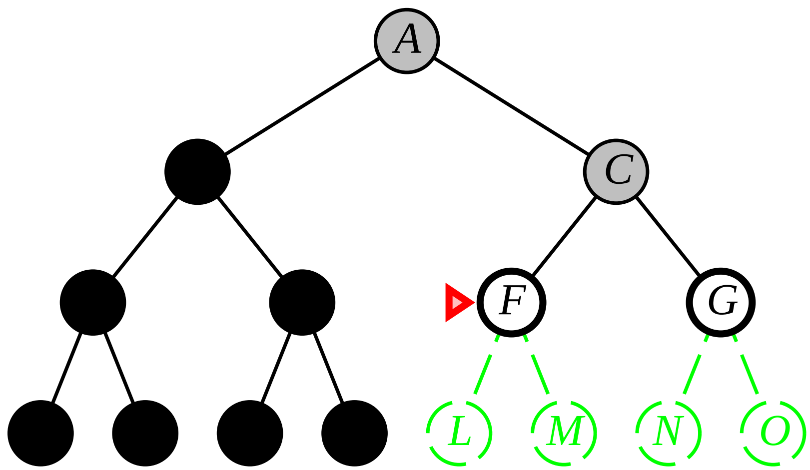


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

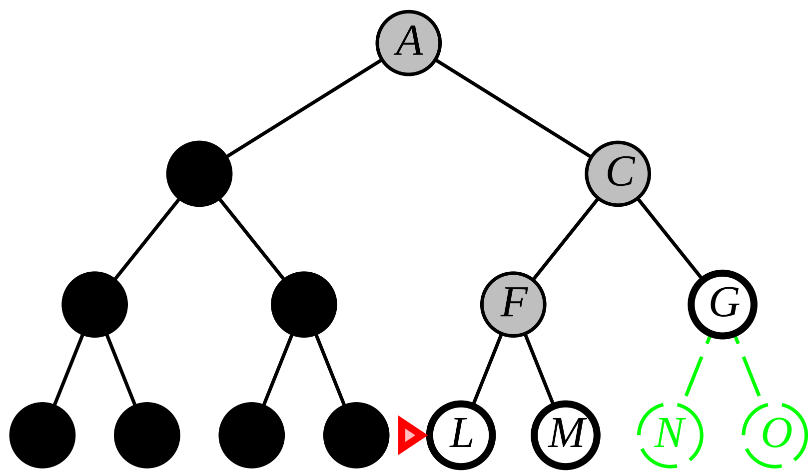


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

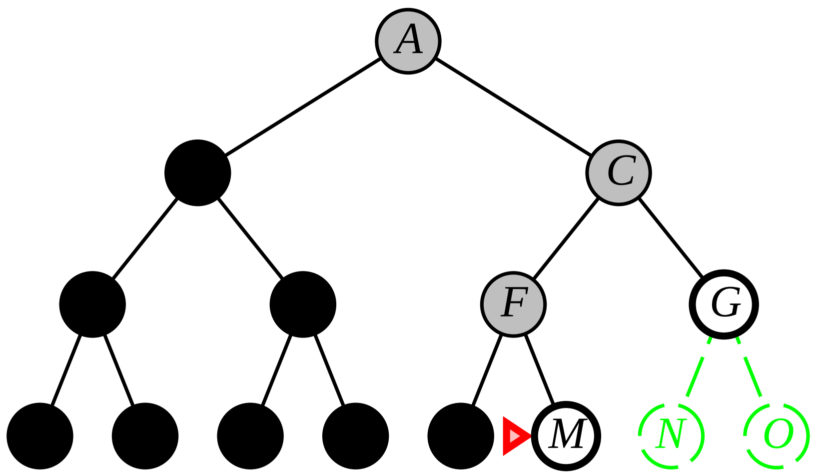


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front



Properties of depth-first search

Complete??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal?? No

Depth-limited search

= depth-first search with depth limit l,
i.e., nodes at depth l have no successors

Recursive implementation:

```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
   RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
   cutoff-occurred? ← false
   if GOAL-TEST(problem, STATE[node]) then return node
   else if DEPTH[node] = limit then return cutoff
   else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
   if cutoff-occurred? then return cutoff else return failure
```

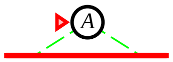
Iterative deepening search

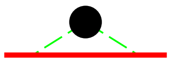
function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution
  inputs: *problem*, a problem

  for *depth* ← 0 **to** ∞ **do**
    result ← DEPTH-LIMITED-SEARCH(*problem, depth*)
    if *result* ≠ cutoff **then return** *result*
  end

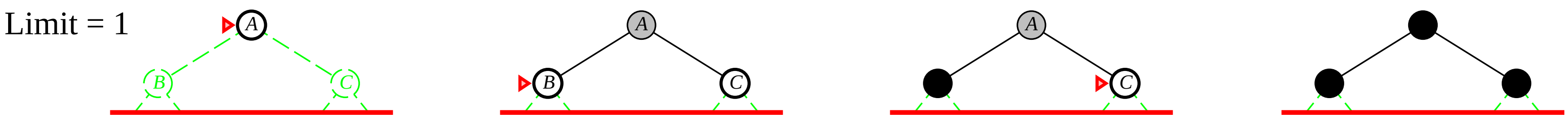
Iterative deepening search $l = 0$

Limit = 0

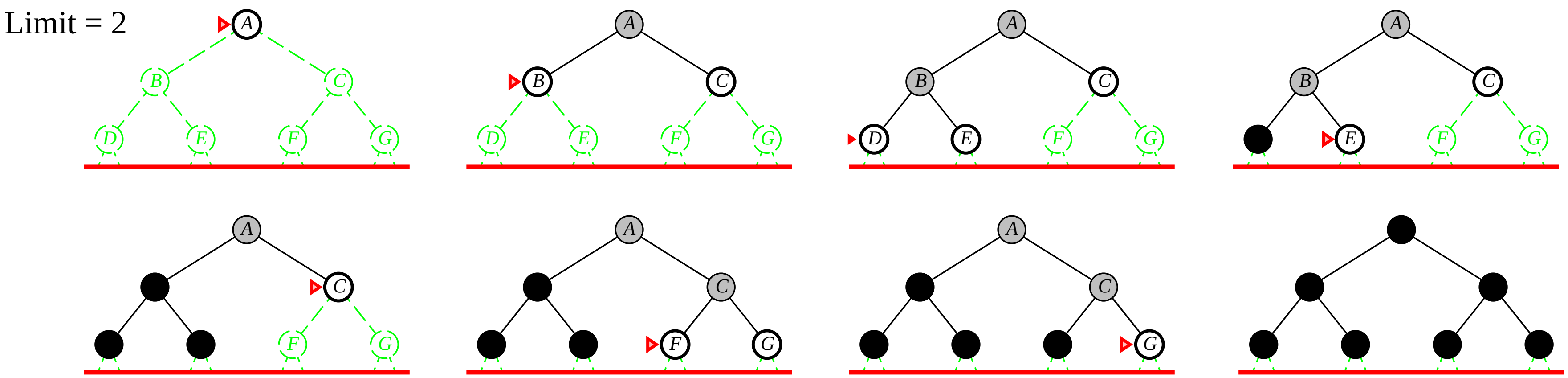




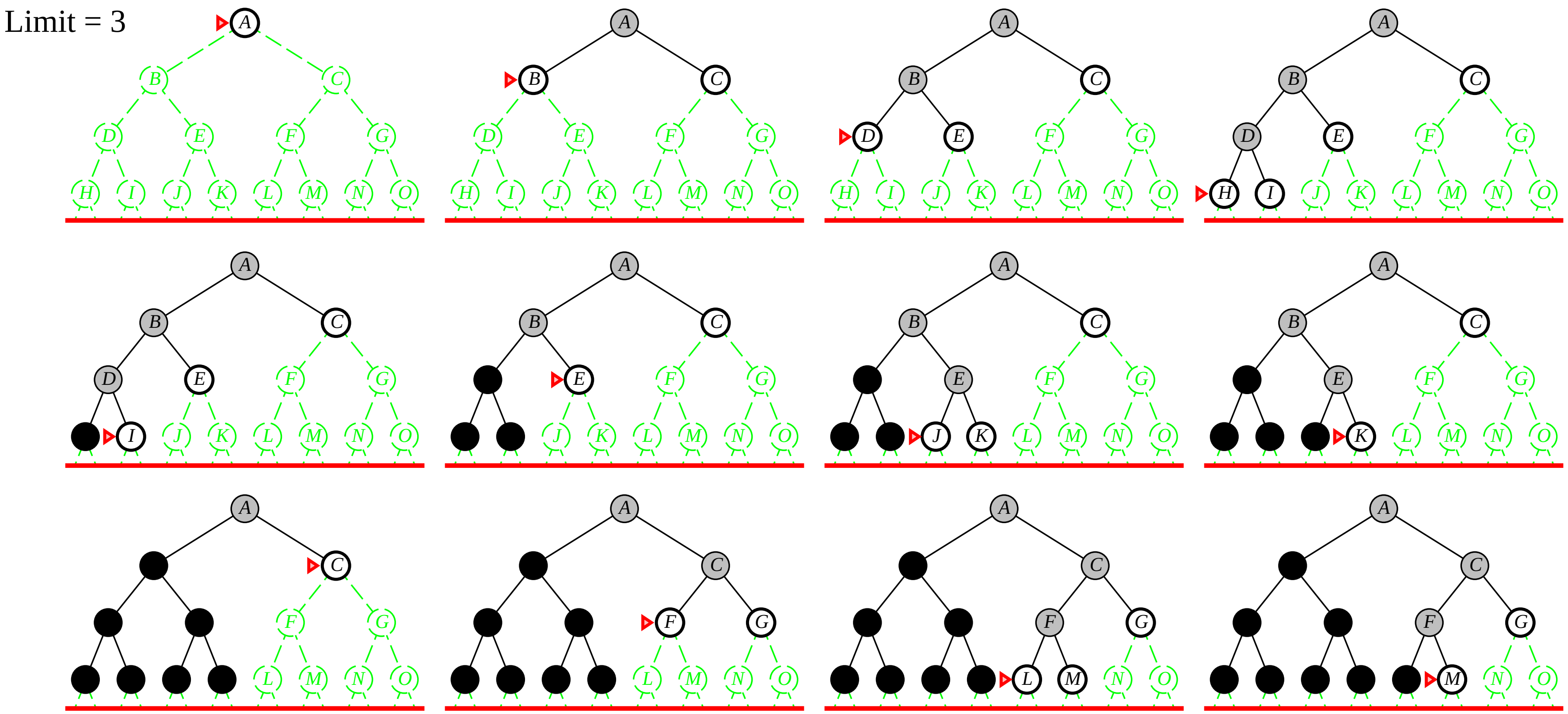
Iterative deepening search $l = 1$



Iterative deepening search $l = 2$



Iterative deepening search $l = 3$



Properties of iterative deepening search

Complete??

Properties of iterative deepening search

Complete?? Yes

Time??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes, if step cost $= 1$
Can be modified to explore uniform-cost tree

Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$
$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$$

IDS does better because other nodes at depth d are not expanded

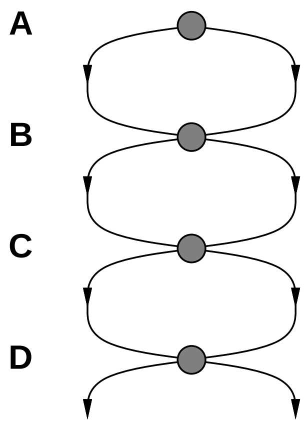
BFS can be modified to apply goal test when a node is **generated**

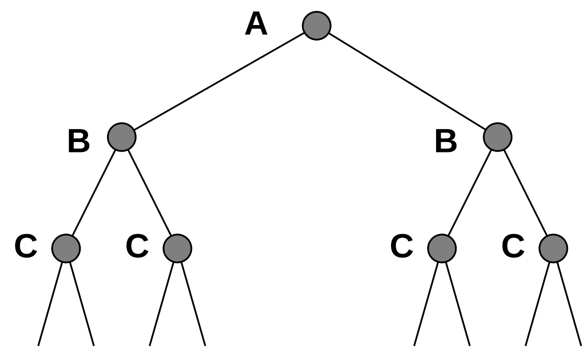
Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!





Graph search

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

Summary

Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored

Variety of uninformed search strategies

Iterative deepening search uses only linear space
and not much more time than other uninformed algorithms

Graph search can be exponentially more efficient than tree search

Informed search algorithms

Chapter 4, Sections 1–2

Outline

◇ Best-first search

◇ A^* search

◇ Heuristics

Review: Tree search

```
function Tree-Search( problem, fringe) returns a solution, or failure
    fringe ← Insert(Make-Node(Initial-State[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← Remove-Front(fringe)
        if Goal-Test[problem] applied to State(node) succeeds return node
        fringe ← InsertAll(Expand(node, problem), fringe)
```

A strategy is defined by picking the **order of node expansion**

Best-first search

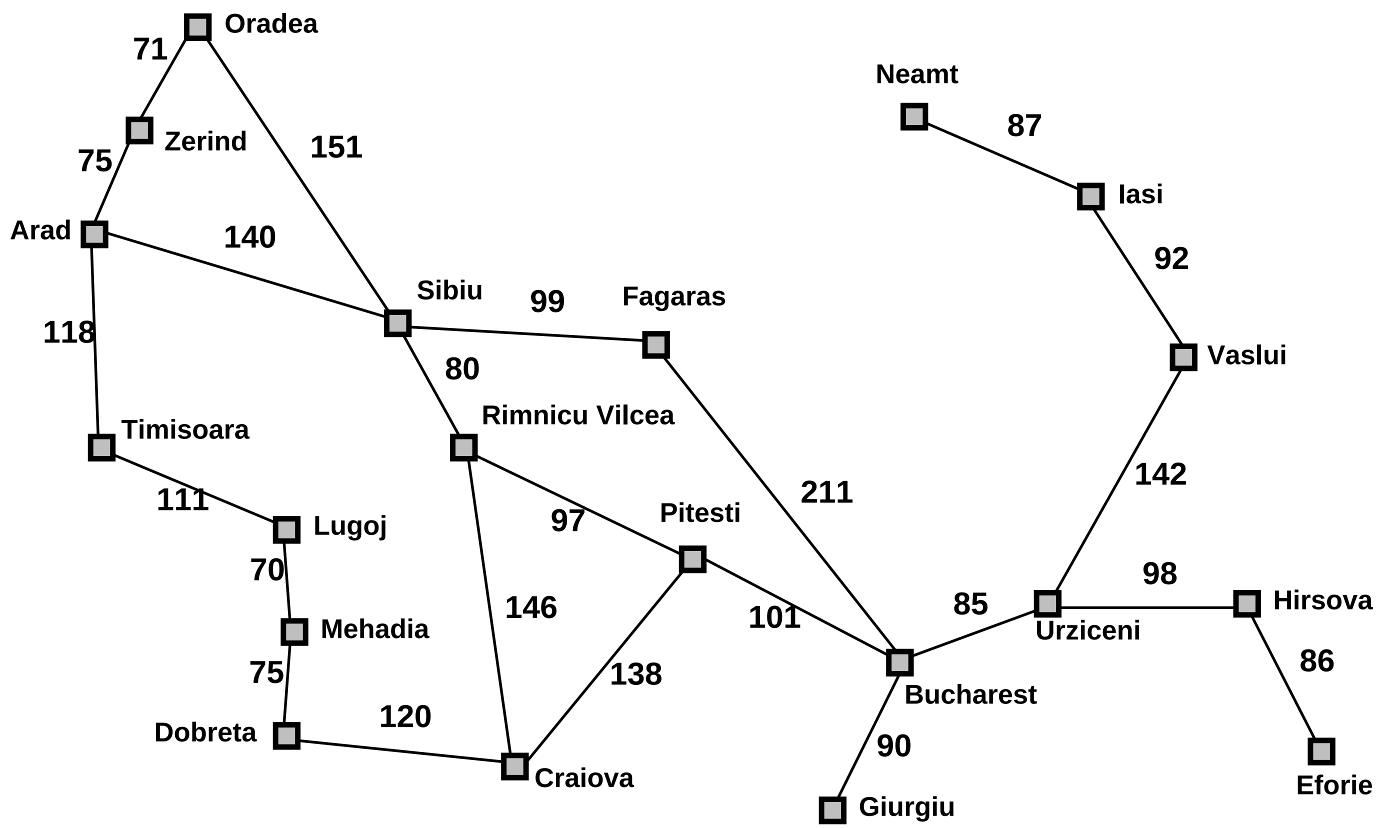
Idea: use an evaluation function for each node
– estimate of "desirability"

⇒ Expand most desirable unexpanded node

Implementation:
fringe is a queue sorted in decreasing order of desirability

Special cases:
greedy search
A^* search

Romania with step costs in km



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

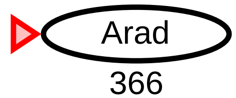
Greedy search

Evaluation function $h(n)$ (**h**euristic)
$= $ estimate of cost from n to the closest goal

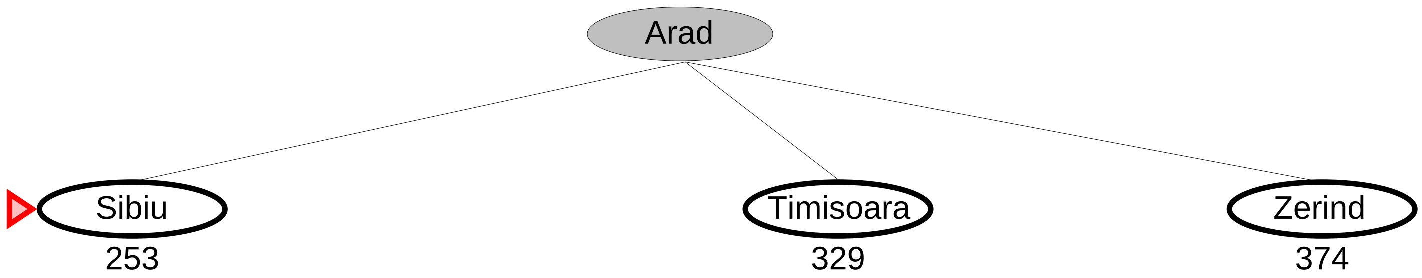
E.g., $h_{\mathrm{SLD}}(n) = $ straight-line distance from n to Bucharest

Greedy search expands the node that **appears** to be closest to goal

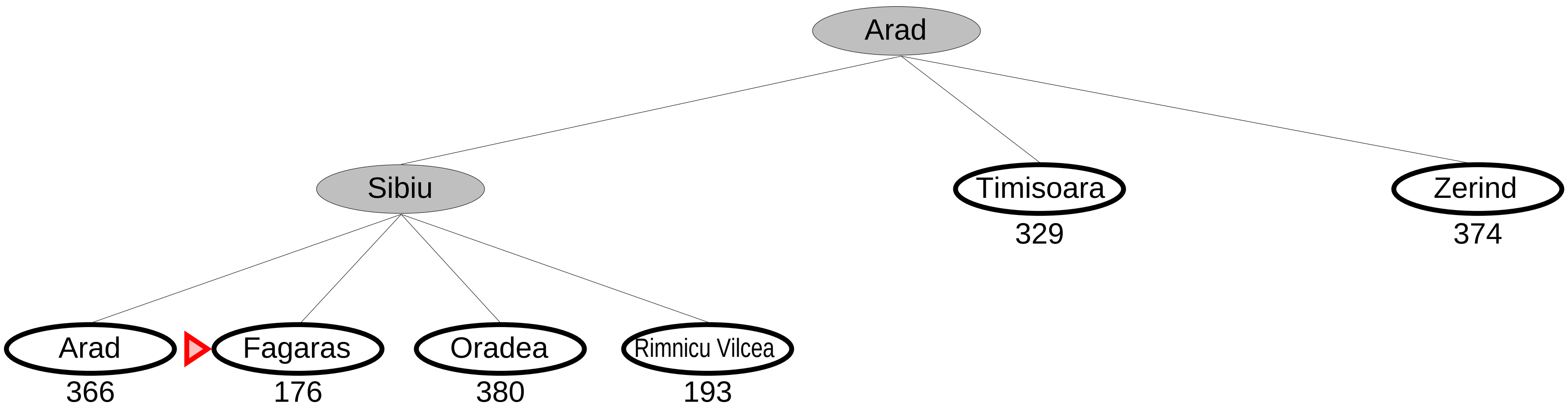
Greedy search example



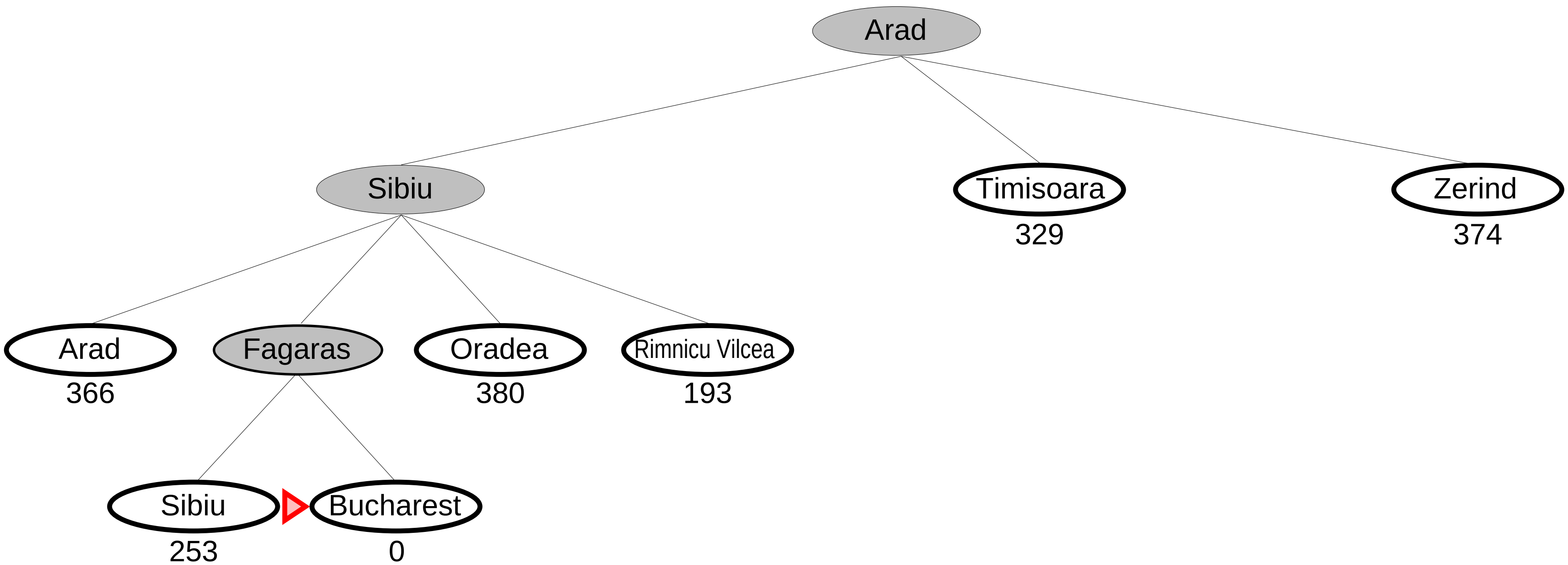
Greedy search example



Greedy search example



Greedy search example



Properties of greedy search

Complete??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with Oradea as goal,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal?? No

A* search

Idea: avoid expanding paths that are already expensive

Evaluation function $f(n) = g(n) + h(n)$

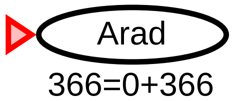
$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

A* search uses an admissible heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the **true** cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

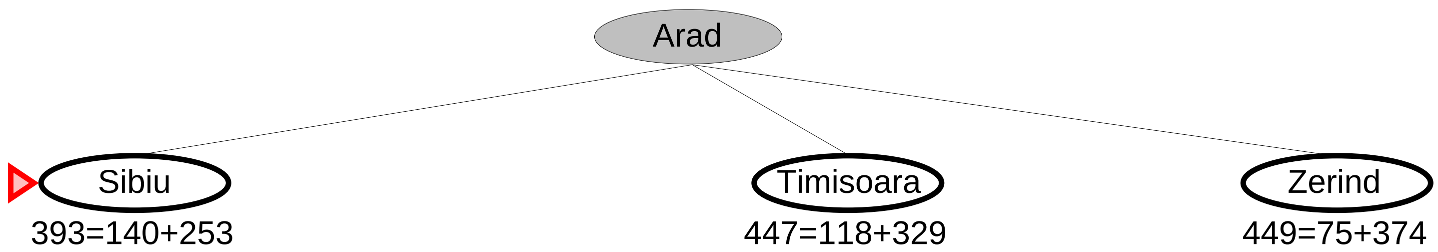
E.g., $h_{\text{SLD}}(n)$ never overestimates the actual road distance

Theorem: A* search is optimal

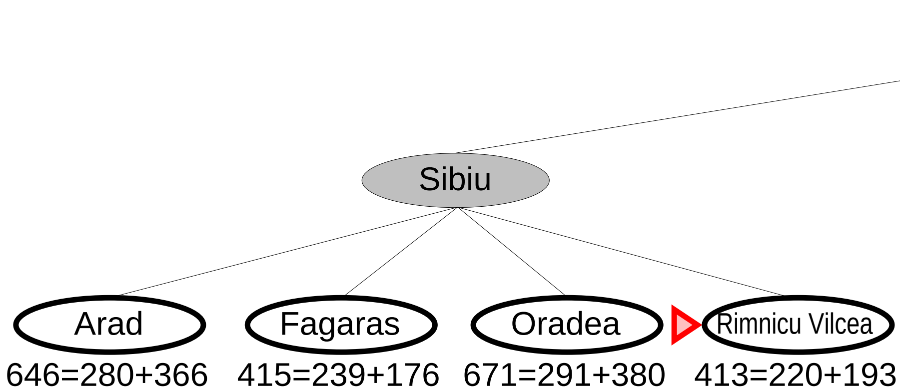
A* search example



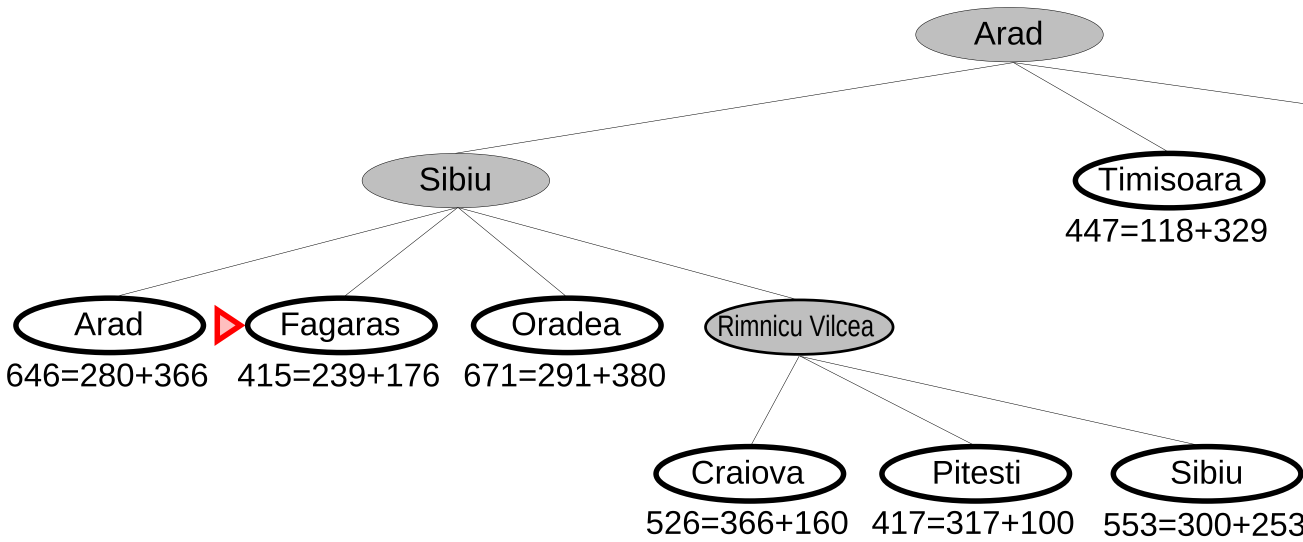
A* search example



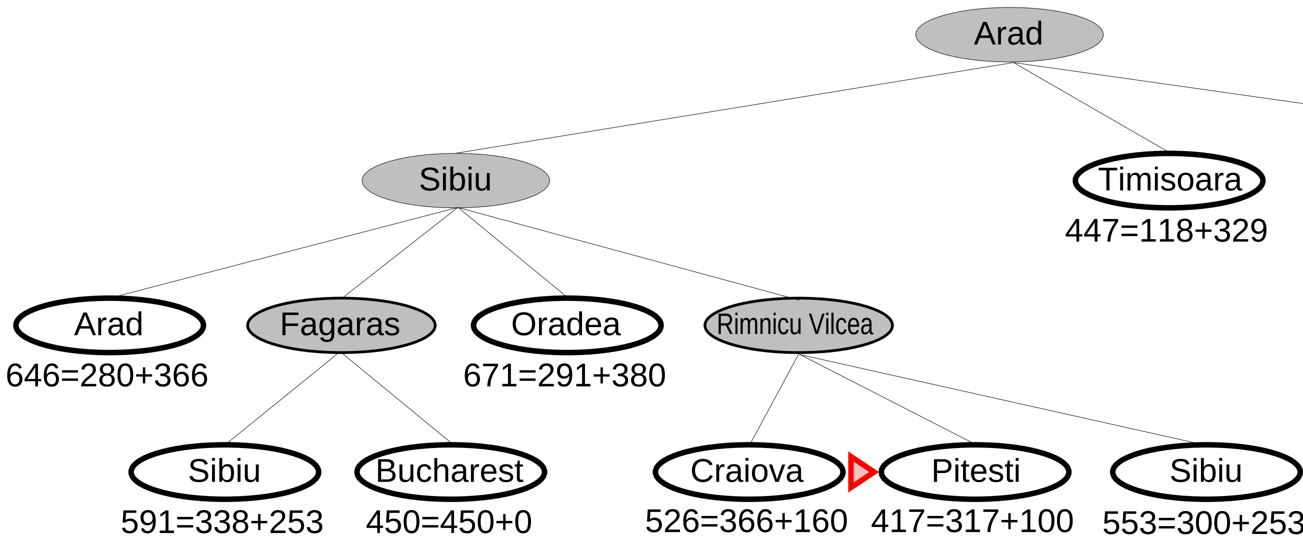
A* search example



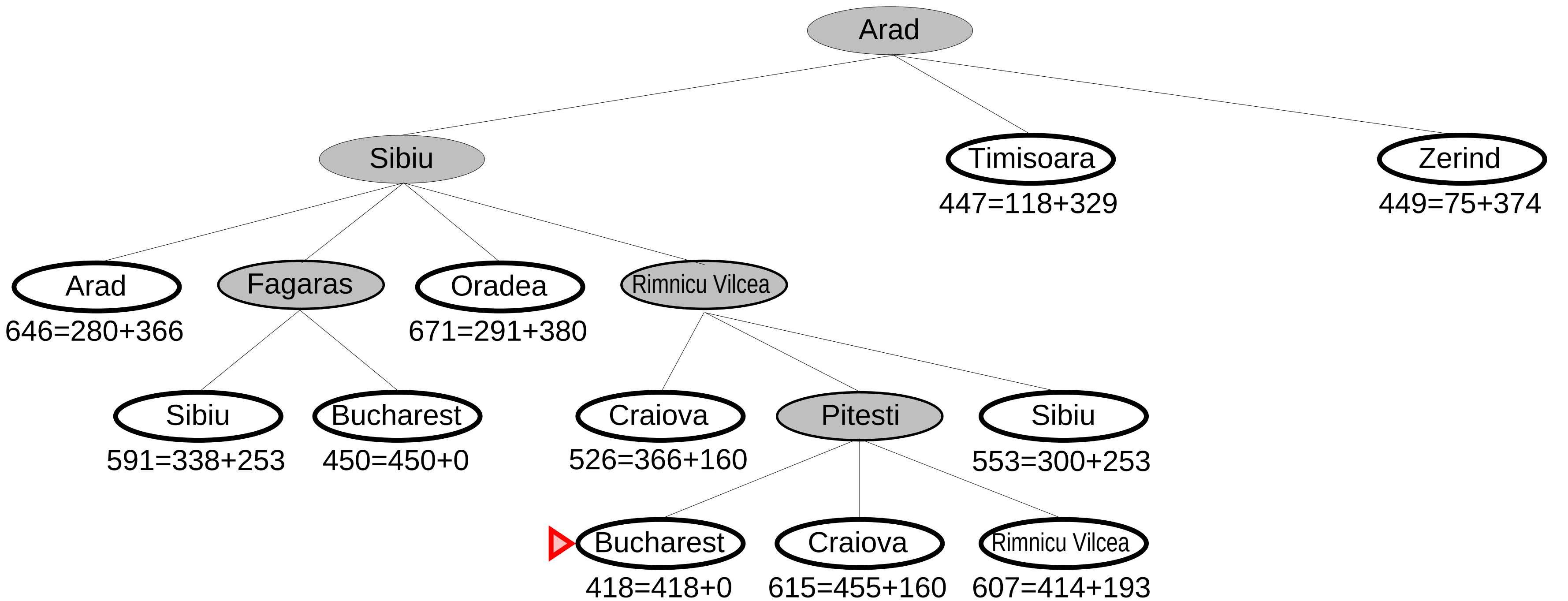
A* search example



A* search example

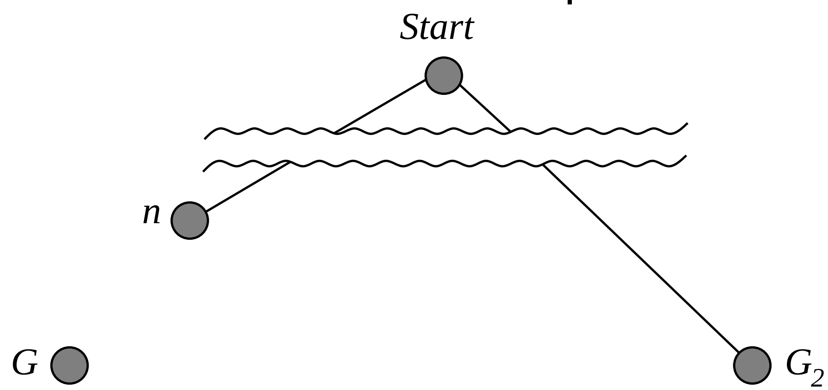


A* search example



Optimality of A* (standard proof)

Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G_1.



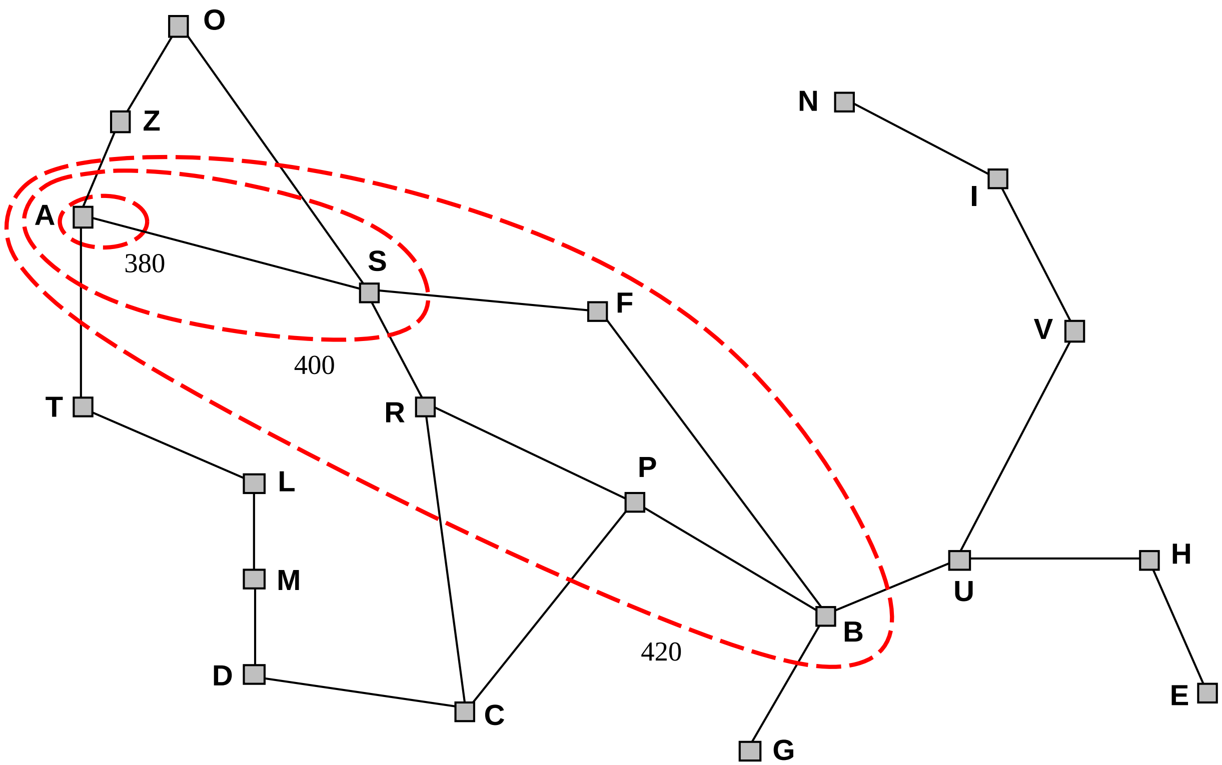
$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Optimality of A* (more useful)

Lemma: A* expands nodes in order of increasing f value*

Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)
Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



Properties of A*

Complete??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h × length of soln.]

Space??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

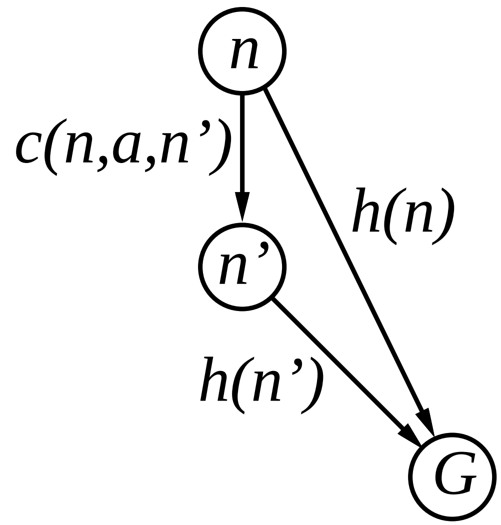
Proof of lemma: Consistency

A heuristic is consistent if

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

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

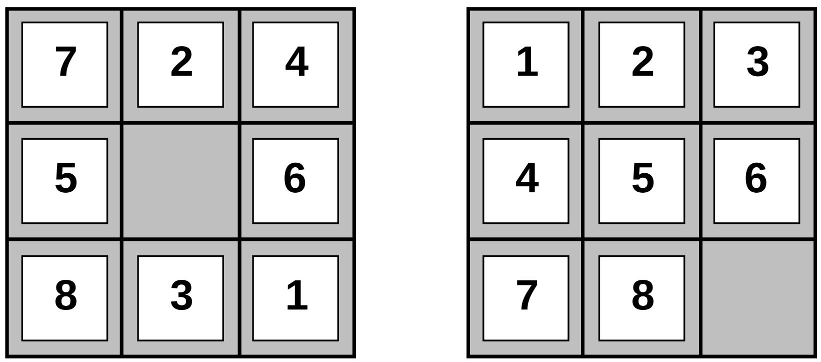


I.e., $f(n)$ is nondecreasing along any path.

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



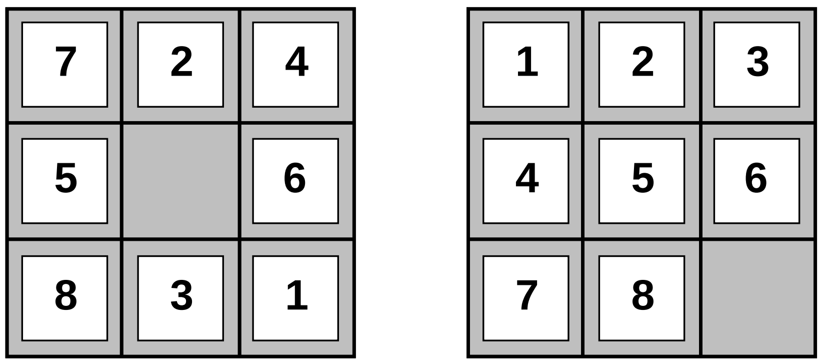
Start State **Goal State**

$\underline{h_1(S) = ??}$
$\underline{h_2(S) = ??}$

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



Start State **Goal State**

$h_1(S) =$?? 6
$h_2(S) =$?? 4+0+3+3+1+0+2+1 = 14

Dominance

If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 dominates h_1 and is better for search

Typical search costs:

$d = 14$	IDS = 3,473,941 nodes
	A$^*(h_1)$ = 539 nodes
	A$^*(h_2)$ = 113 nodes
$d = 24$	IDS $\approx$ 54,000,000,000 nodes
	A$^*(h_1)$ = 39,135 nodes
	A$^*(h_2)$ = 1,641 nodes

Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and dominates h_a, h_b

Relaxed problems

Admissible heuristics can be derived from the **exact**
solution cost of a **relaxed** version of the problem

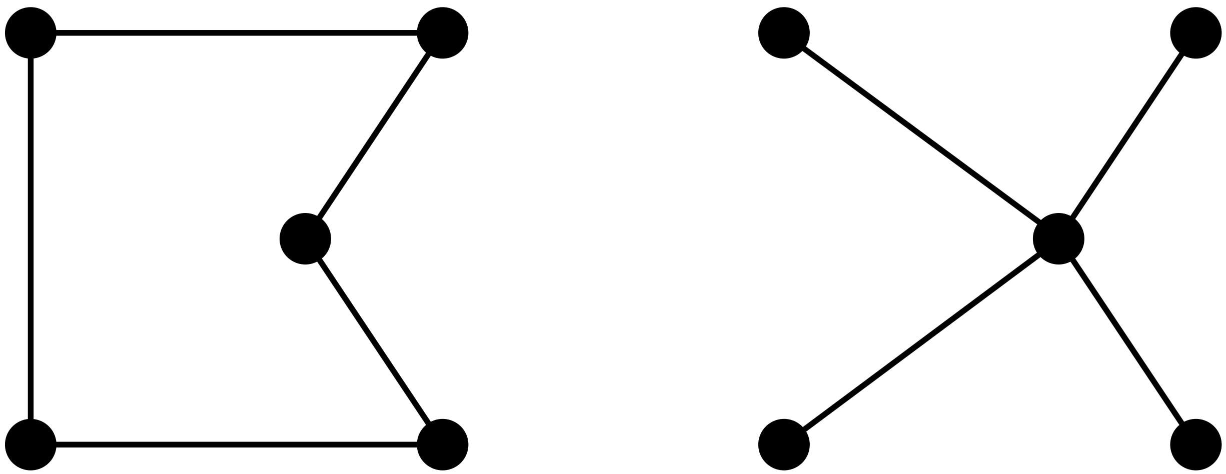
If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**,
then $h_1(n)$ gives the shortest solution

If the rules are relaxed so that a tile can move to **any adjacent square**,
then $h_2(n)$ gives the shortest solution

Key point: the optimal solution cost of a relaxed problem
is no greater than the optimal solution cost of the real problem

Relaxed problems contd.

Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once



Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour

Summary

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy best-first search expands lowest h
- incomplete and not always optimal

A* search expands lowest $g + h$
- complete and optimal
- also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems

Local search algorithms

Chapter 4, Sections 3–4

Outline

- ◇ Hill-climbing
- ◇ Simulated annealing
- ◇ Genetic algorithms (briefly)
- ◇ Local search in continuous spaces (very briefly)

Iterative improvement algorithms

In many optimization problems, **path** is irrelevant;
the goal state itself is the solution

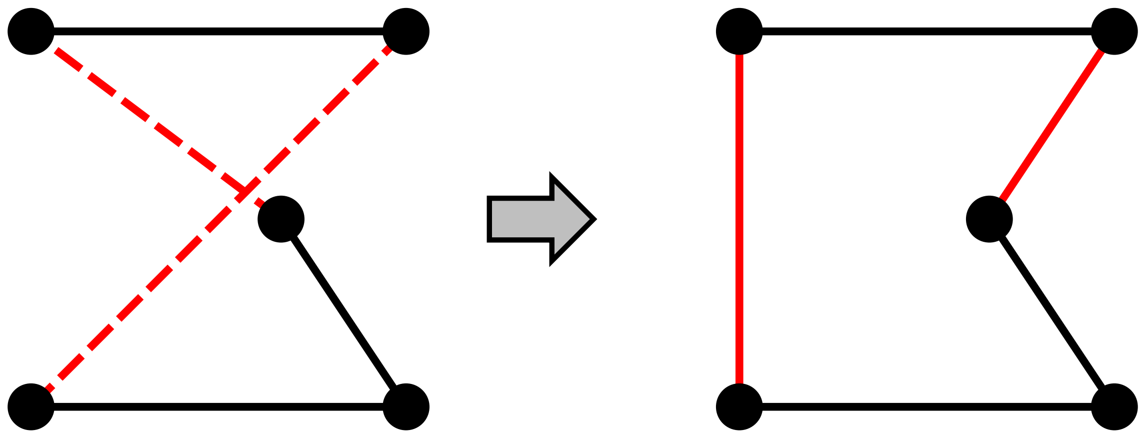
Then state space = set of "complete" configurations;
find **optimal** configuration, e.g., TSP
or, find configuration satisfying constraints, e.g., timetable

In such cases, can use iterative improvement algorithms;
keep a single "current" state, try to improve it

Constant space, suitable for online as well as offline search

Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

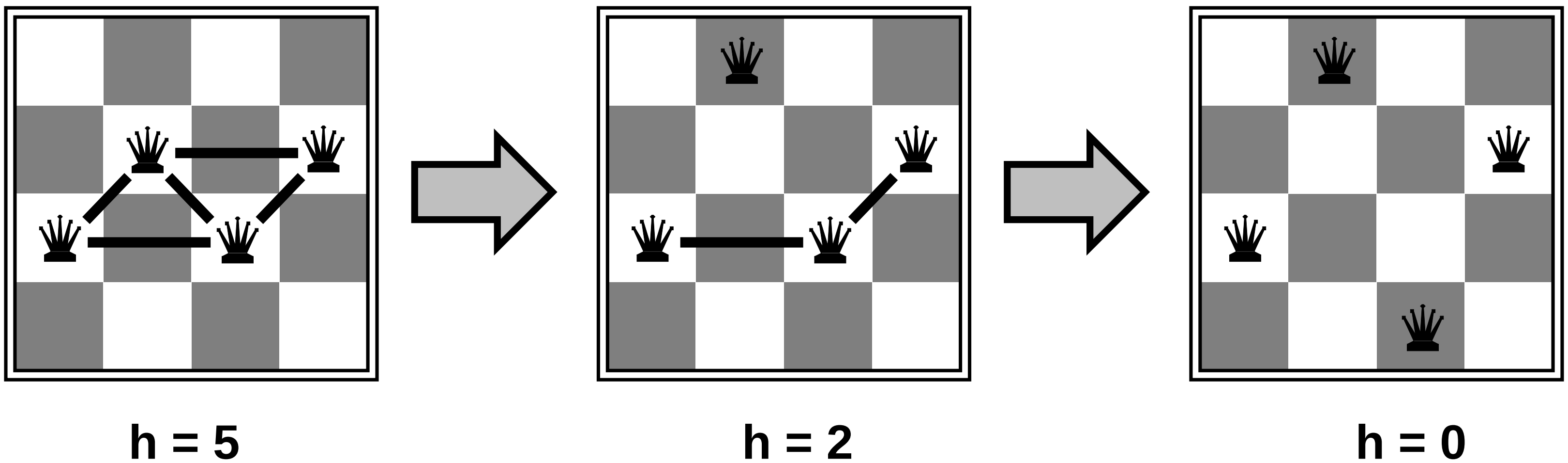


Variants of this approach get within 1% of optimal very quickly with thousands of cities

Example: n-queens

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Move a queen to reduce number of conflicts



Almost always solves n-queens problems almost instantaneously for very large n, e.g., $n = 1 million$

Hill-climbing (or gradient ascent/descent)

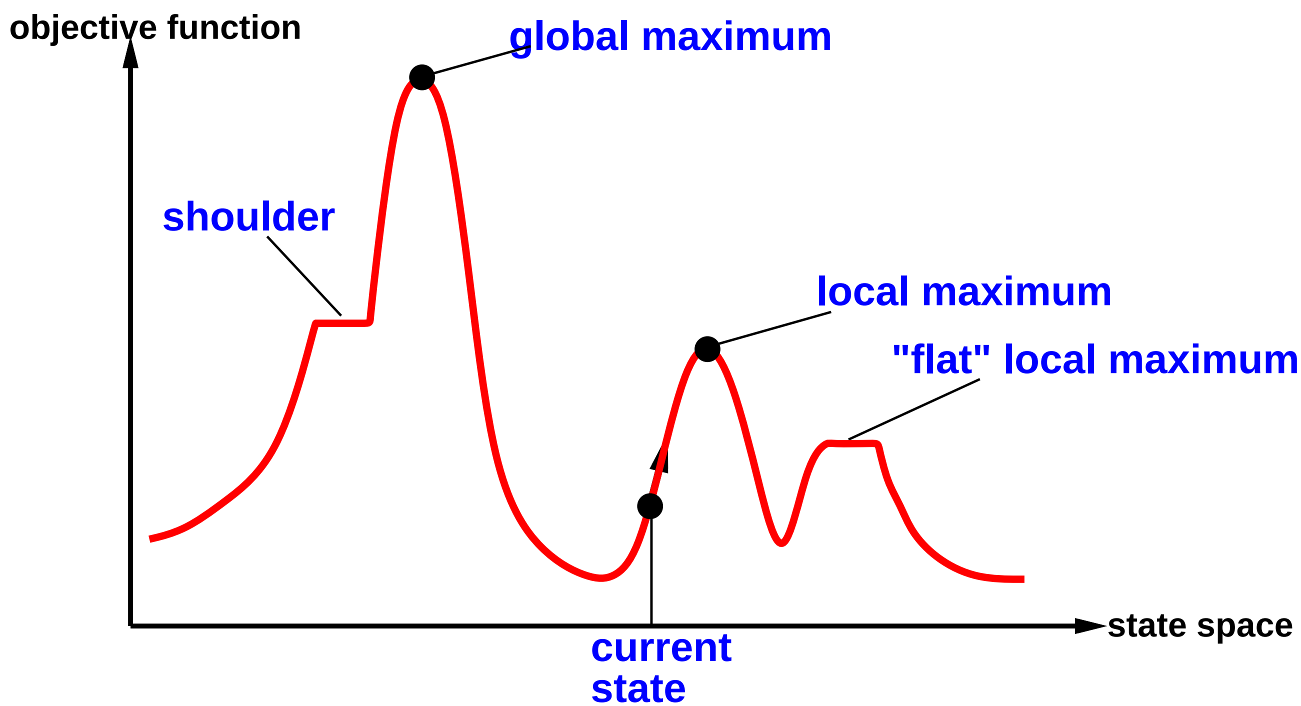
"Like climbing Everest in thick fog with amnesia"

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  inputs: *problem*, a problem
  local variables: *current*, a node
    neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    neighbor ← a highest-valued successor of *current*
    if VALUE[neighbor] ≤ VALUE[current] **then return** STATE[*current*]
    current ← *neighbor*
  end

Hill-climbing contd.

Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves 😊 escape from shoulders ☹ loop on flat maxima

Simulated annealing

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

Properties of simulated annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T decreased slowly enough $\Longrightarrow$ always reach best state x^* because $e^{\frac{E(x^*)}{kT}}/e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.

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

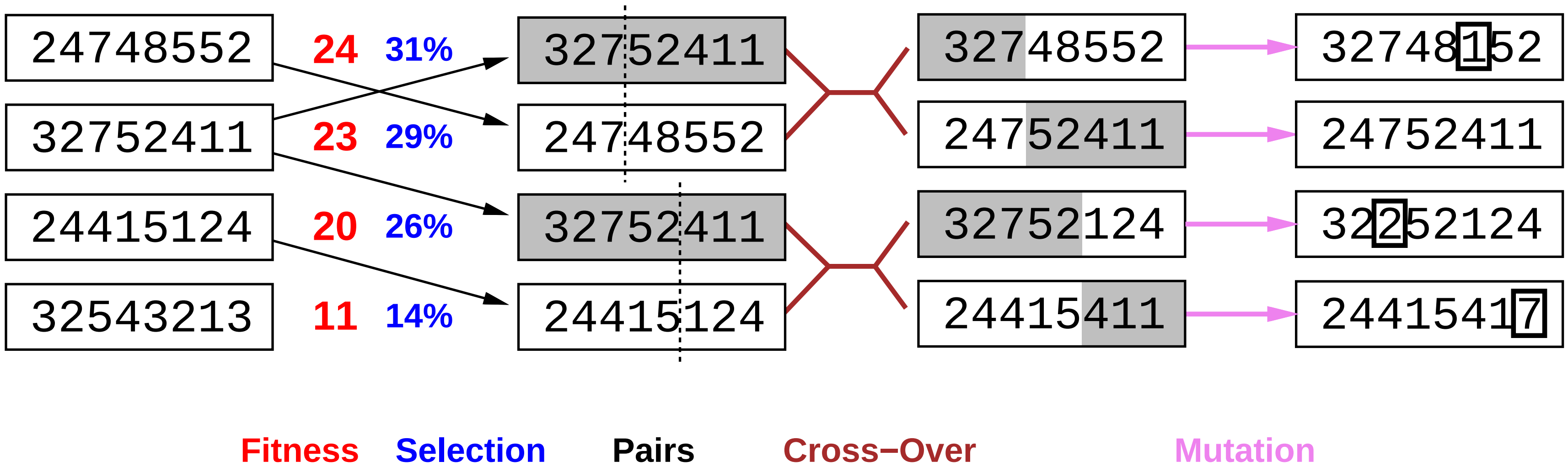
Problem: quite often, all k states end up on same local hill

Idea: choose k successors randomly, biased towards good ones

Observe the close analogy to natural selection!

Genetic algorithms

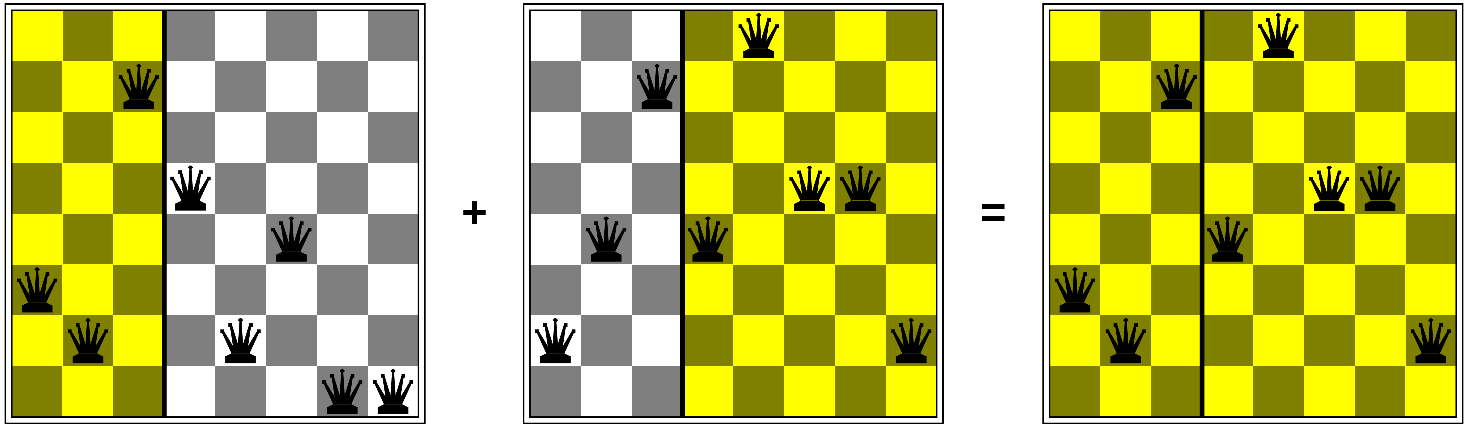
= stochastic local beam search + generate successors from **pairs** of states



Genetic algorithms contd.

GAs require states encoded as strings (GPs use programs)

Crossover helps **iff substrings are meaningful components**



GAs $\neq$ evolution: e.g., real genes encode replication machinery!

Continuous state spaces

Suppose we want to site three airports in Romania:

– 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)

– objective function $f(x_1, y_2, x_2, y_2, x_3, y_3) =$
  sum of squared distances from each city to nearest airport

Discretization methods turn continuous space into discrete space, e.g., empirical gradient considers $\pm\delta$ change in each coordinate

Gradient methods compute

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

to increase/reduce f, e.g., by $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

Sometimes can solve for $\nabla f(\mathbf{x}) = 0$ exactly (e.g., with one city).
Newton–Raphson (1664, 1690) iterates $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$
to solve $\nabla f(\mathbf{x}) = 0$, where $\mathbf{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$