

CSC 442: ARTIFICIAL INTELLIGENCE

PROBLEM SOLVING

TOPICS

- ▶ Intelligent Agents
- ▶ Environment and Problem Descriptors
- ▶ The State Space Search Framework

PERSPECTIVES FROM LAST TIME

- ▶ Acting Humanly – The Turing Test Approach
- ▶ Thinking Humanly – Cognitive Modeling
- ▶ Thinking Rationally – The Laws of Thought
- ▶ Acting Rationally – The Rational Agent Approach

MILESTONES OF AI (1/2)

- ▶ 1943 – McCulloch & Pitts develop Perceptron
- ▶ 1950 – Turing, Computing Machinery and Intelligence, Turing Test
- ▶ 1956 – Dartmouth Workshop, Newell & Simon develop Logic Theorist, General Problem Solver, and IPL.
- ▶ 1958 – McCarthy develops Lisp, proposes “Programs with Common Sense”
- ▶ 1965 – Robinson discovers Resolution Theorem Proving
- ▶ 1966 – US Government cancels AI funding, first AI winter. “The vodka is good but the meat is rotten.”
- ▶ 1969 – Green makes breakthroughs in Question Answering and Planning
- ▶ 1970 – Surge of interest in microwords, especially Blocks World (Shakey the robot, natural language understanding)

MILESTONES OF AI (2/2)

- ▶ 1970s – Expert and Knowledge-based Systems, MYCIN
- ▶ 1980s – Return of Neural Networks, Pearl changes AI with introduction of Bayesian Networks.
- ▶ 1984 – Darkest point of the AI winter.
- ▶ 1990s – Cognitive agents (Laird develops SOAR). Interest in AGI returns.
- ▶ 1997 – IBM's DeepBlue beats Gary Kasparov at chess.
- ▶ 2000s – Beginning of era of big data. Statistical approaches begin to take off, including bootstrapping and unsupervised methods.
- ▶ 2010s – Computing power and size of available datasets explodes. Smartphones ubiquitous. GPU based computing.
- ▶ 2012 – AlexNet revolutionizes computer vision and leads to resurgence of interest in neural networks.
- ▶ 2016 – Deep learning breaks records in machine translation. Machines overtake humans at the game of go.
- ▶ 2017 – AlphaZero beats previous worlds best chess engine.
- ▶ 2018 – GANs and DeepFakes become common knowledge.
- ▶ 2019 – Neural Language models beginning to scale to huge datasets.

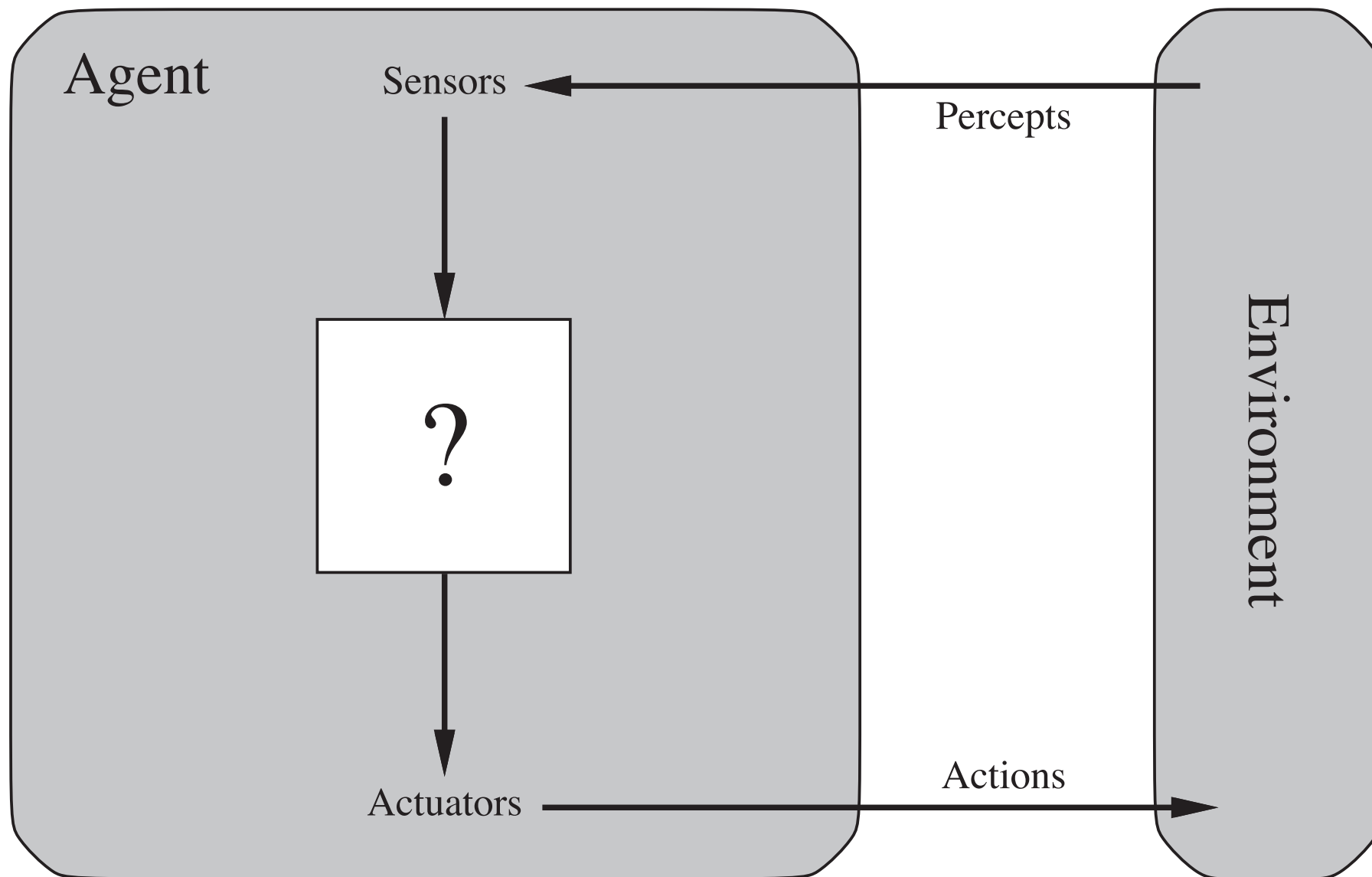
INTELLIGENT AGENTS (AIMA PERSPECTIVE)

- ▶ The job of AI is to design an **agent program** that implements the **agent function** – a mapping from percepts to actions.

INTELLIGENT AGENTS (AIMA PERSPECTIVE)

- ▶ An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**.
- ▶ R&N, p34.

AGENT ARCHITECTURE



INTELLIGENT AGENTS (AIMA PERSPECTIVE)

- ▶ An agent's choice of action at any given instant can depend on the entire **percept sequence** observed to date, but not on anything it hasn't perceived.
- ▶ R&N, p35

RATIONAL AGENTS

- ▶ For each possible percept sequence, a **rational agent** should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

WHAT IS RATIONAL?

- ▶ What is rational at any time depends on four things:
 - ▶ The performance measure defining success.
 - ▶ The agent's prior knowledge of the environment.
 - ▶ The actions the agent can perform.
 - ▶ The agent's percept sequence to date.

ENVIRONMENTS

- ▶ R&N use the term “Task Environment” to encapsulate four key design considerations for AI agents:
 - ▶ Performance Measure
 - ▶ Environment
 - ▶ Actuators
 - ▶ Sensors

TASK ENVIRONMENTS

	Performance Measure	Environment	Actuators	Sensors
Medical Diagnosis	healthy patient, cost of treatment	patient, hospital, staff	display questions, tests, treatments, referrals	keyboard, findings, patient responses
Refinery Controller	purity, yield, safety	refinery, operators	values, pumps, heaters, displays	temperature, pressure, chemical
English Tutor	test scores	students, classroom	display exercises, suggestions, corrections	keyboard entry, microphone, camera
Drone Pilot	follows person, don't crash, stable camera	trees, buildings, sky, air, turbulence	rotors, camera gimbal	airspeed, tilt, gps, gyroscope

	Performance Measure	Environment	Actuators	Sensors
Paper Writing	logic? readability? publishable?	???	???	???
Phone Center Operator	no wait time; speaking more than one sentence; unlimited customers	???	???	???
self driving cars	don't crash; get to proper location	???	???	???
better recommendation systems	purchase rate/ conversions	viewers, movies/ catalog/videos, ratings	???	???
barber/stylist	customer satisfaction	???	???	database of styles/fashion/ hats/haircuts
ai debugging	???	all kinds of programming languages	???	???

WHAT SHOULD WE AIM FOR?

- ▶ Designing a good performance measure is often very difficult!
- ▶ Rationality is not the same as perfection.
Rationality maximizes *expected* performance,
perfection maximizes *actual* performance.

PROPERTIES OF ENVIRONMENTS

- ▶ Several key aspects of environments affect our design decisions:
 - ▶ Is it fully observable, or are some things hidden?
 - ▶ How many agents are involved?
 - ▶ Is it deterministic or stochastic?
 - ▶ Is it episodic or sequential?
 - ▶ Does it change while we deliberate?
 - ▶ Are our actions discrete or continuous?
 - ▶ Do we know the likely results of our actions?

PROPERTIES OF ENVIRONMENTS

	Observable	Agents	Deterministic	Episodic	Static	Discrete
Medical Diagnosis	Partial	Single	No	Sequential	No	No
Chess	Full	Multi	Yes	Sequential	Depends	Yes
Taxi Driving	Partial	Multi	No	Sequential	No	No
Drone Pilot	Partial	Multi	No	Episodic	No	No

PROPERTIES OF ENVIRONMENTS

	Observable	Agents	Deterministic	Episodic	Static	Discrete
???	???	???	???	???	???	???
???	???	???	???	???	???	???
???	???	???	???	???	???	???
???	???	???	???	???	???	???

THE STRUCTURE OF AGENTS

- ▶ Agents may be:
 - ▶ Simple-reflex (table-driven)
 - ▶ Model-based reflex
 - ▶ Goal-based
 - ▶ Utility-based

SIMPLE REFLEX AGENTS

- ▶ Select an action based exclusively on the current percept.
- ▶

```
def simpleReflexAgent(percept, rules):  
    state = interpretInput(percept)  
    rule = ruleMatch(state, rules)  
    return rule.action
```
- ▶ AIMA Fig 2.10

SIMPLE REFLEX AGENTS

- ▶ Simple reflex agents are prone to infinite loops and other unintelligent behavior.
- ▶ They may also require an intractably large set of rules.

MODEL-BASED REFLEX AGENTS

- ▶ The most effective way to handle partial observability is for an agent to keep track of the part of the world it can't see now. I.e., to have some form of memory.
- ▶ A model based agent maintains a state or model of the world, and chooses the next action based on the model.

MODEL-BASED REFLEX AGENTS

```
▶ def modelBasedReflexAgent(percept, rules):  
    static model = initialModel()  
    static prev_act = None  
    state = updateState(state, percept, prev_act)  
    rule = ruleMatch(state, rules)  
    return rule.action
```

DESIGNING INTELLIGENT AGENTS

- ▶ Modeling a general environment can be **very difficult.**
- ▶ Related fields:
 - ▶ Knowledge Representation and Reasoning
 - ▶ Perception

PROBLEM SOLVING AGENTS

- ▶ **Problem Solving Agents**, also known as **Goal-Based Agents**, consider future actions and the desirability of their outcomes, based on a model of the problem at hand.

PROBLEM SOLVING AGENTS

- ▶ A problem solving agent develops an **action sequence** in order to accomplish a goal. The sequence is obtained by **searching** a possible **state-space** together with a set of possible **actions** and an associated **transition model**.
- ▶ Problem solving agents are general purpose and powerful. They can be applied to almost any AI problem, as long as you're flexible about your **level of abstraction**.

PROBLEM SOLVING AGENTS

```
▶ def problemSolvingAgent(percept):  
    static state = initialState()  
    static goal = None  
    static seq = []  
    if goal is None:  
        goal = formulateGoal(state)  
    problem = formulateProblem(state, goal)  
    seq = search(problem)  
    if seq is empty:  
        return Failure  
    else:  
        return first(seq)
```

UTILITY-BASED AGENTS

- ▶ Goals alone often insufficient – many ways to satisfy, some better.
- ▶ Therefore problem solving agents typically also consider **utility** functions.

- ▶ We will focus on problem solving for the next lectures:
 - ▶ The state-space search method
 - ▶ Four Major Search Algorithms
- ▶ Suggested reading:
 - ▶ AIMA Chapters 2 and 3

KEY STEPS

- ▶ **Goal formulation** – what do you want to accomplish?
- ▶ **Problem formulation** – how to formally specify the constraints, assumptions, and rules of the problem.
- ▶ Choices here determine the **level of abstraction**.

- ▶ We will use an **abstraction** called the **state-space search** model.

STATE-SPACE SEARCH PROBLEMS

- ▶ A **problem** can be defined formally by components:
 - ▶ an **initial state**
 - ▶ a set of **possible actions**
 - ▶ a **transition model**
 - ▶ a **goal test** function
 - ▶ an optional **path cost** function

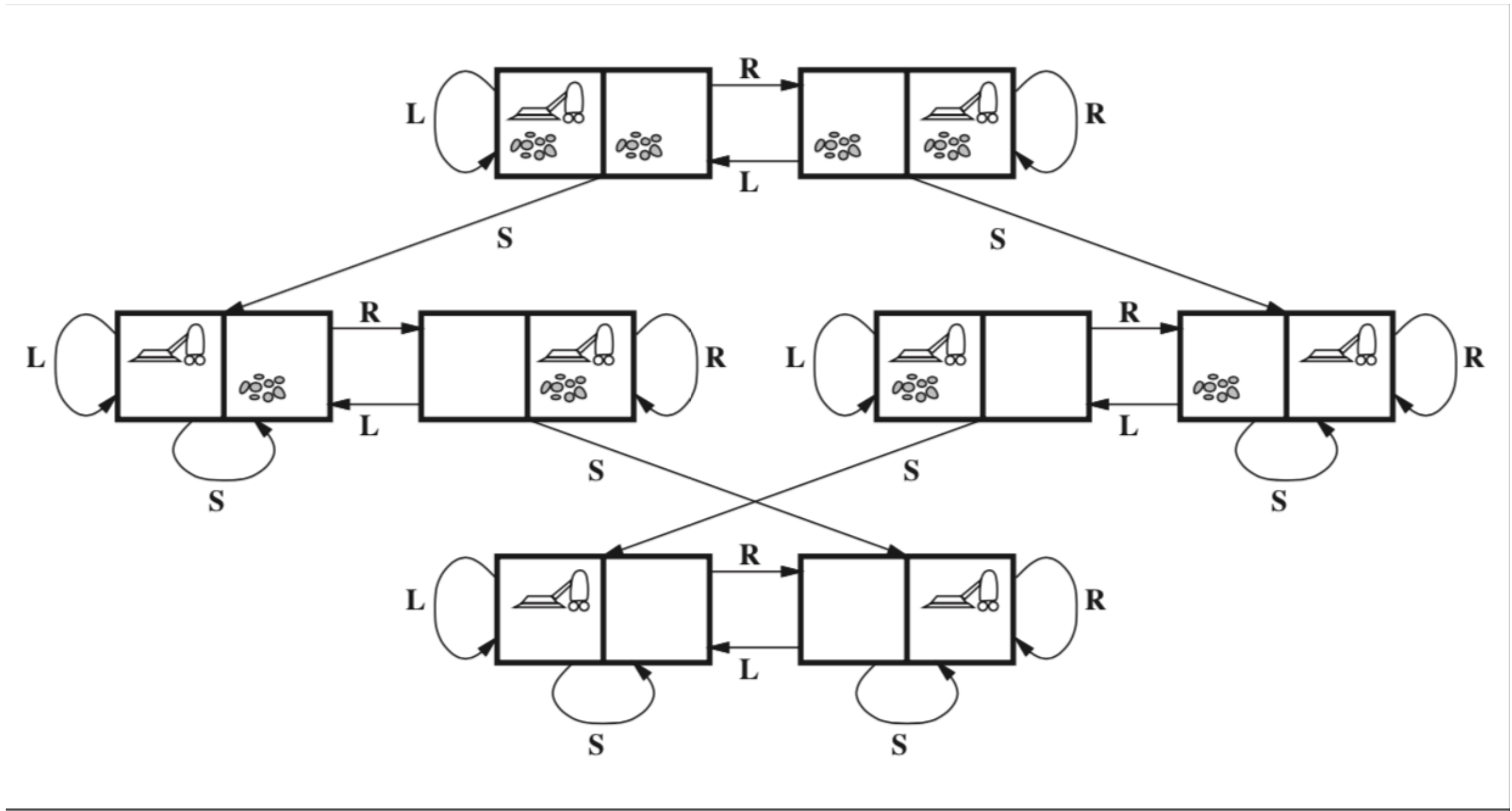
TOY PROBLEMS

- ▶ Let's look at some simple, toy problems to understand the concepts. Then, we'll look at navigation.
- ▶ Vacuum World
- ▶ The N-Queens Puzzle
- ▶ Knuth's Operator Search

VACUUM WORLD

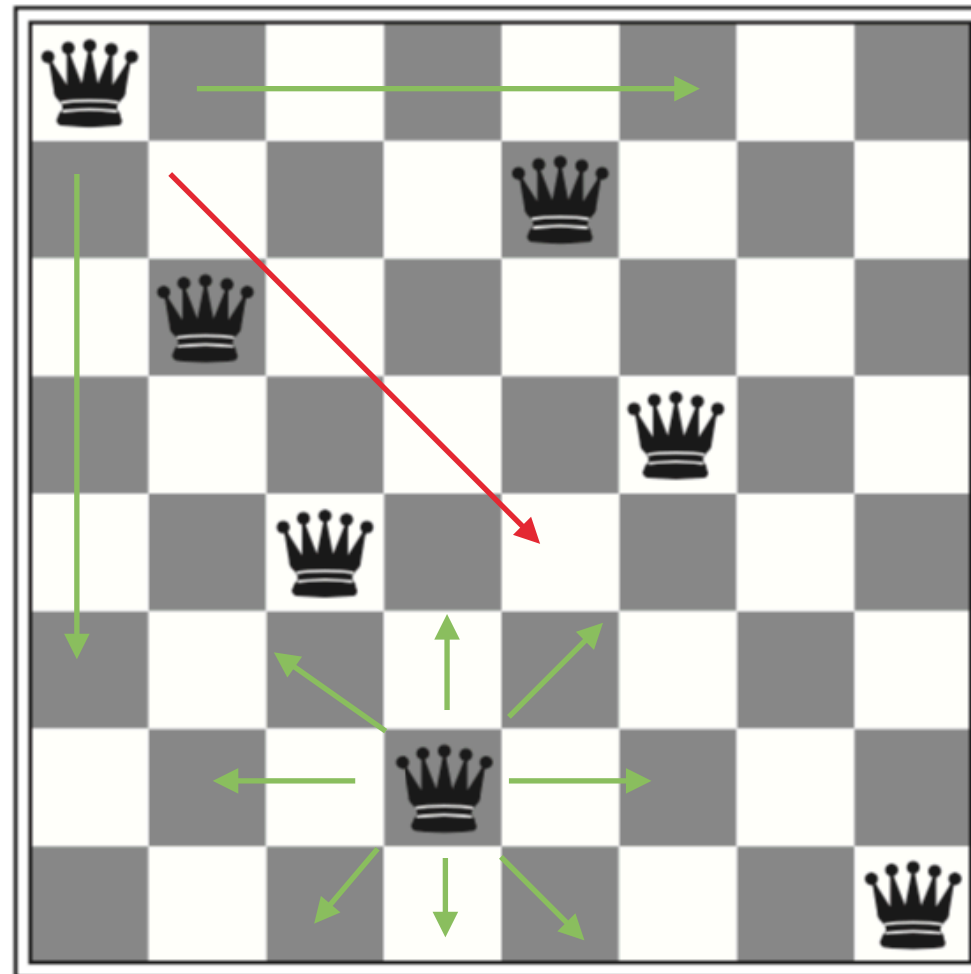
- ▶ Simple world with two locations, which may or may not contain dirt.
- ▶ The robot vacuum can move left, move right, or suck up dirt.
- ▶ What would a formal problem look like?

VACUUM WORLD



THE N-QUEENS PUZZLE

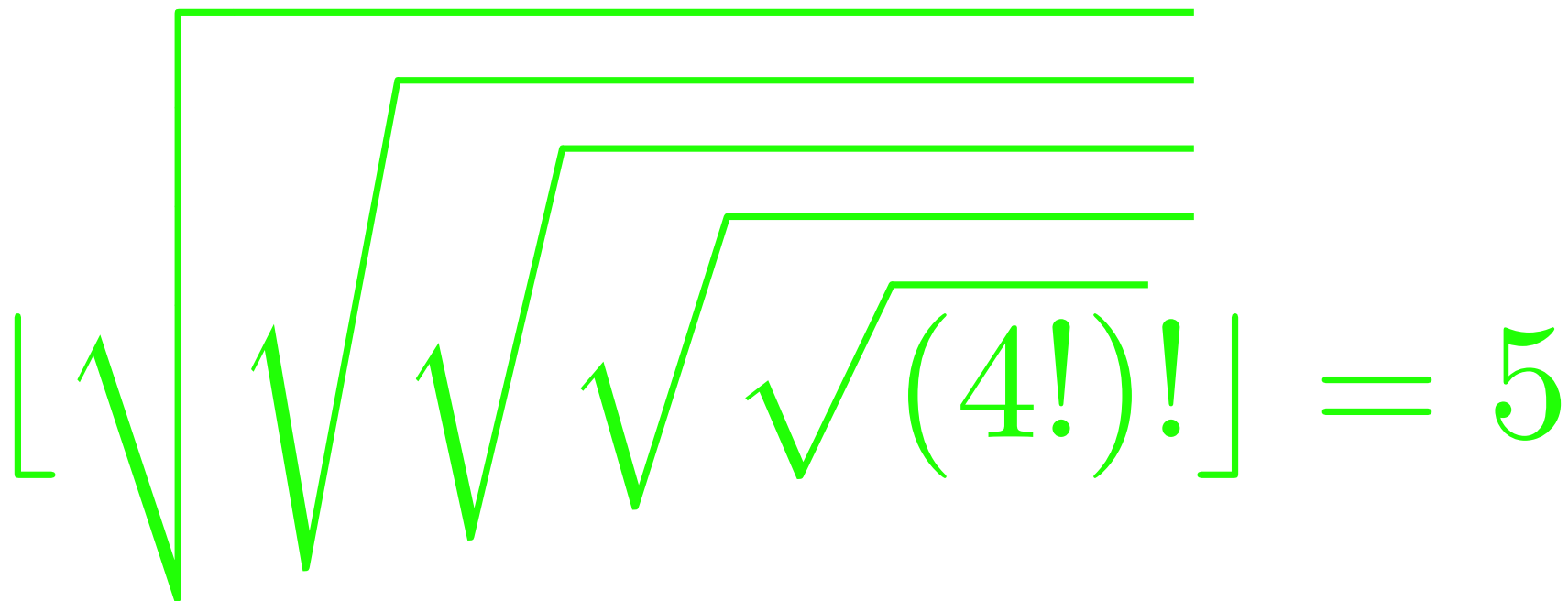
- ▶ Can we place queens in every row and column without attacks?



- ▶ Stopped here Wednesday, Sep 4th.

KNUTH'S CONJECTURE

- ▶ Starting with the number 4, a sequence of factorial, square root, and floor operations will reach any desired positive integer.



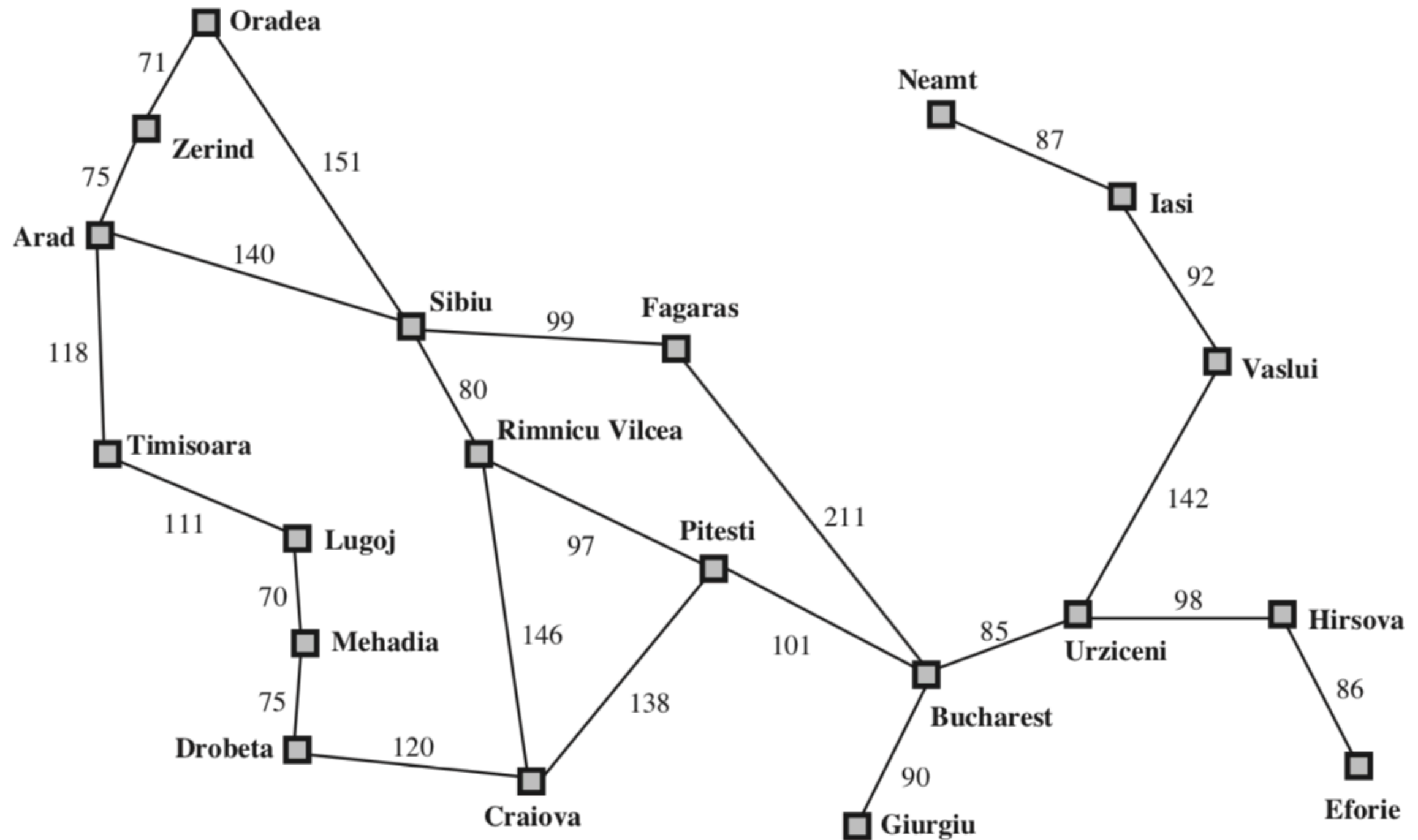
The diagram illustrates the nested square root operation in Knuth's conjecture. It features a series of four nested square root symbols, each represented by a green line. The innermost square root contains the expression $(4!)!$. Above each square root symbol, a horizontal green line extends to the right, and a vertical green line descends from the end of that horizontal line to the top of the next square root symbol, creating a staircase effect. The entire expression is enclosed in large square brackets, and the result is shown as $= 5$.

$$\lfloor \sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}} \rfloor = 5$$

APPLICATIONS

- ▶ Airline booking systems, military and emergency logistics, shipping, traveling sales, robot (taxi?) navigation, CNC milling, circuit and chip design, protein interactions, factory automation, ...

NAVIGATION — ARAD TO BUCHAREST?



PAUSE TO CONSIDER

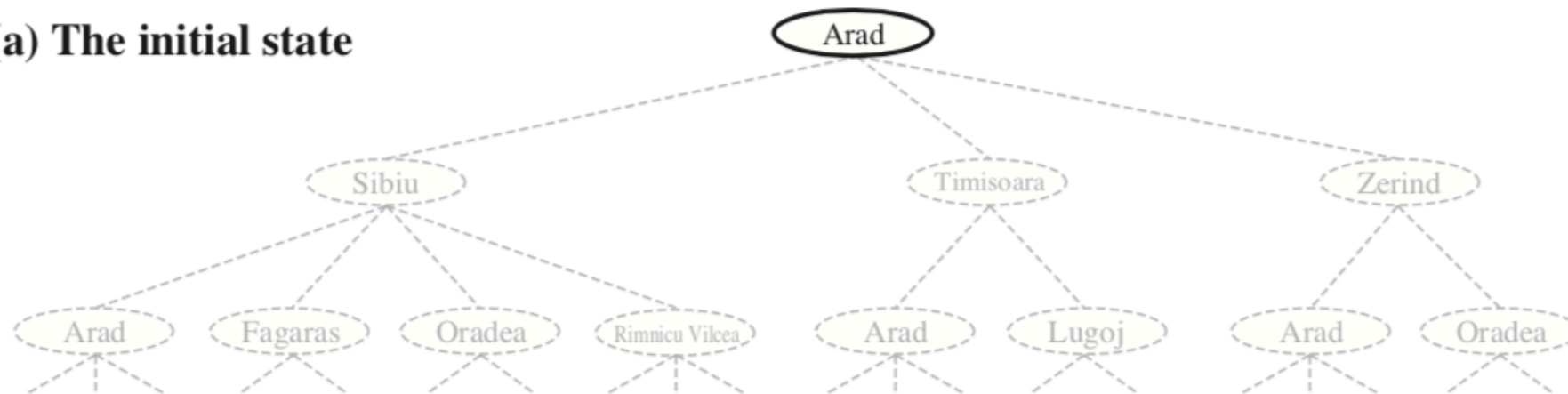
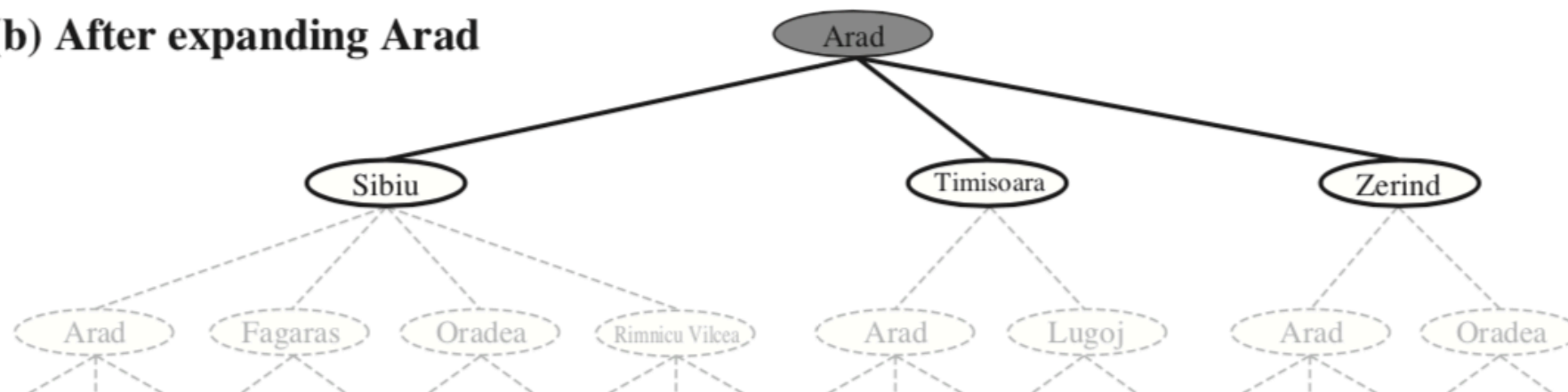
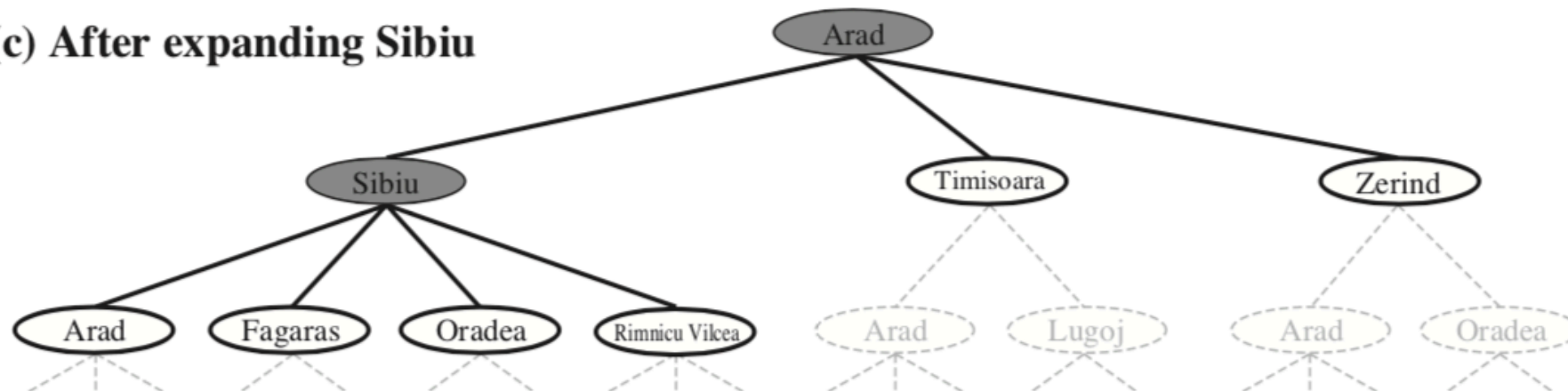
- ▶ Now we've introduced **problems**, so let's think for a moment about how to obtain **solutions**.
- ▶ This slide represents a significant transition.

SOLVING STATE-SPACE SEARCH PROBLEMS

- ▶ Search problems are solved with **search algorithms**.
- ▶ The simplest of these is **depth-first tree search**.

DEPTH-FIRST TREE SEARCH

```
function Tree-Search (problem, state) returns a solution
  if Goal(state), return Empty
  for each possible action a:
    let seq := Tree-Search (problem, Transition(state, a))
    if seq satisfies Goal, return a + seq
  return failure
```

(a) The initial state**(b) After expanding Arad****(c) After expanding Sibiu**

SOLVING STATE-SPACE SEARCH PROBLEMS

- ▶ Tree-search works great in some applications, but may find sub-optimal solutions in others, or worse, may recurse forever.
- ▶ Soon, we will cover the following algorithms:
 - ▶ General Tree and Graph Search
 - ▶ Breadth First, Depth First, Iterative Deepening
 - ▶ Heuristics and A^*

REVIEW YOUR DATA STRUCTURES

- ▶ Implementation of problem solving search requires the user to understand the **stack**, **queue**, and **graph** data structures.
- ▶ Understanding **linked lists** would be helpful.
- ▶ Understanding **dictionaries (hash tables)** would also be helpful.
- ▶ Read for next time: 3.3-3.7

STOP HERE

- ▶ Next up: search algorithms.