CSCI-331
Introduction to Intelligent Systems

Week01
Intro
History
Intelligent Agents

T.J. Borrelli

A note on notes

- **Much of the notes/materials used in this course this semester were developed by a committee of faculty members including but not necessarily limited to:**

- **Leon Reznik**
- **Zack Butler**
- **Roxanne Canosa**
- **Richard Zanibbi**
- **Roger Gaborski**
- **T.J. Borrelli**

- **There will also be supplemental material provided from other sources.**

Questions

- What tasks are machines good at doing that humans are not?
- What tasks are humans good at doing that machines are not?
- What tasks are both good at?
- What does it mean to learn?
- How is learning related to intelligence?
- What does it mean to be intelligent? Do you believe a machine will ever be built that exhibits intelligence?
- Have the above definitions changed over time?
- If a computer were intelligent, how would you know?
- What does it mean to be conscious?
- Can one be intelligent and not conscious or vice versa?

Questions Addressed

Problem Area

- **What *are* intelligent systems and agents?**
- **Why are we interested in developing them?**

Methodologies

- **What kind of software is involved? What kind of math?**
- **How do we develop it (software, repertoire of techniques)?**
- **Who uses AI? (Who are practitioners in academia, industry, government?)**

Artificial Intelligence as a Science

- **What is AI?**
- **What does it have to do with intelligence? Learning? Problem solving?**
- **What are some interesting problems to which intelligent systems can be applied?**
- ***Should I be interested in AI (and if so, why)?***

Today: Brief Tour of AI History

- **Study of intelligence (classical age to present), AI systems (1940-present)**
- **Viewpoints: philosophy, math, psychology, engineering, linguistics**

What is AI again?: descriptive approach

- Although the term of AI has been widely used for quite a long time with steadily increasing amount of research and applications, there is no anonymously accepted definition. AI can mean many things to different people and various techniques are considered as belonging to AI.
- The term coined in 1956 by J. McCarthy at MIT
- Two branches: engineering discipline dealing with the creation of intelligent machines and empirical science concerned with the computational modeling of human intelligence
- The goal of AI is developing methods, which allow producing thinking machines that can solve problems
- Which problems?
- ill-defined and ill-structured
- complicated taxonomy or classifying
- Combinatorial optimization

What is AI again?:

- The great variety of AI techniques have been developed and applied over the history for solving the problems mentioned above.
- Some of these methodologies are"conventional" or "old" methods (1950s):
- Search algorithms
- Probabilistic reasoning
- Natural language processing
- Belief networks

- Others are "new" (1960s) – soft computing and computational intelligence

Foundations of Artificial Intelligence

- 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

- Economics – Utility, decision theory, game theory

- Neuroscience – How do brains process information?

- Psychology – Phenomena of perception and motor control, experimental techniques

- Computer Engineering – Building fast computers

- Control Theory – Design systems that maximize an objective function over time

- Linguistics – Knowledge representation, grammar

What is AI again?
Systematic approach

- **Four Categories of Systemic Definitions**
  - **1. Think like humans**
  - **2. Act like humans**
  - **3. Think *rationally***
  - **4. Act *rationally***

What is AI again? Systematic approach

- **Thinking Like Humans**
  - ***Machines with minds*** **(Haugeland, 1985)**
  - **Automation of "decision making, problem solving, learning..." (Bellman, 1978)**
- **Acting Like Humans**
  - ***Functions*** **that** ***require intelligence when performed by people*** **(Kurzweil, 1990)**
  - **Making computers do things** ***people currently do better*** **(Rich and Knight, 1991)**
- **Thinking Rationally**
  - **Computational models of mental faculties (Charniak and McDermott, 1985)**
  - **Computations that make it possible to** ***perceive***, ***reason***, **and** ***act*** **(Winston, 1992)**
- **Acting Rationally**
  - **Explaining, emulating intelligent behavior via computation (Schalkoff, 1990)**
  - **Branch of CS concerned with automation of intelligent behavior (Luger and Stubblefield, 1993)**

What is AI?

Thinking and Acting Like Humans

Concerns: Human Performance (Figure 1.1 R&N, Left-Hand Side)

- Top: thought processes and reasoning (learning and inference)
- Bottom: behavior (interacting with environment)

Machines With Minds

- Cognitive modeling
  - Early historical examples: problem solvers (see R&N Section 1.1)
  - Application (and one driving force) of cognitive science
- Deeper questions
  - What is intelligence?
  - What is consciousness?

Acting Humanly: The Turing Test Approach

- Capabilities required
  - Natural language processing
  - Knowledge representation
  - Automated reasoning
  - Machine learning
- Turing Test: can a machine appear indistinguishable from a human to an experimenter?

Thinking and Acting Like Humans

- **According to Evolutionary Biologist Richard Dawkins:**

Adaptationism is defined as
'that approach to evolutionary studies which assumes without further proof that all aspects of the morphology, physiology and behavior of organisms are adaptive optimal solutions to problems.'

- **Dawkins does not put much stock in this belief.**
- **Take for example: the human eye**

What is AI? Thinking and Acting Rationally

Concerns: Human Performance (Figure 1.1 R&N, Right-Hand Side)

- **Top: thought processes and reasoning (learning and inference)**
- **Bottom: behavior (interacting with environment)**

Computational Cognitive Modelling

- **Rational ideal**
  - **In this course: rational agents**
  - **Advanced topics: learning, utility theory, decision theory**
- **Basic mathematical, computational models**
  - **Decisions: automata Search**
  - **Concept learning**

Acting Rationally: The Rational Agent Approach

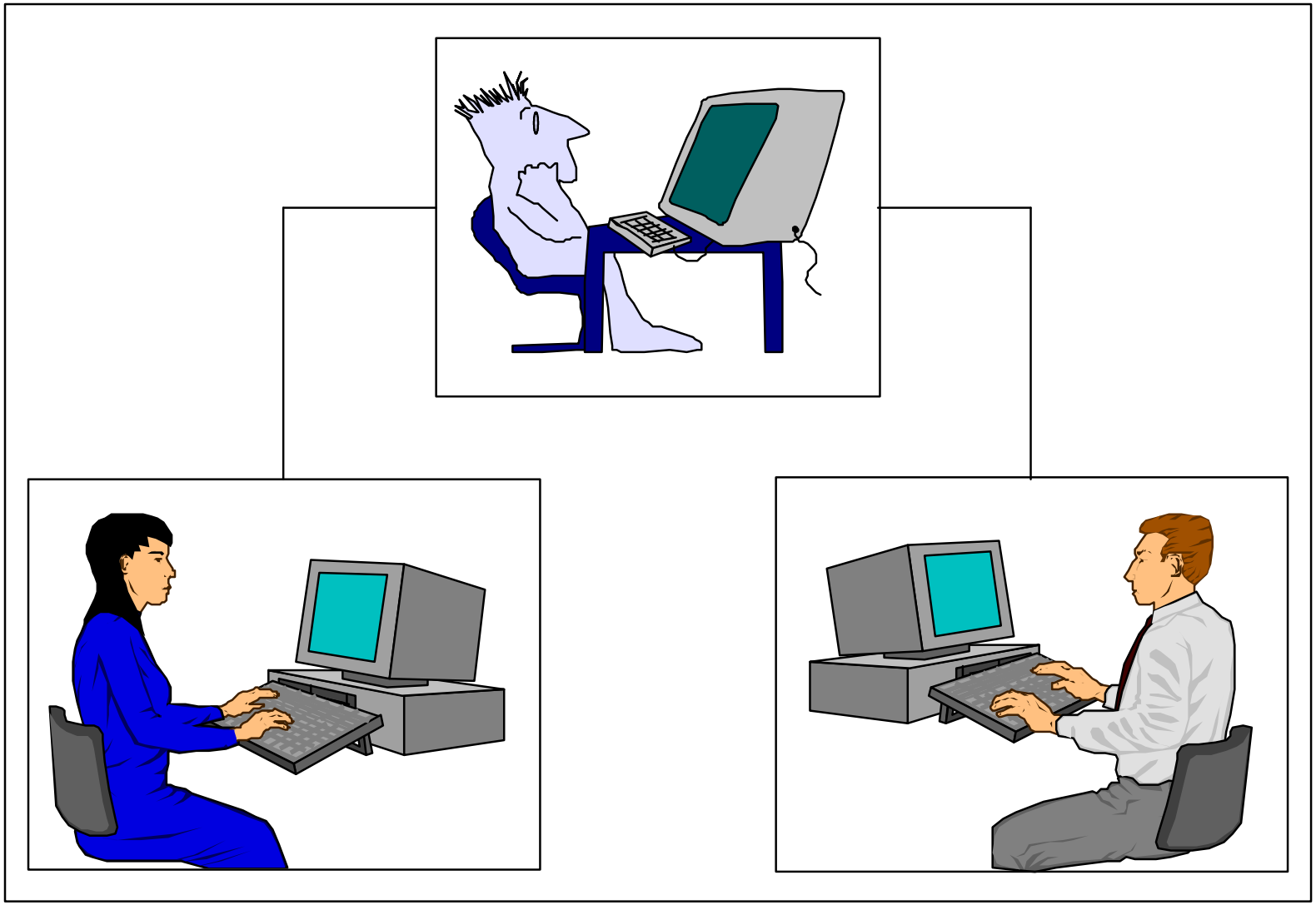
- **Rational action: *acting to achieve one's goals, given one's beliefs***
- **Agent: entity that perceives and acts**
- **Focus of next topic**
  - **"Laws of thought" approach to AI: correct inferences (reasoning)**
  - **Rationality not *limited to* correct inference**

Significant paper on machine intelligence: ***"Computing Machinery and Intelligence" (1950) -*** by the British mathematician ***Alan Turing*** – still stands up "well" under the test of time

He asked: ***Is there thought without experience? Is there mind without communication? Is there language without living? Is there intelligence without life?*** All these questions, as you can see, are just variations on the fundamental question of artificial intelligence, ***Can machines think?***

- Turing did not provide definitions of machines and thinking, he just avoided semantic arguments by inventing a game, the ***Turing Imitation Game***.
- The imitation game originally included two phases.
  - 1st phase - interrogator, a man and a woman are each placed in separate rooms – interrogator's job is to work out who is the man and who is the woman by questioning them.
  - The man should attempt to deceive the interrogator that *he* is the woman, while the woman has to convince the interrogator that *she* is the woman.

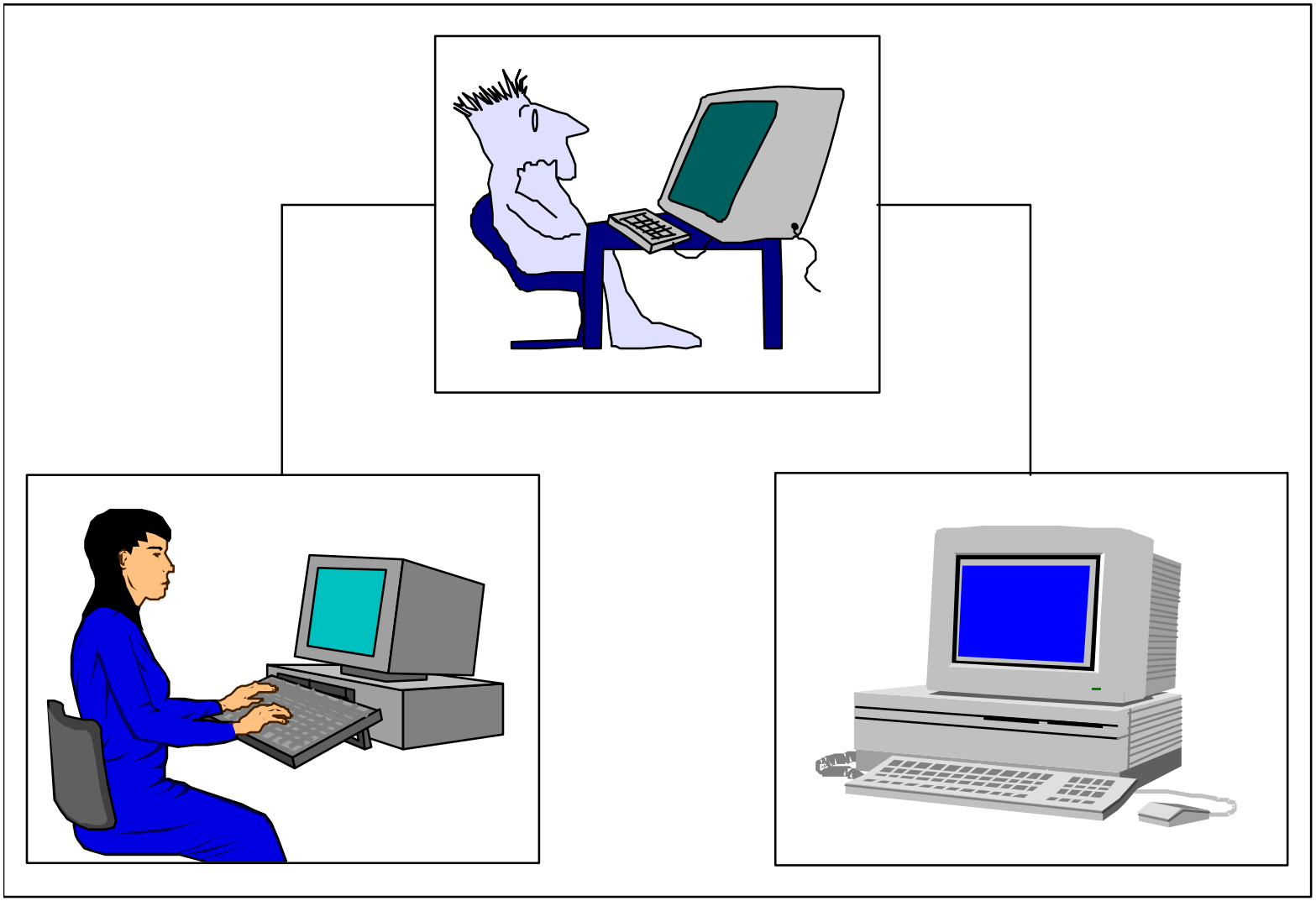
Turing Imitation Game: Phase 1



Turing Imitation Game: Phase 2

- In the second phase of the game, the man is replaced by a computer programmed to deceive the interrogator as the man did
- It would even be programmed to make mistakes and provide fuzzy answers in the way a human would
- If the computer can fool the interrogator as often as the man did, we may say this computer has passed the intelligent behavior test.

Turing Imitation Game: Phase 2



The Turing test has two remarkable qualities that make it interesting.

- By maintaining communication between the human and the machine via terminals, the test gives us an objective standard view on intelligence.
- The test itself is quite independent from the details of the experiment. It can be conducted as a two-phase game, or even as a single-phase game when the interrogator needs to choose between the human and the machine from the beginning of the test.

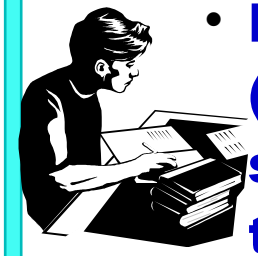
- Turing believed that by the end of the 20th century it would be possible to program a digital computer to play the imitation game
- Although modern computers still cannot consistently pass the Turing test, it provides a basis for the verification and validation of knowledge-based systems.
- **A program thought intelligent in some narrow area of expertise is evaluated by comparing its performance with the performance of a human expert.**
- To build an intelligent computer system, we have to capture, organize and use human expert knowledge in some narrow area of expertise.

AI History (old)

- **Philosophy Foundations (400 B.C. – present)**
  - **Mind: dualism and rationalism(Descartes), materialism (Leibniz), empiricism (Bacon, Locke)**
  - **Thought: syllogism (Aristotle), induction (Hume), logical positivism (Russell)**
  - **Rational agentry (Mill) – acts to achieve the best expected outcome**
- **Mathematical Foundations (c. 800 – present)**
  - **Early: algorithms (al-Khowarazmi, 9th century Arab mathematician), Boolean logic**
  - **Computability (20th century – present)**
    - **Gödel's incompleteness theorem**
    - **Formal computational models: Hilbert's Entscheidungsproblem (decidability)**
    - **Turing Intractability**
    - **NP-completeness**

AI History (not too old)

- Computer Engineering (1940 – present)
- Linguistics (1957 – present)
- Stages of AI
  - Gestation (1943 – c. 1956) – McCulloch and Pitts artificial neuron model, Hebbian learning
  - Birth of AI (1956) – 2 month workshop at Dartmouth (McCarthy, Minsky, Shannon, Rochester)
  - Infancy “Early enthusiasm, great expectations”(c. 1952 – 1969) – some early success in areas such as ANN, Lisp, GPS (general problem solver)



  - Disillusioned early (c. 1966 – 1974), later childhood (1969 – 1979) – the fact that a program can find a solution in principle does not necessarily mean that the program can find it in practice

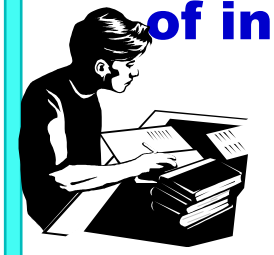
AI History (not too old)

- **"Early" (1969 – 1979) – knowledge based systems (DENDRAL – inferring molecular structure based on mass spec readings; MYCIN – medical diagnosis)**

- **"Middle" adolescence (c. 1980 – present) – AI becomes an industry**

- **"Middle-late" Return of Neural Nets (1986-present)**

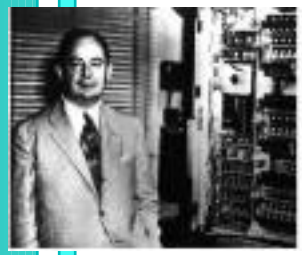
- **"Modern" – AI takes on scientific method; emergence of intelligent agents (1987 - present)**



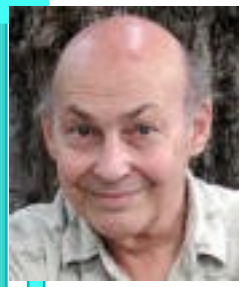
History of AI -- birth

The birth of artificial intelligence (1943 – 1956)

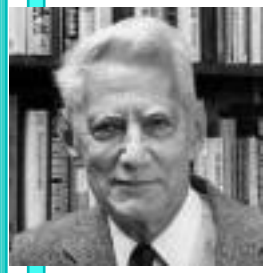
- First work recognized in AI was paper by **Warren McCulloch** and **Walter Pitts** (1943).
- They proposed a model of an artificial neural network
- Demonstrated that simple network structures could learn
- McCulloch – the second "founding father" of AI after Alan Turing, had created the corner stone of neural computing and artificial neural networks (ANN).



- The third founder of AI was **John von Neumann**, (Hungarian-born mathematician).
- In 1930 – joined Princeton, lecturing in mathematical physics.
- Was an adviser for the Electronic Numerical Integrator and Calculator project at the University of Pennsylvania
- Helped to design the **Electronic Discrete Variable Calculator**



- He was influenced by McCulloch and Pitts's neural network model.
- Encouraged **Marvin Minsky** and **Dean Edmonds** (at Princeton math department) to build first neural network computer in 1951



- Another of the first generation researchers was **Claude Shannon**.
- Graduated from MIT, joined Bell Telephone Laboratories in 1941.
- Shannon shared Alan Turing's ideas on the possibility of machine intelligence.
- In 1950, he published a paper on chess-playing machines, which pointed out that a typical chess game involved about 10^120 possible moves (Shannon, 1950).
- Even if the new von Neumann-type computer could examine one move per microsecond, it would take 3 x 10^106 years to make its first move.
- Thus Shannon demonstrated the need to use **heuristics** in the search for the solution.

- In 1956, **John McCarthy**, **Marvin Minsky** and **Claude Shannon** organized a summer workshop at Dartmouth College.
- They brought together researchers interested in the study of machine intelligence, artificial neural nets and automata theory.
- Although there were just ten researchers, this workshop gave birth to a new science called ***artificial intelligence***.

The rise of artificial intelligence, or the era of great expectations (1956 – late 1960s)

- The early works on neural computing and artificial neural networks started by McCulloch and Pitts was continued.
- Learning methods were improved and **Frank Rosenblatt** proved the ***perceptron convergence theorem***, demonstrating that his learning algorithm could adjust the connection strengths of a perceptron.

- One of the most ambitious projects of the era of great expectations was the **General Problem Solver (GPS)**.
- **Allen Newell** and **Herbert Simon** from the CMU developed a general-purpose program to simulate human-solving methods.
- Newell and Simon postulated that a problem to be solved could be defined in terms of ***states***.
- They used the means-end analysis to determine a difference between the current and desirable or *goal state* of the problem, and to choose and apply *operators* to reach the goal state.
- The set of operators determined the solution plan.

- However, GPS failed to solve complex problems.
- The program was based on formal logic and could generate an infinite number of possible operators.
- The amount of computer time and memory that GPS required to solve real-world problems led to the project being abandoned.
- In the sixties, AI researchers attempted to simulate the thinking process by inventing *general methods* for solving *broad classes of problems*.
- They used the general-purpose search mechanism to find a solution to the problem.
- Such approaches, now referred to as ***weak methods***, applied weak information about the problem domain.

- By 1970, the euphoria about AI was gone, and most government funding for AI projects was cancelled.
- AI was still a relatively new field, academic in nature, with few practical applications apart from playing games.
- So, to the outsider, the achieved results would be seen as toys, as no AI system at that time could manage real-world problems.

Unfulfilled promises, or the impact of reality (late 1960s – early 1970s)

The main difficulties for AI in the late 1960s were:

- Because AI researchers were developing general methods for broad classes of problems, early programs contained little or even no knowledge about a problem domain.
- To solve problems, programs applied a search strategy by trying out different combinations of small steps, until the right one was found.
- This approach was quite feasible for simple **toy problems**, so it seemed reasonable that, if the programs could be "scaled up" to solve large problems, they would finally succeed.

Many of the problems that AI attempted to solve were **too broad and too difficult**. A typical task for early AI was machine translation. For example, the National Research Council, USA, funded the translation of Russian scientific papers after the launch of the first artificial satellite (Sputnik) in 1957.

- **The spirit is willing but the flesh is weak**
- **The vodka is good but the meat is rotten**
- ***The story of this specific mistranslation is likely apocryphal**

- In 1971, the British government also suspended support for AI research. Sir **James Lighthill** had been commissioned by the Science Research Council of Great Britain to review the current state of AI.
- He did not find any major or even significant results from AI research, and therefore saw no need to have a separate science called “artificial intelligence”.

Why Study Artificial Intelligence?

New Computational Capabilities

- Advances in uncertain reasoning, knowledge representations
- Learning to act: robot planning, control optimization, decision support
- Database mining: converting (technical) records into knowledge
- Self-customizing programs: learning news filters, adaptive monitors
- Applications that are hard to program: automated driving, speech recognition

Why Study Artificial Intelligence?

- **Better Understanding of Human Cognition**
  - **Cognitive science: theories of knowledge acquisition (e.g., through practice)**
  - **Performance elements: reasoning (inference) and *recommender* systems**
- **Time is Right**
  - **Recent progress in algorithms and theory**
  - **Rapidly growing volume of online data from various sources**
  - **Available computational power**
  - **Growth and interest of AI-based industries (e.g., data mining, planning)**

331 – Intro to Intelligent Systems

(More) History and Intro to AI

Week01b

T.J. Borrelli

Important Features of AI

1. The use of computers to do reasoning, pattern recognition, learning, or some other form of inference.
2. A focus on problems that do not respond to algorithmic solutions.
3. A concern with problem-solving using inexact, missing, or poorly defined information.
4. Reasoning about the significant qualitative features of a situation.
5. An attempt to deal with issues of semantic meaning as well as syntactic form.

Important Features of AI

6. Answers that are neither exact nor optimal, but are in some sense "sufficient".

7. The use of large amounts of domain-specific knowledge in solving problems.

8. The use of meta-level knowledge to effect more sophisticated control of problem-solving strategies.

Intractability and AI

- Some well-known NP-complete problems:
  - Longest Path
  - Hamiltonian Cycle
  - 3-CNF
  - Circuit-Sat
  - Formula-Sat
  - Clique
  - Vertex Cover
  - Traveling Salesperson
  - Subset Sum
  - Graph Coloring
  - Crossword Puzzles
  - Longest Common Subsequence (LCS) for more than 2 strings

Intractability and AI

- More well-known NP-complete problems:
  - 0-1 Knapsack
  - Exam Scheduling and CPU Register Assignment (Graph Coloring)
  - 3D matching
  - Chess
  - Minesweeper ($1,000,000 Grand Challenge to solve it in p-time!)
  - Tetris
  - Rubic's Cube
  - Lloyd Puzzle (8-slide puzzle)
  - Cracker Barrel puzzle

Intractability and AI – Coping with NP-completeness:

1. Brute force - you will need additional resources (parallel machines) and probably will still not be able to solve

2. Efficient non-deterministic machines (quantum computers?)

3. Small inputs

4. Special cases

5. Restrict the problem (parameterize the complexity)

For example, in chess there are 448 possible moves for each play. Therefore, for n plays, there are 448^n possible moves. If you create a game tree with each branch labeled with a possible move, you could win the game by "looking ahead" and following the branches that lead to the winning game configuration. This would require a tree with 448^n branches. Too big. Prune the tree by using a heuristic to cut off unpromising branches, or look ahead a few branches at a time.

Intractability and AI

6. Use a heuristic (a “rule-of-thumb”, usually arising from trial-and-error or experimentation). Neural networks, genetic algorithms, fuzzy logic, simulated annealing, tabu search, hill climbing (problem of local optimum, may not be globally optimal). For example, to navigate through a maze, you could use a heuristic such as the “right-hand-rule” (place your right hand on the wall and always follow the wall that your hand is touching - does not always work). Or you could drop pebbles at intersections to mark corridors that have already been investigated. This will always work, but you may need LOTS of pebbles

7. Randomization (Monte-Carlo techniques). Throw a die. Trade memory for randomness (random walk).

8. Use an approximation algorithm. You will have to settle for a near-optimal solution.

AI in the 20th Century

1943	McCulloch & Pitts: Boolean circuit model of brain
1950	Turing's "Computing Machinery and Intelligence"
1955	Dartmouth meeting: the name "Artificial Intelligence" adopted
1950s	Early AI programs, including Samuel's checkers program,
	Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
1965	Robinson's complete algorithm for logical reasoning
1966-73	AI discovers computational complexity
	Neural network research almost disappears
1969-79	Early development of knowledge-based systems, “Blocks World”
1980	AI becomes an industry
1986	Neural networks return to popularity
1987	AI becomes a science
1995	The emergence of intelligent agents

AI in the 21st Century

- AI is everywhere
- Fuzzy logic is used in elevators, washing machines and cars
- Intelligent agents are used in many software applications
- Robots explore other worlds, and toy robots play with children (and some adults)
- Expert systems diagnose diseases and recommend remedies
- Computer games use AI

Applications of AI

- Game Playing
- Automated Reasoning and Theorem Proving
- Expert Systems
- Natural Language Understanding and Semantic Modeling
- Modeling Human Performance
- Planning and Robotics
- Languages and Environments for AI
- Machine Learning
- Alternative Representations: Neural Nets and Genetic Algorithms
- Computer vision and Image Understanding

Strong AI and Weak AI

- There are two entirely different schools of AI:
- Strong AI:
  - This is the view that a sufficiently programmed computer would actually *be* intelligent and conscious, and would think in the same way that a human does
  - Strong AI is currently the stuff of science fiction, although there are many that believe that machines will indeed be capable of real thought at some point in the future
- Weak AI:
  - This is the use of methods modeled on intelligent behavior to make computers more efficient at solving problems
- This course is concerned with Weak AI

Another Dichotomy in AI

- Another pair of schools of thought in AI is the so called “neats” vs “scruffies”
- Neats:
  - – Those that think AI theories should be rooted in mathematical rigor and appreciate an elegant solution
  - – Notable neats: John McCarthy, Allan Newell (GPS)
- Scruffies:
  - – Those that would rather try out lots of different ideas (trial-and-error) and assess what works best in a pragmatic way
  - Notable scruffies: Marvin Minsky, Terry Winograd (HCI)
- Both approaches are important as sometimes the “elegant” solution is not necessarily the most correct

Chinese Room Counter-Argument

John Searle 1980 – “The Chinese Room” counter-argument

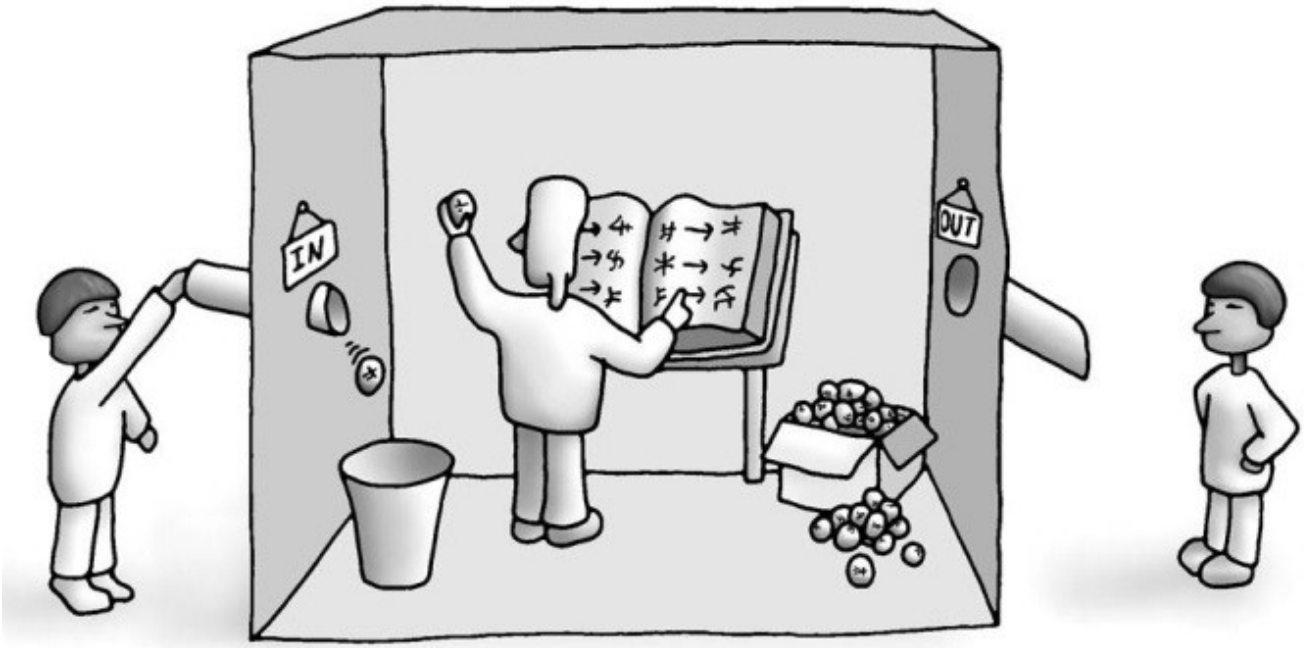


Image from http://10est.com/11/the-chinese-room

HAL – Fantasy or Reality?

- HAL – the computer in the film 2001: A Space Odyssey
  - Plays chess with humans (and wins)
  - Reads people's lips
  - Engages in conversation with humans
  - Eventually goes insane
- Computers can play chess, and beat some players
- Reading lips is very hard to automate.
- The conversational skills of the best systems today are very weak

Strong Methods and Weak Methods

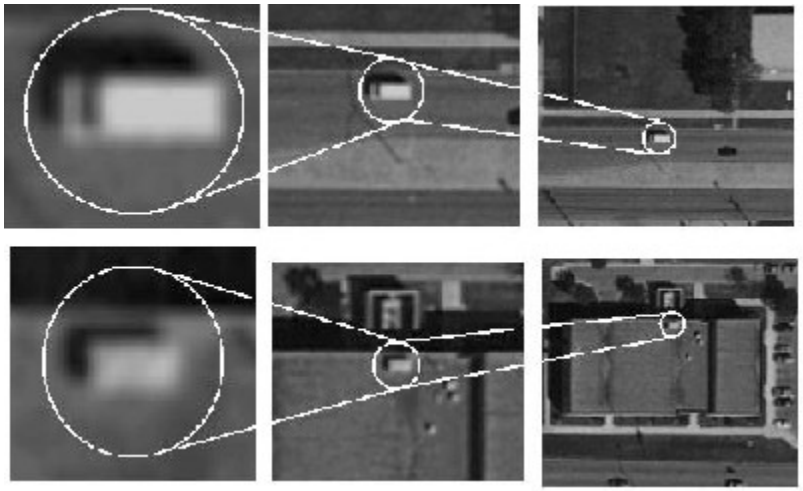
- Not to be confused with Strong AI and Weak AI
- Strong methods use knowledge about the world to solve problems
- Weak methods use logic and other symbolic systems
- Strong method systems rely on weak methods, as knowledge is useless without a way to handle that knowledge
- Weak methods are in no way inferior to strong methods – they simply do not employ world knowledge

Strong Methods Use Context

Count all the chairs

Strong Methods Use Context

Which one is a car?

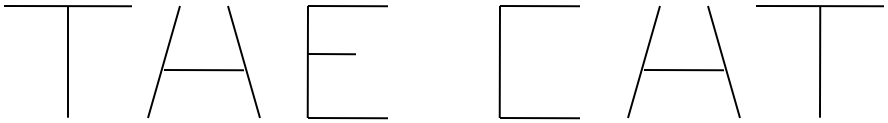


Strong Methods Use Context

Are these letters “A” or “H”?

Strong Methods Use Context

Are these letters “A” or “H”?



331 – Intro to Intelligent Systems
Week01c
Intelligent Agents
R&N Chapter 2

T.J. Borrelli

Agents and Environments

- Last time we spoke about **rational agents** and their central role to AI
- Now we define this more formally with the goal of identifying design principles for building successful agents
- In this context “successful” means a system that can reasonably be thought of an intelligent
- A rational agent should behave “as well as possible” given the environment/conditions it must exist within
- **Agent –** anything that perceives environment through sensors and acts upon environment through actuators

Agents

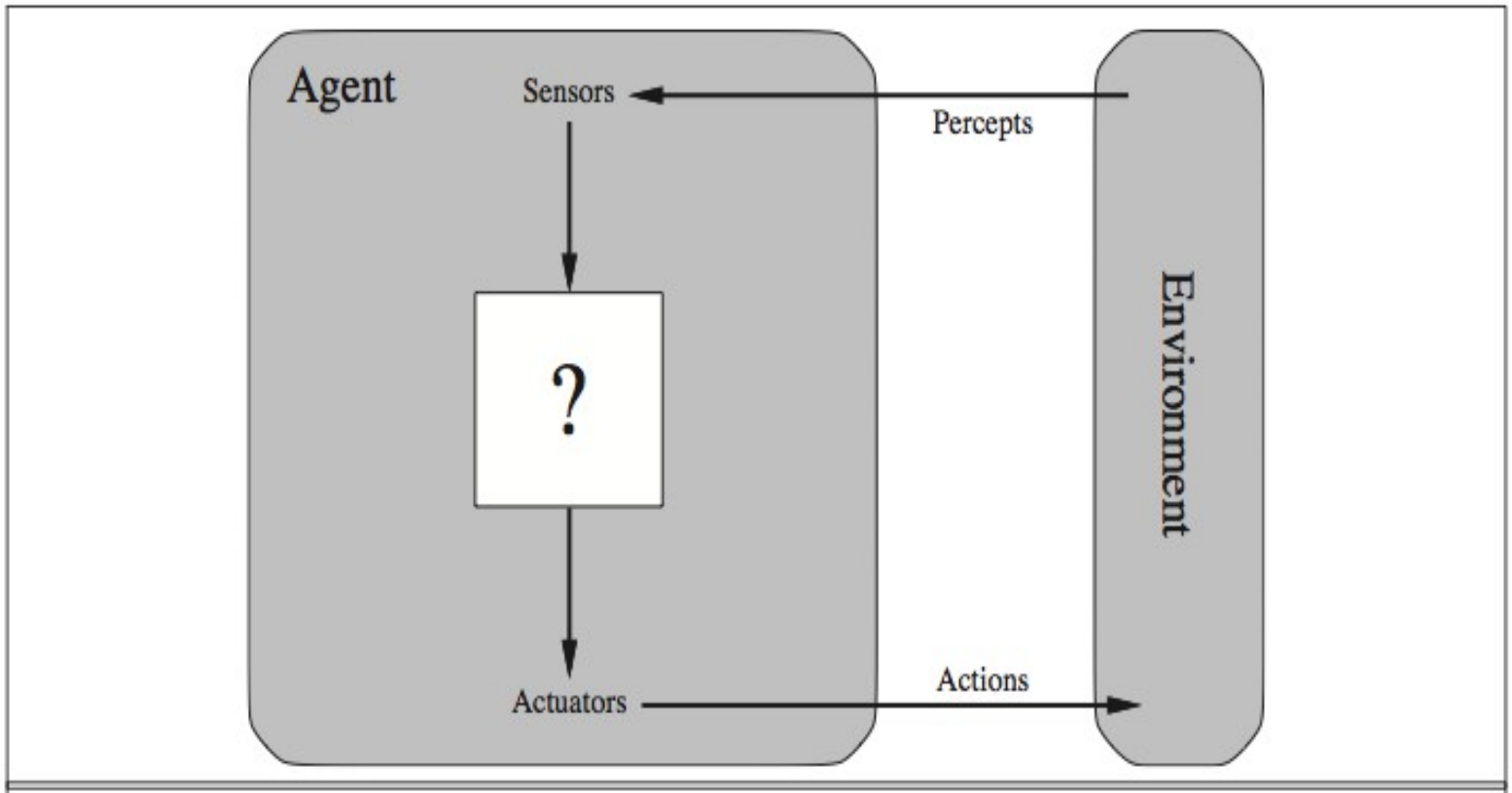


Figure 2.1 FILES: figures/agent-environment.eps (Tue Nov 3 16:22:19 2009). Agents interact with environments through sensors and actuators.

Rational agents

- Rational agents do the “right” thing
- How do we define this?
- In terms of the consequences of a given action
- Performance measure – design these according to the outcome we want in the environment

Rationality

- What is rational at any given time depends on
  - The performance measure (defines success)
  - The agent's prior knowledge of environment
  - The actions that the agent can perform
  - The agent's percept sequence to date (history of things perceived)
- We can now define a **rational agent:** For each possible percept sequence (input), a rational agent should select an action to maximize its performance measure given it percept sequence and other knowledge

Rationality

- Rationality is not the same as perfection
- It is typically not possible to be able to perfectly predict the actual outcomes of every action (that would be omniscience).
- Rationality therefore maximizes *expected* performance

Rational Agents and learning

- Rational agents can often engage in information gathers – performing certain actions in order to modify future percepts
- Exploration is an example of this
- Exploration is useful in environments that are unknown initially
- Our rational agents should also **learn** about their environment
- The initial configuration could reflect some prior knowledge or base rules and we can add to this by exploration

PEAS and the nature of environments

- How do we now build rational agents?
- Need to think about **task environments –** the “problems” our agents are “solutions” for
- Our Task Environment covers
- **P – P**erformance measure (often will involve trade-offs)
- **E – E**nvironment
- **A – A**ctuators
- **S – S**ensors

PEAS example

- Example of an automated taxi driver (open-ended)

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits,	Roads, other traffic, pedestrians, customers	Steering, acceleration, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors

Properties of task environments

- Fully observable vs. partially observable
- Single agent vs. multiagent
- Deterministic vs. stochastic
- Episodic vs. sequential
- Static vs. dynamic
- Discrete vs. continuous
- Known vs. unknown

Fully observable vs. Partially observable

- Fully observable: If the agent's sensors give it access to the complete state of the *relevant* information from the environment at each point in time
- Nice because we don't need to keep track of lots of state information
- Partially observable when we get a subset of information from environment
- Also possible that the environment is unobservable – can still sometimes achieve goals

Single agent vs. multiagent

- Are we working against other agents (competition)
- Or are we working in conjunction with other agents (cooperation)
- Are we the only agent engaged in the environment

Deterministic vs. stochastic

- **Deterministic** – If the next state of the environment is completely determined by the current state and the action executed by the agent
- **Stochastic** – otherwise

- An environment is **uncertain** if it is not fully observable or not deterministic

Episodic vs. Sequential

- **Episodic** – agent's experience is divided into atomic (indivisible) parts; in each part agent receives a percept and performs a single action; next episode does not depend on actions taken in previous

- **Sequential** – The current decision could affect all future decisions (e.g. chess and taxi driving)

Static vs. Dynamic

- **Static** – environment does not change while agent is deciding on next action (e.g. crossword puzzles)
- **Dynamic** – environment may change while agent is deliberating (e.g. taxi driving)
- **Semi-dynamic** – environment does not change but the agent's performance score does

Discrete vs. Continuous

- **Discrete** – state of environment and time are separate and distinct slices
- For example, Chess has distinct set of percepts and actions

- **Continuous** –state of environment and time are along a spectrum of possible values (e.g. taxi driving)

Known vs. Unknown

- **Known** – agent state of knowledge about the rules/laws of physics in the environment
- The outcomes (or probabilities of outcomes) are known for all actions
- **Unknown** – If environment is unknown agent will have to learn how it works to make good decisions
- Note that the distinction between known and unknown environments is not the same as observable and partially observable
- It is possible for known environments to be partially observable (solitaire – know the rules, but can't see all the cards)
- Converse is also true, possible for an unknown environment to be fully observable (a new video game – screen shows all elements but don't know what buttons to press until you try them)

Tough combination

- Hardest case of the above is partly observable (or unobservable), multiagent, stochastic, sequential, dynamic, continuous and unknown
- Our taxi driving agent represents all of these

What is each of the following

	Crossword puzzle	Spam filter	Chess w/ clock	Image analysis	Poker	Interactive tutor
Observable						
Single vs multi						
Deterministic						
Episodic						
Static						
Discrete						
Known						

Simple agent pseudocode

function TABLE-DRIVEN-AGENT(percept) **returns** action

persistent: *percepts,* a sequence, initially empty

table, a table of actions, indexed by percept sequences, initially fully specified

append *percept* to end of *percepts*

action ← LOOKUP(*percepts, table)*

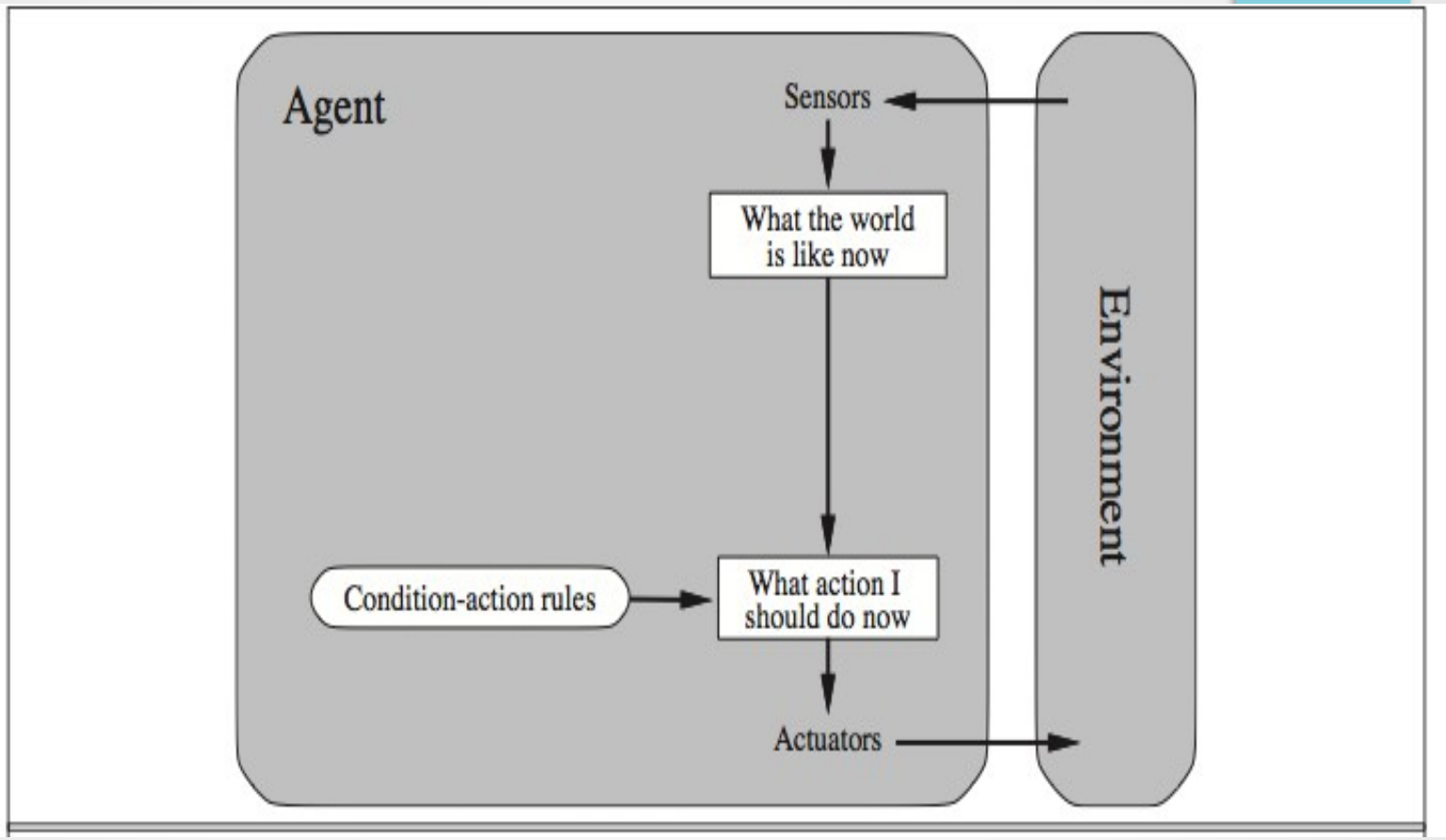
return *action*

- Describes a trivial agent program that keeps track of the percept sequence and then uses it to index into a table of actions
- Table represents the agent function that the agent program embodies
- To build a rational agent – we must construct a table that contains the appropriate action for each percept sequence

Basic Agent Architectures

- Simple reflex
- Reflex with state
- Goal-based
- Utility-based

Simple Reflex Agent



- Responds directly to percepts – acts according to a rule whose condition matches the current state

Simple Reflex Agent pseudocode

function SIMPLE-REFLEX-AGENT(*percept*) **returns** action

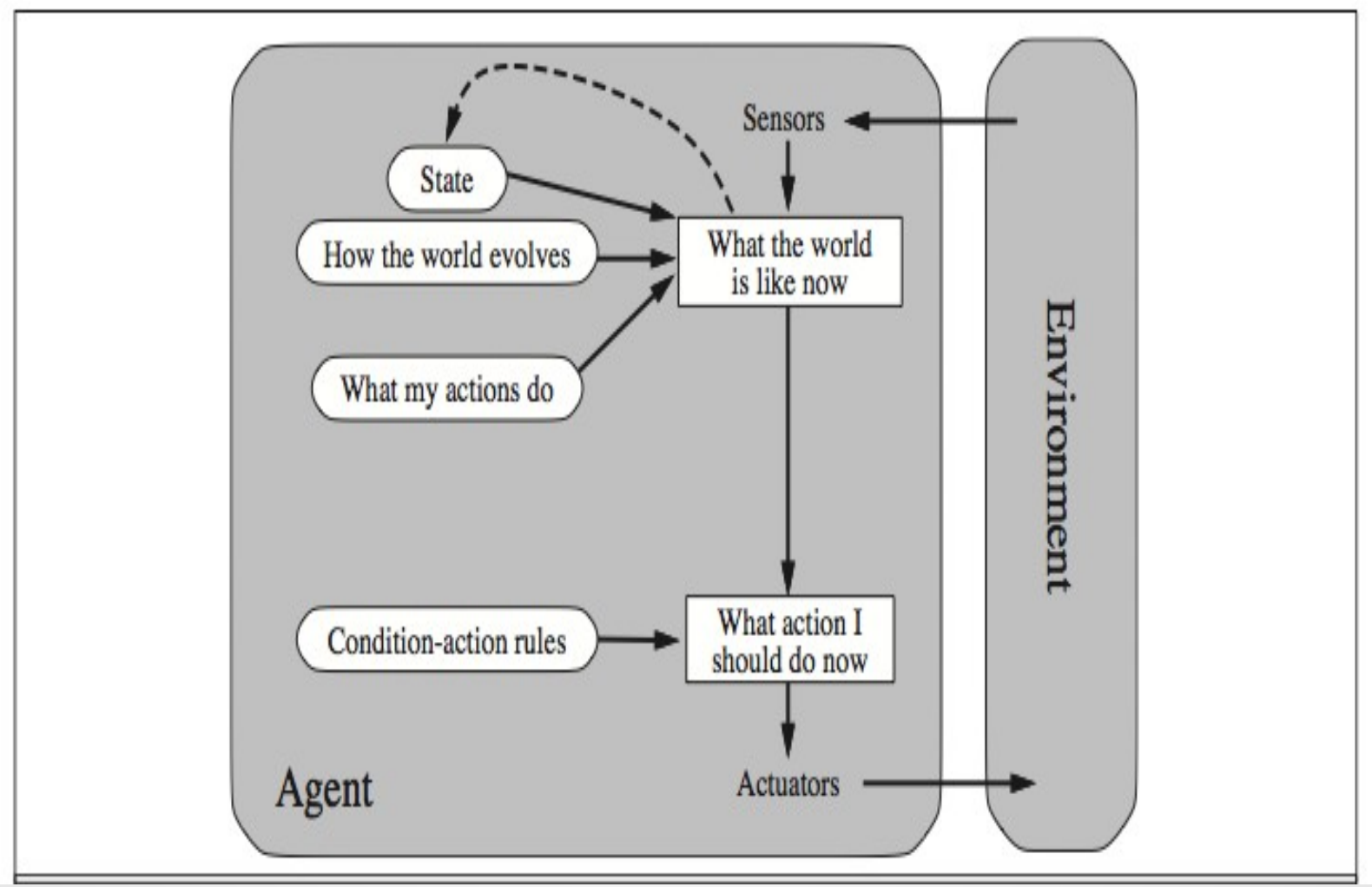
persistent: *rules,* set of condition-action rules

state ← INTERPRET-INPUT(*percepts*)

action ← RULE-MATCH(*state, rules)*

return *action*

Model-based reflex agent



- Maintain internal state to track aspects of world that are not evident in current percept

Model-Based Agent pseudocode

function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** action

persistent: *state,* the agent's current belief of world state

rules, set of condition-action rules

action, the most recent action, initially none

model, desc. how next state depends on cur. state & action

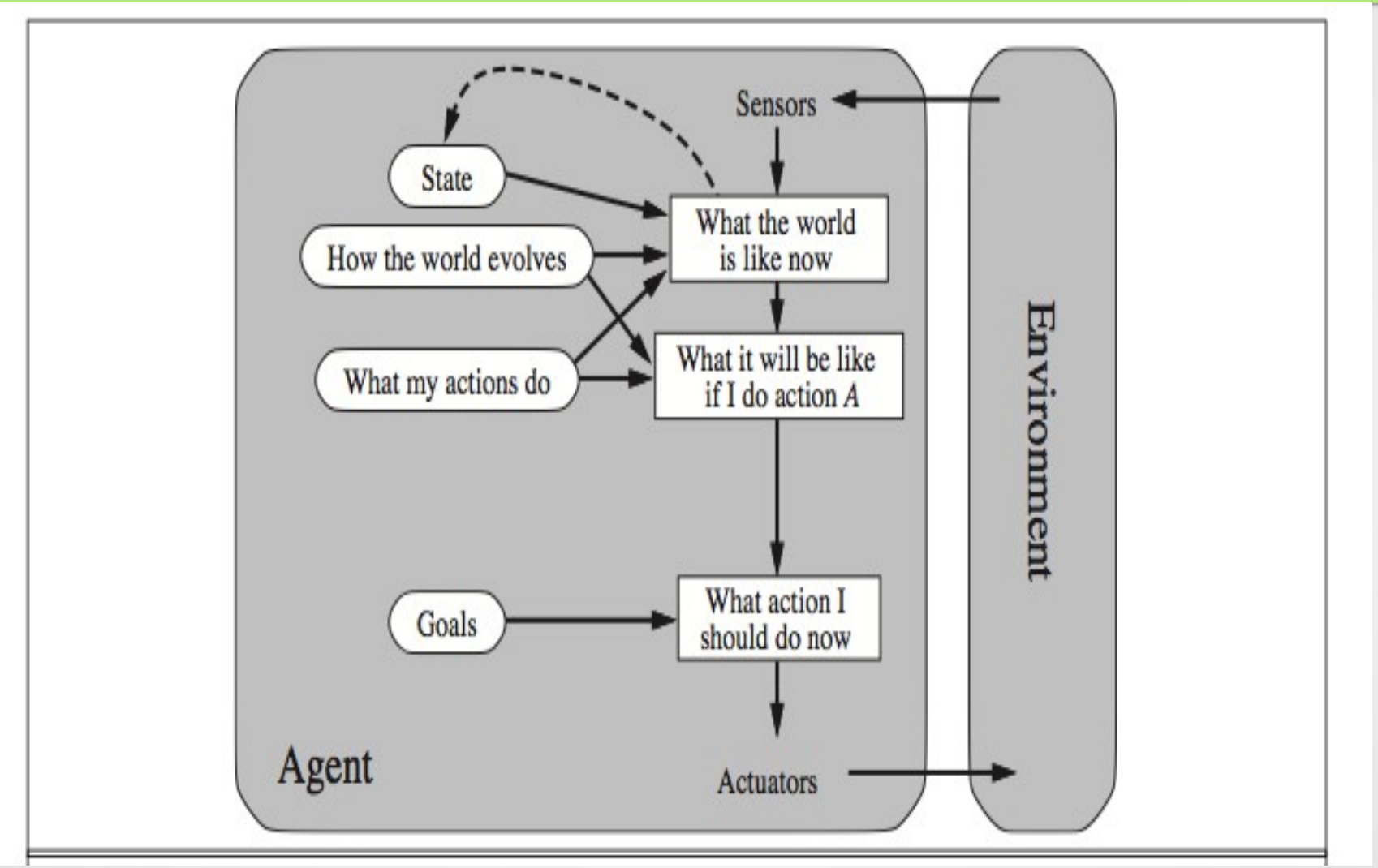
state ← UPDATE-STATE(*state, action, percept, model*)

rule ← RULE-MATCH(*state, rules)*

action ← *rule.*ACTION

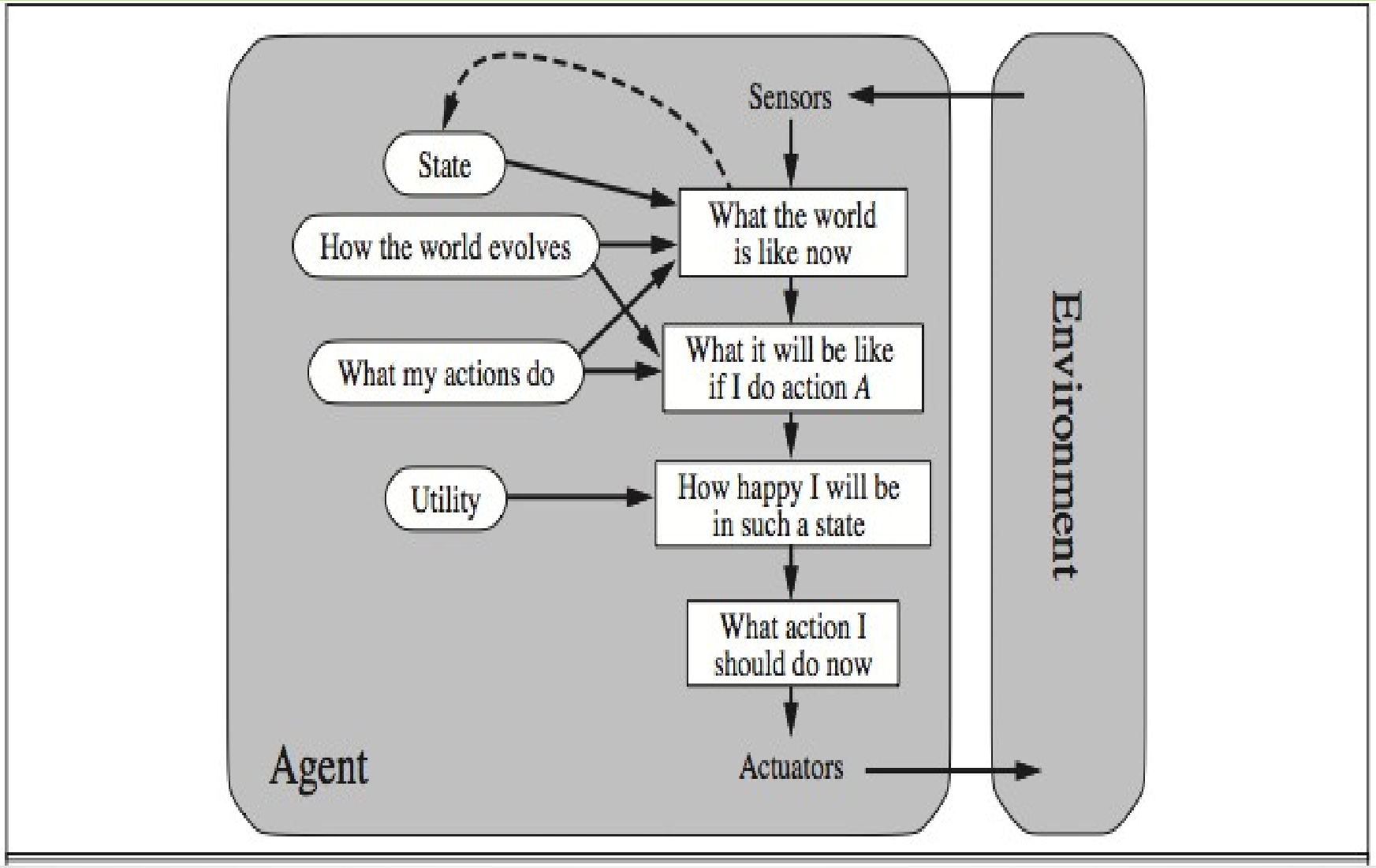
return *action*

Model-based, Goal-based agent



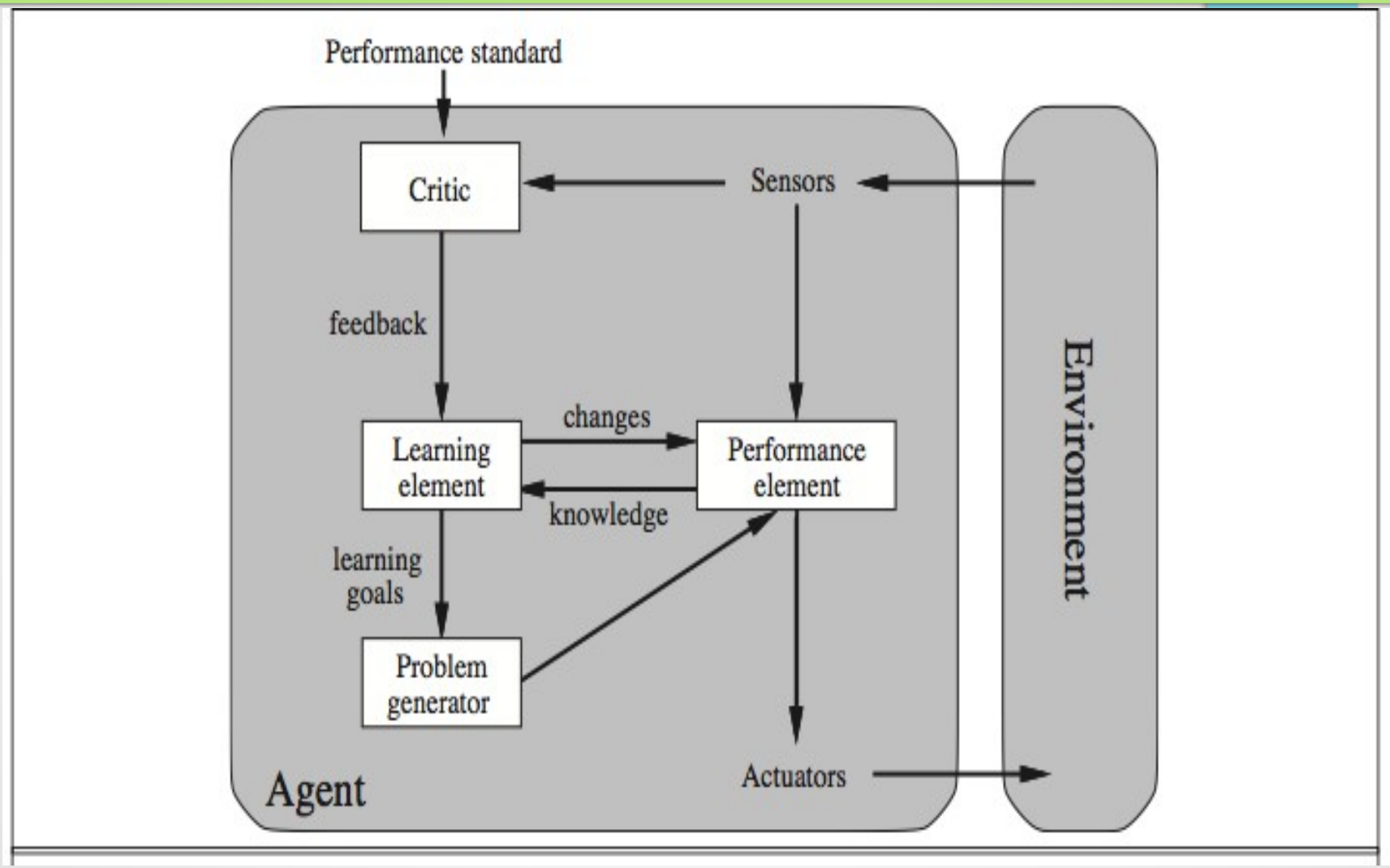
- Acts to achieve certain goals

Utility-based Agent



- Tries to maximize their own expected utility (“happiness”)

General learning agent



- Goal of learning: to improve performance

Summary

- Agents interact with the environment using sensors and actuators
- Performance measure evaluates environment state sequence
- A perfectly rational agent maximizes (expected) performance
- PEAS descriptions define task environments
- Environments categorized along different dimensions
  - Observable, deterministic, episodic, static, discrete, single/multi . . .
- Basic agent architectures
  - Simple reflex
  - Reflex with state
  - Goal-based
  - Utility-based

331 – Intro to AI
Week02
(R&N Chapter 3)
Uninformed Search

T.J. Borrelli

Uninformed vs informed search

- Uninformed search
  – Can only distinguish goals from non-goal states

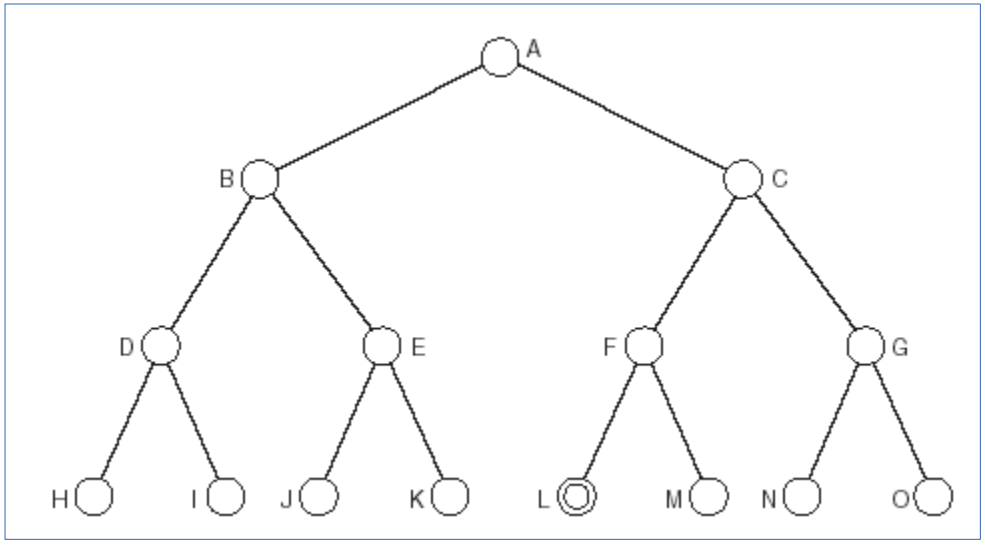
- Informed search
– Have information about progress towards the optimal solution
– can drastically improve time complexity

State Space Representation of a Problem

- In the *state-space representation* of a problem, the nodes of a tree (called the search tree) correspond to partial problem solution states, and the links correspond to steps in a problem-solving process
- The root of the tree is the start state
- Goal states are leaf nodes (but not the other way around)
- A state-space search characterizes problem-solving as the process of finding a solution path from the start state to a goal

Search Tree

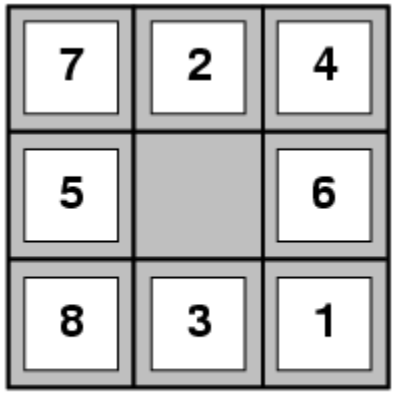
- A is the root node (start state).
- L is the goal node (goal state).
- H, I, J, K, M, N and O are other leaf nodes.
- There is only one solution path: A, C, F, L



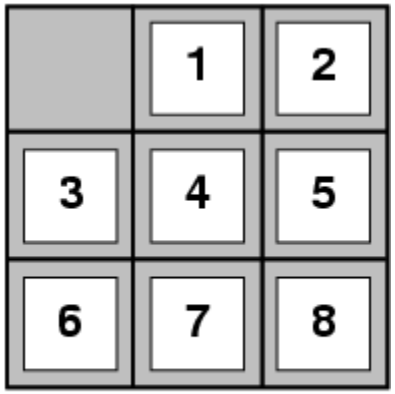
Generating States

- The generation of new states is done by applying operators (e.g., legal moves) to existing states on a path
- The path from the start state to a goal state contains the series of operations that lead to the problem solution

The 8-Puzzle



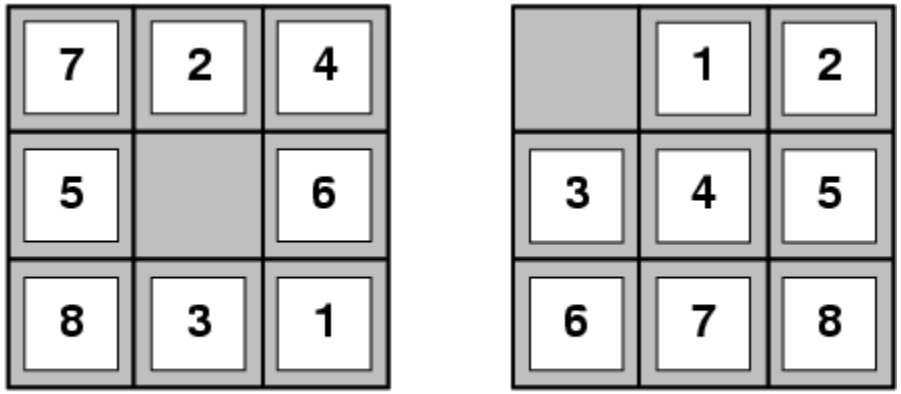
Start State



Goal State

- states?
- actions?
- goal test?
- path cost?

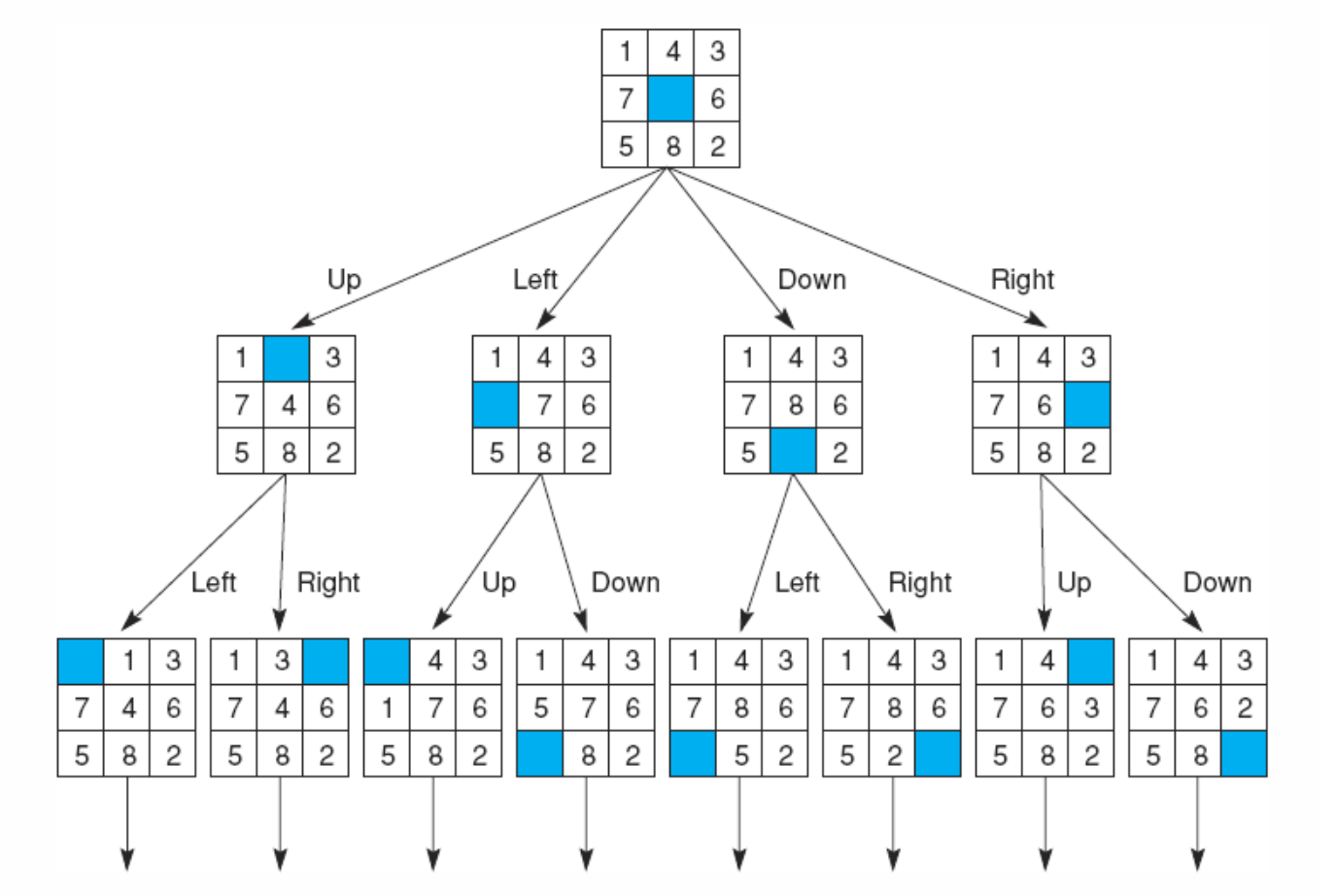
The 8-Puzzle



- states? locations of tiles (2D array)
- actions? move blank left, right, up, down
- goal test? goal state (given)
- path cost? 1 per move

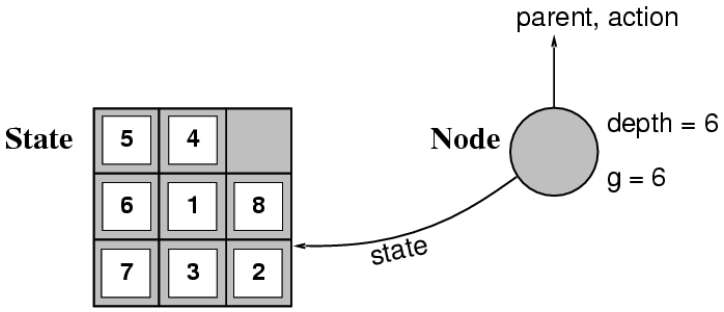
[Note: optimal solution of *N*-Puzzle family is NP-hard]

The 8-Puzzle



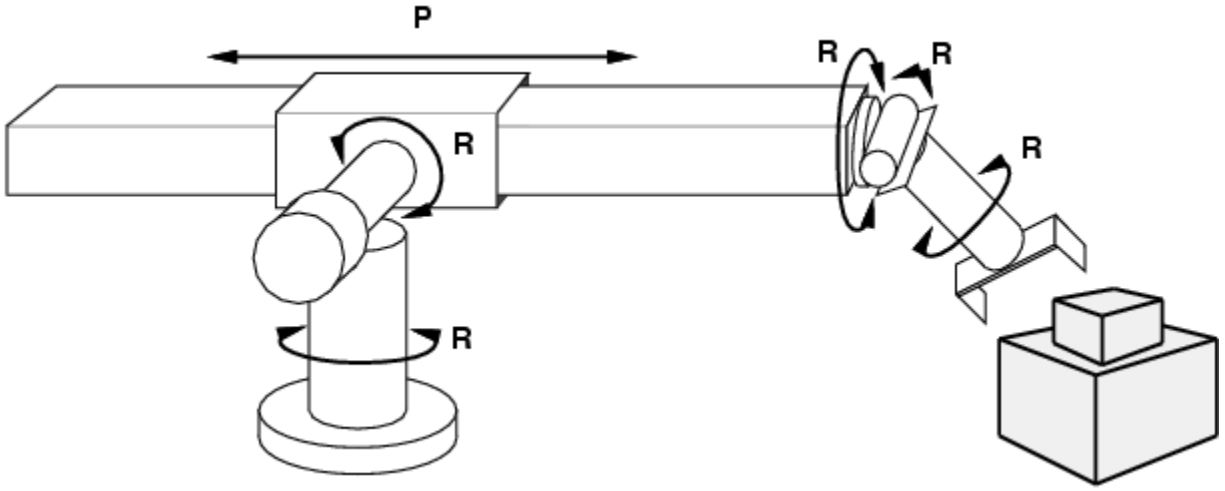
States vs. Nodes

- A *state* is a representation of a physical configuration
- A *node* is a data structure that represents part of a search tree. A node includes state, parent node, action, the path cost *g(x)*, and depth



- The Expand function creates new nodes, filling in the various fields and using the Successor Function of the problem to create the corresponding states

Robotic Assembly



states? coordinates of robot joints and angles
actions? new coordinates of robot joints and angles
goal test? complete assembly of part
path cost? time to execute

Missionaries and Cannibals

Problem Description:
Three missionaries and three cannibals are on one side of a river, with a canoe.
They all want to get to the other side of the river.
The canoe can only hold one or two people at a time.
At no time should there be more cannibals than missionaries on either side of the river as this would probably result in the missionaries being eaten.
How can they all cross the river and end up on the other side safely?

Missionaries and Cannibals

- The first step in solving the problem is to choose a suitable representation
- Represent the number of cannibals, missionaries, and canoes on each side of the river as a three element array of integers

  [number of cannibals, number of missionaries, number of canoes]

- Start state is therefore:

left side of river	right side of river
3,3,1	**0,0,0**

- An example of a state that must be avoided is **2,1,1**

Missionaries and Cannibals

- In fact, since the system is closed, we only need to represent one side of the river, as we can deduce the other side
- We will represent the right side of the river, and omit the left side
- So start state is:

0,0,0

Missionaries and Cannibals

Now we have to choose suitable operators that can be applied:

1. Move one cannibal across the river
2. Move two cannibals across the river
3. Move one missionary across the river
4. Move two missionaries across the river
5. Move one missionary and one cannibal

Missionaries and Cannibals

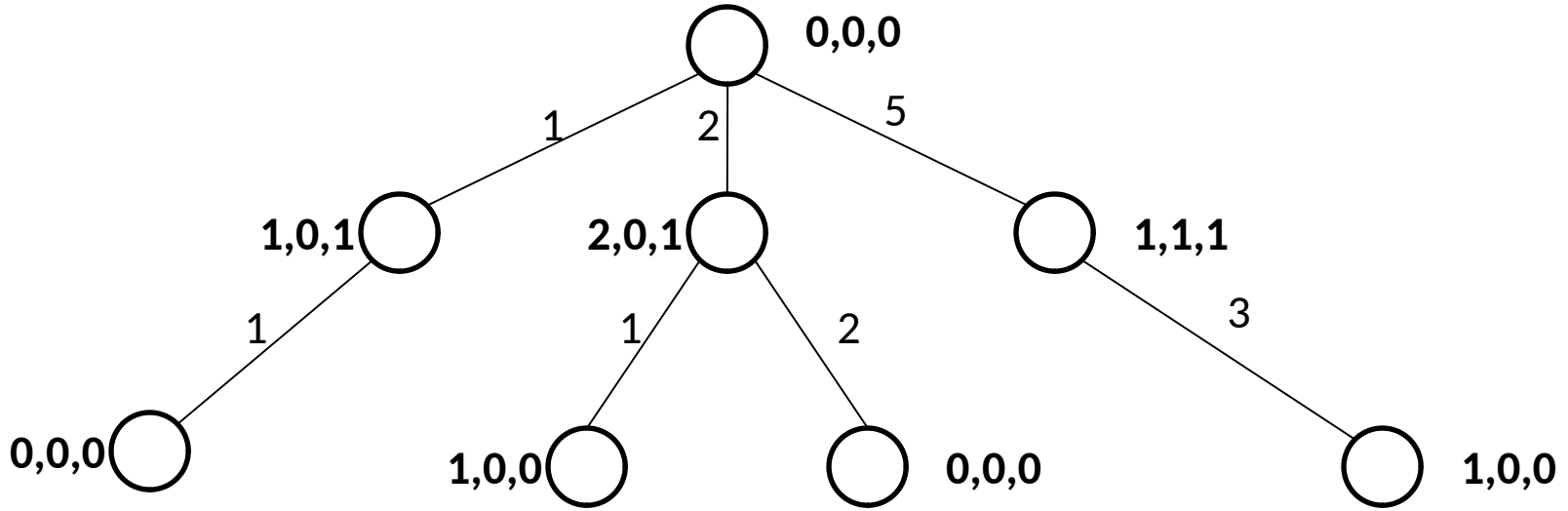
- So if we applied operator 5 (move one cannibal and one missionary to the right side) to the state represented by **1,1,0** then we would result in state **2,2,1**. One cannibal, one missionary, and the canoe have now moved to the right side
- Applying operator 3 (move one missionary to the left side) to this state would lead to an illegal/failing state **2,1,0**

Missionaries and Cannibals

- In addition, we need to have a test that can identify if we have reached the goal state **3,3,1** (or 3,3,0 but this is not possible)
- The path cost could be the number of steps taken, or the number of times an operator is applied
  - In some cases it is desirable to find a solution that minimizes cost

Missionaries and Cannibals

Below is the first three levels of the search tree for this problem (edges are labeled with the operator that has been applied):



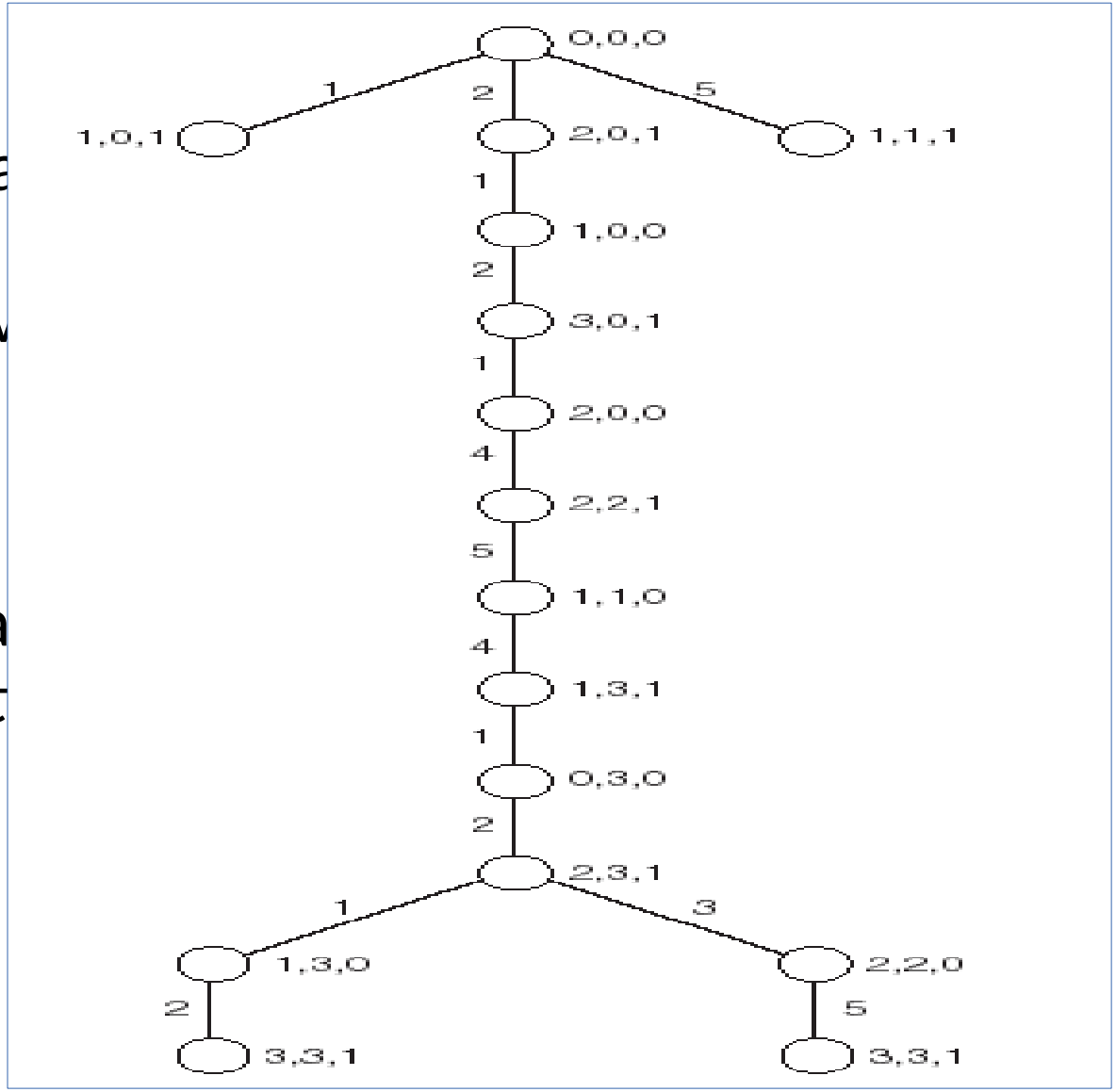
By extending this path to include all possible paths, and the states those paths lead to, a solution can be found. A solution to the problem would be represented as a path from the root node to a goal node.

Missionaries and Cannibals

- Note that this tree has some cycles
  - Applying operator 1 (moving one cannibal to the other side), and then applying the same operator again, we return to the start state
- This is a perfectly valid way to try to solve the problem, but not a very efficient one
- A more effective representation would be one that did not include any cycles

Missionaries And Cannibals Search Tree

- Cycles and repea
  states
- have been remov
- Nodes represent
- edges represent
  operators
- There are two pa
- lead to the solut



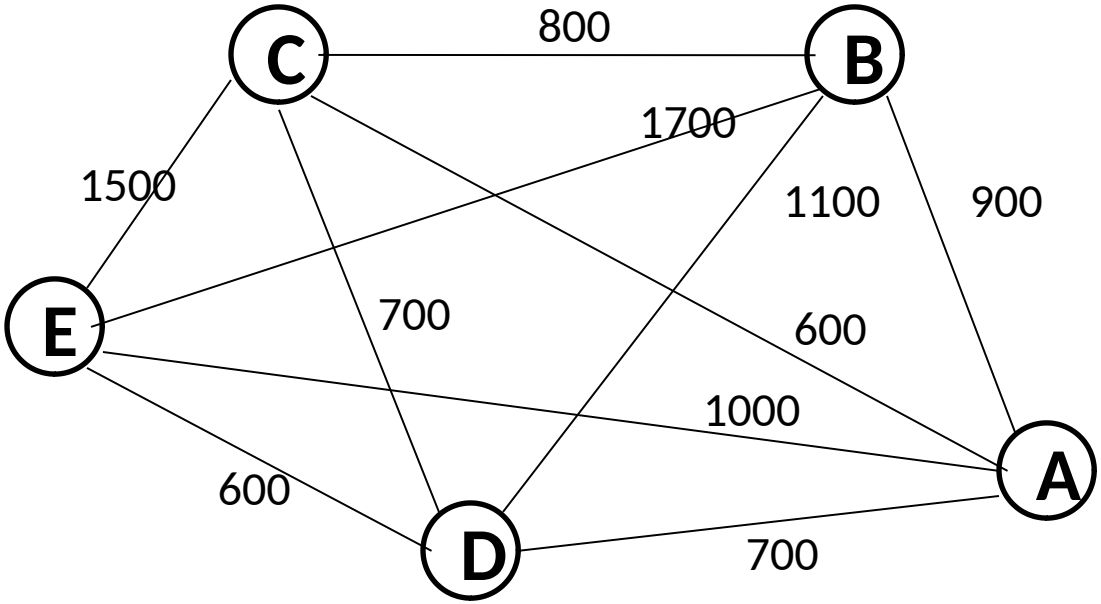
Traveling Salesperson (TSP)

Problem Description:

A salesperson must visit each of a set of cities and then return home. The problem is to find the shortest path that lets the salesperson visit each city. The salesperson lives in Atlanta and is traveling by plane (assume that direct flights are possible between any pair of cities).

City A – Atlanta
City B – Boston
City C – Chicago
City D – Dallas
City E – El Paso

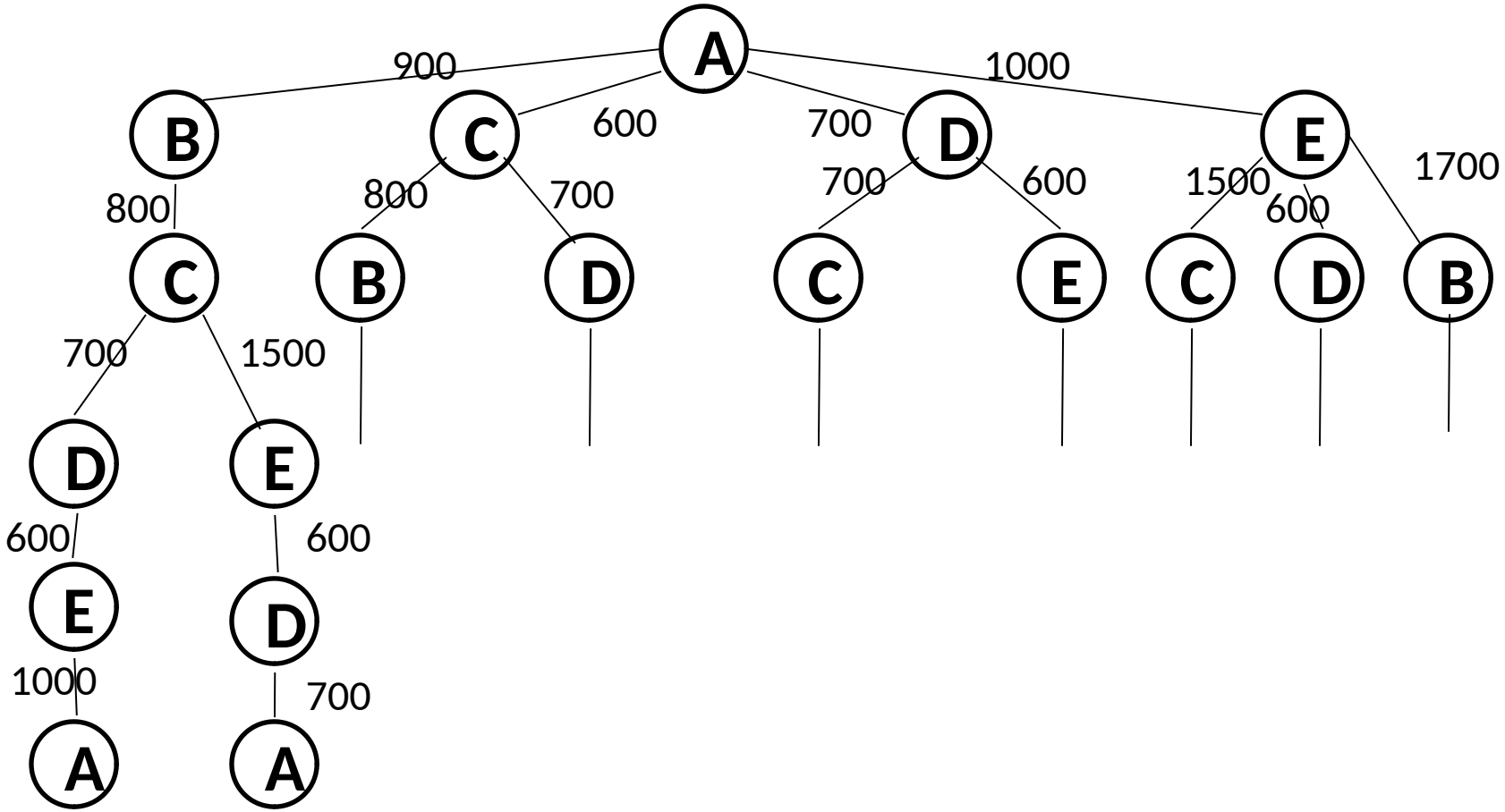
Traveling Salesperson (TSP)



Traveling Salesperson (TSP)

- To solve this problem using search, a different representation is needed
- A node in the search tree represents a city that has been reached by the path up to that point
  - Each node represents the path from city A to the city named at that node
- Edges represent the distance to the next city on the path

Traveling Salesperson



Traveling Salesperson (TSP)

- As with the Missionaries and Cannibals (M&C) example, cyclical paths have been excluded from the tree
- Unlike the M&C tree, this tree does allow repeated states
  - This is because in this problem each state must be visited once, and so a complete path must include all states

Combinatorial Explosion

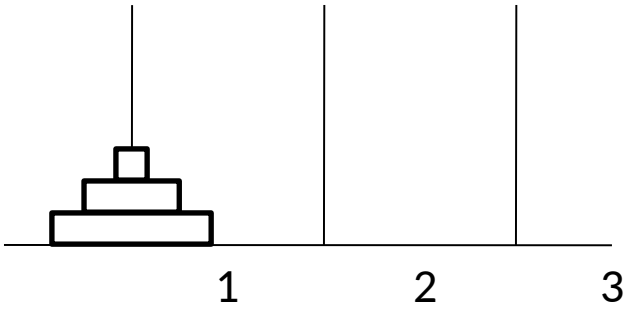
- In total there will be (n-1)! possible paths for a Traveling Saleperson Problem (TSP) with n cities
  - This is because we are constrained in our starting city and, thereafter, have a choice of any combination of (n-1) cities
  - In problems with a small number of cities such as 5 or even 10 this means that the complete search tree can be evaluated by a computer program without much difficulty
  - If the problem consists of 40 cities, then there would be 39! paths, which is roughly $2*10^{46}$
  - Methods that try to examine all of these paths are called **brute-force search methods**
  - To solve search problems with large trees **heuristics** must be used

Towers of Hanoi

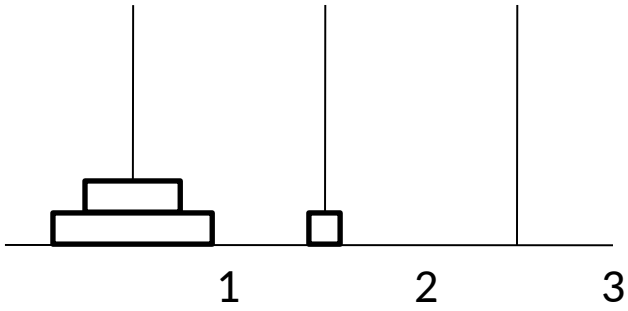
Problem Description:

You have three pegs and a number of disks of different sizes. The aim is to move from the starting state, where all the disks are on the first peg (in size order smallest on top) to the goal state where all the pegs are on the third peg, also in size order. You are allowed to move one disk at a time, as long as there are no disks on top of it, and as long as you do not move it on top of a peg that is smaller than it.

Towers of Hanoi



Start state



State after one disk has been moved from peg 1 to peg 2

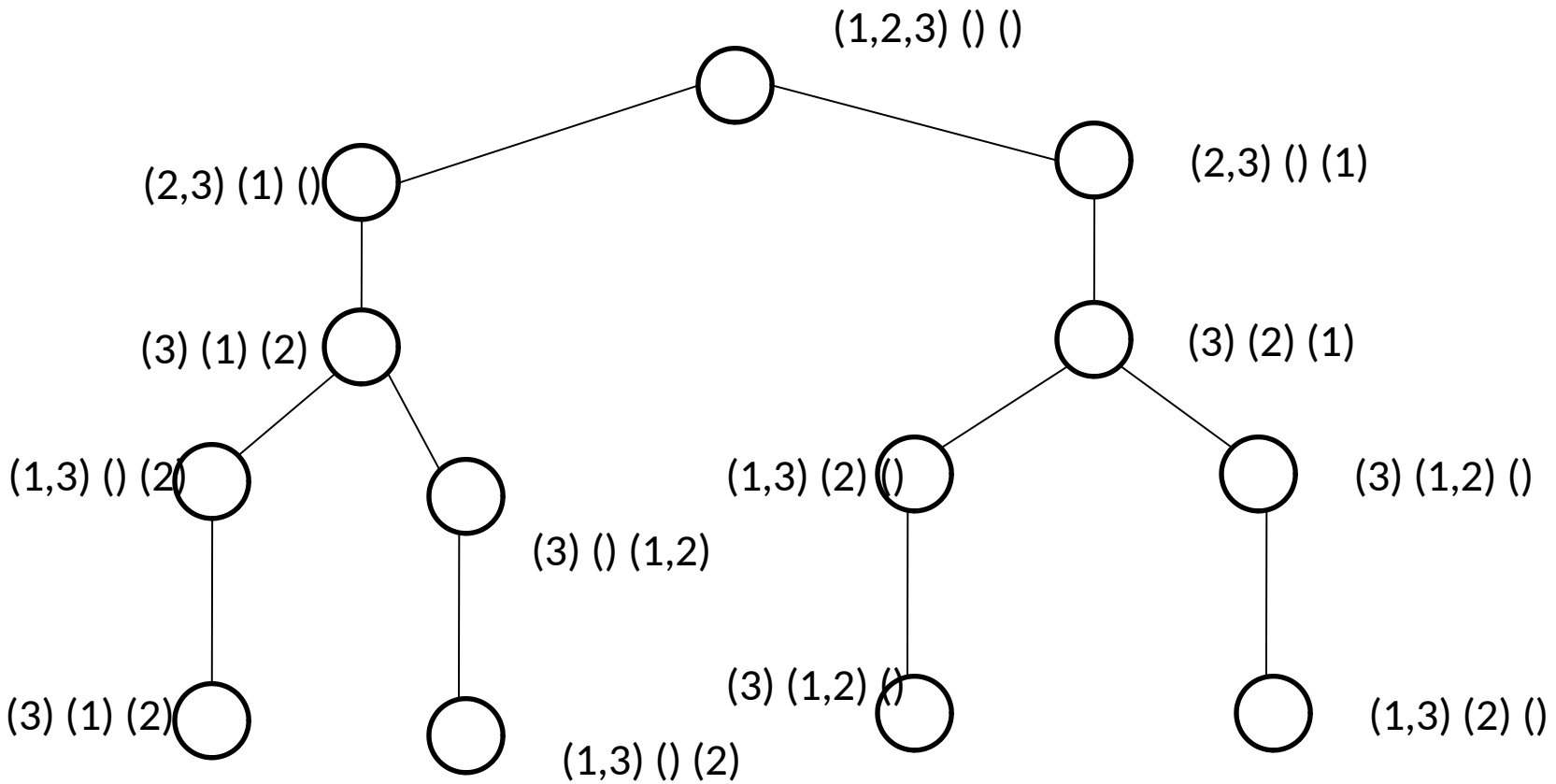
Towers of Hanoi

Operators:

1. Move disk from peg 1 to peg 2
2. Move disk from peg 1 to peg 3
3. Move disk from peg 2 to peg 3
4. Move disk from peg 1 to peg 2
5. Move disk from peg 3 to peg 1
6. Move disk from peg 3 to peg 2
7. Move disk from peg 1 to peg 2

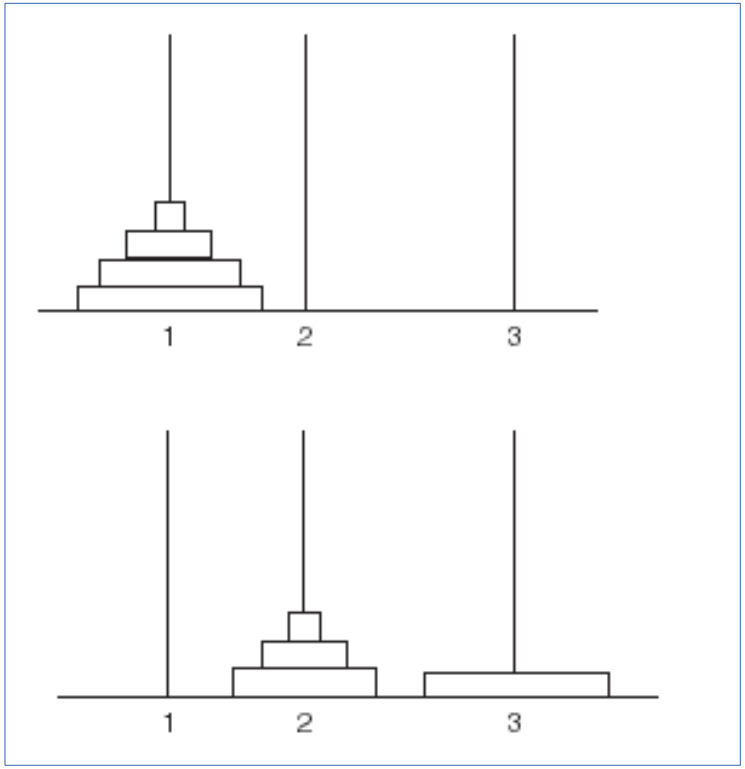
Represent a state using vectors of numbers, where 1 represents the smallest disk and 3 the largest disk.
Starting state: (1,2,3) () () and goal state: () (1,2,3) ()
State on right in previous slide: (2,3) (1) ()

Towers of Hanoi



Problem Reductions

- To solve the Towers of Hanoi problem with 4 disks, you can first solve the same problem with 3 disks
- The solution to the overall problem is solved recursively on smaller sized problems:
- Step 1: Move N-1 disks from Peg 1(source) to Peg 2 (spare)
- Step 2: Move **one** disk from Peg 1 (source) to Peg 3 (destination)
- Step 3: Move N-1 disks from Peg 2 (spare) to Peg 3 (destination)



Search Strategies

A state space may be searched in two directions:

- From the given data of a problem instance toward a goal (data-driven, or *forward chaining*)
- From a goal back to the data (goal-driven or *backward chaining*)

Search Strategies

Use data-driven search if:

1. All or most of the data are given in the initial problem statement
2. There are a large number of potential goals, but there are only a few ways to use the facts and given information of a particular problem instance
3. It is difficult to form a goal or hypothesis

Search Strategies

Use goal-driven search if:

1. A goal or hypothesis is given in the problem statement or can easily be formulated (e.g., math theorem or medical diagnosis)
2. There are a large number of rules that match facts of the problem and thus produce an increasing number of conclusions or goals
3. Problem data is not given and must be acquired

Evaluating Search Strategies

- Strategies are evaluated along the following dimensions:
  - Completeness: Does it always find a solution if one exists?
  - Optimality: Does it always find a least-cost solution?
  - Time complexity: The number of nodes generated
  - Space complexity: The maximum number of nodes in memory
- Time and space complexity are measured in terms of
  - *b:* Maximum branching factor of the search tree (i.e. number of successive nodes of a parent)
  - *d:* Depth of the least-cost solution
  - *m*: Maximum depth of the state space (may be ∞)

Brute-Force Search

- Search methods that examine every node in the search tree – also called **exhaustive**
- "Generate and Test" is the simplest brute force search method:
  - **Generate possible solutions to the problem**
  - **Test each one in turn to see if it is a valid solution**
  - **Stop when a valid solution is found**
- The method used to generate possible solutions must be carefully chosen

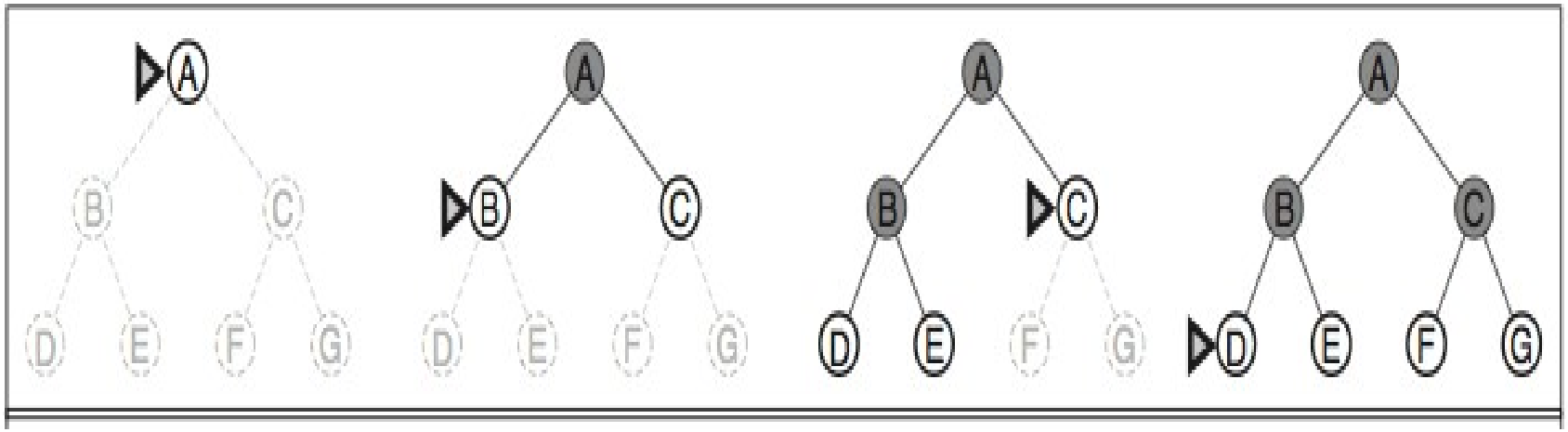
Uninformed Search Strategies

- Uninformed search strategies use only the information available in the problem definition
  - Breadth-first search
  - Depth-first search
  - Uniform-cost search (Dijkstra's shortest path)
  - Depth-limited search
  - Iterative deepening search

Breadth-First Search

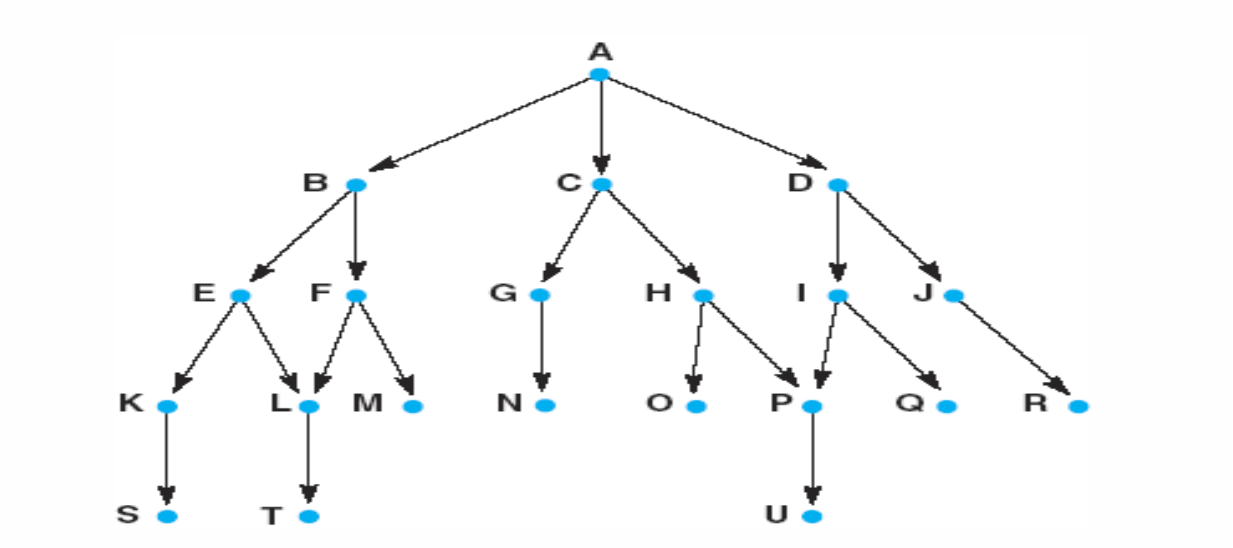
```
function breadth_first_search:

    open := [Start]                                          % initialize
    closed := [ ]
    while open ≠ [ ] do                                      % states remain
        Remove (dequeue) leftmost state from open, call it X
        if X is goal then return SUCCESS                     % goal found
        else
            generate children of X
            if any child of X is goal then return SUCCESS    % goal found
            else
                put X on closed
                discard children of X if on open or closed   % loop check
                put remaining children on right end of open (enqueue)
    return FAILURE
```

Breadth-First Search

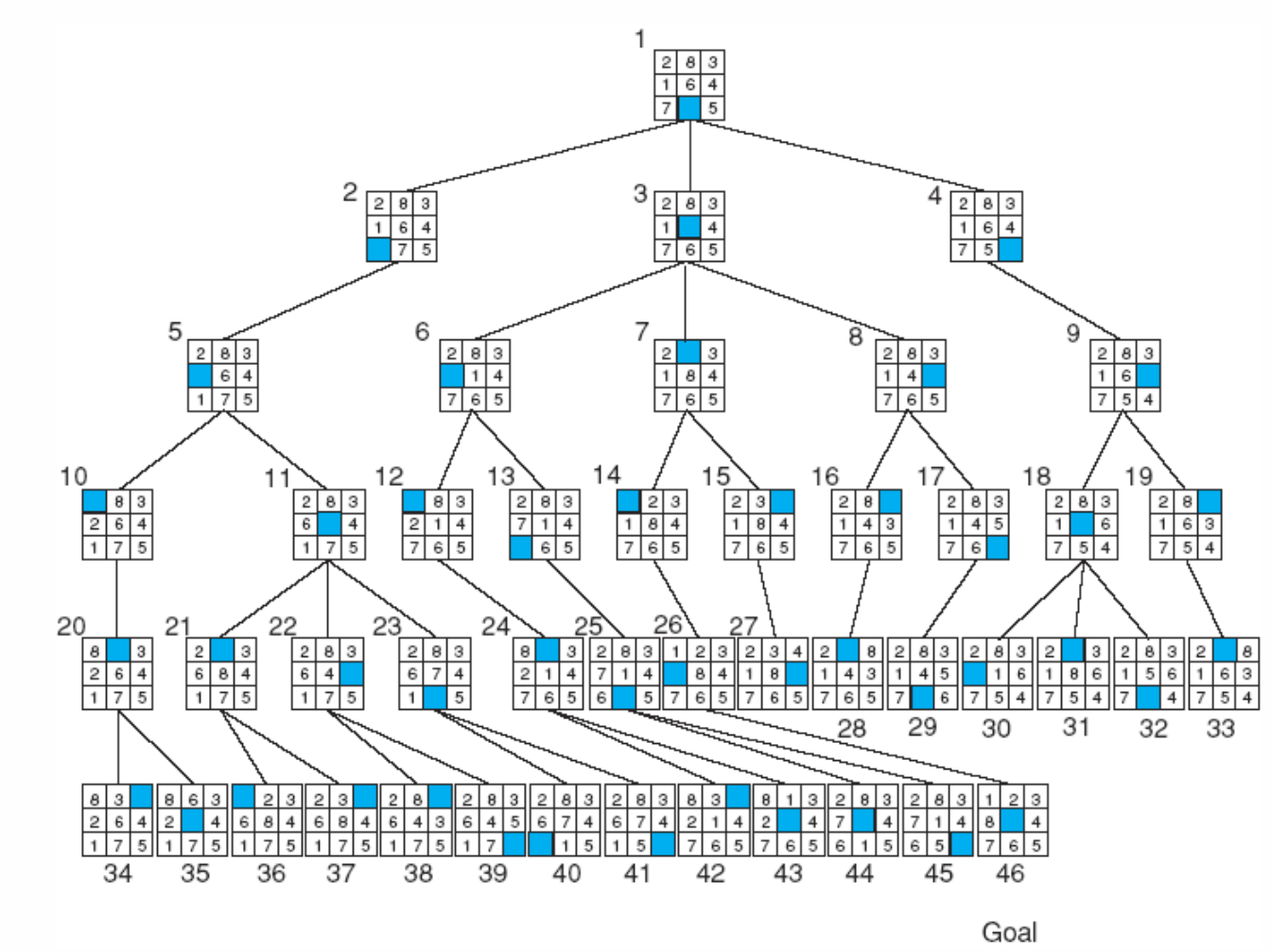


Breadth-First Search

1. **open = [A]; closed = [ ]**
2. **open = [B,C,D]; closed = [A]**
3. **open = [C,D,E,F]; closed = [B,A]**
4. **open = [D,E,F,G,H]; closed = [C,B,A]**
5. **open = [E,F,G,H,I,J]; closed = [D,C,B,A]**
6. **open = [F,G,H,I,J,K,L]; closed = [E,D,C,B,A]**
7. **open = [G,H,I,J,K,L,M]** (as L is already on open); **closed = [F,E,D,C,B,A]**
8. **open = [H,I,J,K,L,M,N]; closed = [G,F,E,D,C,B,A]**
9. and so on until either U is found or **open** = [ ]



Breadth-First Search of 8-Puzzle



Properties of Breadth-First Search

- Complete?
  - Yes (if *b* is finite)
- Time?

$1 + b + b^2 + b^3 + \ldots + b^d + b^{d+1} = O(b^{d+1})$

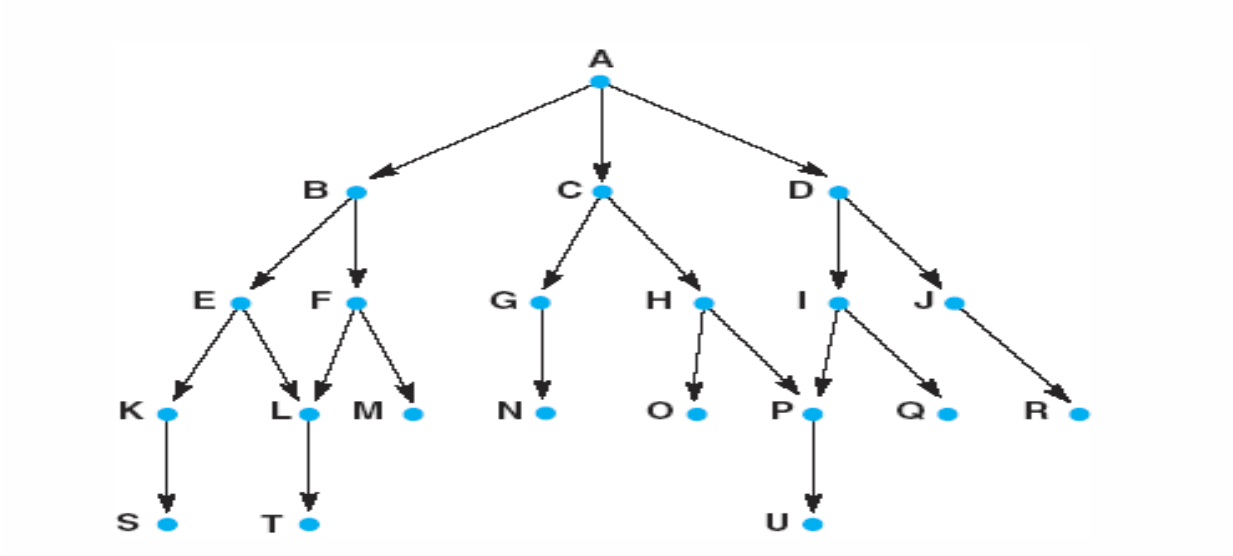
- Space?
  - $O(b^d)$ (keeps every node in memory)
- Optimal?
  - Yes (if cost = 1 per step)
- Space is the bigger problem (more than time)

Depth-First Search

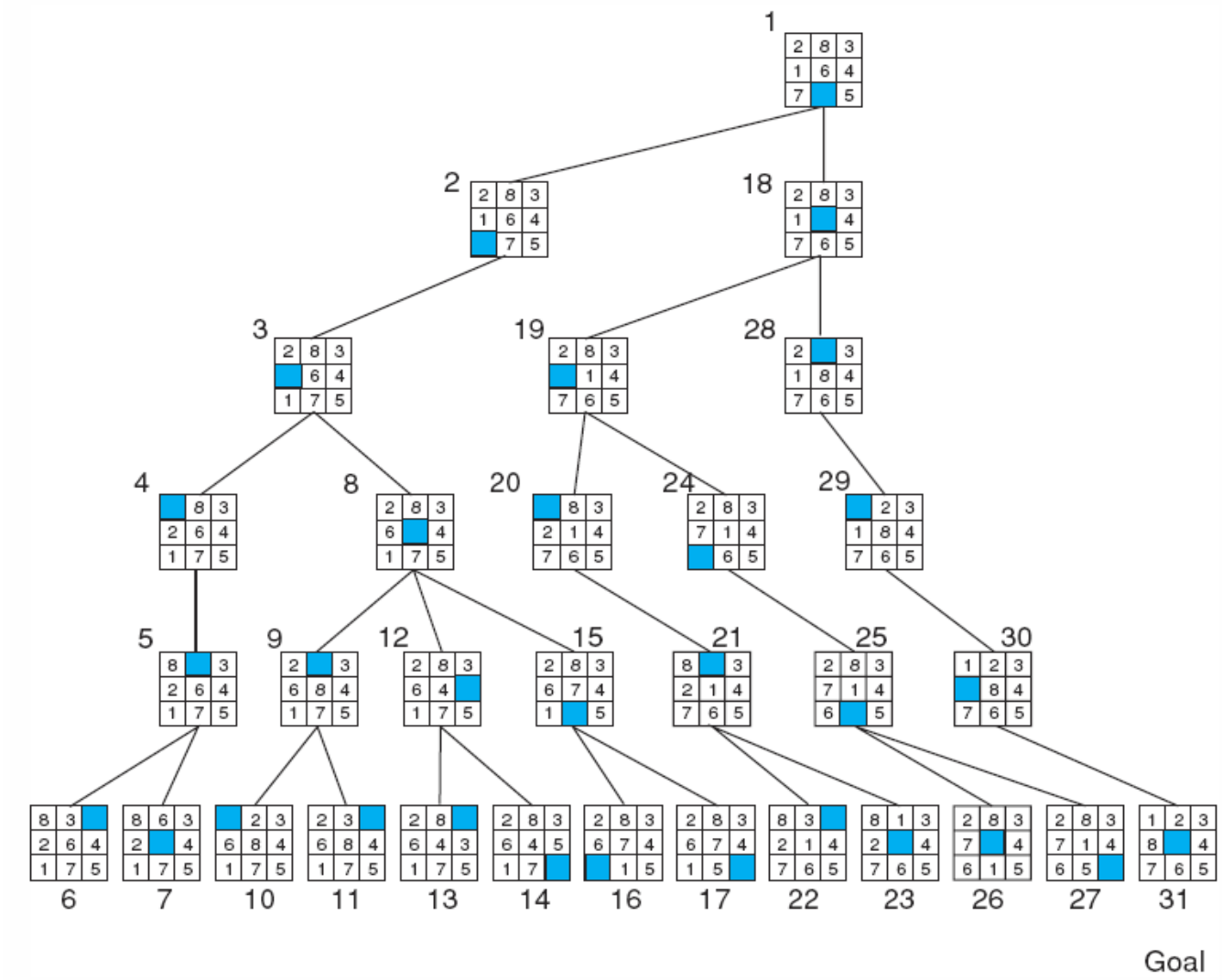
```
function depth_first_search:
    open := [Start]                                          % initialize
    closed := [ ]
    while open ≠ [ ] do                                      % states remain
        remove leftmost state from open, call it X
        if X is goal then return SUCCESS                     % goal found
        else
            generate children of X
            if any child of X is goal then return SUCCESS    % goal found
            else
                put X on closed
                discard children of X if on open or closed   % loop check
                put remaining children on left end of open
    return FAILURE
```

Depth-First Search

1. **open = [A]; closed = [ ]**
2. **open = [B,C,D]; closed = [A]**
3. **open = [E,F,C,D]; closed = [B,A]**
4. **open = [K,L,F,C,D]; closed = [E,B,A]**
5. **open = [S,L,F,C,D]; closed = [K,E,B,A]**
6. **open = [L,F,C,D]; closed = [S,K,E,B,A]**
7. **open = [T,F,C,D]; closed = [L,S,K,E,B,A]**
8. **open = [F,C,D]; closed = [T,L,S,K,E,B,A]**
9. **open = [M,C,D],** as L is already on **closed**; **closed = [F,T,L,S,K,E,B,A]**
10. **open = [C,D]; closed = [M,F,T,L,S,K,E,B,A]**
11. **open = [G,H,D]; closed = [C,M,F,T,L,S,K,E,B,A]**

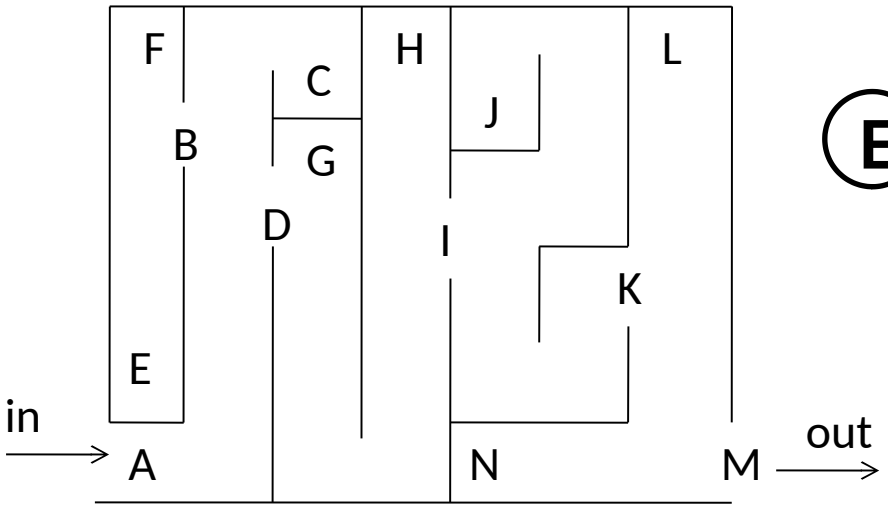


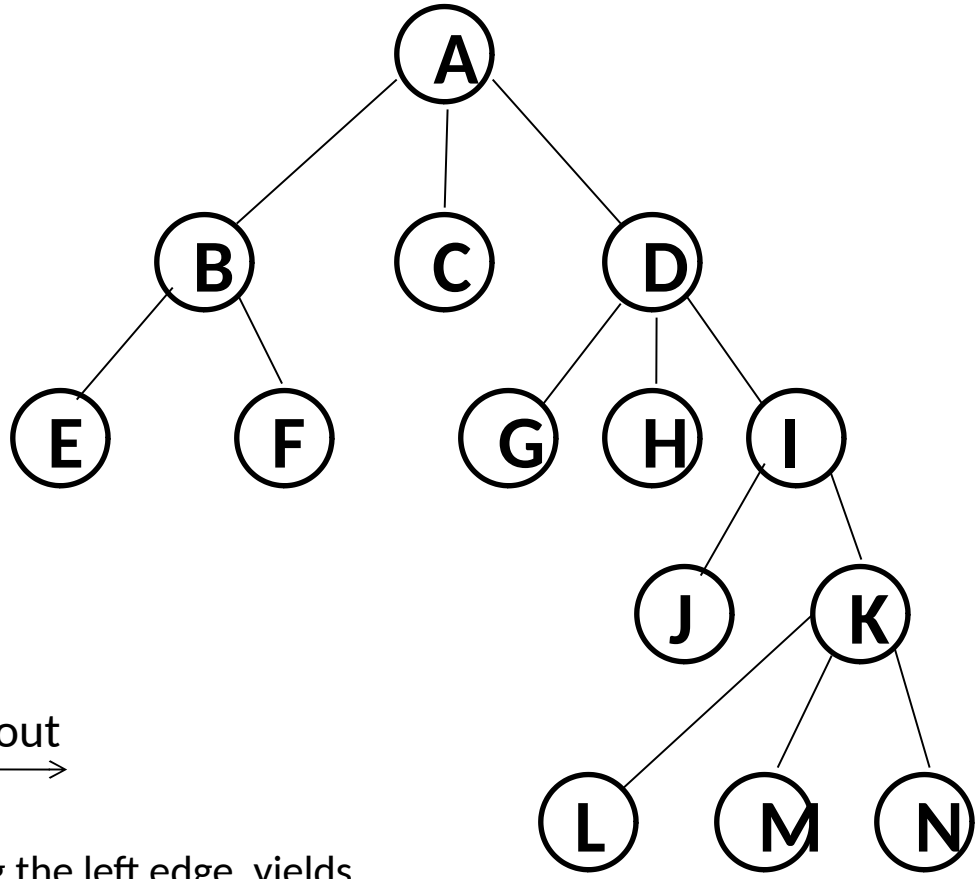
Depth-First Search of 8-Puzzle



Depth-First Search of a Maze

A is the entrance to the maze
M is the exit from the maze
C, E, F, G, H, J, L, and N are dead ends
B, D, I, and K are points in the maze where a choice can be made as to which direction to go next





Running one's hand along the left edge, yields
A, B, E, F, C, D, G, H, I, J, K, L, M

Properties of Depth-First Search

- Complete? No: fails in infinite-depth spaces and spaces with loops
  - Modify to avoid repeated states along path

→ 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

Breadth-First vs. Depth-first

Scenario	Depth-First	Breadth-First
Some paths are extremely long, or even infinite	Performs badly	Performs well
All paths are of similar length	Performs well	Performs well
All paths are of similar length, and all paths lead to a goal state	Performs well	Wasteful of time and memory
High branching factor	Performance depends on other factors	Performs badly

Uniform-Cost Search

- Instead of expanding the shallowest node, as in breadth-first search, expand the node with the *lowest path cost*
  - Note that if all step costs are equal, this is identical to breadth-first search
  - May get stuck in an infinite loop if it ever expands a node with a zero-cost action leading back to the same state
  - Fix by guaranteeing that every step is greater than or equal to some small positive constant, ε. (this means that the cost of a path always increases as we go along a path)
  - The nodes will be expanded in order of increasing path cost
  - Thus, the first goal node selected for expansion is the optimal solution

Properties of Uniform-Cost Search

- Complete?
  - Yes (if step cost >= ε for positive ε)
- Time?
  - $O(b^{\lfloor C^*/\varepsilon \rfloor + 1})$ where C^* is the cost of the optimal solution
- Space?
  - $O(b^{\lfloor C^*/\varepsilon \rfloor + 1})$
- Optimal?
  - Yes

Depth-Limited Search

- Depth-first search with depth limit *l*, i.e., nodes at depth *l* have no successors
- Recursive implementation

Iterative Deepening Search

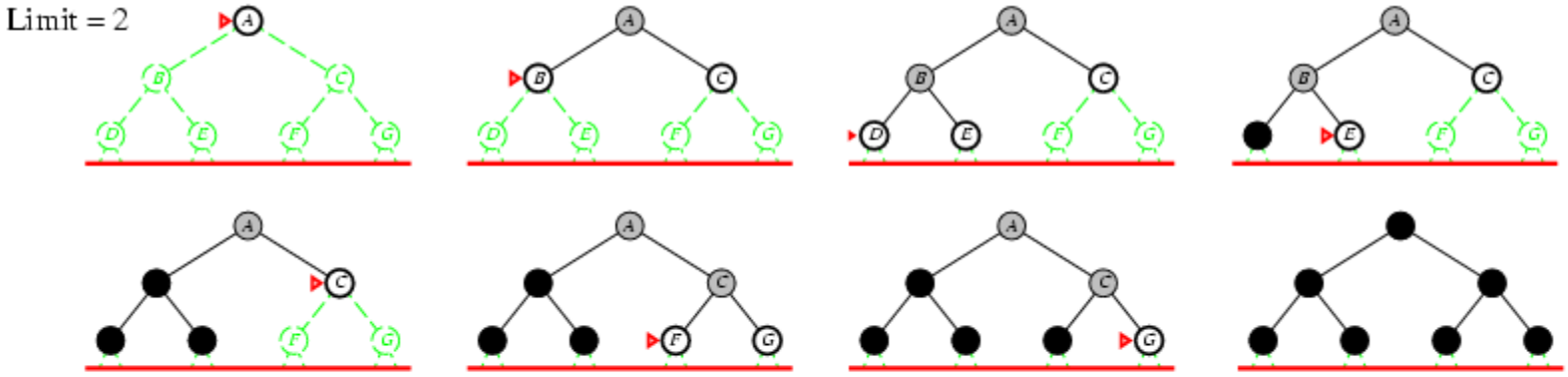
- An exhaustive search method based on both depth-first and breadth-first search
- Carries out depth-first search to depth of 1, then to depth of 2, 3, and so on until a goal node is found
- Efficient in memory use, and can cope with infinitely long branches
- Not as inefficient in time as it might appear, particularly for very large trees, in which it only needs to examine the largest row (the last one) once

Iterative Deepening Search, $D = 0$

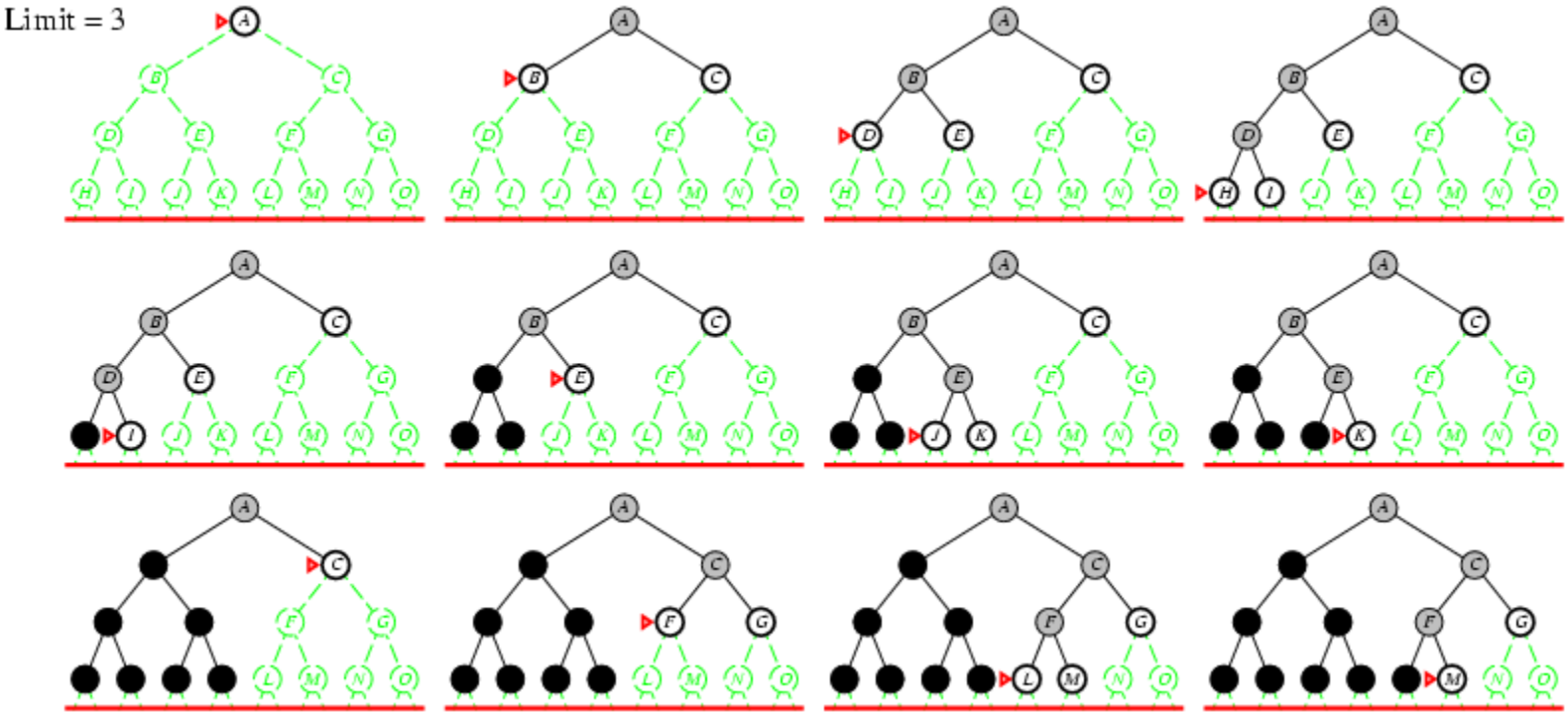
Limit = 0

Iterative Deepening Search, *D* = 1

Iterative Deepening Search, *D* = 2



Iterative Deepening Search, *D* = 3



Iterative Deepening Analysis

- IDS combines the efficiency of memory use of depth-first search, with the advantage that branches of the search tree that are infinite or extremely large will not sidetrack the search
- It also shares the advantage of breadth-first search in that it will always find the path that involves the fewest steps through the tree
  - Although this is not necessarily the best path

Iterative Deepening Analysis

- It appears that IDS would be an extremely inefficient way to search a tree because we are repeatedly starting the search at the root
- It is almost as efficient as depth-first or breadth-first because for most trees, the majority of nodes are in the deepest level
  - All three approaches spend most of their time examining these nodes

Iterative Deepening Analysis

For a tree of depth d and a branching factor b, the total number of nodes is: 1 root node, b nodes in the first layer, b^2 nodes in the second layer, … b^n nodes in the n^{th} layer. Hence, the total number of nodes is:

$$1 + b + b^2 + b^3 + \ldots + b^d = (b^{d+1} - 1) / (b - 1)$$

For example, for a tree with a depth of 2 and a branching factor of 2, there are (8-1)/(2-1) = 7 nodes. Using depth-first search or breadth-first search, this means the total number of nodes to be examined is 7.

Iterative Deepening Analysis

Using Iterative Deepening Search, nodes must be examined more than once, resulting in the following progression:

$(d + 1) + b(d) + b^2 (d - 1) + b^3 (d - 2) + \ldots + b^d$

= time complexity of $O(b^d)$

= space complexity of $O(bd)$

For a small tree, $d = 2$ and $b = 2$, IDS is less efficient in time than depth-first search or breadth-first search:

$(2+1) + 2(2) + 2^2 = 11$ nodes visited

Iterative Deepening Analysis

However, for a larger tree, $d = 4$ and $b = 10$, the tree has the following number of nodes: $(10^5 - 1) / (10 - 1) = 11{,}111$.

IDS will examine the following number of nodes:
$(4+1) + 10(4) + 100(3) + 1{,}000(2) + 10{,}000 = 12{,}345$ nodes.

Hence, as the tree gets larger, we see that the majority of nodes to be examined (in this case, 10,000 out of 12,345) are in the last row, which needs to be examined at most once.

Properties of Iterative Deepening

- Complete?
  - Yes
- Time?
  - $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space?
  - $O(bd)$
- Optimal?
  - Yes, if step cost = 1

Summary of Search Algorithms

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

331 – Intro to Intelligent Systems
Week 03b
Adversarial Search
R&N Chapter 5.1-5.3

T.J. Borrelli

Adversarial Search

- We now move on to situations where we have multiple agents in competitive environments
- The agents goals will be in conflict, thus "adversarial search"
- Also known as games! :)

Games

- Mathematical game theory (economics) views any multiagent environment as a game provided the impact of each agent on the others is “significant”
- Regardless of whether the agents are cooperative or competitive
- In AI, the most common games are deterministic, turn-taking, two-player, zero-sum games of perfect information (like chess)
- This is deterministic, fully-observable, discrete and the utility value at the end are equal and opposite
- Zero-sum – each players gain (or loss) is balanced by loss (or gain) of other player – if I win, you lose

Games

- We are not talking about physical games (hockey, basketball) as the description of such games is complex (exception: robot soccer)

- Rather games where the state of the game can be easily represented and there are precise actions with deterministic outcomes

Games

- Unlike most of the toy problems seen earlier, games are interesting because they are so hard to solve

Chess, for example has an average branching factor of 35, and games often go 50+ moves by each player ($35^{100} \sim 10^{154}$) – although the search graph is "only" about 10^{40} distinct nodes

- Games, like the real world, require us to make some decisions even when calculating the optimal decision is infeasible
- Games penalize inefficiency

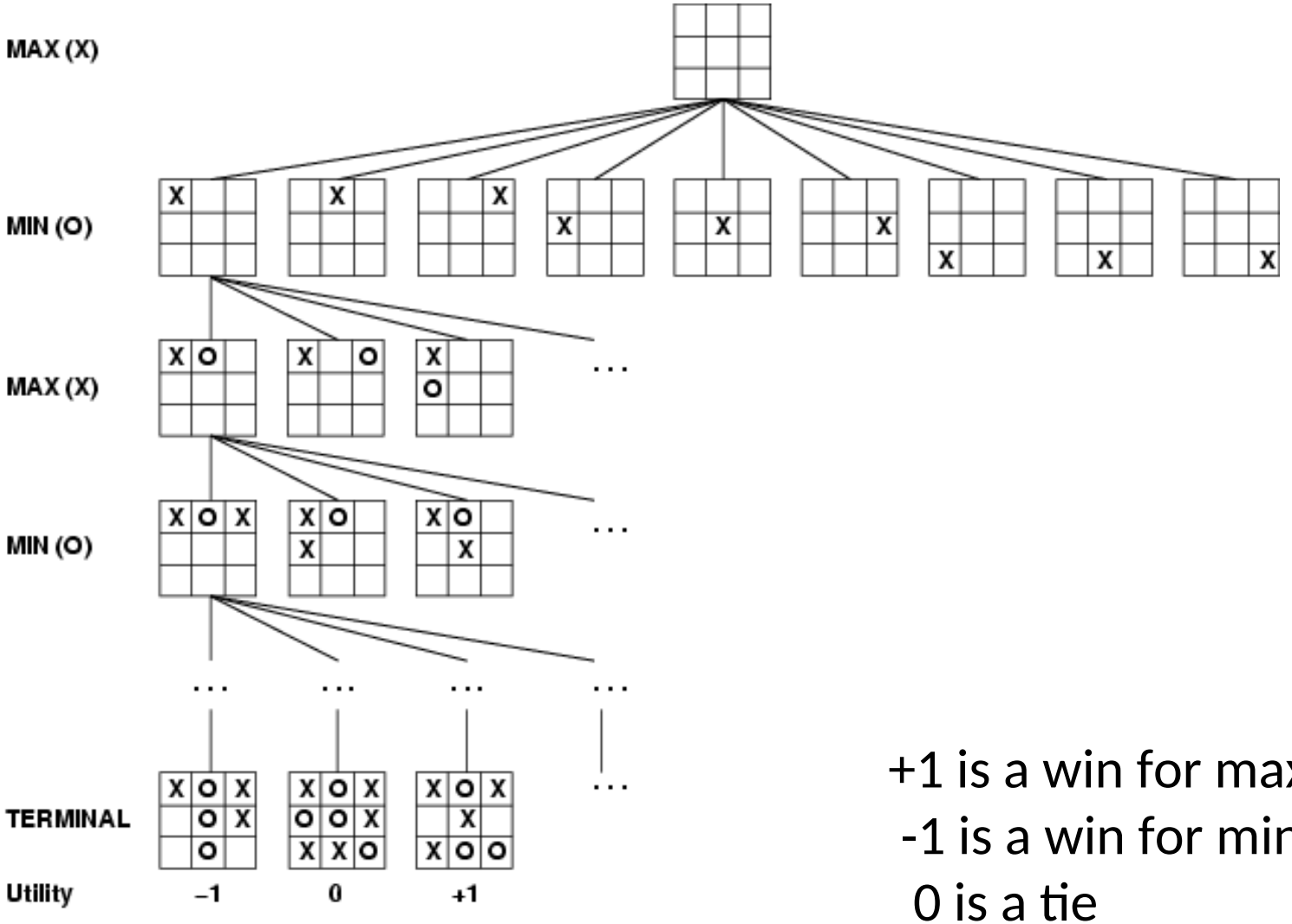
Game formalization

- A game can be formally defined as a kind of search problem with the following:
- S_0 – the initial state
- PLAYER(s) – defines whose turn it is
- ACTION(s) – returns the set of legal moves given the state
- RESULT(s, a) – defines the result of an action/move
- TERMINAL-TEST(s) – determines if the game is over
- UTILITY(s, p) – defines the numeric value for a game that ends In a terminal state s, for player p

Heuristics for Games of Strategy

- *Minimax* is a method used to evaluate game trees, where the goal is to maximize a utility function
- This will result in *perfect play* for a 2-player, deterministic, zero-sum game
- A static evaluator is applied to leaf nodes, and values are passed back up the tree to determine the best score a player can obtain against a rational opponent

Partial Game Tree for Tic-Tac-Toe



+1 is a win for max (X)

-1 is a win for min (O)

0 is a tie

Tic-Tac-Toe

- The tree is relatively small (fewer than 9! = 362,880 terminal nodes)
- But it is not often feasible to list all of them
- So, we should think of game tree as a theoretical construct
- We use the term **search tree** for a tree that is superimposed on the full game tree
- The search tree will examine enough nodes to allow a player to determine what move to make

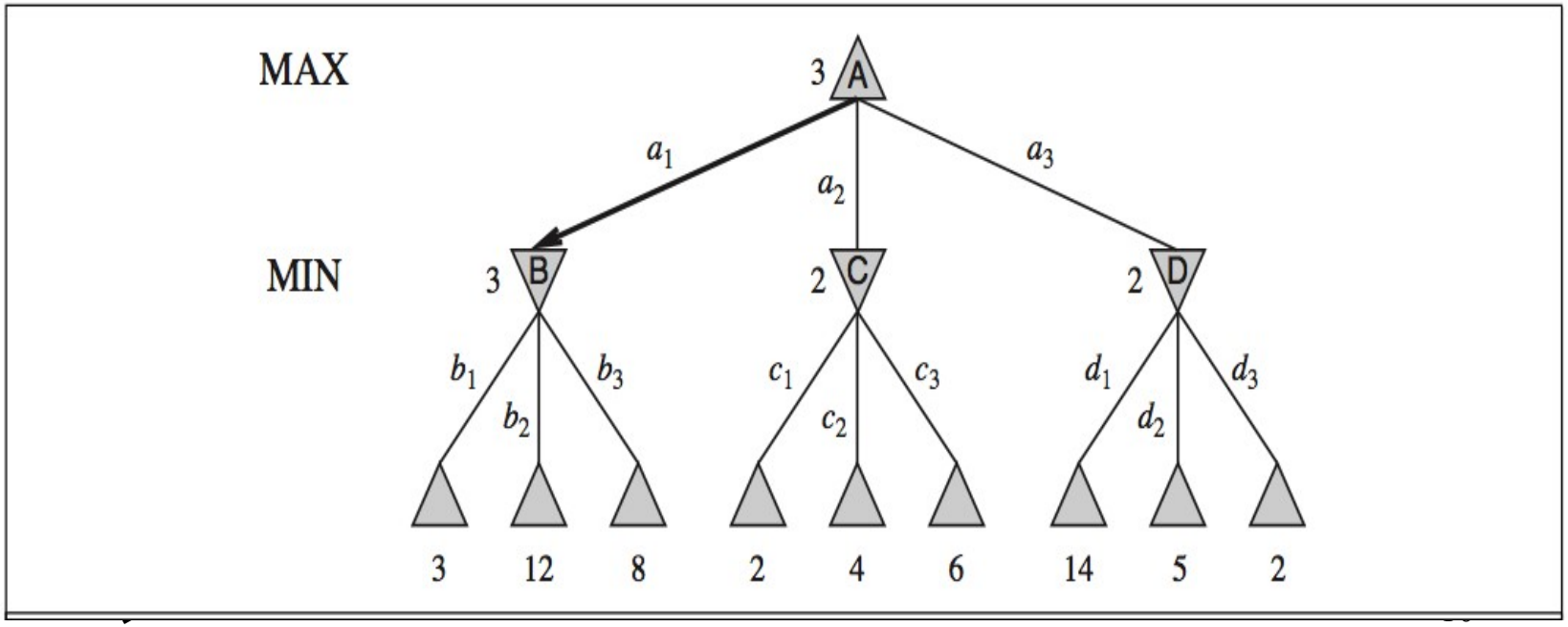
Minimax

- 2 players: “max”, who wants to maximize his/her own score and “min” who wants to minimize max’s score
- Example: 2-ply game, with “max” taking the first turn (assume heuristic values at leaves are known by both players):

goes first

takes a turn

Max’s scores



Minimax

- Minimax was originally about a two-player zero-sum game
- It essentially says that there are optimal strategies for such games
- Since MIN is also moving, MAX must find a contingent strategy
- Minimax strategies are essentially pessimistic (Murphy's Law) strategies
  - What would you do if you knew that your opponent was going to make his best response against you?
  - What is the best strategy for you to play?

Minimax Algorithm

- MINIMAX(s)

```
If TERMINAL-TEST(s)
   return UTILITY(s)

else if PLAYER(s) == MAX
   return MAX(RESULT(s,a)

else  // PLAYER(s) == MIN
   return MIN(RESULT(s,a)
```

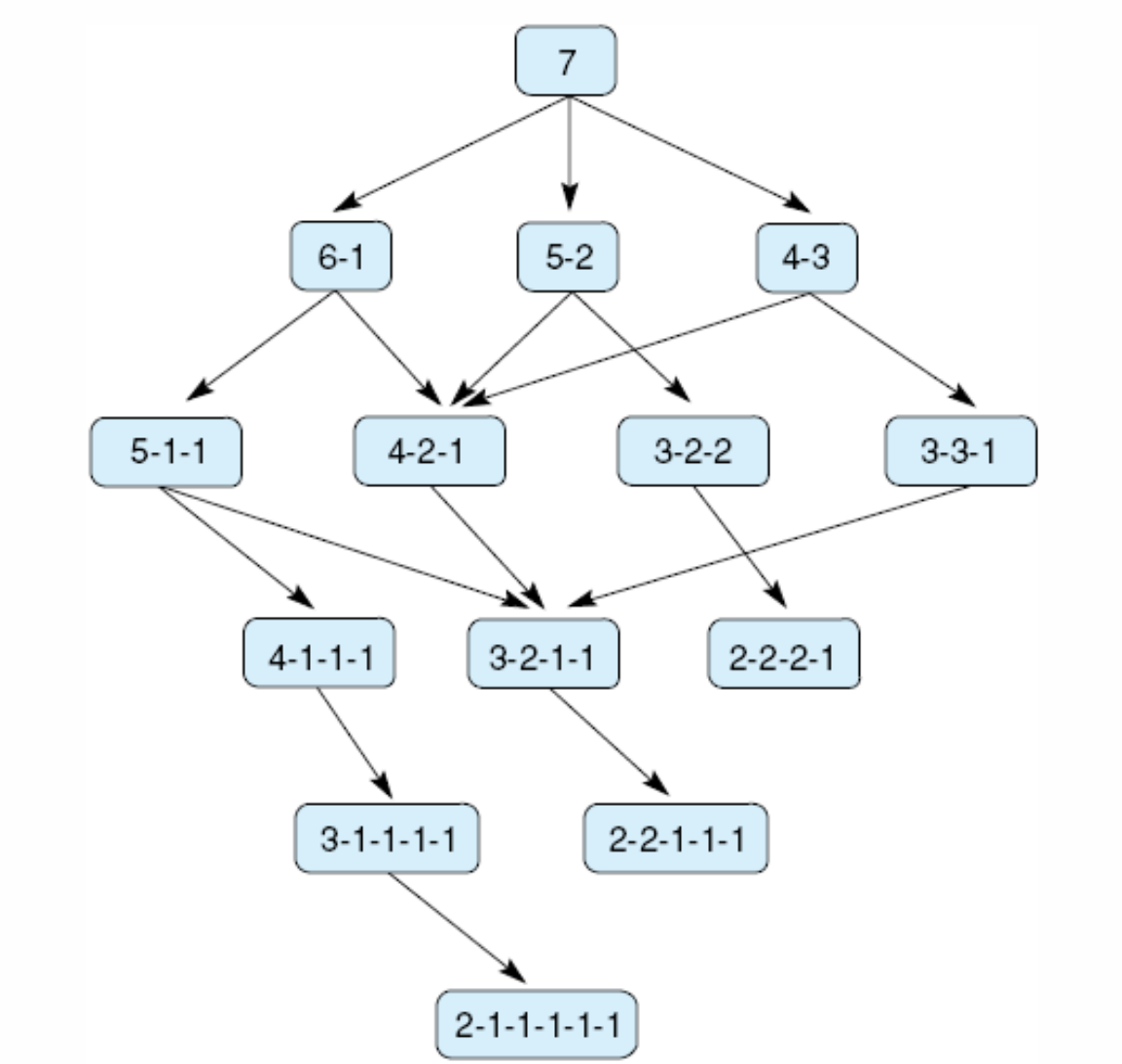
Minimax

- Roughly – an optimal strategy leads to outcomes at least as good as any other strategy when one is playing an infallible opponent
- This approach maximizes the worst-case outcome for MAX
- If MIN does not play optimally, MAX does even better

Example: Game of “Nim”

- A number of tokens are placed on a table between the two players
- At each move a player must divide a pile of tokens into two non-empty piles of different sizes
  - For example, 6 tokens may be divided into piles of 5 and 1, or 4 and 2, but not 3 and 3
- The first player who can no longer make a move loses the game

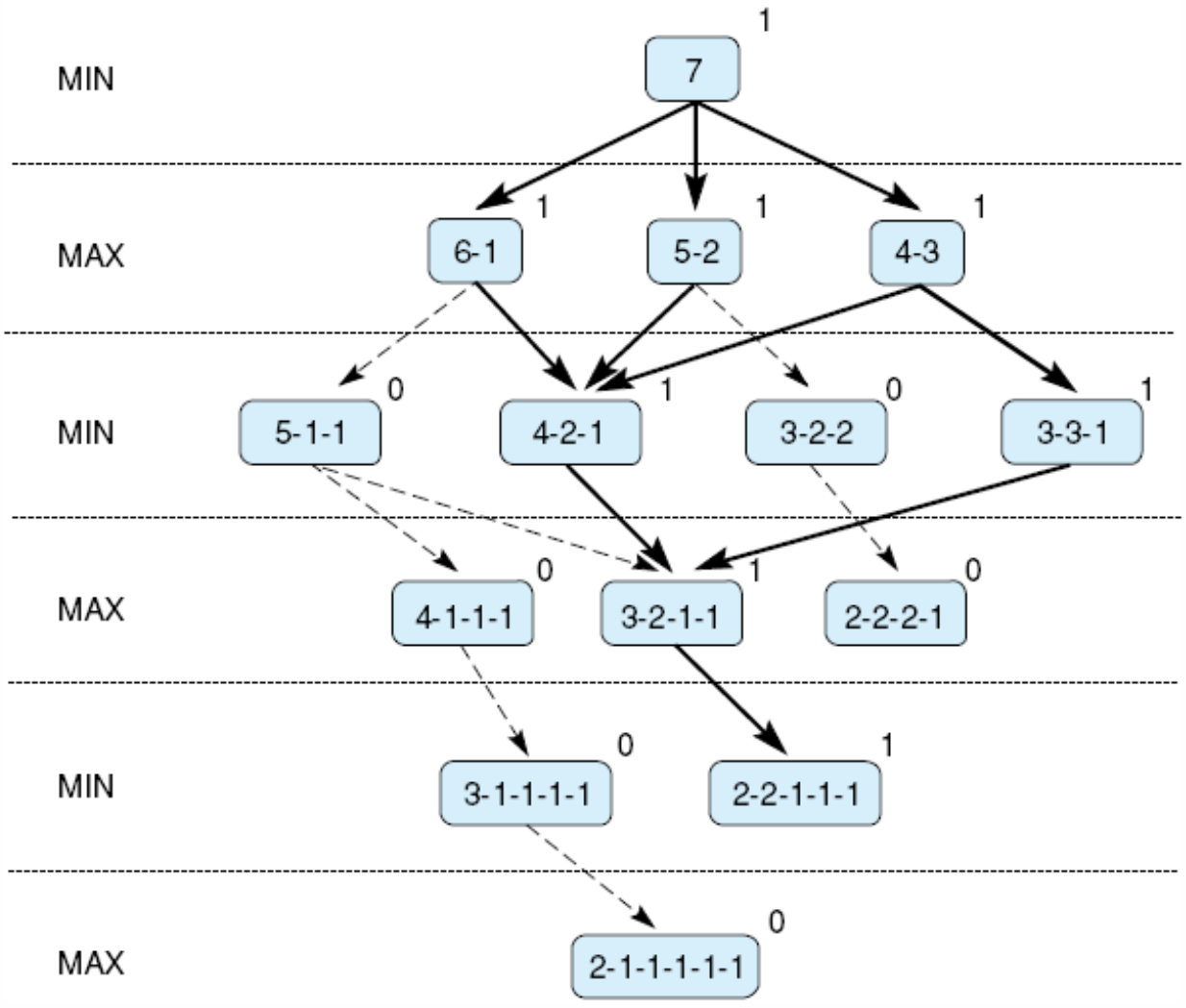
Game Tree for “Nim” with 7 Tokens



Game Tree for “Nim” with 7 Tokens

- In this game, assume Min moves first
- Give each leaf node a value of 1 or 0, depending on if it is a win for Max (1) or for Min (0)
- Propagate the values up the tree through successive parent nodes according to the rule:
  - If the parent state is a Max node, give it the maximum value among its children
  - If the parent state is a Min node, give it the minimum value among its children

Game Tree for “Nim” with 7 Tokens



Game Tree for “Nim” with 7 Tokens

- Because all of Min’s possible first moves lead to nodes with a derived value of 1, the second player, Max, can always force the game to a win, regardless of Min’s first move
- Min can choose any of the first move alternatives and will still be guaranteed to lose
- The resulting “win paths” for Max are in bold
- Min can only win if Max played foolishly

Properties of Minimax

- Complete?
  - Yes (if tree is finite)
- Optimal?
  - Yes (against an optimal opponent)
- Time complexity?
  - $O(b^m)$
  - For chess, b ≈ 35, m ≈100 for "reasonable" games
  - → exact solution completely infeasible!
- Space complexity?
  - O(bm)

Bounded Look-Ahead

- For trees with high depth or very high branching factor, minimax cannot be applied to the entire tree
- In such cases, *bounded look-ahead* is applied:
  - When search reaches a specified depth, the search is cut off, and the static evaluator applied
- Must be applied carefully:
  - In some positions a static evaluator will not take into account significant changes that are about to happen

α-β Pruning

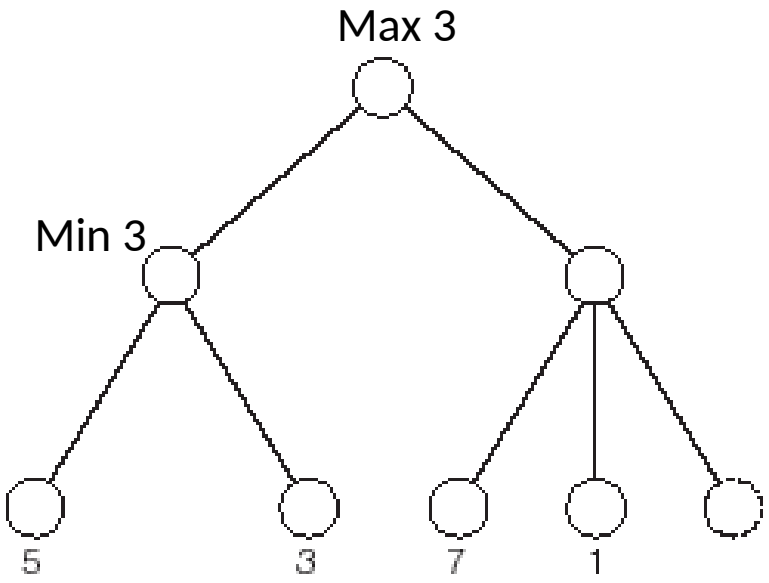
- Problem with minimax search is that the number of game states it has to examine is (still) exponential in the depth of the tree
- Unfortunately , we can't eliminate the exponent, but we can effectively cut it in half
- We can compute the correct minimax decision without looking at every node in the game tree

α-β Pruning

- A method that can often cut off a large part of the game tree
- Based on the idea that if a move is clearly bad, there is no need to follow the consequences of it

α-β Pruning

- In this tree, having examined the leaf nodes with values 7 and 1, there is no need to examine the final leaf node (we can prune the tree at this node)
- To see why, notice that min is going to pass up to max the maximum of (3,1) which is 3
- But notice also that min is going to always choose the minimum value of its children, so once min sees a 1 in the right sub-tree, it knows that it will never pass anything from that tree up to max, because 1 < 3.



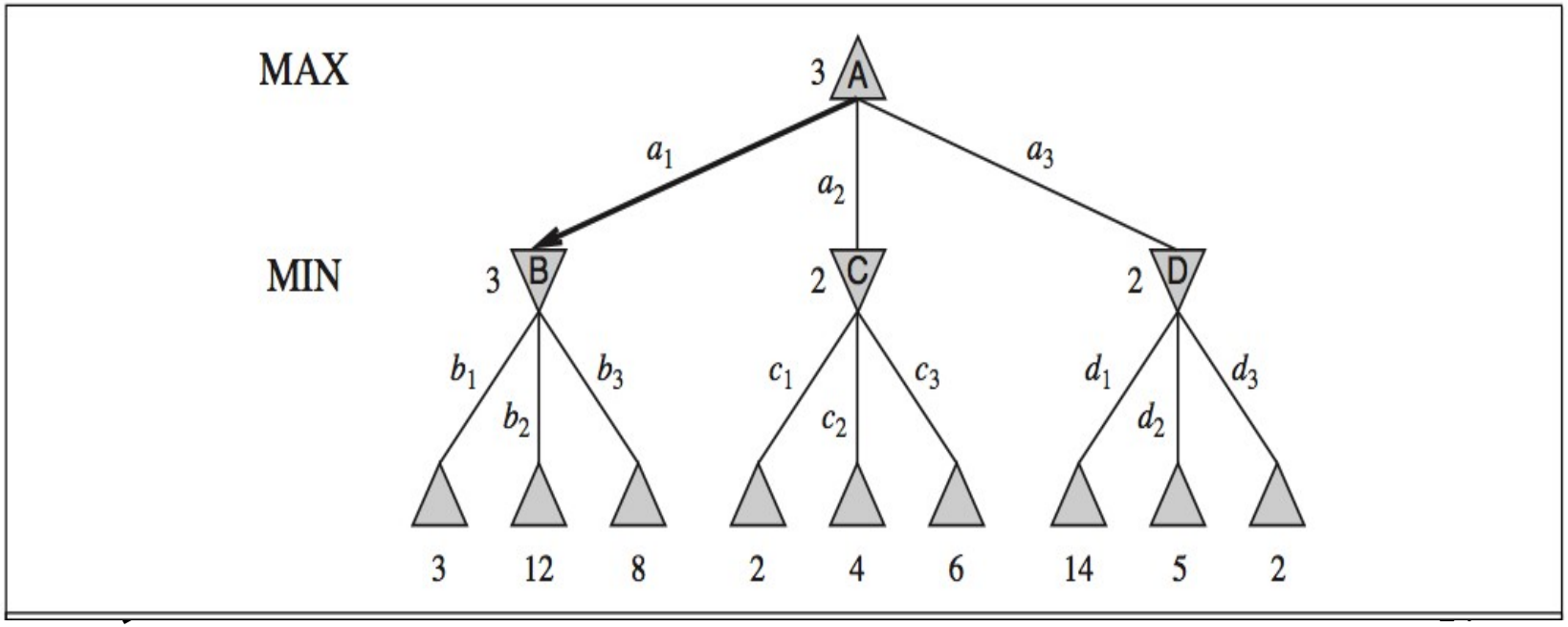
α-β Pruning

- Let's look again at our tree from before

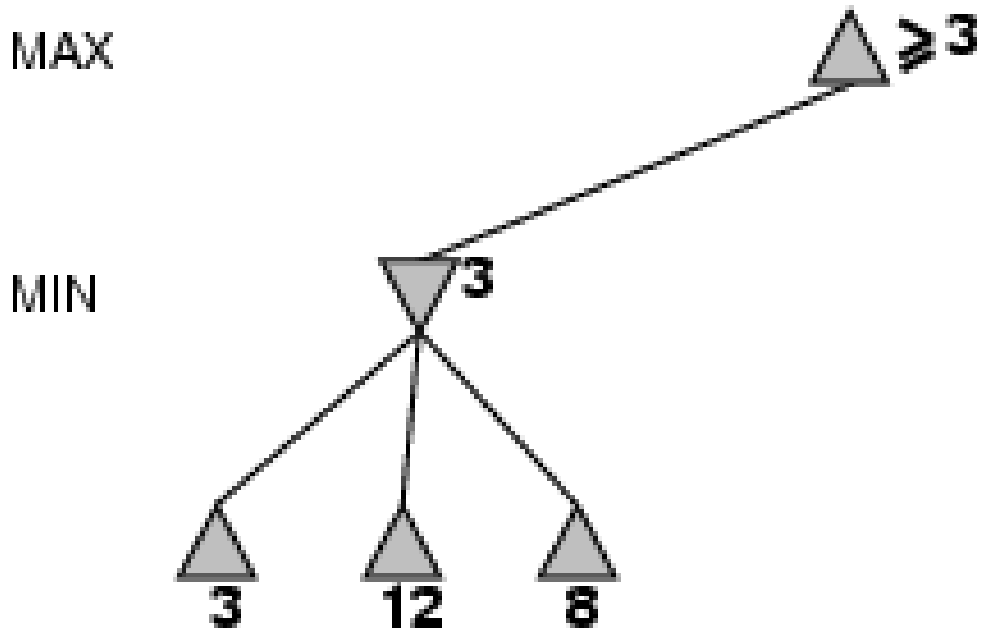
goes first

takes a turn

Max's scores

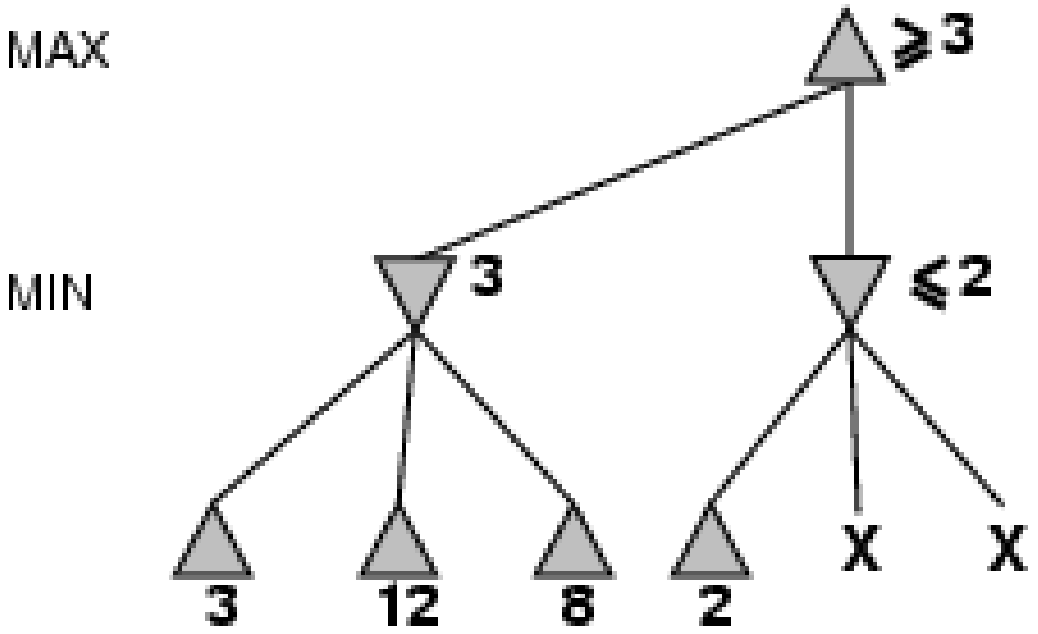


α-β Pruning



Max is waiting for a value that is >= 3.

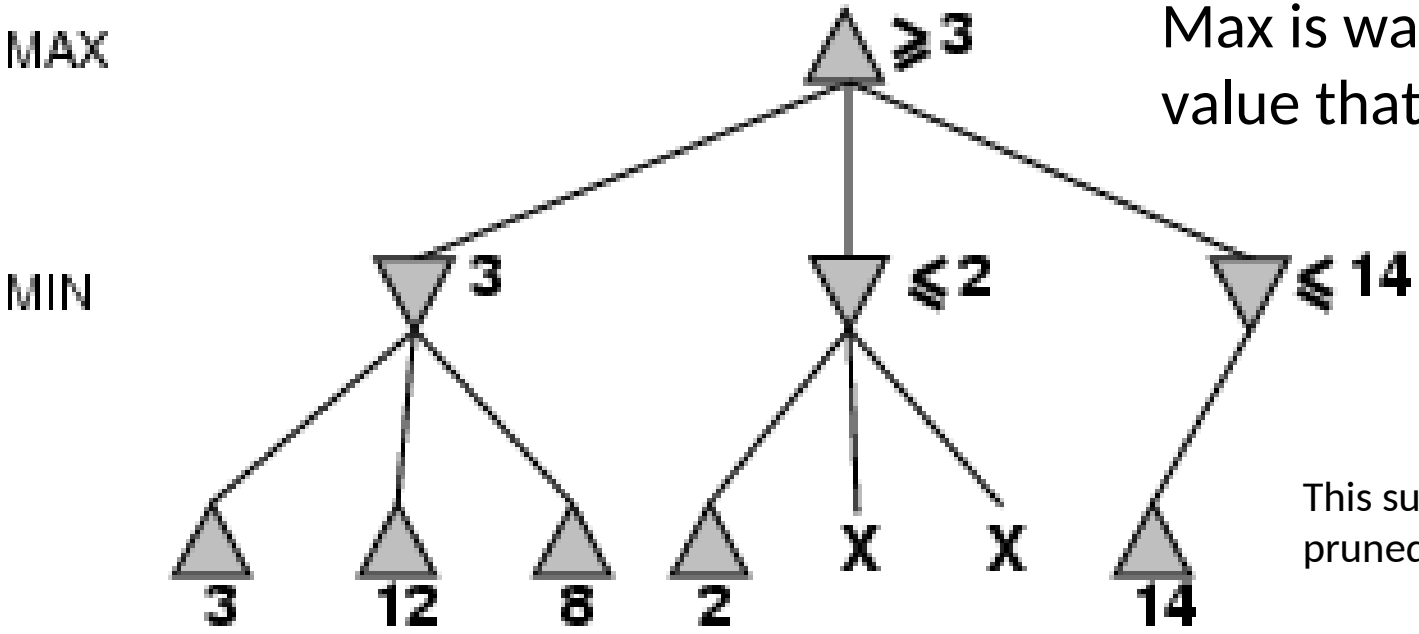
α-β Pruning



Max is waiting for a value that is >= 3.

This sub-tree can be pruned because 2 < 3

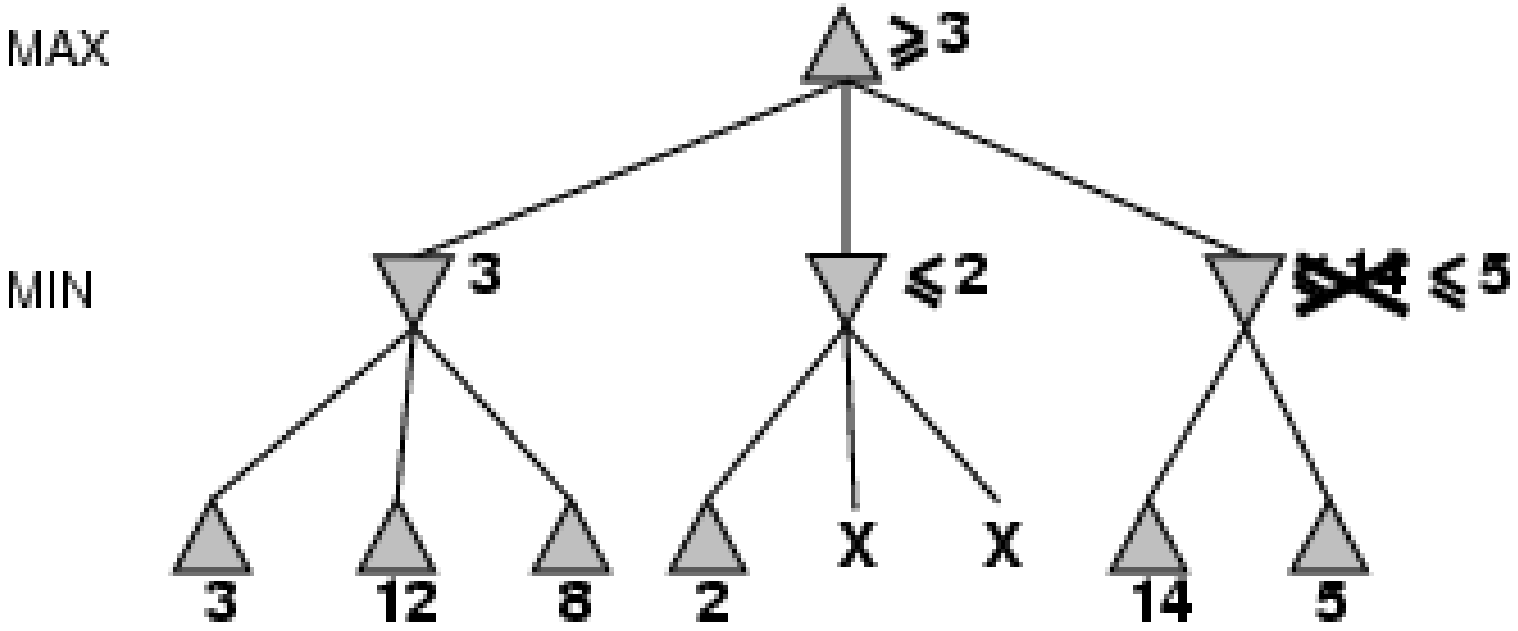
α-β Pruning



Max is waiting for a value that is >= 3.

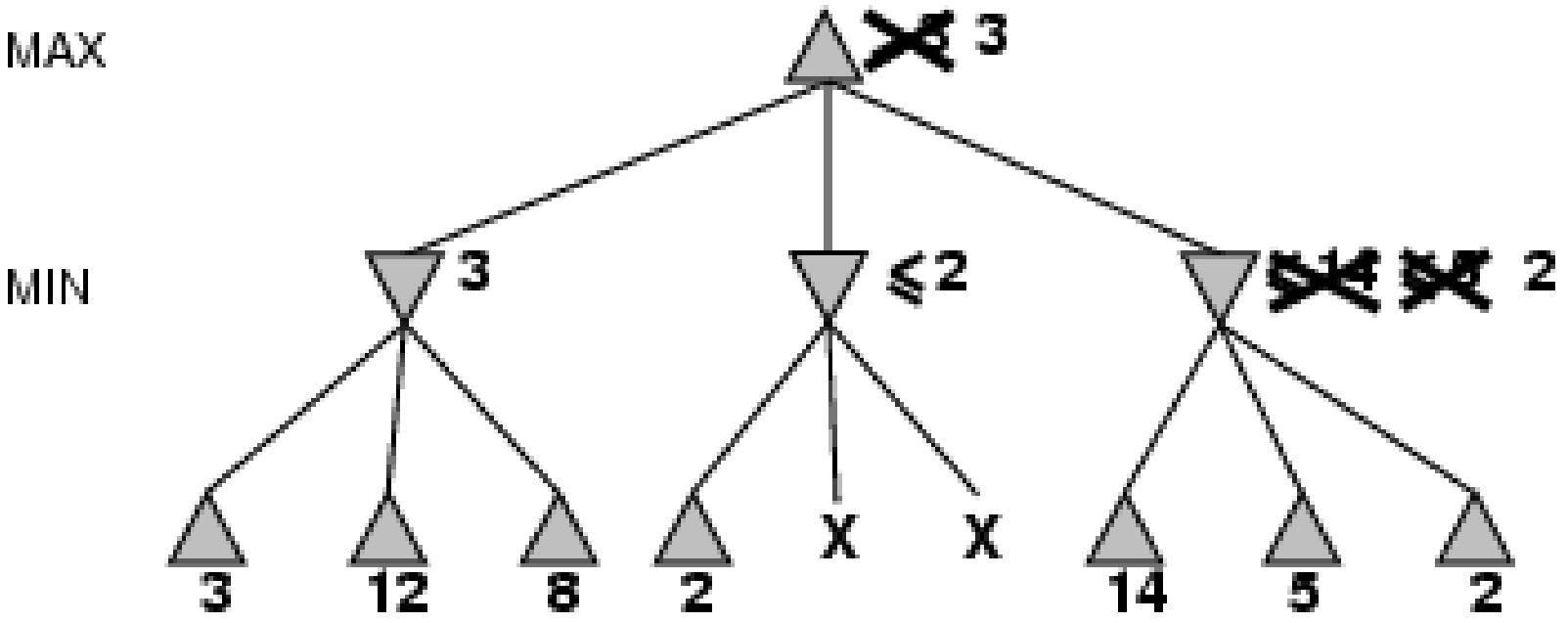
This sub-tree cannot be pruned because 14 > 3

α-β Pruning



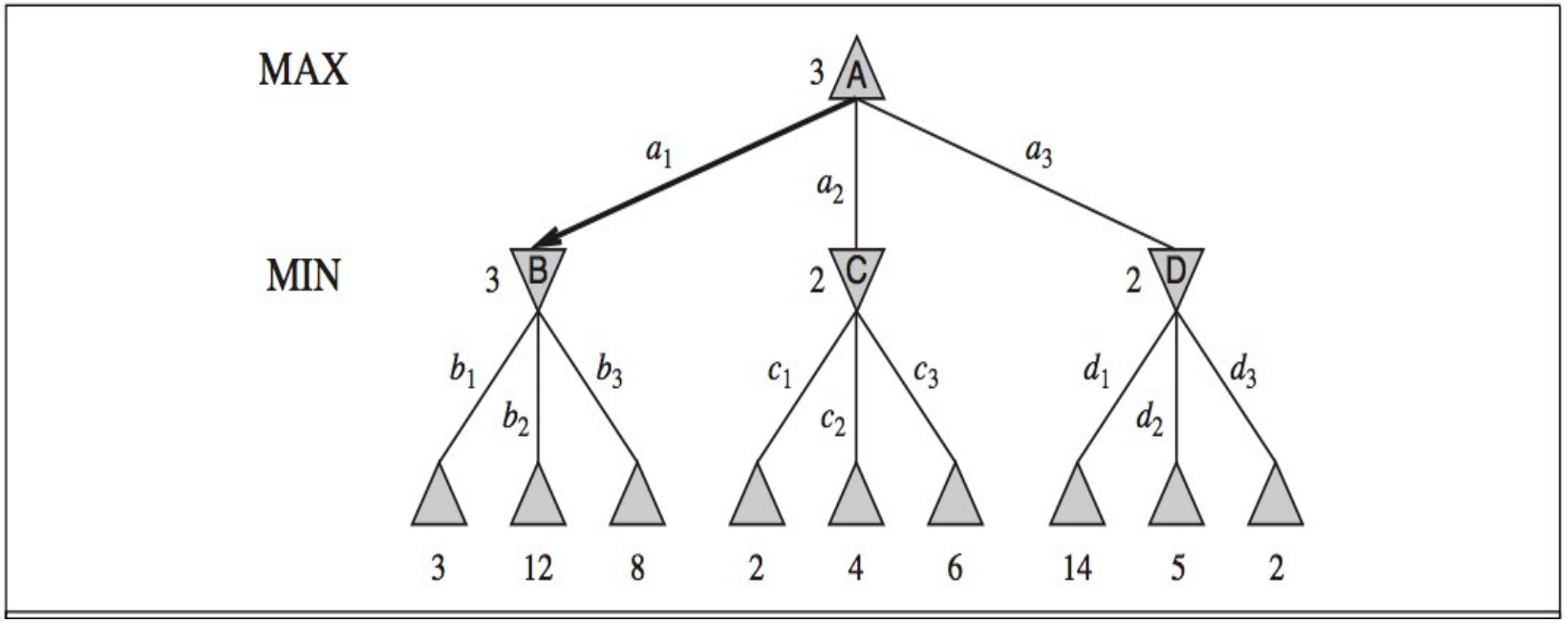
Min is looking for a value <=14 but >= 3

α-β Pruning



Min is looking for a value <=14 but >= 3. Since 2 < 5 Min will choose that, and Max will have to settle for 3.

α-β Pruning



- MINIMAX(root) = max(min(3,12,8), min(2,x,y), min(14,5,2)
  = max(3, min(2,x,y), 2)
  = max(3,z,2) where z = min(2,x,y)<=2
  = 3

Application: playing world class chess

- Current PCs can evaluate ~200 million nodes / 3 min
- Minimax search: ~5 ply lookahead
- With α-β pruning: ~10 ply
- Further improvements:
- Only evaluate “stable” positions
- Transposition tables: Remember states evaluated before
- Null move heuristic: Get lower bound by letting opp. move 2x
- Precompute endgames (all 5, some 6 piece positions)
- Opening library (up to ~30ply in first couple moves)
- Hydra: 18 ply lookahead (on 64 processor cluster)

Properties of α-β

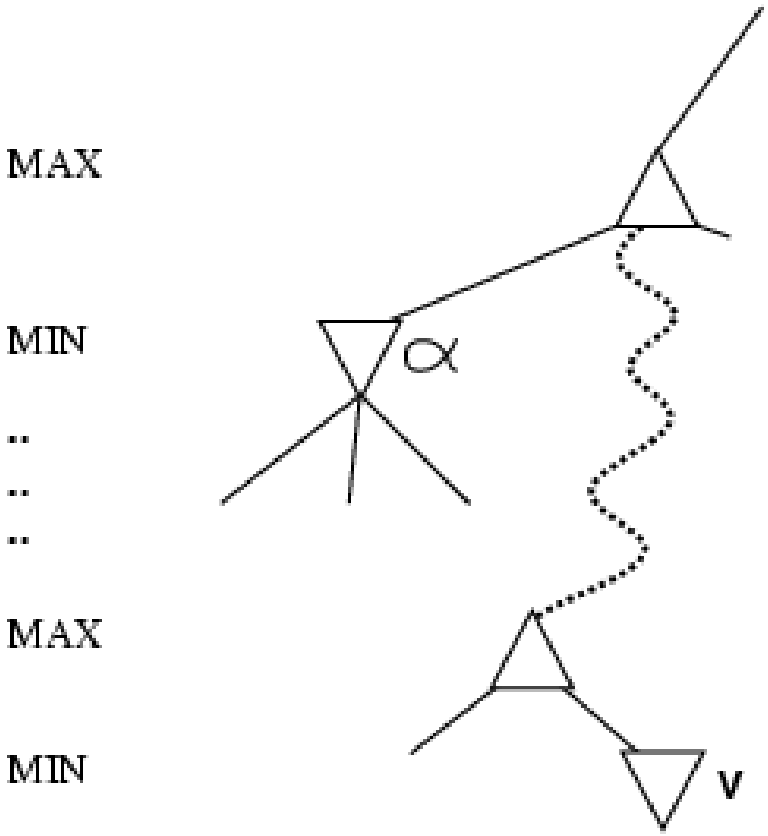
- Pruning does not affect the final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  - Reduces number of states that have to be checked by $\sqrt{b}$
  - For chess, the effective branching factor becomes ~ 6 instead of 35
- A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning)
- Alpha-beta can solve a tree roughly twice as deep as minimax in the same amount of time.

Properties of α-β

- Move ordering (evaluation order of moves) matters a lot!
- Worst case we have $O(b^m)$
- Best case "perfect ordering" we can get $O(b^{m/2})$
- With random ordering we have $O((b/\log b)^m)$

Why is it Called α-β?

- α = the value of the best (i.e., highest-value) choice found so far at any choice for *max*
- If *v* is worse than α, *max* will avoid it
  —Prune that branch
- β = the value of the best (lowest-value) choice we have found so far at any choice for for *min*



Application: checkers

- First classic game fully played by computer – Christopher Stratchey 1952
- Also in 1952 Arthur Samuel (IBM) developed checkers program (~20KB memory, 1 KHz processor) that learned its own evaluation function by playing itself thousands of times
- This program started out as a novice and in a few days could beat Samuel and in 1962 beat a checkers champion
- In 1990 checkers program entered world championship and almost won
- Did win in 1994 (*human player had to withdraw due to health reasons)

Application: checkers

- In 2007 checkers was “solved”
- Checkers involves analyzing about 500 quadrillion positions ($5^* 10^{17}$)
- Completed endgame tables for all checker positions with 10 or fewer pieces (over 39 trillion)
- From there, were able to do alpha-beta search

331 – Intro to AI
Week 03
Heuristics
R&N Chapter 3 (Section 3.5, 3.6), Chapter 4.1

T.J. Borrelli

Heuristics

- Heuristic: a rule or other piece of information that is used to make methods such as search more efficient or effective
- In search, often use a heuristic evaluation function, f(n):
  - f(n) tells you the approximate distance of a node, n, from a goal node
- f(n) may not be 100% accurate, but it should give better results than pure guesswork

Heuristics

- A heuristic should reduce the number of nodes that need to be examined
- The more *informed* a heuristic is, the better it will perform
- Heuristics are used for solving *constraint satisfaction problems*
  - Generate a possible solution, and then make small changes to bring it closer to satisfying constraints

Best-First Search

- Pick the *most likely* node (based on some heuristic value) from the partially expanded tree at each stage
- Tends to find a path more quickly/efficiently than depth-first or breadth-first search, but does not guarantee to find the best path
- This is a greedy algorithm – may converge to local optimum

Best-First Search

- Use an evaluation function *f(n)* for each node
  - *f(n)* is an estimate of "desirability"
  - Expand the most desirable unexpanded node
- Implementation:
  - Order the nodes in fringe (candidate nodes) in order of decreasing desirability
- Special cases:
  - Greedy best-first search
  - A* search

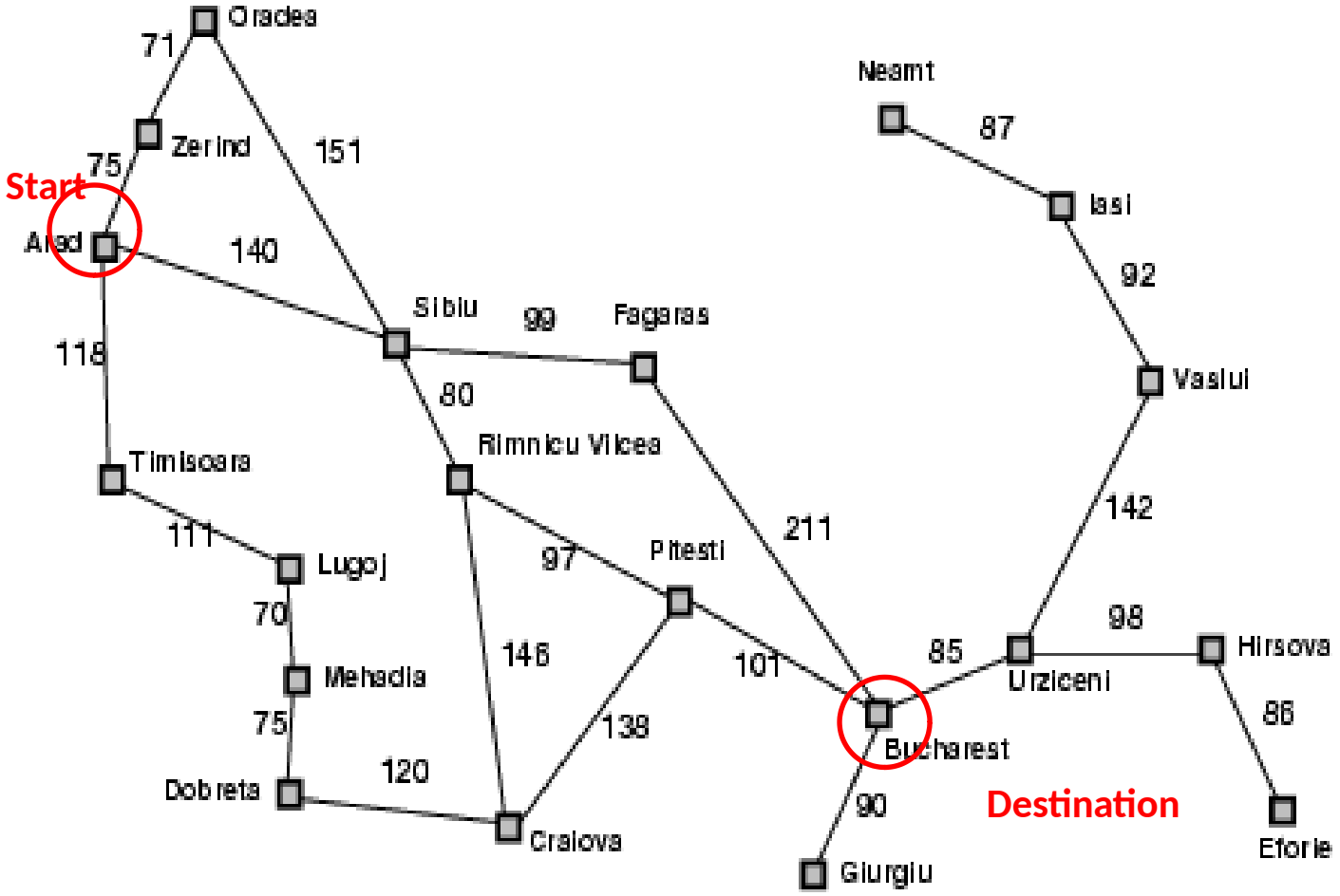
Greedy Best-First Search

- Greedy Best-First Search always expands the node that appears to be closest to the goal
- Not optimal, and not guaranteed to find a solution at all
- Can easily be fooled into taking poor paths

Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (heuristic)
- The heuristic is the estimate of cost from node n to the goal
  - e.g., $h_{SLD}(n)$ = straight-line distance ("as the crow flies") from start node to destination node
- Greedy best-first search expands the node that *appears* to be closest to goal

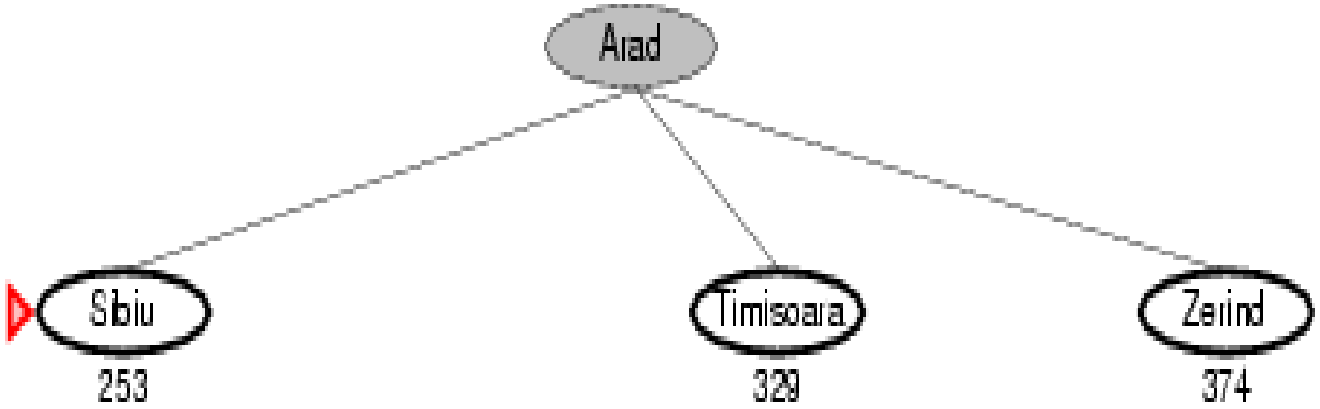
Romania with Step Costs in km



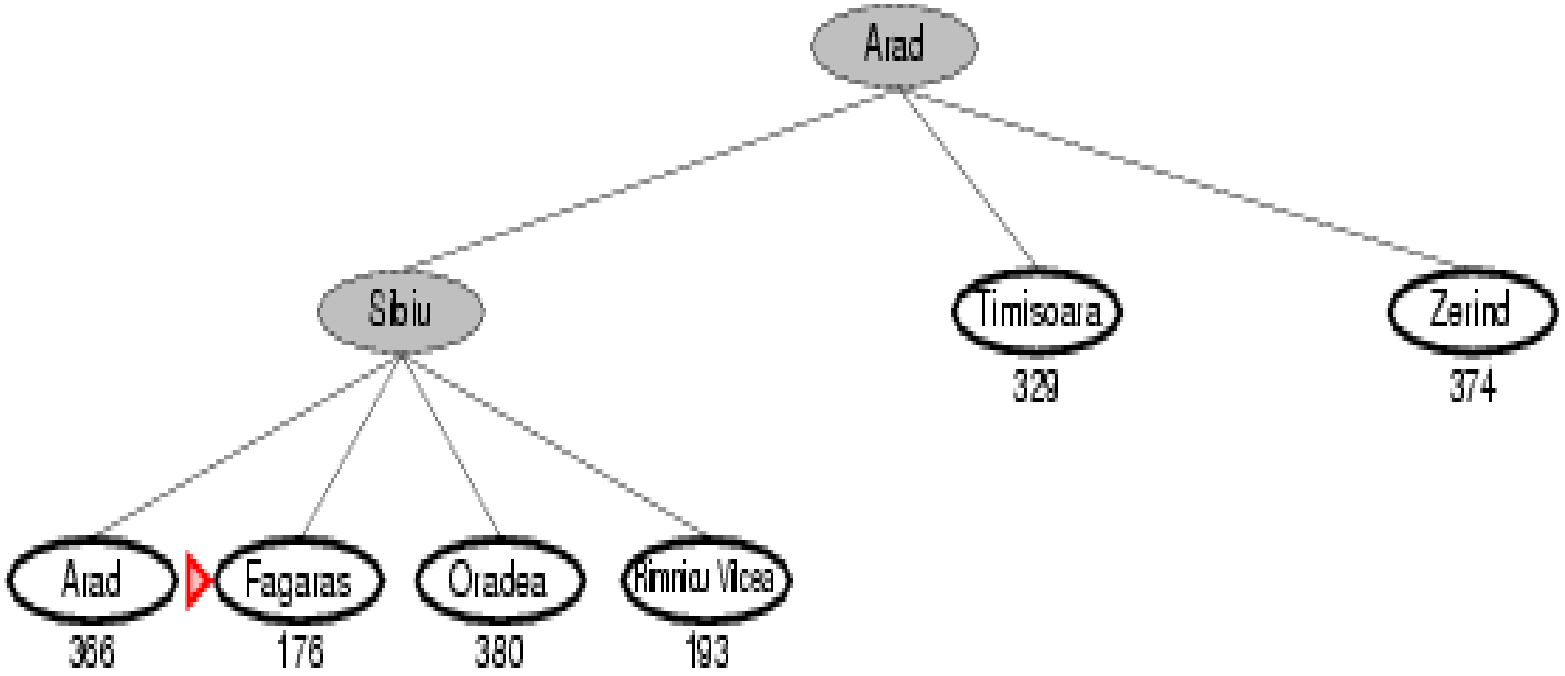
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-First Search Example

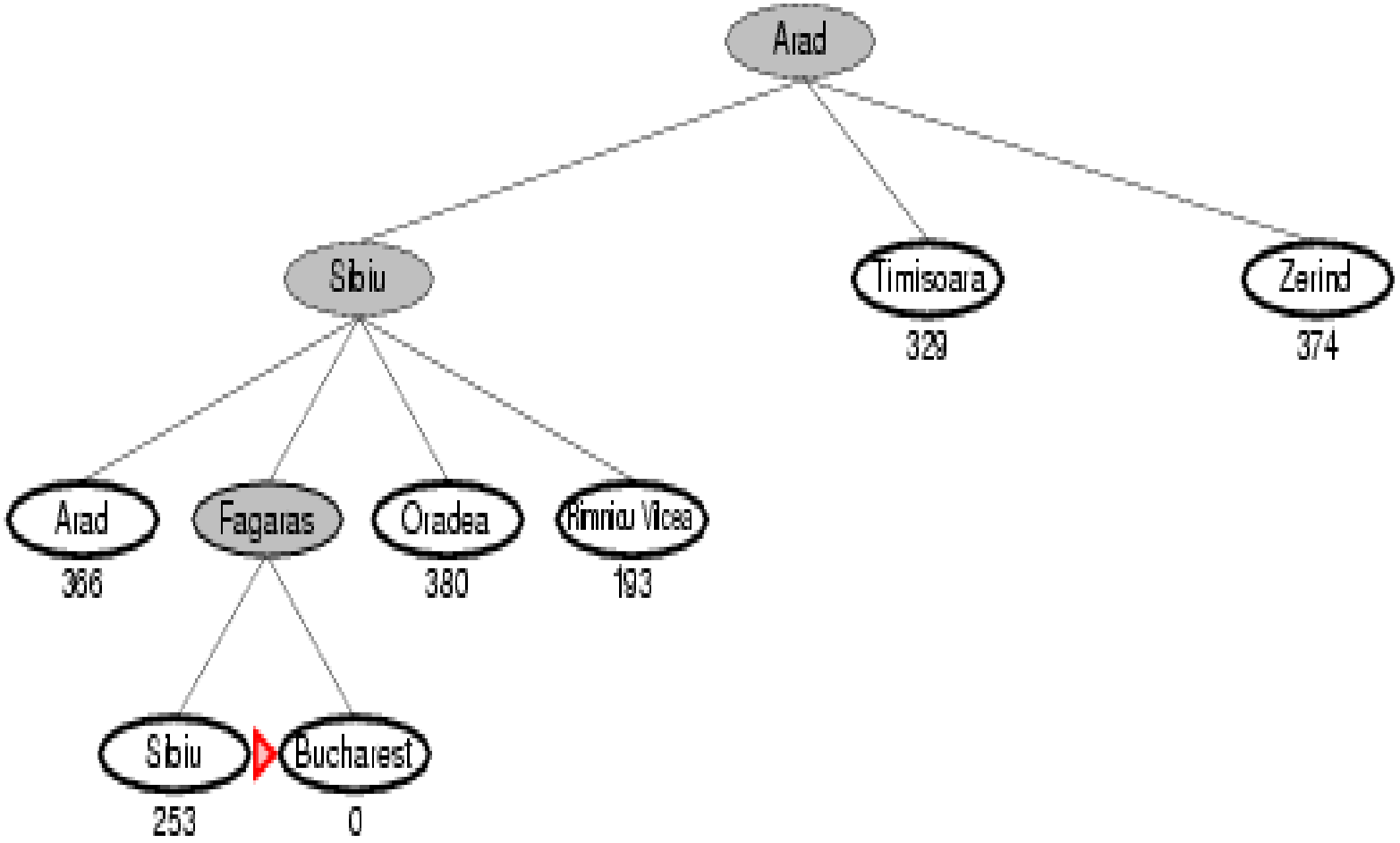
Greedy Best-First Search Example



Greedy Best-First Search Example



Greedy Best-First Search Example



Properties of Greedy Best-First Search

- Complete?
  - No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt → etc.
- Time?
  - $O(b^m)$, but a good heuristic can give dramatic improvement
- Space?
  - $O(b^m)$ -- keeps all nodes in memory
- Optimal?
  - No

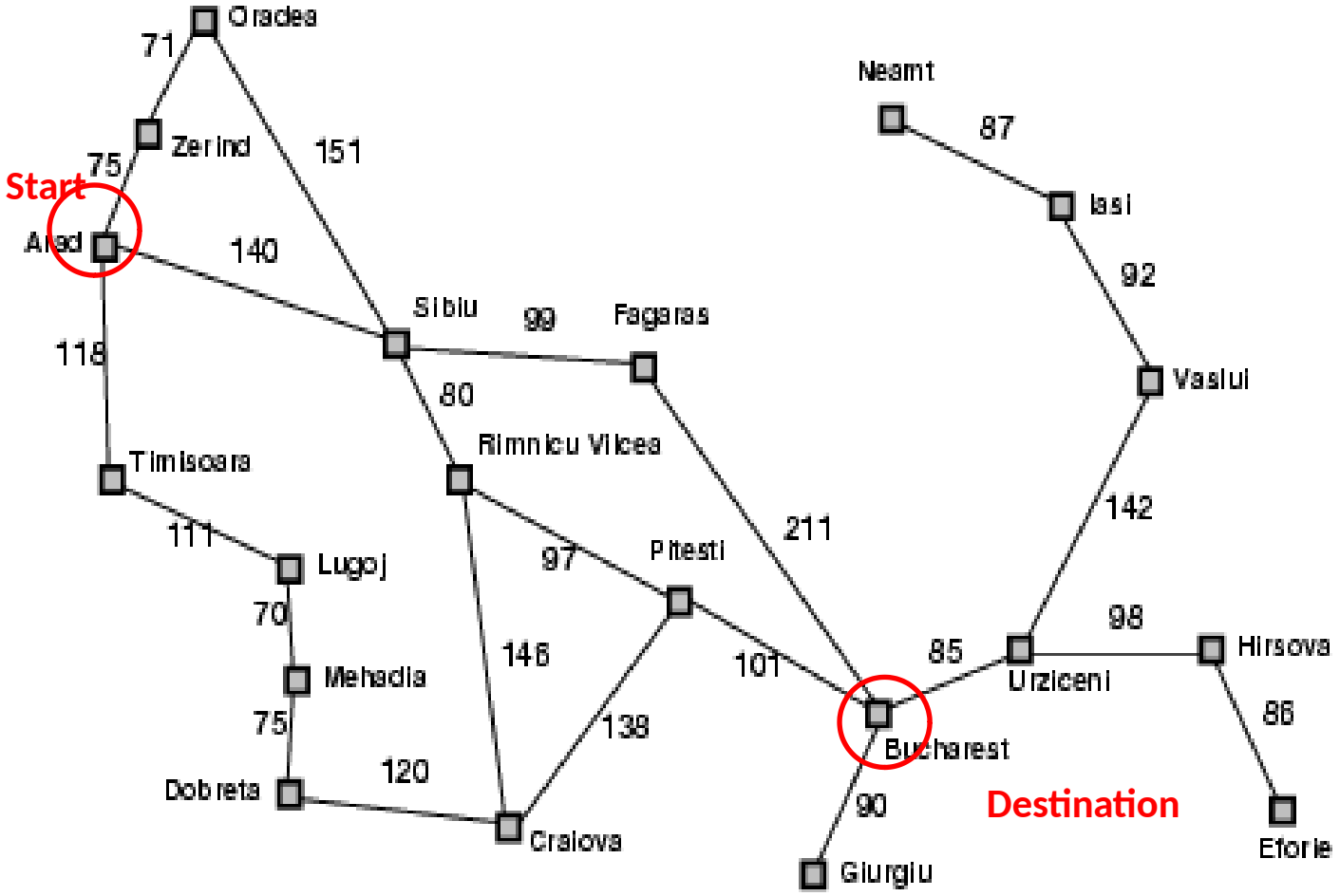
A* Search

- Idea: Avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $f(n)$ = estimated total cost of path through n to goal
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal

A* Search Example

- Choose h(n) to be the straight-line distance to the goal(Bucharest), as with Greedy Best-First
- Choose g(n) to be distance from the start node to the current node

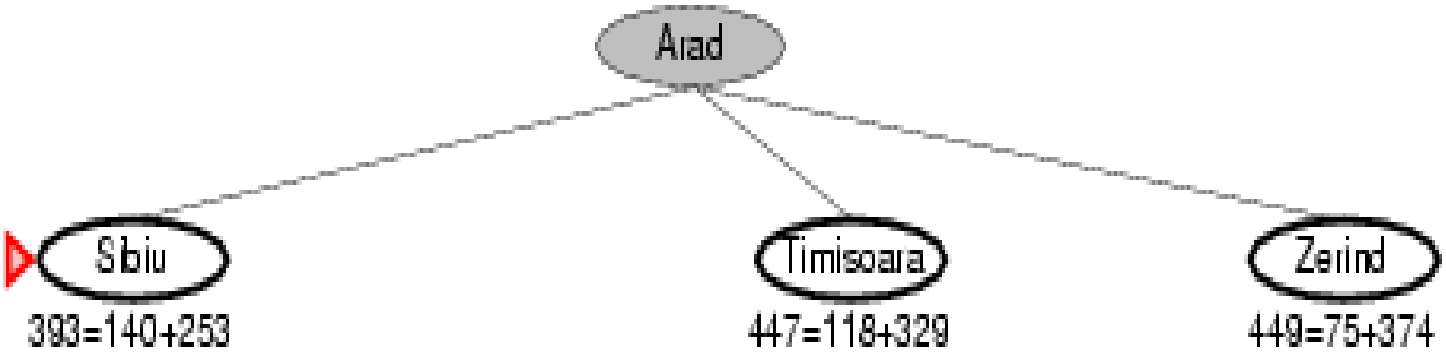
A* Search Example



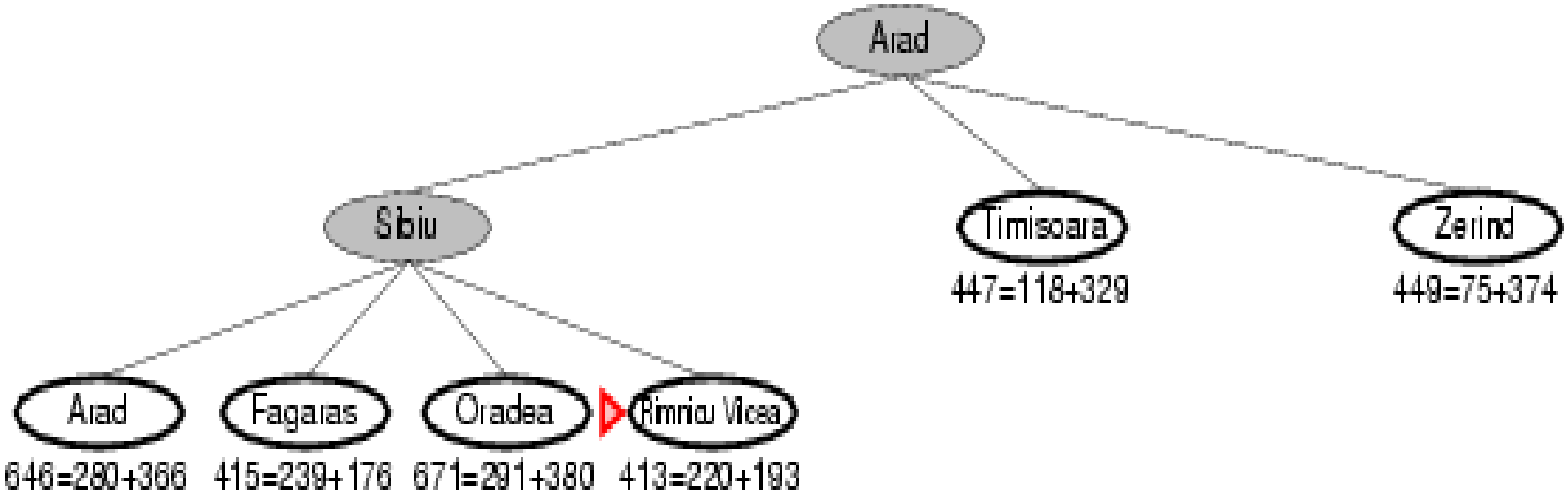
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Search Example

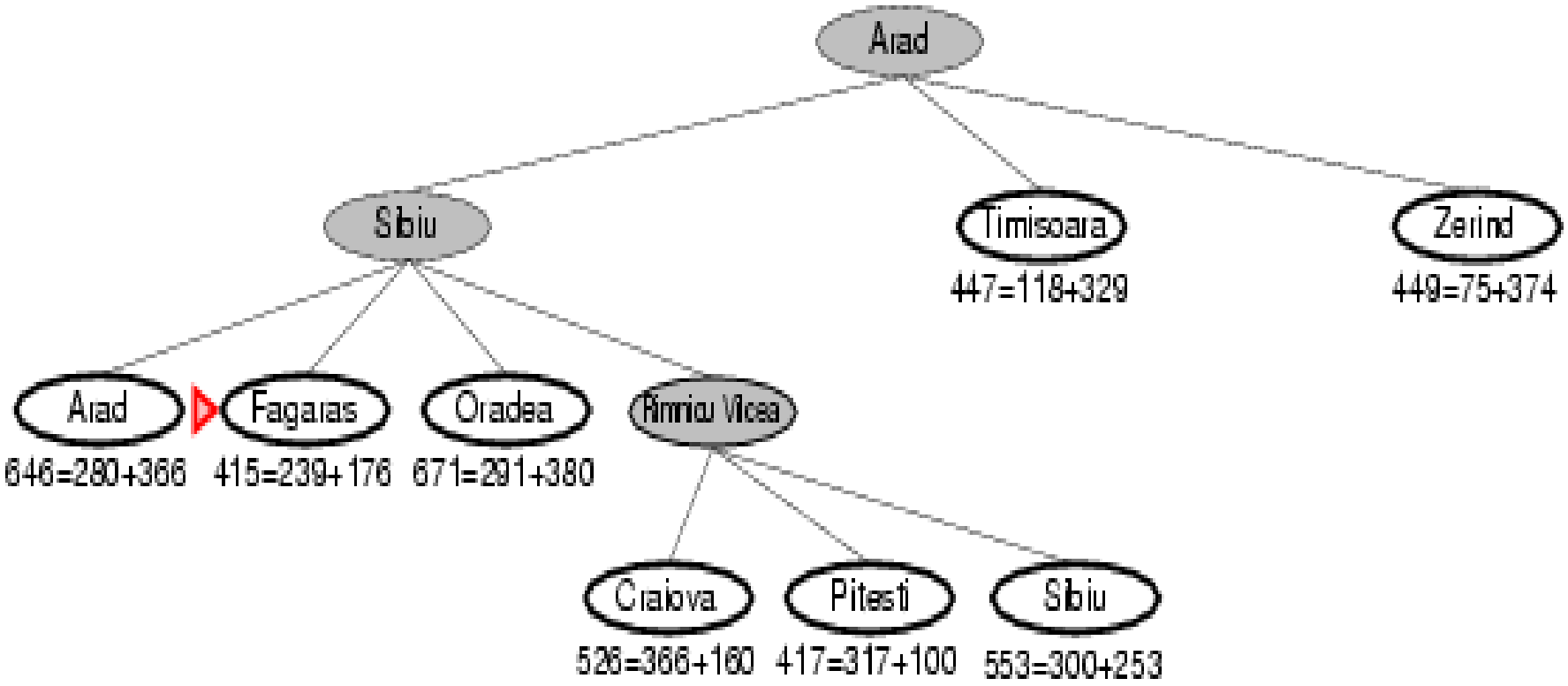
A* Search Example



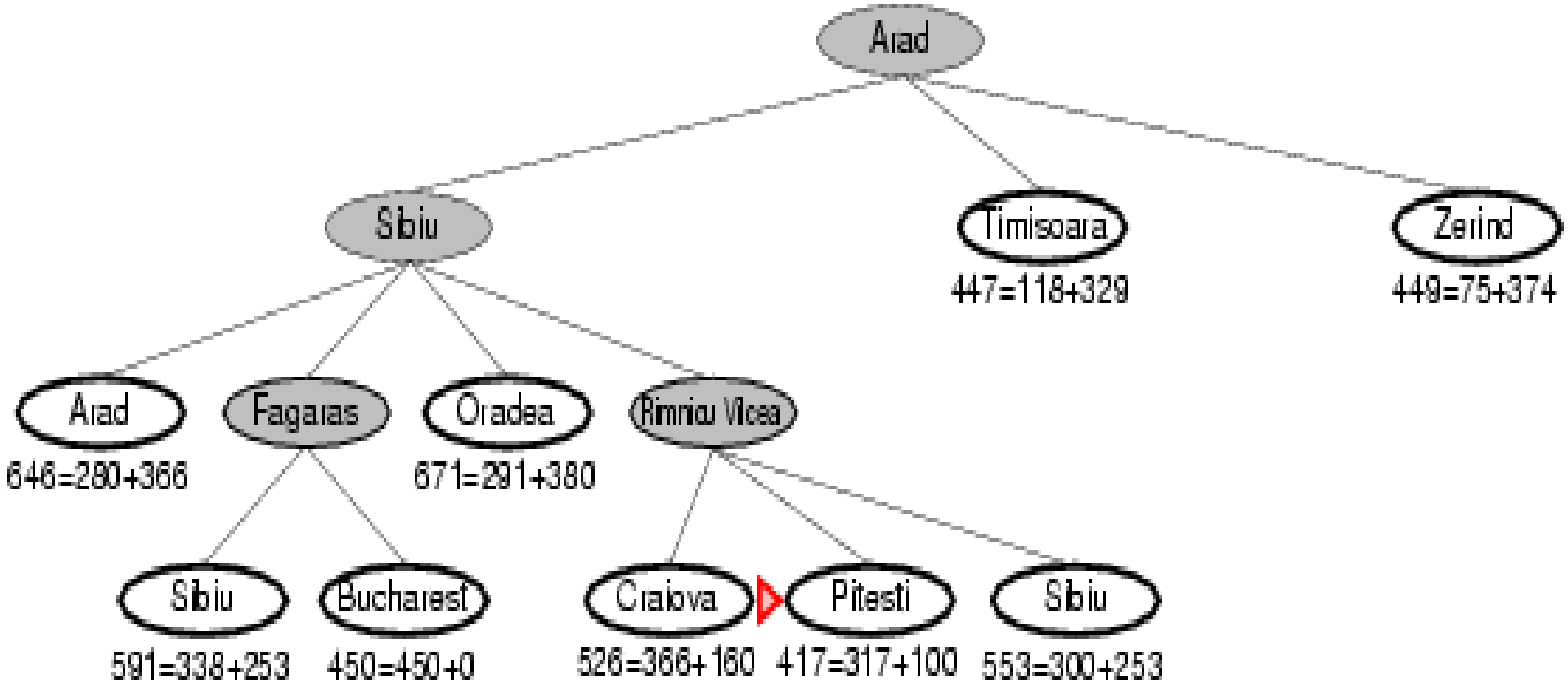
A* Search Example



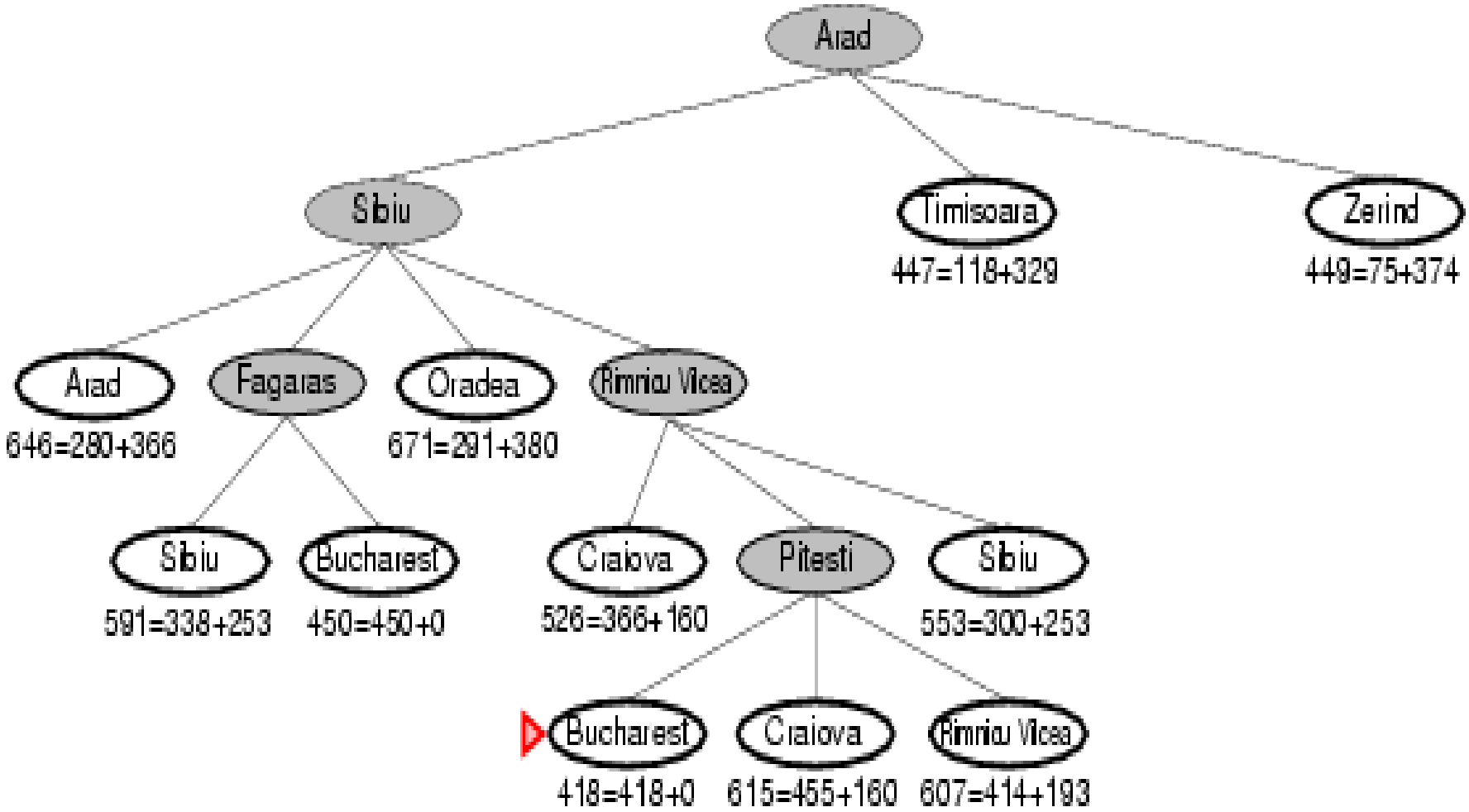
A* Search Example



A* Search Example



A* Search Example



A* Search

- A* algorithms are optimal:
  - They are guaranteed to find the shortest path to a goal node, provided *h* is never an overestimate
    - i.e., *h* is an *admissible heuristic*
- A* methods are also optimally efficient – they expand the fewest possible paths to find the right one
- If *h* is not admissible, the method is called A, rather than A*

Admissible Heuristics

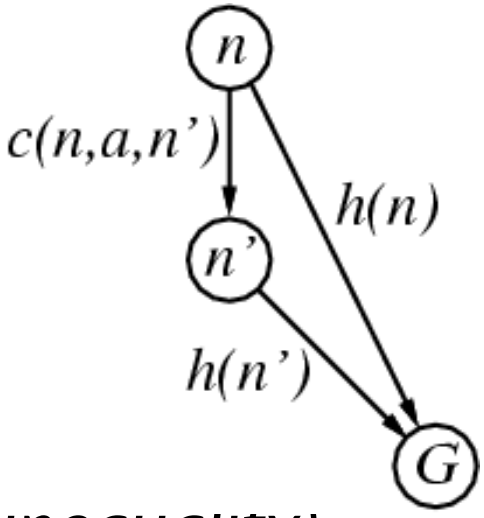
- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h(n)$ = "*straight-line distance*" never overestimates the actual road distance

Admissible Heuristics

- A heuristic *h* is *consistent* if for every node *n*, every successor *n'* of *n* generated by any action *a*,

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

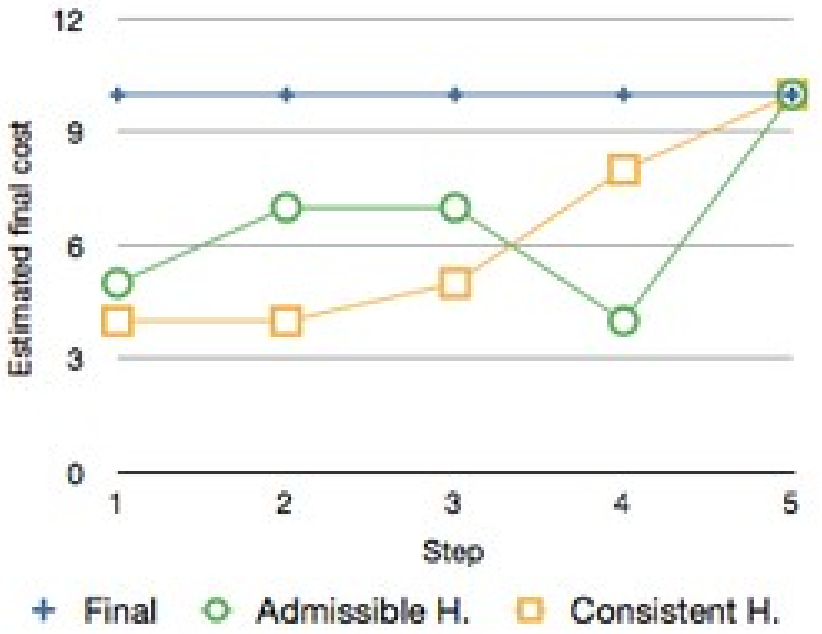
- If *h* is consistent, then for every node *n* and every successor *n'* of *n* generated by any action *a'*, the estimated cost of reaching the goal fro *n* is no greater than the step cost of get to n' plus the estimated cost of reachin goal from n' (also know as the *triangle inequality)*



- i.e., *f(n)* is non-decreasing along any path (*f(n)* is *monotonic)*

Admissible Heuristics

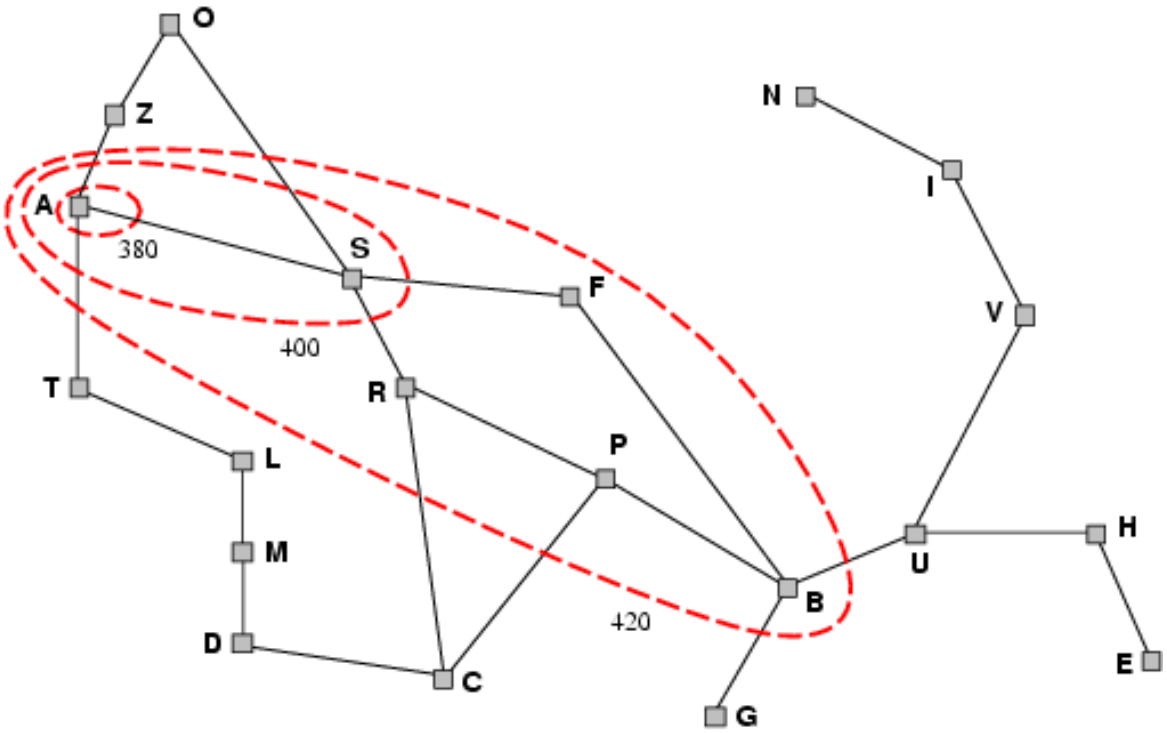
- All consistent heuristics are admissible, but not all admissible heuristics are consistent



http://en.wikipedia.org/wiki/Consistent_heuristic

Admissible Heuristics

- A* expands nodes in order of increasing *f* value
- It gradually adds "*f*-contours" of nodes, where contour *i* has all nodes with $f_i < f_{i+1}$

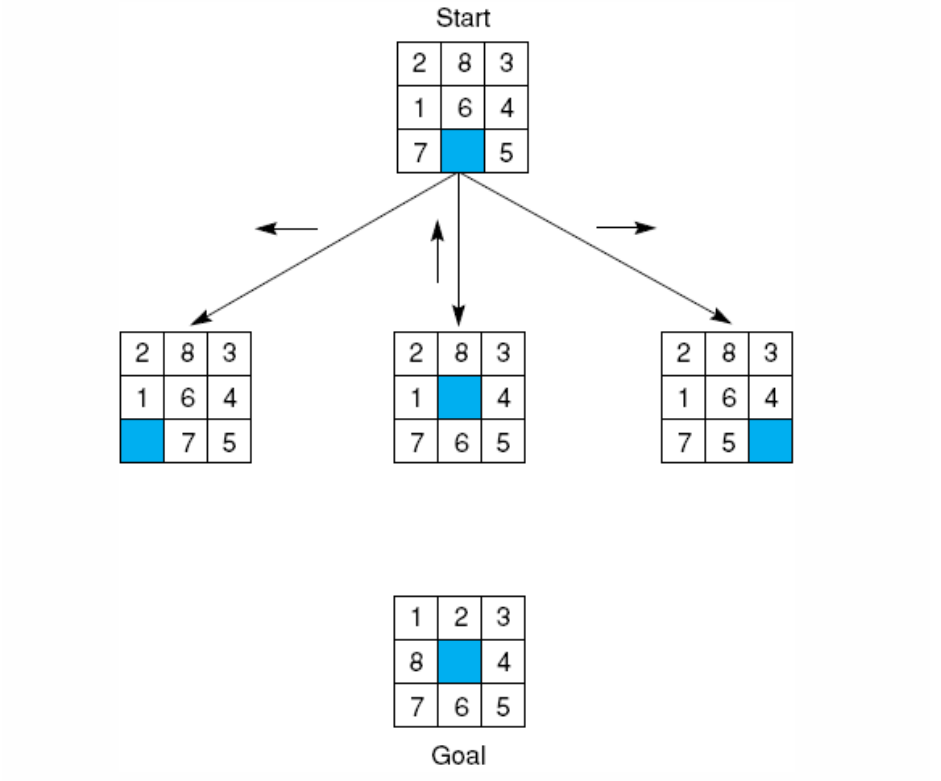


Another note on A*

- The choice of heuristic function h(n) can change the behavior of A*
- We can use different h(n) values or weights
- You may have noticed that h(n) = 0 is admissible (since h(n) <= h*(n) the actual cost)
- This is one extreme in which only g(n) plays a role and reduces A* to Dijkstra's shortest path algorithm, which is guaranteed to find the shortest path, but we may lose some efficiency
- On the other extreme is to use h(n) such that it is very high relative to g(n), in this case A* is reduced to Greedy Best First Search (remember that this is not optimal nor complete)

Other Admissible Heuristics

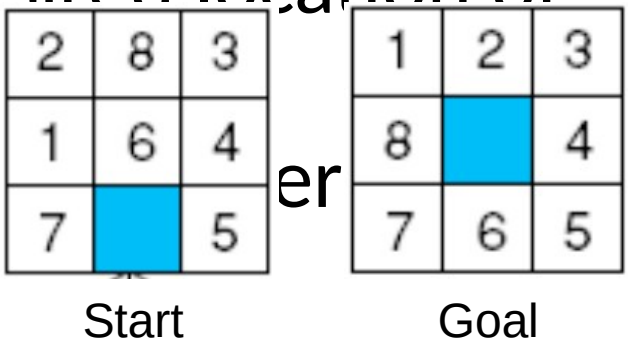
The start state, first move, and goal state for our example 8-puzzle:



Other Admissible Heuristics

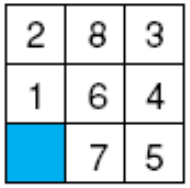
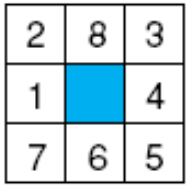
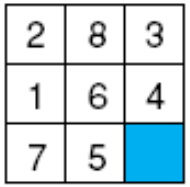
For the 8-puzzle:

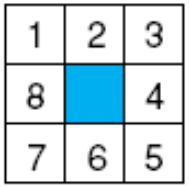
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance (i.e., the number of squares from desired location of each tile)
- $h_3(n)$ = 2 * number of direct[illegible]er



- $h_1(S) = 4$
- $h_2(S) = 1+1+0+0+0+2+0+2 = 6$
- $h_3(S) = 0$

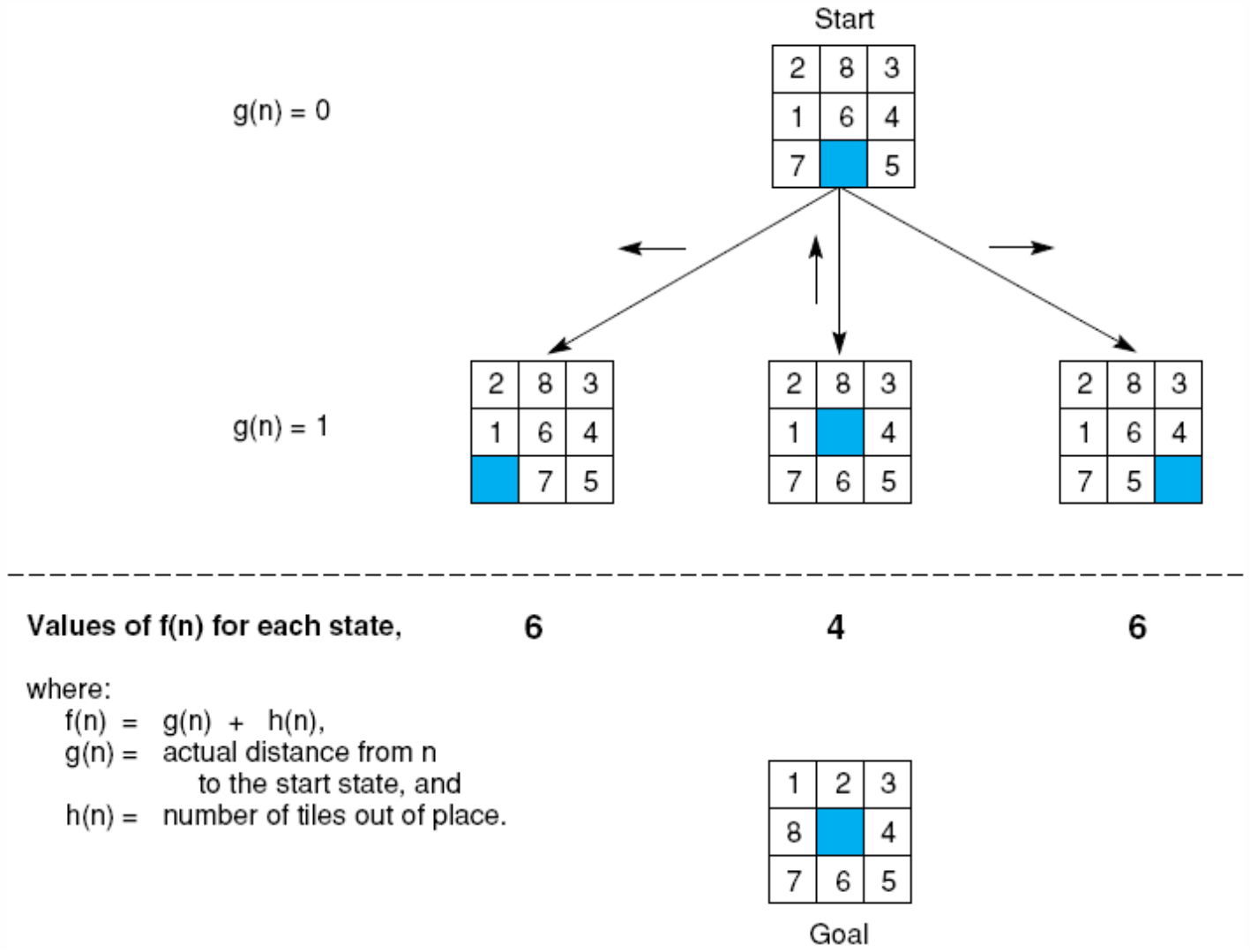
Other Admissible Heuristics

	**5**	**6**	**0**
	**3**	**4**	**0**
	**5**	**6**	**0**
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals

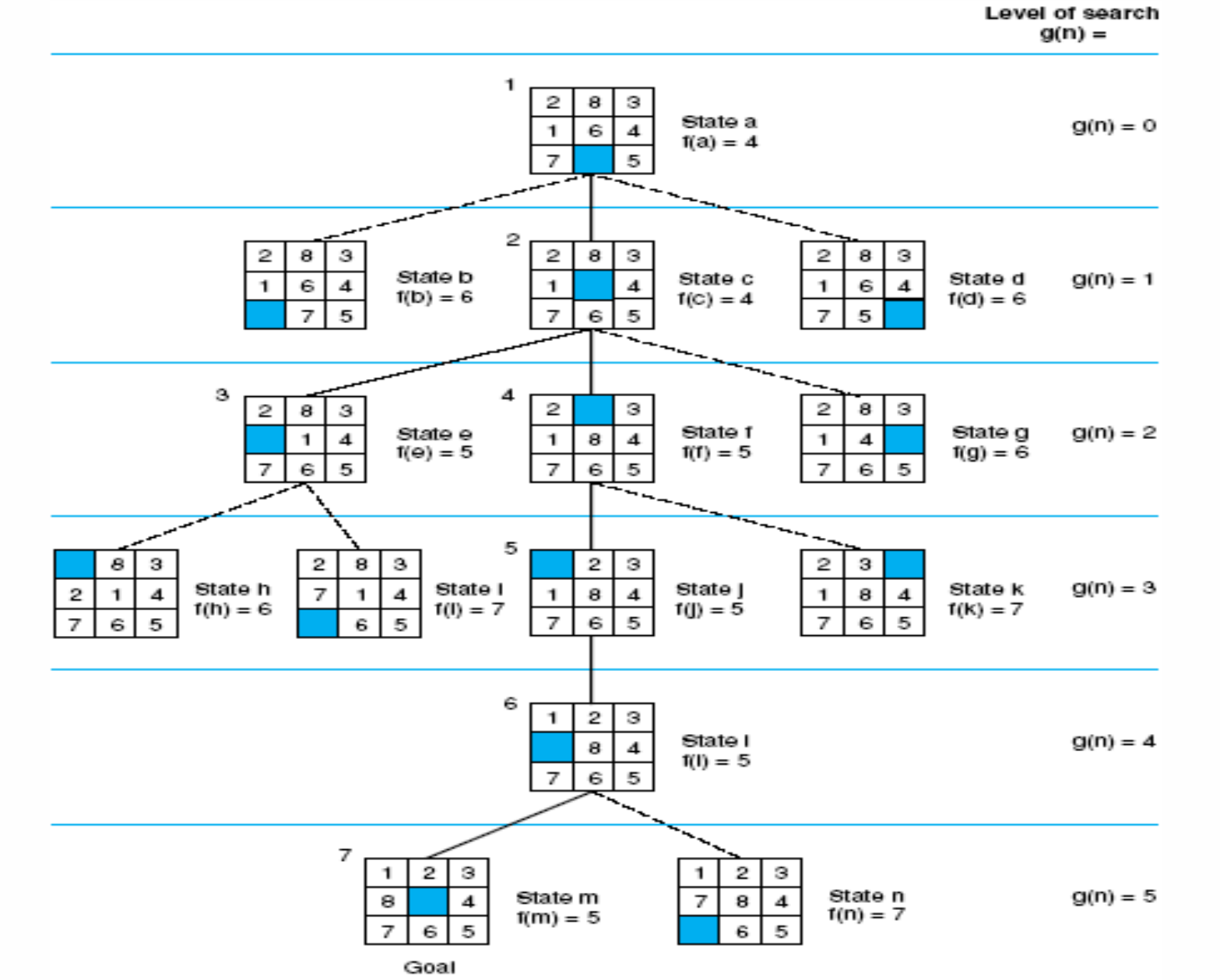


Goal

Other Admissible Heuristics



Other Admissible Heuristics



Properties of A*

- Complete?
  - Yes (unless there are infinitely many nodes with f ≤ *f(G)*)
- Time?
  - Exponential
- Space?
  - Keeps all nodes in memory
- Optimal?
  - Yes (if h(n) is admissible)

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (and both are admissible) then h_2 dominates h_1 (h_2 is better)
- Typical search costs (average number of nodes expanded) for iterative deepening search (IDS), $A^*(h_1)$ and $A^*(h_2)$ for the 8-puzzle :
- *depth = 12:* IDS = 3,644,035 nodes
- $A^*(h_1)$ = 227 nodes
- $A^*(h_2)$ = 73 nodes
- *depth = 24:* IDS = too many nodes (intractable!)
- $A^*(h_1)$ = 39,135 nodes
- $A^*(h_2)$ = 1,641 nodes

Relaxed Problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the exact solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the exact solution

Relaxed Problems

- If a problem is written down in a formal language, it is possible to construct heuristics automatically. Consider the following rule:
  - "A tile can move from square A to square B if A is horizontally or vertically adjacent to B and B is blank"
- We can generate three heuristics by removing one or both of the conditions from the above rule:

a) "A tile can move from square A to square B"
b) "A tile can move from square A to square B if A is adjacent to B"
c) "A tile can move from square A to square B if B is blank"

Constraint Satisfaction Problems

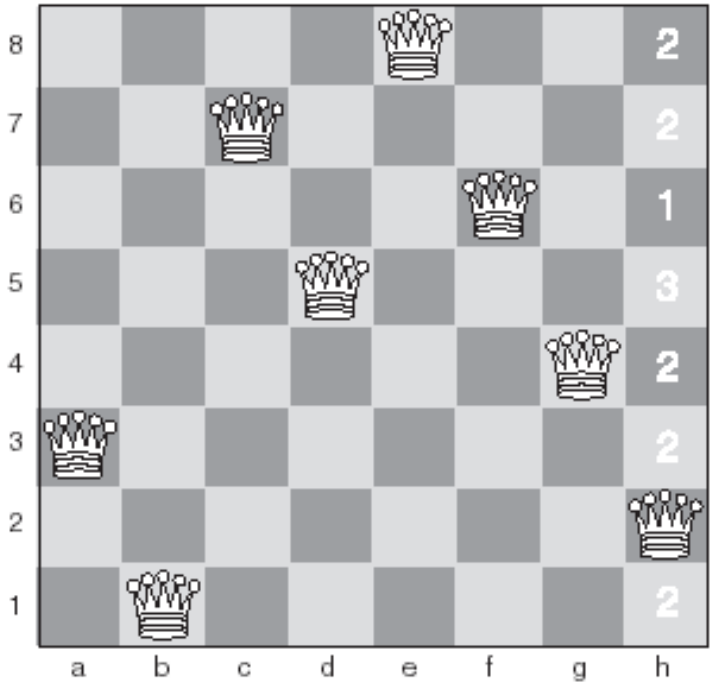
- A constraint satisfaction problem is a combinatorial optimization problem with a set of constraints
- Can be solved using search
- With many variables, it is essential to use heuristics (brute-force is too costly!)

The Eight Queens Problem

- Example of a constraint satisfaction problem:
  - Place eight queens on a chess board so that no two queens are on the same row, column or diagonal
- Can be solved by search, but the search tree is large
- *Heuristic repair* (constraint satisfaction) is very efficient at solving this problem

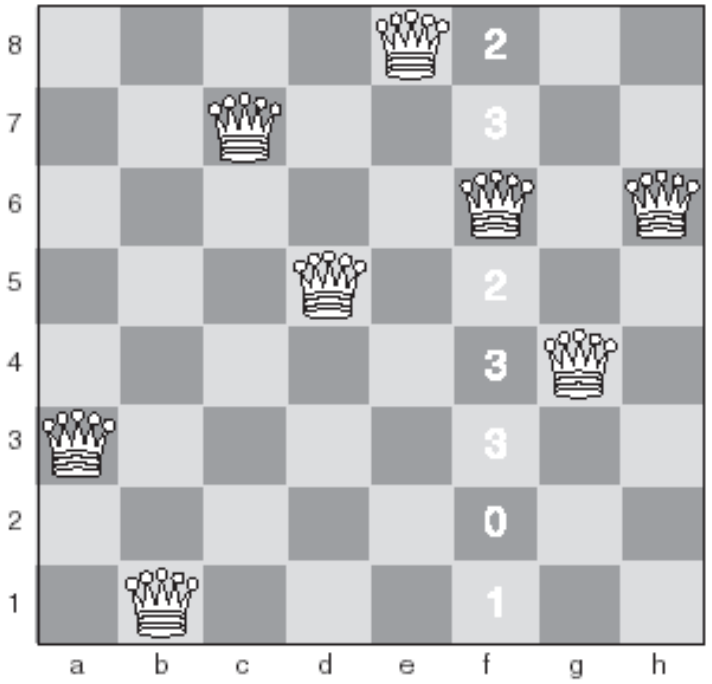
The Eight Queens Problem

- Initial state – one queen is conflicting with another (locations c7 and h2)
- Constrain the problem by only moving a queen in the same column (column h)
- The numbers in the right-most squares indicate how many conflicts will arise if we move the queen at h2 to that row
- Move the queen at h2 to the square with the fewest conflicts (h6)



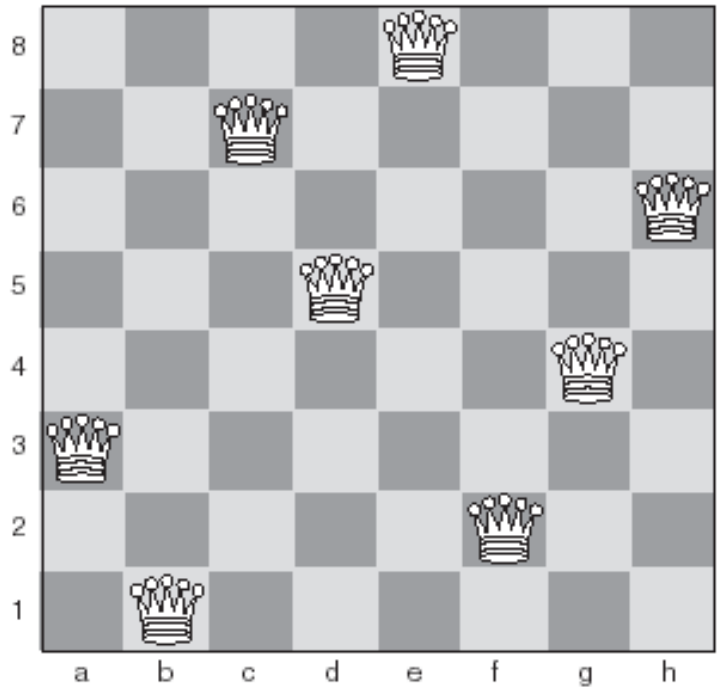
The Eight Queens Problem

- Second state – after moving the queen at h2 to h6
- Now the queen at f6 is conflicting with the queen at h6, so we'll move f6 to the square with the fewest conflicts (f2)



The Eight Queens Problem

- Final state – a solution!



Local Search Algorithms

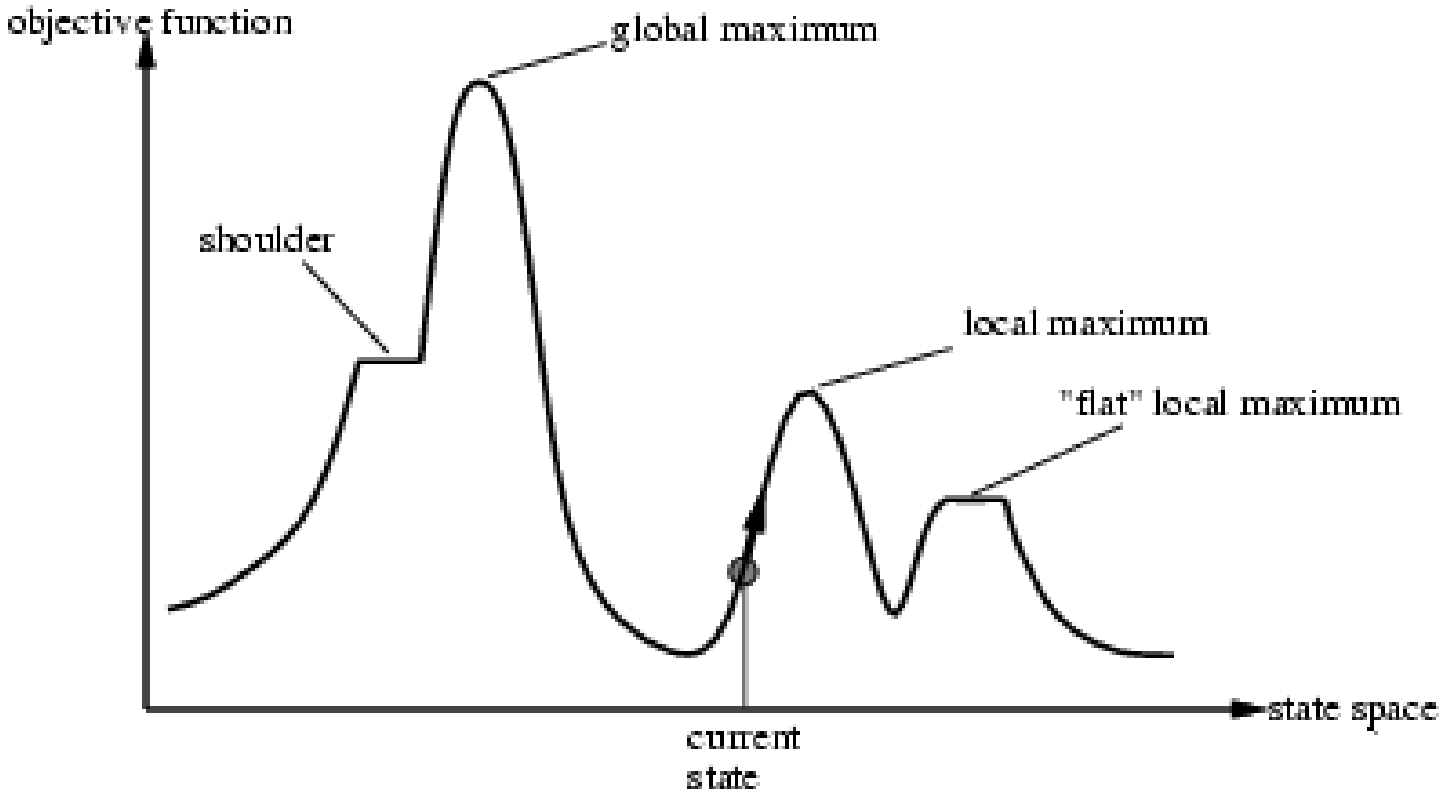
- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- Like heuristic repair, local search methods start from a random state, and make small changes until a goal state is achieved
- Most local search methods are susceptible to local maxima, like *hill-climbing*

Hill-Climbing

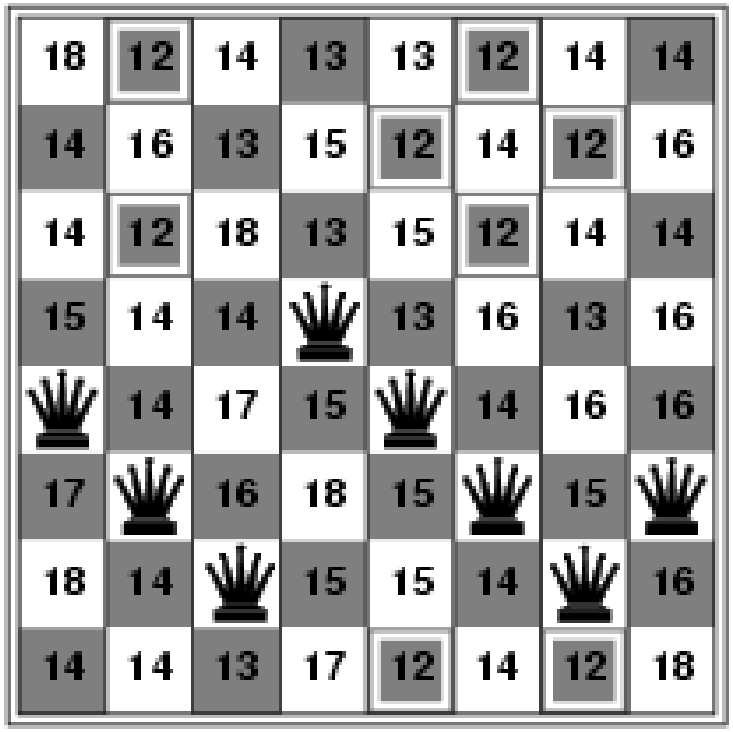
- Hill-climbing is an informed, irrevocable (choices cannot be “un-done”) search method (once you go down a path, there's no turning back)
- Easiest to understand when considered as a method for finding the highest point in a three dimensional search space:
  - Check the height one unit away from your current location in each direction
  - As soon as you find a position whose height is higher than your current position, move to that location, and restart the algorithm

Hill-Climbing

- Problem: depending on the initial state, you can get stuck in local maxima

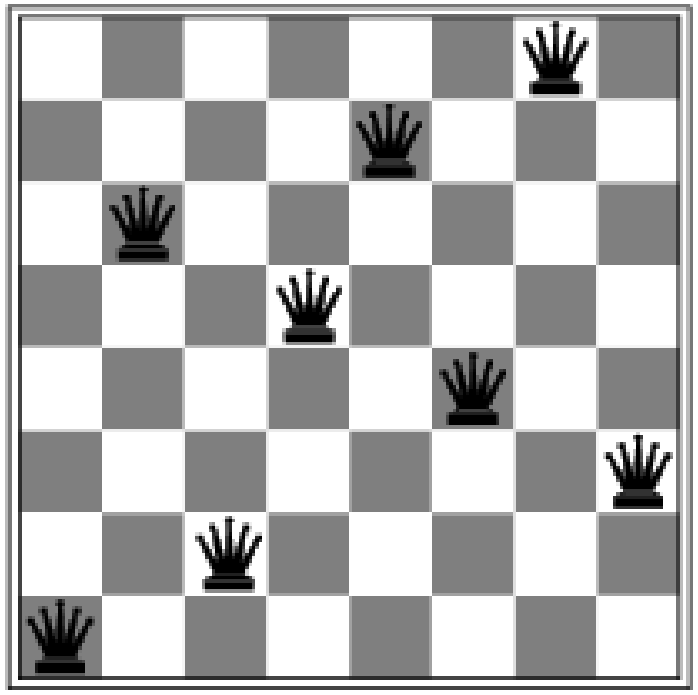


Hill-Climbing: 8-Queens Problem



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly.
- *h = 17* for the state shown here
- The number in each square indicates how many conflicts will result by moving the queen in that column to the indicated row

Hill-Climbing: 8-Queens Problem



A local minimum with *h = 1*

Improved Heuristics For The Eight Queens Problem

- One way to solve this problem is to have a tree that is 8-ply deep (a *ply* is a level), with a branching factor of 64 for the first level (because there are 64 squares to place the first queen on an empty board), 63 for the next level, and so on, down to 57 for the eighth level

Improved Heuristics For The Eight Queens Problem

- This can be improved further by noting that each row and column must contain exactly one queen
  - If we assume the first queen is placed in row 1, the second in row 2, etc. the first level will have a branching factor of 8, the next 7, the next 6, etc.
  - In addition, as each queen is placed on the board, it "uses up" a diagonal, meaning that the branching factor is only 5 or 6 after the first choice has been made

Simulated Annealing Search

- Since hill-climbing never moves “downhill” can get stuck at local maximum
- And purely random walk would be inefficient
- What if we tried combining both in a way to yield efficient and complete
- A method borrowed from metallurgy (statistical physics), based on the way in which metal is heated and then cooled very slowly in order to make it extremely strong
- Goal is to obtain a minimum value for some function of a large number of variables
  – This value is known as the *energy of the system*

Simulated Annealing Search

- Simulated annealing escapes local minima by allowing some "bad" moves but gradually decreases the frequency with which they are allowed

```
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)
```

Simulated Annealing Search

- A random start state is selected
- A small random change is made to this state
  - If this change lowers the system energy, it is accepted
  - If it increases the energy, it may be accepted, depending on a probability called the Boltzmann acceptance criteria:
    - $e^{(-dE/T)}$

Simulated Annealing Search

- Because the energy of the system is sometimes allowed to increase, simulated annealing is able to escape from local minima
- Simulated annealing is a widely used local search method for solving problems with a very large numbers of variables
  - For example: scheduling problems, traveling salesperson (TSP), placing VLSI (chip) components

331 – Intro to Intelligent Systems
Week 04
Game Theory I
Pure strategies in Adversarial search
R&N Chapter 17.5, 17.6

T.J. Borrelli

Algorithmic Game Theory

- Game theory is the study of *strategic interactive decision-making among rational agents*
- There are three major components of any game:
  1. Players – the agents who play the game
  2. Strategies – what the agents do, how they will respond in *every possible* situation
  3. Payoffs – how much each player likes the result

Algorithmic Game Theory

- Includes:
  - Sequential games
  - Simultaneous games
  - Threats, promises, commitments
  - Credibility, deterrence, compellence
  - Signaling and screening
  - Incentives
  - Voting, auctions, bargaining

Algorithmic Game Theory

- Modern game theory began in 1944 with the publication of *Theory of Games and Economic Behavior* by John von Neumann and Oskar Morgenstern
- Their goal was to allow economics to be studied as a science, similar to physics
- Many Nobel Prizes in economics have been awarded to people for their work in game theory

Some Applications of Game Theory

- Consumer behavior
- Elections
- War
- Terrorism
- Dating
- Global warming
- Traffic congestion – human and network
- Computer games
- Interactions among multi-agent robotic systems
- Machine learning
- Many, many, more

Strategies

- Pure strategies
  - Do not involve any element of chance
    - In other words, you are not going to flip a coin to decide what to do
- Mixed strategies
  - Involve chance
  - The strategy must be kept a secret from your opponent
  - You must keep your opponent guessing by “mixing it up”

Mixed strategy

- Mixed strategies
  - Chooses some action *a*
  - With probability *p*
  - If there are only two actions then the probability of *b* is *1-p*
  - Some games only have solutions with a mixed strategy

Payoffs

- Once the strategies interact and play themselves out, the game is over and the players receive a payoff
- The payoff represents how much each player likes the outcome of the game
- The bigger the payoff, the more the player likes the outcome (i.e., an outcome of 4 is better than an outcome of 2)
- Negative values are also sometimes used to show utility/disutility

Rational Decision-Making

- Example: Let's say that I would like to sell you a vase for $11 that I had previously bought for $8. You believe the vase is worth $18
- If the deal does not go through then both of us have a profit of $0 (ignoring the residual value of the vase)
- If it does go through, then I have a profit of $3 and you have a profit of $7

Rational Decision-Making

- Payoffs represent what ***each player*** cares about, not what another player thinks the other player should care about
  - Therefore, payoffs for different players cannot always be directly compared
  - In the previous example, if all each player cares about is money, then the payoffs are $3 and $7 respectively
  - If one player feels that he or she is getting “ripped off” then the payoff changes to reflect the real value of the transaction

Rational Decision-Making

- Being rational means that each player makes decisions based on what that player believes will lead to the best expected payoff *for them!*
- Example – The Ultimatum Game: Take it or leave it?
- Even when considering $$, one dollar may not be equivalent to another

Finite vs. Infinite Games

- Finite games
  - Must be guaranteed to eventually end
  - Must have a finite number of choices for each player
  - Played with the goal of winning
  - Debates, sports, receiving a degree, etc.
- Infinite (non-finite) games
  - No definite beginning or ending
  - Played with the goal of continuing to play
  - Beginning to play does not require volunteering or conscious thought, continuing to play does
  - Life

Ordinal vs. Cardinal Payoffs

- Ordinal payoffs
  - —You only need to know the ordering, or preferences of the outcomes, i.e., first choice, second choice, third choice, etc.
  - —Consider an ice cream payoff: first choice = vanilla, second choice = chocolate, third choice = horseradish ripple
- Cardinal payoffs
  - —Are on an interval scale, i.e., the difference between 10 and 20 is the same as the difference between 30 and 40
  - —You must know more than just the ordering of the outcomes
  - —The ice cream payoffs shown above are ordinal, not cardinal

Common Knowledge

- Assume that the rules of the game and the rationality of the players is common knowledge
  - In other words, everyone knows the rules of the game
- In addition, everyone knows that everyone knows the rules of the game
- And everyone knows that everyone knows that everyone knows the rules of the game, etc.

Sequential Games

- Sequential games represent events unfolding over time
  - Also called “dynamic games”
  - Players have full knowledge of other players’ moves
  - Chess, monopoly, open auctions, etc.
- Simultaneous games represent events occurring at the same time
  - Also called “static games”
  - Players do not know what other players are doing
  - Silent auctions, clicker game, etc.

First-Mover Advantage?

- Do first movers always have an advantage in sequential games?
  - Not necessarily
  - Going first means *committing to a course of action*
  - Not going first means *flexibility of response*
- Does order make a difference in the payoffs to the individual players?
  - In general, yes – going 2nd allows the player to weigh options against a fixed decision from the other player

Non-Cooperative Games

- Cooperative games imply that binding agreements between the players are possible

- Non-cooperative games imply that binding agreements are not possible

Dominant Strategy

- A strategy that **strongly** dominates does so if the outcome of the strategy for the player is better than any other outcome
- A strategy **weakly** dominate if it is no worse than any other
- A dominant strategy is a strategy that dominates all others
- It is irrational to play a dominat**ed** strategy
- It is irrational not to play a dominant strategy if one exists

Pareto

- An outcome is **Pareto optimal** - If there is no other outcome that **all** players would prefer

- An outcome is **Pareto dominated** by another outcome if all players would prefer the other outcome

The Roll-Back Approach

- In order for the roll-back approach to work, the game must be finite (no randomness), non-cooperative, sequential, and must have *perfect information*
  - Perfect information means that all players know the potential payoffs for each player before any moves are made, and all decisions are made public (no secrets)
  - Chess is an example of a game of perfect information, poker is a game of imperfect information

Nash Equilibrium

- If a game outcome is an equilibrium, then no player can gain from unilaterally changing his or her strategy
  - No regrets! Even if you don't end up with exactly what you wanted.
  - John Forbes Nash invented the idea of the "Nash Equilibrium"
  - If every player is playing the Nash equilibrium, then you might as well also because you will not gain anything by changing your strategy

Nash Equilibrium

- Nash equilibrium is essentially a local optimum
- Nash (the mathematician) proved that a game has at least one **mixed strategy** equilibrium
- But not necessarily a pure strategy equilibrium

Simultaneous Games

- In simultaneous games all of the players make their decisions at essentially the same time
- Players do not know what other players are doing at the time they make their decisions
- No one "goes first"
- Use a payoff matrix instead of a game tree
  - 2 x 2 matrix
  - Each element in the matrix contains the cardinal value of each player's preferences (the payoff)

Simultaneous Games

- The Coordination Game
- The Battle of the Sexes
- The Game of Chicken
- The Prisoner’s Dilemma

Simultaneous Games

- Label the rows of the payoff matrix with the choices of player 1, and the columns with the choices of player 2 (book uses slightly different convention)

	Player 2 : Choice 1	Player 2 : Choice 2
Player 1 : Choice 1	P1=payoff, P2=payoff	P1=payoff, P2=payoff
Player 1 : Choice 2	P1=payoff, P2=payoff	P1=payoff, P2=payoff

The Coordination Game

- Consider the example of two firms choosing whether to use standard X or standard Y for their joint software project
- They both prefer standard X over standard Y, but the least favorite option is to disagree with the other firm (one chooses X and the other chooses Y, or vice-versa)

	Firm B : standard X	Firm B : standard Y
Firm A : standard X	A=2, B=2	A=0,B=0
Firm A : standard Y	A=0, B=0	A=1,B=1

The Coordination Game

- Assume common knowledge – both players know all of the game matrix payoffs
- Assume both players are rational – their stated preferences are their true preferences
- Then choosing the “standard X/standard X” option is best

The Battle of the Sexes

- What if we change the situation slightly – Player A prefers Opera over Football, and Player B prefers Football over Opera, but both still want to work together (coordinate)

	Player B : Opera	Player B : Football
Player A : Opera	A=2, B=1	A=0,B=0
Player A : Football	A=0, B=0	A=1,B=2

- No good solution – unless you can find a Schelling (focal) point (i.e. a point that players will tend to choose in absence of communication)

The Game of Chicken (anti-coordination)

- Suppose Player A and B are fierce competitors and the success of their products rely on using different standards; however, the standard X action is clearly superior to any other on the market (there are several others)

	Player B : standard X	Player B : other standard
Player A : standard X	A=0, B=0	A=3,B=1
Player A : other standard	A=1, B=3	A=2,B=2

The Game of Chicken

- What is the best thing to do?
- It may seem most fair for both firms to choose another standard, but if Player A (or B) knew that the other Player was going with the other standard, then Player A (B) would prefer to stay with standard X to get the payoff
- Therefore (2,2) is NOT a Nash equilibrium
- Solution again is to create a Schelling point

The Prisoner's Dilemma

- Two criminals committed a crime together, and are being interrogated in separate cells. If neither one confesses, they'll each get a year in prison. If one confesses, that one goes free but the other one gets five years. If they both confess, they both get three years.
- Assign values to these outcomes (free = 5, 1 year = 3, 3 years = 1, 5 years = 0)

The Prisoner's Dilemma

- What if both players refuse to defect if the other player gets to use the better choice (for spite)?

	Prisoner 2 : confess	Player 2 : keep silent
Prisoner 1 : confess	P1=1, P2=1	P1=5,P2=0
Prisoner 1: keep silent	P1=0, P2=5	P1=3,P2=3

  - No matter what Prisoner 2 does, it is better for Prisoner 1 to choose action "confess"
  - Therefore, both firms will choose "confess" (payoff = 1,1)
  - Notice that this payoff is *lower for both players (!)* than if both chose to keep silent (payoff = 3,3)

The Prisoner's Dilemma

- Both will confess and both will get three years, whereas they would have been better off if they had both kept silent!
- The *dilemma* in the prisoner's dilemma is that the equilibrium outcome is worse for both players than the outcome they would get if they both refused
- (1, 1) outcome for (confess, confess) is Pareto dominated by (3, 3) outcome of (silent, silent)

The Prisoner's Dilemma

Consider 1964 when the federal government banned cigarette advertising on television. Before the ban:

	Company B: advertise	Company B: don't advertise
Company A : advertise	A=1150, B=1150	A=2020, B=630
Company A: don't advertise	A=630, B=2020	A=1500, B=1500

The Prisoner's Dilemma

- The Soviet Union exploded its atomic bomb in 1949
  —The Prisoner's Dilemma was discovered in 1950
- Preferences (in order)?
  1. We nuke them
  2. No one nukes anyone
  3. Everyone nukes everyone
  4. They nuke us
- If this is the order of preferences, then one equilibrium is nuclear war – both sides launch missiles
- In the years following 1950 many people (including von Neumann) thought that nuclear war with the U.S.S.R. was inevitable

No Dominant strategy

- Here is a game where there is no dominant strategy
- However, there are *two* Nash equilibria: (bluray, bluray) and (dvd,dvd).
- These are both Nash equilibria because if either player unilaterally moves to a different strategy, that player will be worse off

	Best: bluray	Best: dvd
Acme: bluray	A=9, B=9	A=-4, B=-1
Acme: dvd	A=-3, B=-1	A=5, B=5

Tragedy of the Commons

- Let's consider another game where countries set their policy on controlling air pollution.
- Each country has a choice:
  - they can *reduce* pollution at a cost of -10 for implementing the changes
  - They can *continue* to pollute which gives them a net utility of -5 (in added health costs etc.) AND contributes -1 to every other country because air shared

Tragedy of the Commons

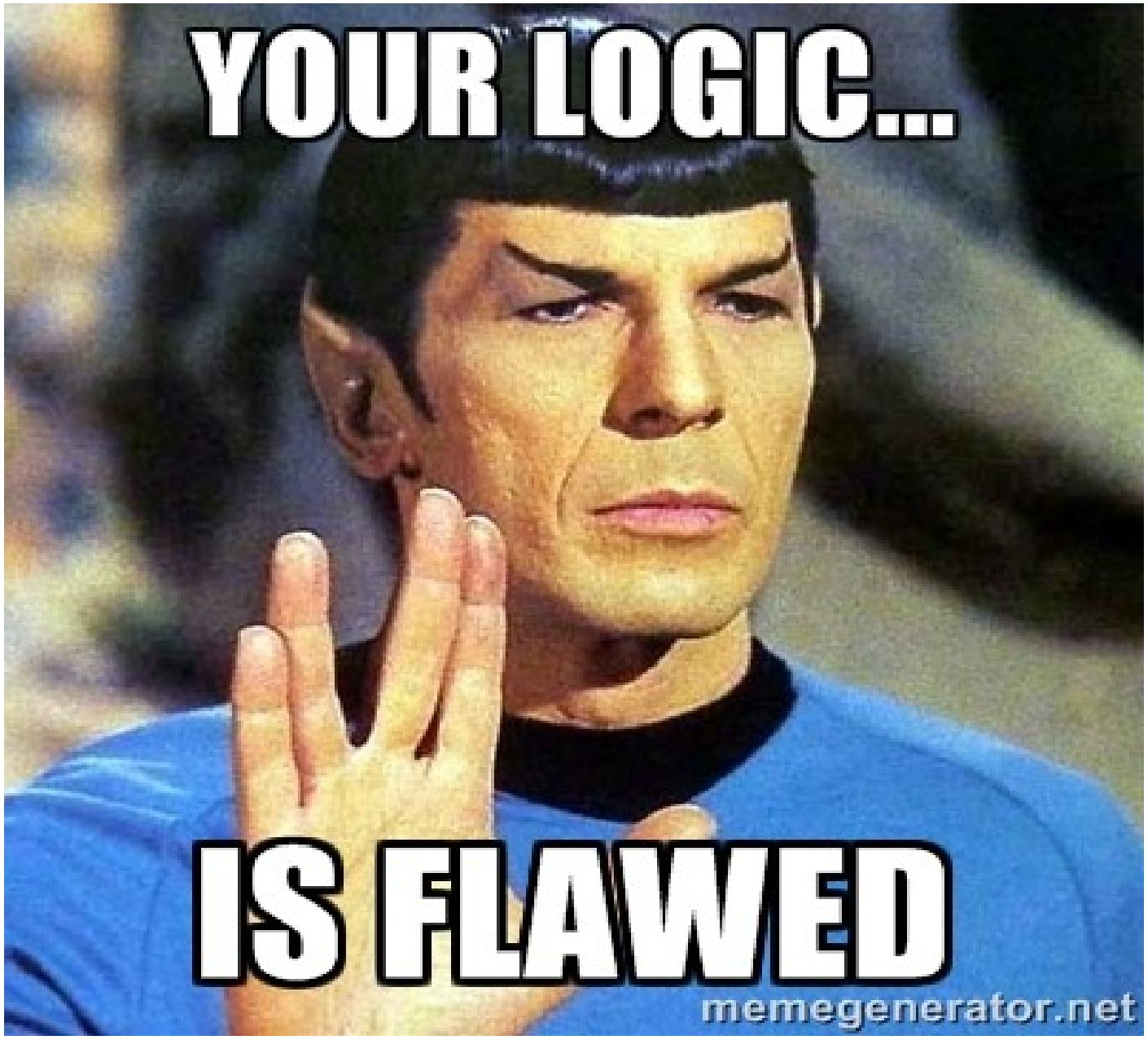
- Clearly, the dominant strategy for each country is “continue to pollute”
- This is bad enough if there are two countries (similar to the prisoner's dilemma) however, let's assume we have 100 countries
- If everyone follows this policy, then each country gets a total utility value of -104
- Whereas if every country reduced pollution, they would each have a utility of -10
- This is the tragedy of the commons: if nobody has to pay for a common resource, it tends to be exploited in a way that leads to a lower total utility for all

The Tragedy of the Commons

- From the individual game-theoretic perspective, shirking is the right thing to do because it is the dominant strategy
- From a social good perspective, it is a tragedy
- Self interest, in this case does not maximize the common good (contrary to Adam Smith's philosophy) because the cost that the shirkers create have to be borne by others

331 – Intro to Intelligent Systems
Week06
Propositional Logic
R&N Chapter 7.1 – 7.5

T.J. Borrelli


YOUR LOGIC...
IS FLAWED
memegenerator.net

What is Logic?

- Reasoning about the validity of arguments.
- An argument is **valid** if its conclusions follow logically from its premises – even if the argument doesn't actually reflect the real world:
  - All lemons are blue
  - Mary is a lemon
  - Therefore, Mary is blue

How is Logic Used in Intelligent Systems?

- Logic is used as a representational method for communicating concepts and theories
- Logic allows us to reason about negatives ("the book is not red") and disjunctions ("he's either a soldier or a sailor")
- Logic is used in systems that attempt to understand and analyze human language

Weaknesses of Logic

- Formal logics are unable to deal with uncertainty
  - Logical statements must be expressed in terms of truth or falsehood, not possibilities
- Formal logics are not well suited to deal with change
- Formal logics are not well suited to deal with events unfolding over time

Logical Operators

And	$\wedge$
Or	$\vee$
Not	$\neg$
Implies	$\rightarrow$ (if… then…)
Iff	$\Leftrightarrow$ (if and only if)

Truth Tables

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Truth Tables

- Truth table demonstrating the equivalence of $P \rightarrow Q$ and $\neg P \vee Q$:

P	Q	¬P	P → Q	¬P ∨ Q
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Truth Tables

- Truth table demonstrating the non-equivalence of $A \wedge (B \vee C)$ and $(A \wedge B) \vee C$:

A	B	C	$A \wedge (B \vee C)$	$(A \wedge B) \vee C$
T	T	T	T	T
T	T	F	T	T
T	F	T	T	T
T	F	F	F	F
F	T	T	F	T
F	T	F	F	F
F	F	T	F	T
F	F	F	F	F

English vs. Logic

- Facts and rules need to be translated into logical notation
- For example:
  - It is raining and it is Thursday:
  - R means “It is raining”, T means “it is Thursday”
  - R Λ T

English vs. Logic

- Sentences in predicate calculus are created using predicates along with logical operators and quantifiers
- For example, the English sentence, “Whenever he eats sandwiches that have pickles in them, he ends up either asleep at his desk or singing loud songs” can be expressed as:

$$s(Y) \wedge e(X, Y) \wedge p(Y) \rightarrow a(X) \vee (s(X, Z) \wedge o(Z))$$

s(Y) refers to the sandwich (Y)

e(X, Y) means that he (X) eats the sandwich (Y)

p(Y) means that the sandwich (Y) has pickles in it

a(X) means that he (X) ends up asleep at his desk

s(X, Z) means that he (X) sings songs (Z)

o(Z) means that those songs (Z) are loud

Entailment

- Entailment means that one thing follows from another:
- KB $\models \alpha$

•A knowledge base KB entails sentence α if and only if α is true in all worlds where KB is true

- KB is a subset of α
- KB is a stronger assertion than α since it rules out more worlds
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics
  - For example, the KB containing “the Giants won” and “the Bills won” entails “the Giants won and the Bills won”

Tautology

- The expression $A \vee \neg A$ is a tautology.
- This means the expression is always true, regardless of the value of A
- A is a tautology is written as $\mathbb{A}$
- A tautology is true under any interpretation
- An expression which is false under any interpretation is contradictory

A	$A \vee \neg A$
false	true
true	true

Properties of Logical Systems

- Completeness: Every tautology is a theorem
- Soundness: Every theorem is valid
- Decidability: An algorithm exists that will determine if a well-formed formula is valid
- Monotonicity: A valid logical proof cannot be made invalid by adding additional premises or assumptions

Logical Equivalence

- Two expressions are equivalent if they always have the same logical value under any interpretation:
  - $A \wedge B \equiv B \wedge A$
- Equivalences can be proven by examining truth tables
- Two sentences are logically equivalent iff they are true in the same models (knowledge base):
  - $\alpha \equiv ß$ iff $\alpha \models \beta$ and $\beta \models \alpha$

Logical Equivalence

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$
$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$
$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$
$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{de Morgan}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{de Morgan}$$
$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

Propositional Logic

- A proposition is a statement that is either true or false, given some state of the world
- Propositional logic is a logical system that deals with propositions
- Propositional calculus is the language we use to reason about propositional logic
- A legal sentence in propositional logic is called a well-formed formula (wff)

Propositional Logic

The following are wff's:

P, Q, R…

true, false

(A)

$\neg A$

$A \wedge B$

$A \vee B$

$A \rightarrow B$

$A \Leftrightarrow B$

Propositional Logic: Syntax

- Propositional logic is the simplest logic
  - It illustrates basic ideas
- Rules for constructing legal sentences (well-formed formulae) in propositional logic (the proposition symbols S_1 and S_2 are sentences):
  - If S is a sentence, ¬S is also a sentence (negation)
  - If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
  - If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
  - If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional Logic: Semantics

A specific model is an assignment of true or false to each proposition symbol.

For example, assume A , B, and C are statements in propositional logic. With these 3 symbols, $2^3 = 8$ possible models can be enumerated automatically

One possible model assigns each statement a specific value:

A = false B = true C = false

A simple recursive process can now evaluate a sentence:

¬A ∧ (B ∨ C) = *true* ∧ (*true* ∨ *false*) = *true* ∧ *true* = *true*

Deduction

- Deduction is the process of deriving a conclusion from a set of assumptions
- If we deduce a conclusion C from a set of assumptions (facts), we write:

$$\{A_1, A_2, \ldots, A_n\} \vdash C$$

- To derive a conclusion from a set of assumptions, we apply a set of *inference rules*
- To distinguish an inference rule from a set of assumptions, we often write $A \vdash B$ as $\frac{A}{B}$

Inference Rules

- ¬ ¬ Elimination : if we have a sentence that is negated twice, we can conclude the sentence itself, without the negation:

$$\frac{\neg\ \neg\ A}{A}$$

Inference Rules

- And-Introduction (Conjunction): given sentences A and B, we can deduce A ∧ B:

  $$\frac{A,\ B}{A \wedge B}$$

- And-Elimination (Simplification): given A ∧ B, we can deduce A and we can deduce B separately:

  $$\frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B}$$

Inference Rules

- Or-Introduction (Addition): given sentence A, we can deduce the disjunction of A with *any* other sentence:

  $$\frac{A}{A \vee B}$$

- Modus Ponens (M.P.): given sentence A and the fact that A implies B, we can derive sentence B:

  $$\frac{A \rightarrow B,\ A}{B}$$

Inference Rules

- Hypothetical Syllogism (H.S.)

  $A \rightarrow B \wedge B \rightarrow C$

  $A \rightarrow C$

- Disjunctive Syllogism (D.S.)

  $A \vee B, \neg A$

  B

Inference Rules

- $\rightarrow$ Introduction: if, in carrying out a proof, we start from assumption A and derive a conclusion C, then we can conclude that $A \rightarrow C$:

$$\frac{A \ldots C}{A \rightarrow C}$$

Indirect Proof

- Reductio Ad Absurdum: if we assume A is incorrect (negate A) and this leads to a contradiction, then we can conclude that A is correct (proof by contradiction):

$$\frac{\neg A \quad \ldots \quad \bot}{A}$$

$\bot$ is called *falsum*

Careful!

- An invalid argument that looks similar to M.P. is as follows:

  $\underline{A \rightarrow B\ ,\ B}$

  A

- This is known as the "Fallacy of Affirming the Consequent"

Deduction Example 1

- Prove the following: $\{A, \neg A\} \vdash \bot$

First, note that , $\neg A \equiv (A \rightarrow \bot)$ This can be seen by comparing the truth tables for $\neg A$ and for $A \rightarrow \bot$. Hence we can take as our set of assumptions $\{A, A \rightarrow \bot\}$. Thus, our proof using modus ponens is as follows:

$$\frac{A \quad , \quad A \rightarrow \bot}{\bot}$$

Deduction Example 2

- Prove the following: $\{A \wedge B\} \vdash A \vee B$

$$\frac{A \wedge B}{A} \text{ by And-Elimination}$$

$$A \vee B \text{ by Or-Introduction}$$

Deduction Example 3

- Prove the following:

$\{\neg A, \neg A \rightarrow B, \neg B\} \vdash (\neg A \rightarrow B) \rightarrow (\neg B \rightarrow A)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg A \quad \neg A \rightarrow B}{B} \quad \neg B}{B} \quad B \rightarrow \bot}{\bot}}{A}}{\neg B \rightarrow A}}{(\neg A \rightarrow B) \rightarrow (\neg B \rightarrow A)}$$

assumptions
modus ponens
rewriting $\neg B$
modus ponens
reductio ad absurdum
$\rightarrow$ introduction
$\rightarrow$ introduction

331 – Intro to Intelligent Systems
Week07b
First-Order Predicate Logic
R&N Chapter 8

T.J. Borrelli

First-Order (Predicate) Logic

- Propositional logic assumes the world contains facts
- Predicate logic (like natural language) assumes the world contains:
  - Objects: people, houses, numbers, colors, baseball games, wars, …
  - Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  - Functions: father of, best friend, one more than, plus, …

Why FOL?

- Why do we need First-order logic?
- Propositional logic is too weak a language to represent knowledge of complex environments in a concise way
- We already saw this to some degree with the Wumpus world example – we would need many Rules (R_n) in order to make valid conclusions
- FOL has sufficient expressive power to deal with partial information

FOL

- We adopt the foundation of propositional logic:
- Declarative – knowledge and inference are separate, and inference is entirely domain independent
- Compositionality – the meaning of a sentence is a function of the meaning of its parts.
- Context-independent and unambiguous
- FOL can easily express facts about **some** or **all** objects in the universe

Logics

- **Ontological commitment – what the logic assumes about the nature of reality

- **Epistemological commitments – the possible states of knowledge that it allows with respect to each fact
- ** more on this if you put on exam.

Propositional Logic VS FOL

Logic	Ontological commitment (what exists in the world)	Epistemological Commitment (what an agent believes about facts)
Propositional Logic	facts	True/false/unknown
First-order Logic	Facts, objects, relations	True/false/unknown

Syntax of Predicate Calculus

Symbols represent constants, variables, functions, or predicates:

- Constants begin with a lower-case letter
  - bill, john, etc.
  - The constants ‘true’ and ‘false’ are reserved as truth symbols
- Variables begin with an upper-case letter
  - For example, X, is a variable that can represent *any* constant
- Functions have input and produce an output
  - plus(5,4)
  - The *arity* of a function is the number of arguments
  - For example, ‘plus’ has an arity of 2
- Predicates are similar to functions, but return true
  - A predicate names a relationship between objects in the world
  - For example, father(bill, john) means that “bill is the father of john”
  - father(bill, X) refers to any child of bill

Semantics of Predicate Calculus

- Quantification of variables is important
- When a variable appears in a sentence, that variable serves as a *placeholder*
  - For example, the “X” in likes(george, X)
  - Any constant allowed under the interpretation can be substituted for it in the expression
  - Substituting “kate” or “susie” for “X” forms the statement likes(george, kate) or likes(george, susie)

Sentences in Predicate Calculus

- Sentences are created by combining predicates using logical operators (the same as used for propositional calculus) and *quantifiers*
- Examples:

  likes(george, kate) ∧ friend(X, george)

  ¬helps(X, X)

  has_children(ben, plus(2, 3))

  friend(father_of(david, X), father_of(andrew, Y))

- *likes*, *friend*, *helps* and *father_of* and *has_children* are predicates
- *plus* is a function

Sentences in Predicate Calculus

- When a variable appears in a sentence, it refers to unspecified objects in the domain. First order predicate calculus includes two additional symbols, the variable quantifiers ∀ and ∃, that constrain the meaning of a sentence:

  ∃Y friend(Y, peter)

  ∀X likes(X, ice_cream)

Quantifiers ∀ and ∃

- ∀ The universal quantifier

  ∀X p(X) is read "For all X, p(X) is true"

- ∃ The existential quantifier

  ∃X p(X) is read "There exists an X such that p(X) is true"

- Relationship between the quantifiers:

  $\exists X\ p(X) \equiv \neg(\forall X)\neg p(X)$

  "There exists an X for which p(X) is true" is equivalent to "it is not true that for all X p(X) does not hold"

Universal Quantification

- Typically, → is the main connective with ∀

  Example: "Everyone at RIT is smart":

  $$\forall X \; at(X, rit) \rightarrow smart(X)$$

- Common mistake: using ∧ as the main connective with ∀:

  $$\forall X \; at(X, rit) \wedge smart(X)$$

  means "Everyone is at RIT and everyone is smart"

Existential Quantification

- Typically, $\wedge$ is the main connective with $\exists$

  Example: "Someone at RIT is smart":

  $$\exists X \text{ at}(X, \text{rit}) \wedge \text{smart}(X)$$

- Common mistake: using $\rightarrow$ as the main connective with $\exists$:

  $$\exists X \text{ at}(X, \text{rit}) \rightarrow \text{smart}(X)$$

  This means:

  "If RIT has a student then that student is smart."

Properties of Quantifiers

- ∀X ∀Y is the same as ∀Y ∀X
- ∃X ∃Y is the same as ∃Y ∃X
- ∃X ∀Y is not the same as ∀Y ∃X
- ∃X ∀Y loves(X,Y)
  - "There is a person who loves everyone"
- ∀Y ∃X loves(X,Y)
  - "Everyone is loved by someone"
- Quantifier duality: each can be expressed using the other:
  - ∀X likes(X, iceCream)= ¬∃X ¬likes(X, iceCream)
  - ∃X likes(X, broccoli) = ¬∀X ¬likes(X, broccoli)

Equality

- $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object

- For example, the definition of *sibling* in terms of *parent* is:

$$\forall X,Y\ (\neg(X = Y) \land \exists M,F\ \neg(M = F) \land parent(M,X) \land parent(F,X) \land parent(M,Y) \land parent(F,Y) \rightarrow sibling(X,Y))$$

Using First-Order Logic

First-order rules for the kinship domain:

- A brother is a male sibling
- $\forall X,Y\ (male(X) \wedge sibling(X,Y) \rightarrow brother(X,Y))$

- "sibling" is symmetric
- $\forall X,Y\ (sibling(Y,X) \rightarrow sibling(X,Y))$

- One's mother is one's female parent
- $\forall M,C\ (female(M) \wedge parent(M,C) \rightarrow mother(M,C))$

Using First-Order Logic

For example, given the following facts:

mother(eve,abel)
mother(eve,cain)
father(adam,abel)
father(adam,cain)

And the following rules:

$\forall X,Y\ (father(X,Y) \vee mother(X,Y) \rightarrow parent(X,Y))$
$\forall X,Y,Z\ (parent(X,Y) \wedge parent(X,Z) \rightarrow sibling(Y,Z))$

One can conclude that cain and abel are siblings (or half-siblings)

Examples of First-Order Predicates

- “If it doesn’t rain on Monday, Tom will go to the mountains.”

  ¬weather(rain, monday) → go(tom, mountains)
- “Emma is a Doberman pinscher and a good dog.”

  is_a(emma,doberman) ∧ good_dog(emma)
- “All basketball players are tall.”

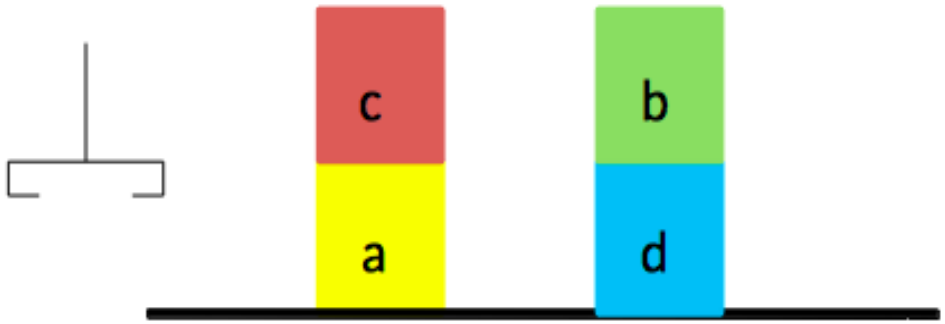
  ∀ X (basketball_player(X) → tall(X))
- “Some people like anchovies.”

  ∃ X (person(X) ∧ likes(X, anchovies))
- “If wishes were horses, beggars would ride.”

  equal(wishes, horses) → ride(beggars)
- “Nobody likes taxes.”

  ¬∃ X likes(X, taxes)

Blocks World Example



A collection of logical clauses describes the important properties and relationships in a Blocks World:

on(c,a)
on(b,d)
onTable(a)
onTable(d)
clear(b)
clear(c)

Suppose you want to define a test to determine whether **all** blocks are clear (have nothing stacked on top of them):

$\forall X (\neg \exists Y \text{ on } (Y, X)) \rightarrow \text{clear}(X)$.

"For all X, if there does not exist a Y such that Y is on X, then X is clear."

331 - Intro to Intelligent Systems
Week07c
Inference
R&N Chapter 9

T.J. Borrelli

Resolution in Propositional Logic

- Deductive reasoning can be used to make proofs in propositional and predicate logic
- It is not clear how this process can be automated because at each stage some initiative is required to choose the right next step
- *Resolution* is a proof method that can be automated because it involves a fixed set of steps
- Resolution requires that the problem be expressed in *conjunctive normal form*

Conjunctive Normal Form

- A well-formed formula (wff) is in *conjunctive normal form* (CNF) if it is a conjunction of disjunctions:
  - $X_1 \wedge X_2 \wedge X_3 \wedge \ldots \wedge X_n$

  where each clause, $X_{i,}$ is of the form:
  - $Y_1 \vee Y_2 \vee Y_3 \vee \ldots \vee Y_n$
    - The Ys are literals
- For example, $A \wedge (B \vee C) \wedge (\neg A \vee \neg B \vee \neg C \vee D)$
- Similarly, a wff is in *disjunctive normal form* (DNF) if it is a disjunction of conjunctions.
- For example, $A \vee (B \wedge C) \vee (\neg A \wedge \neg B \wedge \neg C \wedge D)$

Conversion to CNF

- Any wff can be converted to CNF by using the following equivalences:

(1) $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$

(2) $A \rightarrow B \equiv \neg A \vee B$

(3) $\neg(A \wedge B) \equiv \neg A \vee \neg B$

(4) $\neg(A \vee B) \equiv \neg A \wedge \neg B$

(5) $\neg\neg A \equiv A$

(6) $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

- Importantly, this can be converted into an algorithm – this will be useful for *automating resolution*

Example 1 Conversion to CNF

- Convert $(A \rightarrow B) \rightarrow C$ to CNF:

  $\neg(A \rightarrow B) \vee C$

  $\neg(\neg A \vee B) \vee C$

  $(A \wedge \neg B) \vee C$

  $(A \vee C) \wedge (\neg B \vee C)$

- Now express the CNF form as a set of clauses:

  $\{(A, C), (\neg B, C)\}$

Example 2 Conversion to CNF

- Convert $A \leftrightarrow (B \wedge C)$:

  $(A \rightarrow (B \wedge C)) \wedge ((B \wedge C) \rightarrow A)$

  $(\neg A \vee (B \wedge C)) \wedge (\neg(B \wedge C) \vee A)$

  $(\neg A \vee (B \wedge C)) \wedge (\neg B \vee \neg C \vee A)$

  $(\neg A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee \neg C \vee A)$

- Now express the CNF form as a set of clauses:

  $\{(\neg A, B), (\neg A, C), (\neg B, \neg C, A)\}$

The Resolution Rule

- The resolution rule is written as follows:

$$\frac{A \vee B \qquad \neg B \vee C}{A \vee C}$$

- This tells us that if we have two clauses that have a literal and its negation, we can combine them by removing that literal
- For example, if we have {(A, B), (¬B, C)} we would apply resolution to get {(A, C)}

The Resolution Rule

- Example: {(A,B,C), D, (¬A,D,E),(¬D,F)} can be resolved to {(B,C,D,E), D, (¬D,F)} which can be further resolved to {(B,C,D,E), F} or {(B,C,E,F), D}
- Note that at the first step, we also had a choice and could have resolved to {(A,B,C),D,(¬A,E,F)} which can be further resolved to {(B,C,E,F), D}

The Resolution Rule

- If wff P resolves to wff Q, we write $P \models Q$
- For example, resolve $(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee C)$:

  $\{(A,B), (\neg A,C), (\neg B,C)\}$

  $\{(B,C), (\neg B,C)\}$

  $\{C\}$
- We can express this as

  $(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee C) \models C$
- The *resolvent* is a logical consequence of the clauses

Resolution Refutation

- Let us resolve: {(¬A,B), (¬A,¬B,C), A, ¬C}
- We begin by resolving the first clause with the second clause, thus eliminating B and ¬B:

  {(¬A,C), A, ¬ C}

  {C, ¬C}
- Now we can resolve both remaining literals, which gives falsum (a contradiction):

  ⊥
- If we reach falsum, we have proved that our initial set of clauses were inconsistent
- This is written:

  {(¬A,B),(¬A,¬B,C),A,¬C} ⊨ ⊥

Proof by Refutation

- If we want to prove that a logical argument is valid, we negate its conclusion, convert it to clause form, and then try to derive falsum using resolution
- If we derive falsum, then our clauses were inconsistent, meaning the original argument was valid, since we negated its conclusion

Example 1 of Proof by Refutation

- Prove the following 3 statements by refutation:

  $(A \wedge \neg B) \rightarrow C$

  $A \wedge \neg B$

  $\therefore C$

- Negate the conclusion and convert to clauses:

  $\{(\neg A, B, C),\ A,\ \neg B,\ \neg C\}$

- Now resolve:

  $\{(B, C),\ \neg B,\ \neg C\}$

  $\{C,\ \neg C\}$

  $\bot$

- We have reached falsum, so our original argument was valid

Example 2 of Proof by Refutation

- Prove or disprove the following:

  $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E \vee F$

  $\therefore A \rightarrow F$

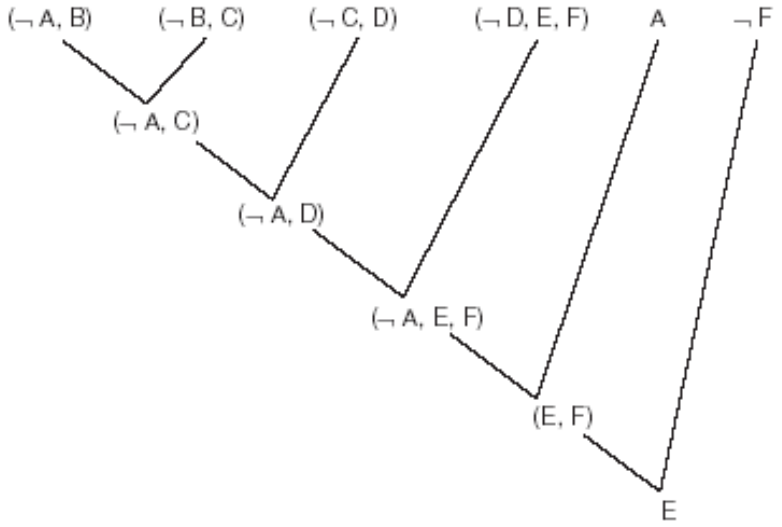
- First, we negate the conclusion to give $\neg(A \rightarrow F)$
- Next we convert to clause form:

  $D \rightarrow E \vee F \equiv \neg D \vee (E \vee F)$ and $\neg(A \rightarrow F) \equiv \neg (\neg A \vee F)$
  $\equiv A \wedge \neg F$

- So our clauses are: $\{(\neg A,B), (\neg B,C), (\neg C,D), (\neg D,E,F), A, \neg F\}$
- This will resolve to $\{E\}$; we cannot reach falsum. Hence, we can conclude that our original conclusion was not valid. (You can prove this for yourself by using a truth table.)

Refutation Proof in Tree Form

- It is often helpful to represent a refutation proof in tree form:



- In this case, the proof has failed, as we are left with {E} instead of falsum

Example: Graph Coloring

- Resolution refutation can be used to determine if a solution exists for a particular combinatorial problem
- For example, for graph coloring, we represent the assignment of colors to the nodes and the constraints regarding edges as propositions, and attempt to prove that the complete set of clauses is consistent
- This does not tell us how to color the graph, simply that it is possible

Example: Graph Coloring

Available colors are r (red), g (green) and b (blue).

A_r means that node A has been colored red.

Each node must have exactly one color:

$A_r \vee A_g \vee A_b$

$\neg A_r \vee \neg A_g \quad (\equiv A_r \rightarrow \neg A_g)$

$\neg A_g \vee \neg A_b$

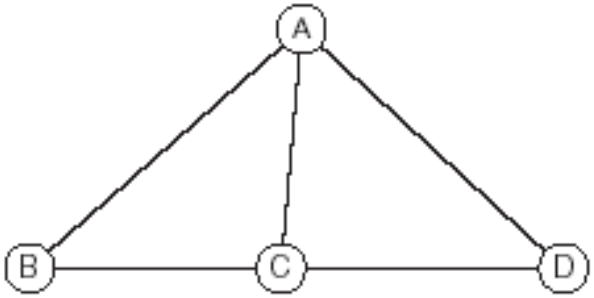
$\neg A_b \vee \neg A_r$

If (A,B) is an edge, then:

$\neg A_r \vee \neg B_r$

$\neg A_b \vee \neg B_b$

$\neg A_g \vee \neg B_g$



Now we construct the complete set of clauses for our graph, and try to see if they are consistent, using resolution

Resolution in Propositional Logic: Truth-Tellers and Liars

Suppose that liars always speak what is false, and truth-tellers always speak what is true. Further, suppose that Amy, Bob, and Cal are each either a liar or a truth-teller. Amy says, “Bob is a liar.” Bob says, “Cal is a liar.” Cal says, “Amy and Bob are liars.” Which, if any, of these people are truth-tellers?

Resolution in Propositional Logic: Truth-Tellers and Liars

The atomic sentences for this problem are:

A, for “Amy is a truth-teller.”

B, for “Bob is a truth-teller.”

C, for “Cal is a truth-teller.”

A literal is an atomic sentence or its negation. For example, A, ¬A, B, ¬B, C, ¬C are all literals.

A sentence is either an atomic sentence or a complex sentence.

Resolution in Propositional Logic: Truth-Tellers and Liars

The set of sentences that represent our knowledge is called our *knowledge base*. The knowledge base for this problem is:

$\{A \leftrightarrow \neg B, B \leftrightarrow \neg C, C \leftrightarrow (\neg A \wedge \neg B)\}$

(Amy is a truth-teller if and only if Bob is not a truth-teller)

Note that these sentences could all be combined with conjunction into a single sentence which expresses all knowledge of the knowledge base:

$(A \leftrightarrow \neg B) \wedge (B \leftrightarrow \neg C) \wedge (C \leftrightarrow (\neg A \wedge \neg B))$

Resolution in Propositional Logic: Truth-Tellers and Liars

Convert the knowledge base to CNF:

1. Re-write {A ↔ ⌐B, B ↔ ⌐C, C ↔ (⌐A ∧ ⌐B)} as

{(A→⌐B), (⌐B→A), (B→⌐C), (⌐C→B), (C→(⌐A∧⌐B)), ((⌐A∧⌐B)→C)}

2. Eliminate →

{⌐A ∨ ⌐B, ⌐⌐B ∨ A, ⌐B ∨ ⌐C, ⌐⌐C ∨ B, ⌐C ∨ (⌐A∧⌐B), ⌐(⌐A∧⌐B) ∨ C}

3. Apply DeMorgan and eliminate ⌐⌐

{⌐A ∨ ⌐B, B ∨ A, ⌐B ∨ ⌐C, C ∨ B, ⌐C ∨ (⌐A∧⌐B), A ∨ B ∨ C}

4. Distribute ∨ over ∧

{⌐A ∨ ⌐B, B ∨ A, ⌐B ∨ ⌐C, C ∨ B, ⌐C ∨ ⌐A, ⌐C ∨ ⌐B, A ∨ B ∨ C}

Resolution in Propositional Logic: Truth-Tellers and Liars

Thus, the knowledge base consists of:

{(¬A,¬B), (B,A), (¬B,¬C), (C,B), (¬C,¬A), (¬C,¬B), (A,B,C)}

It is important to note that a model in CNF is a truth assignment that makes *at least one literal in each clause true*.

Resolution in Propositional Logic: Truth-Tellers and Liars

Consider the two clauses (B,A) and (¬B,¬C) from the knowledge base. The first reads, “Bob is a truth-teller or Amy is a truth-teller.” The second reads, “Bob is not a truth-teller or Cal is not a truth-teller.” Consider what happens according to Bob’s truthfulness:

- **Bob is a truth-teller.** In this case, the first clause is satisfied. The second clause is satisfied if and only if Cal is not a truth-teller
- **Bob is not a truth-teller.** In this case, the second clause is satisfied. The first clause is satisfied if and only if Amy is a truth-teller

Resolution in Propositional Logic: Truth-Tellers and Liars

- Since one case or the other holds, we know that in any model of *both* clauses Amy is a truth-teller or Cal is not a truth-teller
- In other words, from clauses (B,A) and (¬B,¬C), we can derive the clause (A,¬C) and add it to our knowledge base
- This is a specific application of the Resolution Rule

Resolution in Propositional Logic: Truth-Tellers and Liars

- Note that not all possible resolution rule derivations are useful
- Consider what happens when we apply the resolution rule to the first two clauses of the knowledge base
- From (¬A,¬B) and (B,A) we can derive either (¬A,A) or (¬B,B) depending on which atomic sentence we use for the resolution
- In either case we derive a tautology

Resolution in Propositional Logic: Truth-Tellers and Liars

Let's use proof by contradiction to prove that Cal is a liar (¬C). In addition to the knowledge base, we assume the negation of what we wish to prove (¬¬C, that is, C).

	(¬A, ¬B)		Knowledge base
	(B, A)		
	(¬B, ¬C)		
	(C, B)		
	(¬C, ¬A)		
	(¬C, ¬B)		
	(A, B, C)		
	(C)		Assumed negation
	(¬A)	(5, 8)	Resolution Rule
	(B)	(2, 9)	Resolution Rule
	(¬C)	(3, 10)	Resolution Rule
(12)	⊥	(8, 11)	Contradiction!

Resolution in Propositional Logic: Truth-Tellers and Liars

- Recall that at least one literal of each clause must be true for a truth assignment to be a model. The last clause (the empty clause) has no literals at all and represents a clear contradiction. It cannot be the case that C is true as we had assumed. Thus ¬C logically follows from our knowledge base.
- It should be noted that a contradictory knowledge base can derive an empty clause without any need of assumptions. One can thus prove any sentence using proof by contradiction, starting with a contradictory knowledge base.

Resolution in Propositional Logic: Truth-Tellers and Liars

- When we derive a sentence s_2 from s_1, we denote it as $s_1 \vdash s_2$
- A proof procedure that derives only what is entailed is called sound
- A proof procedure that can derive anything that is entailed is called complete
- Resolution theorem proving is both sound and complete

Improving Performance

- Reasoning performance can be greatly improved by changing the knowledge base
  - A proof is essentially a search through a maze of possible resolutions to a contradiction, so compacting the knowledge base can greatly simplify search
- Reasoning engines often have a pre-processing stage that eliminates unnecessary clauses: *equivalent clauses*, *subsumed clauses*, and *tautology clauses*
- Reasoning engines can also add new knowledge produced through reasoning to speed future reasoning

Improving Performance

(¬A, ¬B)
(B, A)
(¬B, ¬C)
(C, B)
(¬C, ¬A)
(¬C, ¬B)
(A, B, C)

Notice that (3) and (6) are logically equivalent.
Also, note that (2) entails (7). This means that (2) *subsumes* (7), and we can eliminate (7).
Delete (6) and (7) from the knowledge base.

Improving Performance

Using our previous example with the unnecessary clauses eliminated, let's try to prove that Amy is a liar:

	(¬A, ¬B)		Knowledge base
	(B, A)		
	(¬B, ¬C)		
	(C, B)		
	(¬C, ¬A)		
	(A)		Assumed negation
	(¬B)	(1, 6)	Resolution Rule
	(C)	(4, 7)	Resolution Rule
	(¬A)	(5, 8)	Resolution Rule
(10)	⊥	(6, 9)	Contradiction!

Once we have proven the contradiction, we can add ¬A to our knowledge base. Furthermore, we can eliminate all clauses that are subsumed by ¬A.

Improving Performance

After adding ⌐A and deleting subsumed clause (⌐A, ⌐B) and (⌐C, ⌐A), let's try to prove that Bob is a truth-teller:

	(1)	(B, A)		Knowledge base
	(2)	(⌐B, ⌐C)		
	(3)	(C, B)		
	(4)	(⌐A)		
	(5)	(⌐B)		Assumed negation
	(6)	(A)	(1, 5)	Resolution rule
	(7)	⊥	(4, 6)	Contradiction!

Improving Performance

With the new knowledge that Bob is a truth-teller, we can again remove subsumed clauses (B,A) and (C,B) and prove that Cal is a liar:

	(1)	(⌐B, ⌐C)		Knowledge base
	(2)	(⌐A)		
	(3)	(B)		
	(4)	(C)		Assumed negation
	(5)	(⌐B)	(1, 4)	Resolution rule
	(6)	⊥	(3, 5)	Contradiction!

Improving Performance

New knowledge that Cal is a liar subsumes clause (1) so our knowledge base is now:

(1)	(¬A)	Knowledge base
(2)	(B)	
(3)	(¬C)	

This is the answer to the original question: Amy is a liar, Bob is a truth-teller, and Cal is a liar.

Improving Performance

Consider this: The number of possible (non-tautology) clauses of length l for *a* atomic sentences is $2^l \binom{a}{l}$

a	= 1	= 2	= 3	= 4	= 5
1	2	0	0	0	0
2	4	4	0	0	0
3	6	12	8	0	0
4	8	24	32	16	0
5	10	40	80	80	32
10	20	180	960	3360	8064
20	40	760	9120	77520	496128
40	80	3120	79040	1462240	21056256
80	150	12640	657260	25305280	769280512

Normal Forms in Predicate Calculus

- To allow inferences to be automated, the logical database must be expressed in an appropriate form
- To apply resolution to first-order predicate logic expressions, we first need to deal with the presence of the quantifiers ∀ and ∃

Normal Forms in Predicate Calculus

- The requirement is that all variables must be universally quantified
  - This allows full freedom in computing substitutions
- The method that is used is to move the quantifiers to the beginning of the expression, resulting in an expression that is in *prenex normal form*

Normal Forms in Predicate Calculus

When converting a wff in predicate calculus to Prenex Normal Form, we use the same rules as we used to convert a wff to CNF:

(1) $a(X) \leftrightarrow b(X) \equiv (a(X) \rightarrow b(X)) \wedge (b(X) \rightarrow a(X))$

(2) $a(X) \rightarrow b(X) \equiv \neg a(X) \vee b(X)$

(3) $\neg(a(X) \wedge b(X)) \equiv \neg a(X) \vee \neg b(X)$

(4) $\neg(a(X) \vee b(X)) \equiv \neg a(X) \wedge \neg b(X)$

(5) $\neg\neg a(X) \equiv a(X)$

(6) $a(X) \vee (b(X) \wedge c(X)) \equiv (a(X) \vee b(X)) \wedge (a(X) \vee c(X))$

Normal Forms in Predicate Calculus

In addition, we use the following rules to move the quantifiers to the front:

$\neg(\forall X)\ a(X) \equiv (\exists X)\ \neg a(X)$

$\neg(\exists X)\ a(X) \equiv (\forall X)\ \neg a(X)$

$(\forall X)\ a(X) \wedge b(Y) \equiv (\forall X)(a(X) \wedge b(Y))$

$(\forall X)\ a(X) \vee b(Y) \equiv (\forall X)\ (a(X) \vee b(Y))$

$(\exists X)\ a(X) \wedge b(Y) \equiv (\exists X)\ (a(X) \wedge b(Y))$

$(\exists X)\ a(X) \vee b(Y) \equiv (\exists X)\ (a(X) \vee b(Y))$

$(\forall X)\ a(X) \wedge (\forall Y)\ b(Y) \equiv (\forall X)\ (\forall Y)\ (a(X) \wedge b(Y))$

$(\forall X)\ a(X) \wedge (\exists Y)\ b(Y) \equiv (\forall X)\ (\exists Y)\ (a(X) \wedge b(Y))$

$(\exists X)\ a(X) \wedge (\forall Y)\ b(Y) \equiv (\exists X)\ (\forall Y)\ (a(X) \wedge b(Y))$

$(\exists X)\ a(X) \wedge (\exists Y)\ b(Y) \equiv (\exists X)\ (\exists Y)\ (a(X) \wedge b(Y))$

*Note that rules 9, 10, 11, and 12 can be used only if X does not occur in b

Example of Converting to Prenex NF

Convert to PNF: $(\forall X)\ (a(X) \rightarrow b(X)) \rightarrow (\exists Y)\ (a(Y) \wedge b(Y))$

$\neg(\forall X)\ (a(X) \rightarrow b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 2 (imp. elim)

$\neg(\forall X)\ (\neg a(X) \vee b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 2 (imp. elim.)

$(\exists X)\ \neg(\neg a(X) \vee b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 7 (DeM Quant.)

$(\exists X)\ (\neg\ \neg a(X) \wedge \neg b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 4 (DeM)

$(\exists X)\ (a(X) \wedge \neg b(X)) \vee (\exists Y)\ (a(Y) \wedge b(Y))$ Rule 5 (Double Neg)

$(\exists X)\ (\exists Y)\ \ ((a(X) \wedge \neg b(X)) \vee (a(Y) \wedge b(Y)))$ Rule 16 (logical equiv)

$(\exists X)\ (\exists Y)\ ((\mathbf{(a(X) \vee a(Y))} \wedge \mathbf{(a(X) \vee b(Y))} \wedge \mathbf{(\neg b(X) \vee a(Y))} \wedge \mathbf{(\neg b(X) \vee b(Y))}$

Rule 6 (distribute)

The resulting clauses are highlighted in bold

Skolemization

- Before resolution can be carried out on a wff, we need to eliminate all of the existential quantifiers (∃) from the wff
- This is done by replacing a variable that is existentially quantified by a constant:
  - For example, ∃(X) p(X) would be converted to p(c), where c is a constant that has not been used elsewhere in the wff
  - Although p(c) is not logically equivalent to ∃(X) p(X), we can make this substitution because we are only interested in seeing if a solution exists (i.e., assume there exists some X for which p(X) holds)

Skolemization

- Skolemization must proceed slightly differently in the case where the ∃ follows a ∀ as in: (∀X) (∃Y) (p(X,Y))
- In this case, rather than replacing Y with a skolem constant, we must replace it with a skolem function, such as in: (∀X) (p(X, f(X)))
- Note that the skolem function is a function of the variable that is universally quantified (X)
  - Once the existentially quantified variables have been removed from a logical database, unification may be used to match sentences in order to apply inference

Example 1 of Skolemization

Skolemize the following expression:

$(\forall X)\ (\exists Y)\ (\forall Z)\ (p(X) \wedge q(Y,Z))$

Solution:

$(\forall X)\ (\forall Z)\ (p(X) \wedge q(f(X),Z))$

Note that Y has been replaced by f(X) because ∃Y occurred *after* ∀X

Example 2 of Skolemization

Skolemize the following expression:

$(\forall W)\ (\forall X)\ (\exists Y)\ (\forall Z)\ (p(X) \wedge q(W, Y, Z))$

Solution: $(\forall W)\ (\forall X)\ (\forall Z)\ (p(X) \wedge q(W, f(W, X), Z))$

– Note that Y has been replaced by a function of both W and X, f(W, X) because $\exists Y$ occurred after $\forall W$ and $\forall X$

- To proceed with resolution, this wff must now be represented as a set of clauses
- To do this we first drop the universal quantifiers
- Hence, the expression above will be represented as: {(p(X)), (q(W, f(W,X), Z))}

Unification

- To resolve clauses we often need to make substitutions. For example:

  {(p(W,X)), (¬p(Y,Z))}

- To resolve, we need to substitute Y for W, and Z for X, in other words, {W/Y, X/Z}, giving:

  {(p(Y,Z)), (¬p(Y,Z))}

- These resolve to give falsum

Unification

For example, generate legal substitutions (bindings) of the expression “team(X, bob, friend(Y))” using the following unifications:

{X/code_monkeys, Y/Z} unifies the expression to:

team(code_monkeys, bob, friend(Z))

{X/W, Y/jack} unifies the expression to:

team(W, bob, friend(jack))

{X/Z, Y/cousin(Z)} unifies the expression to:

team(Z, bob, friend(cousin(Z)))

Unification

Issues with unification:

A constant is considered a “ground instance” and cannot be replaced

A variable cannot be unified with a term containing the variable

- For example, X cannot be replaced with p(X) as this creates an infinite expression: p(p(p(p(…X)))

Unifying substitutions must be made consistently across all occurrences within the scope of the variable in both expressions being matched

Unification

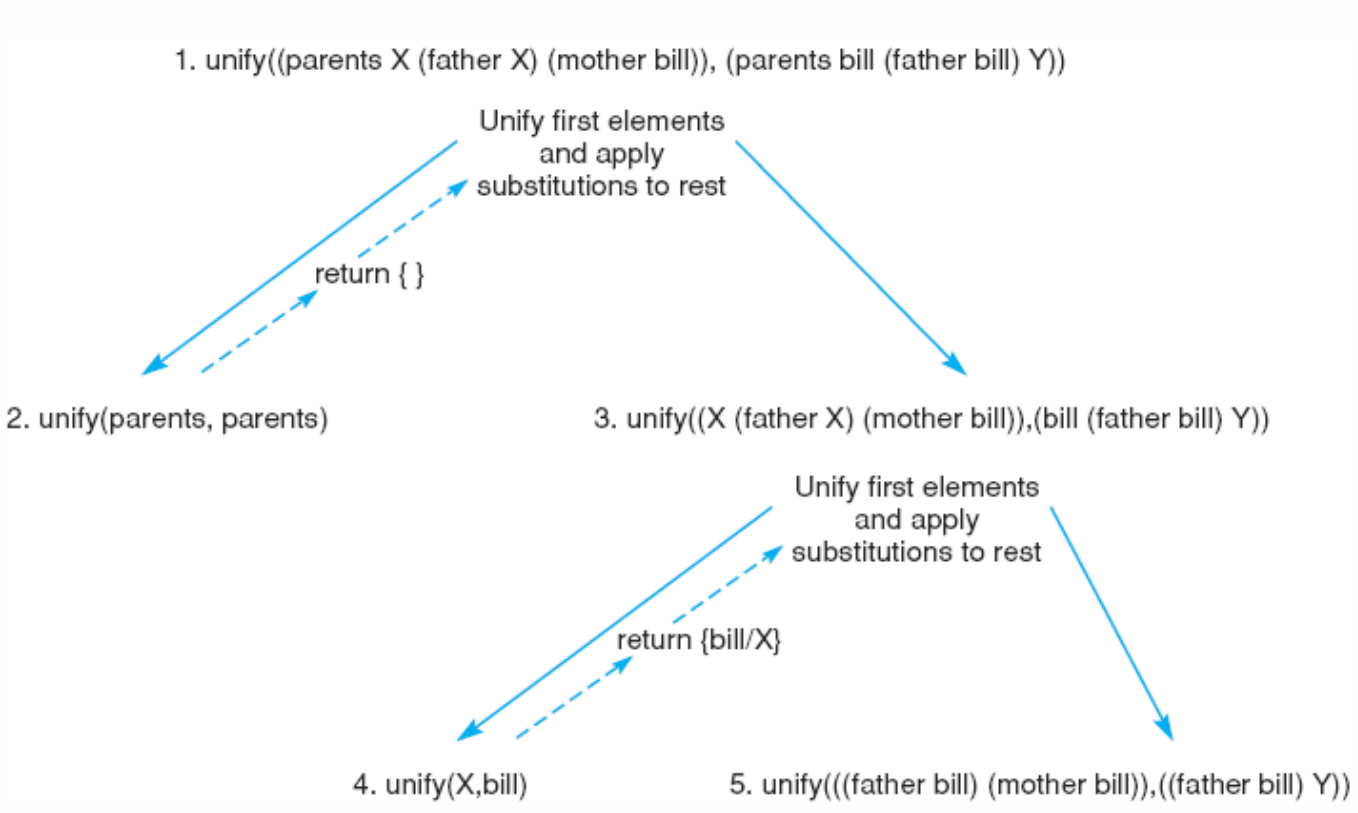
Once a variable is bound to a constant, that variable may not be given a new binding in a future unification

For example, suppose p(adam,X) unifies with the premise of p(Y,Z) → q(Y,Z) using the substitution {Y/adam, Z/X}, then modus ponens lets us infer q(adam,X) under the same substitution.

If we then match this result with the premise of q(W,bob) → r(W,bob), we must infer r(adam,bob).

Example of Unification

First 5 steps in the unification of:
(parents X (father X) (mother bill)) and (parents bill (father bill) Y)



Example of Unification

When unify is first called, because neither argument is an atomic symbol, the function will attempt to recursively unify the first elements of each expression, calling:

unify(parents, parents).

This unification succeeds, returning the empty substitution { }. Applying this to the remainder of the expression creates no change. The algorithm then calls:

unify((X (father X) (mother bill), (bill (father bill) Y)).

Example of Unification

In the second call to unify, neither expression is atomic, so the algorithm separates the expression into its first component and the remainder of the expression. This leads to the call:

unify(X, bill).

This call succeeds because both expressions are atomic and one of them is a variable. The call returns the substitution {bill/X}. This substitution is applied to the remainder of each expression and unify is called on the results: unify(((father bill) (mother bill)), ((father bill) Y)).

Example of Unification

The result of this call is to unify (father bill) with (father bill). This leads to the calls:

unify(father, father)
unify(bill, bill)
unify(( ), ( )).

All of these succeed, returning the empty set of substitutions. Unify is then called on the remainder of the expressions:
unify(((mother bill)), (Y)).

Example of Unification

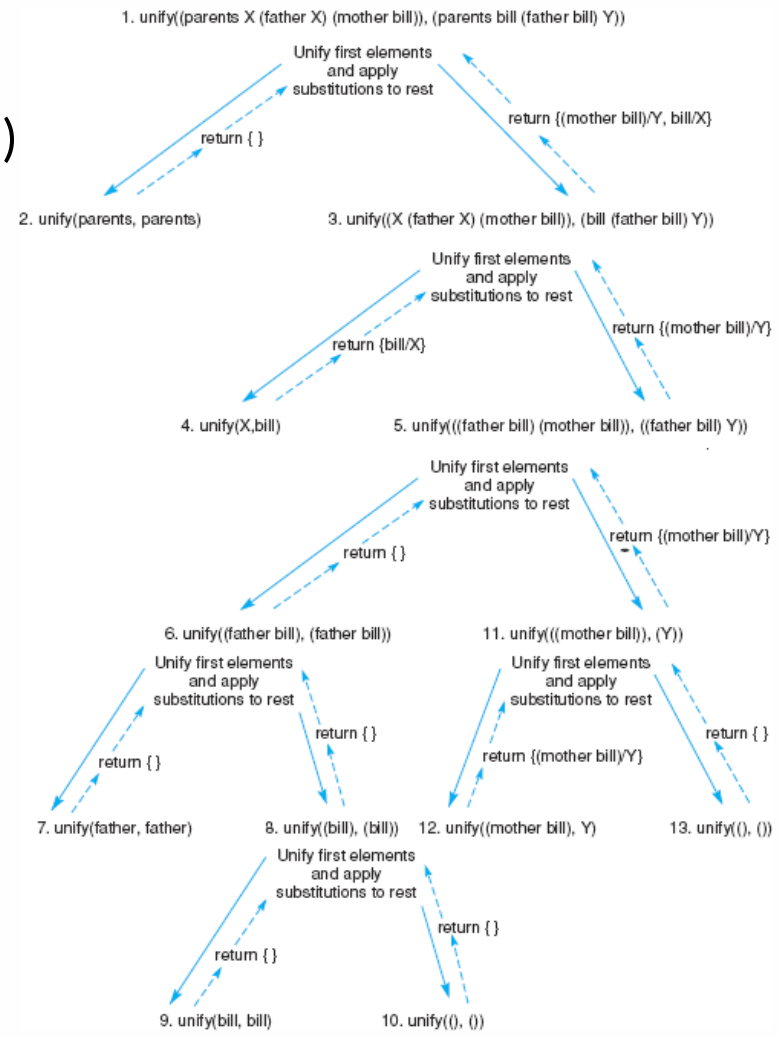
This in turn leads to calls:

unify((mother bill), Y) and
unify(( ), ( )).

In the first of these, (mother bill) unifies with Y. Notice that unification substitutes the whole structure (mother bill) for Y. Thus, unification succeeds and returns the substitution {(mother bill)/Y}. The call unify(( ), ( )) returns { }. All the substitutions are composed as each recursive call terminates, to return the answer {bill/X (mother bill)/Y}.

Example of Unification

Final trace of the unification of
(parents X (father X) (mother bill))
and
(parents bill (father bill) Y)



The Resolution Algorithm

First, negate the conclusion and add it to the list of assumptions

Now convert the assumptions into Prenex Normal Form (PNF)

Next, skolemize the resulting expression

Now convert the expression into a set of clauses

Now resolve the clauses using suitable unifiers

This algorithm means we can write programs that automatically prove theorems using resolution!

Resolution in Predicate Logic

Consider the following set of premises:

- Some children like any food
- No children like food that is green
- All children like food made by Cadbury's

We now wish to prove that the following conclusion follows from these premises:

- "No food made by Cadbury's is green"

Resolution in Predicate Logic

First, we need to represent the premises and the conclusion in predicate calculus:

c(X) means “X is a child”

f(X) means “X is food”

l(X, Y) means “X likes Y”

g(X) means “X is green”

m(X, Y) means “X makes Y”

c means “Cadbury’s”

Resolution in Predicate Logic

The premises can be represented as:

$(\exists X)\ (c(X) \wedge (\forall Y)\ (f(Y) \rightarrow l(X, Y)))$

$(\forall X)\ (c(X) \rightarrow (\forall Y)\ ((f(Y) \wedge g(Y)) \rightarrow \neg l(X, Y)))$

$(\forall X)\ ((f(X) \wedge m(c, X)) \rightarrow (\forall Y)\ (c(Y) \rightarrow l(Y, X)))$

Represent the conclusion as:

$(\forall X)\ ((f(X) \wedge m(c, X)) \rightarrow \neg g(X))$

Resolution in Predicate Logic

First, we must negate the conclusion and add it to the set of premises, which means we must now prove that the following expression cannot be satisfied:

$$(\exists X)\,(c(X) \wedge (\forall Y)\,(f(Y) \rightarrow l(X, Y))) \wedge$$
$$(\forall X)\,(c(X) \rightarrow (\forall Y)\,((f(Y) \wedge g(Y)) \rightarrow \neg l(X, Y))) \wedge$$
$$(\forall X)\,((f(X) \wedge m(c, X))) \rightarrow (\forall Y)\,(c(Y) \rightarrow l(Y, X))) \wedge$$
$$\neg((\forall X)\,((f(X) \wedge m(c, X)) \rightarrow \neg g(X)))$$

Next, convert this expression into a set of clauses.

Resolution in Predicate Logic

Expression 1: $(\exists X)\ (c(X) \wedge (\forall Y)\ (f(y) \rightarrow l(X, Y)))$

Eliminate $\rightarrow$: $(\exists X)\ (c(X) \wedge (\forall Y)\ (\neg f(Y) \vee l(X, Y)))$

Bring the quantifiers to front: $(\exists X)(\forall Y)\ (c(X) \wedge (\neg f(Y) \vee l(X, Y)))$

Skolemize: $(\forall Y)\ (c(a) \wedge (\neg f(Y) \vee l(a, Y)))$

Create set of clauses: $\{c(a), (\neg f(Y), l(a, Y))\}$

Resolution in Predicate Logic

Expression 2: $(\forall X)\,(c(X) \rightarrow (\forall Y)\,((f(Y) \wedge g(Y)) \rightarrow \neg l(X,Y)))$

Eliminate $\rightarrow$: $(\forall X)(\neg c(X) \vee (\forall Y)\,(\neg(f(Y) \wedge g(Y)) \vee \neg l(X, Y)))$

DeMorgan's law: $(\forall X)\,(\neg c(X) \vee (\forall Y)\,(\neg f(Y) \vee \neg g(Y) \vee \neg l(X,Y)))$

Quantifiers to front: $(\forall X)\,(\forall Y)(\neg c(X) \vee \neg f(Y) \vee \neg g(Y) \vee \neg l(X,Y))$

Create (single) clause: $\{(\neg c(X), \neg f(Y), \neg g(Y), \neg l(X,Y))\}$

Resolution in Predicate Logic

Expression 3: $(\forall X)\ ((f(X) \wedge m(c, X)) \rightarrow (\forall Y)\ (c(Y) \rightarrow l(Y, X)))$

Eliminate $\rightarrow$: $(\forall X)\ (\neg(f(X) \wedge m(c, X)) \vee (\forall Y)\ (\neg c(Y) \vee l(Y, X)))$

DeMorgan's law: $(\forall X)(\neg f(X) \vee \neg m(c, X) \vee (\forall Y)(\neg c(Y) \vee l(Y, X)))$

Quantifiers: $(\forall X)(\forall Y)(\neg f(X) \vee \neg m(c, X) \vee \neg c(Y) \vee l(Y, X))$

Single clause: $\{(\neg f(X), \neg m(c, X), \neg c(Y), l(Y, X))\}$

Resolution in Predicate Logic

Conclusion (which has been negated):

$\neg(\forall X)\ ((f(X) \wedge m(c, X)) \rightarrow \neg g(X))$

Eliminate $\rightarrow$: $\neg(\forall X)\ (\neg(f(X) \wedge m(c, X)) \vee \neg g(X))$

Move $\neg$ from front: $(\exists X)\ \neg(\neg(f(X) \wedge m(c, X)) \vee \neg g(X))$

DeMorgan's law: $(\exists X)\ (\neg\neg(f(X) \wedge m(c, X)) \wedge \neg\neg g(X))$

Remove $\neg\neg$: $(\exists X)\ (f(X) \wedge m(c, X) \wedge g(X))$

Skolemize: $f(b) \wedge m(c, b) \wedge g(b)$

Create set of clauses: $\{f(b), m(c, b), g(b)\}$

Now we have arrived at a set of clauses, upon which resolution can be applied

Resolution in Predicate Logic

The clauses we have are the following:

c(a)

(¬f(Y), l(a, Y))

(¬c(X), ¬f(Y), ¬g(Y), ¬l(X, Y))

(¬f(X), ¬m(c, X), ¬c(Y), l(Y, X))

f(b)

m(c, b)

g(b)

Resolution in Predicate Logic

We now apply resolution as follows:

First we unify clauses 1 and 3 using {X/a} and resolve to give:

(¬f(Y), ¬g(Y), ¬l(a, Y))

Similarly, the unifier {Y/b} can be applied to resolve clauses 2 and 5:

l(a, b)

Now we apply {Y/b} to resolve clause 5 with 8 to give:

(¬g(b), ¬l(a, b))

Next, clauses 9 and 10 can be resolved to give:

¬g(b)

Clause 11 can be resolved with 7 to give the empty set, or:

falsum

$\square$

Resolution in Predicate Logic

- We have proven that the set of clauses derived from the premises 1, 2, and 3, and the negation of the conclusion, are unsatisfiable
- Thus we have successfully proved that the conclusion does indeed follow from the premises, and so the argument is a valid one

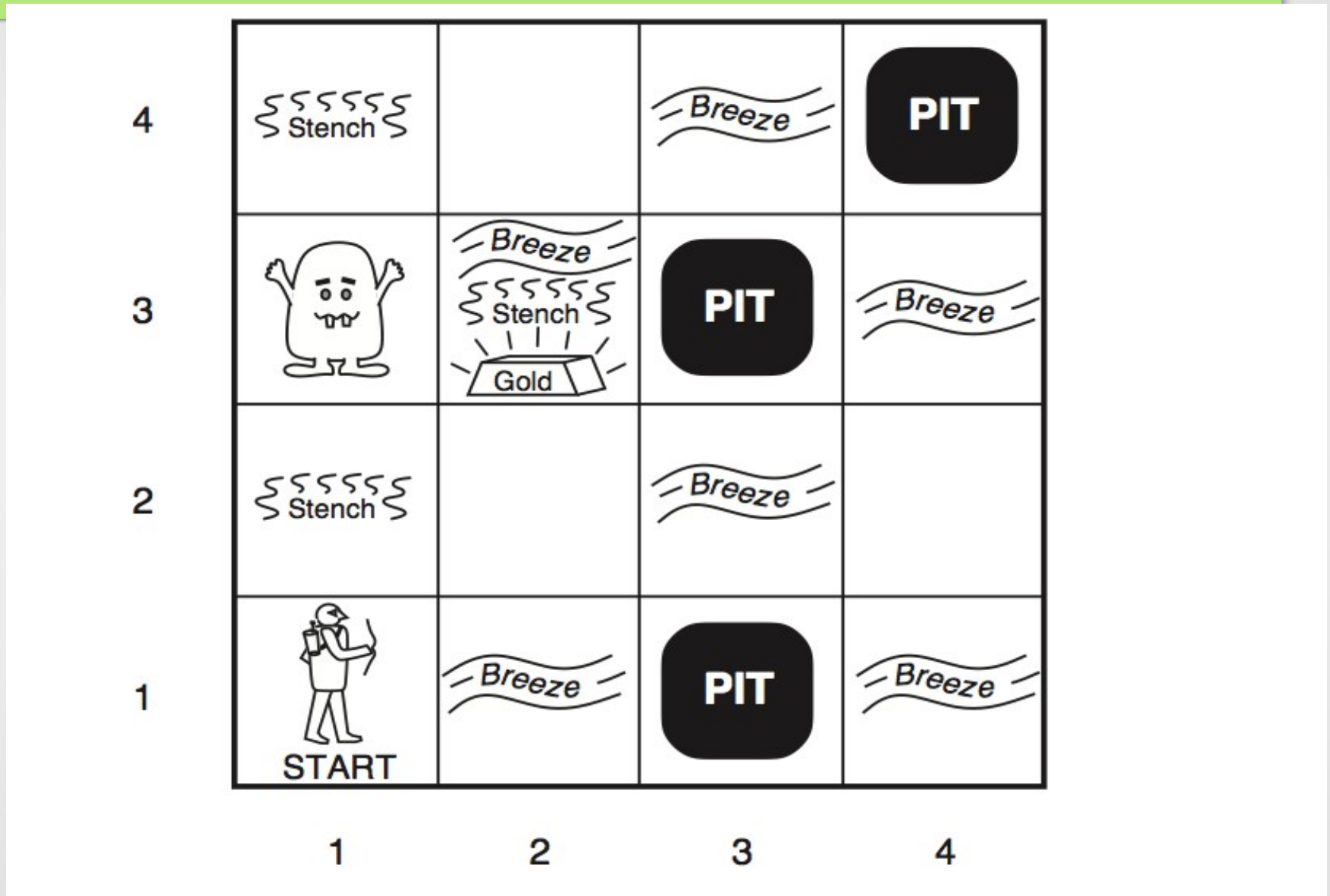
331 – Intro to Intelligent Systems
Week07
Logic
Wumpus World
R&N Chapter 7.2

T.J. Borrelli

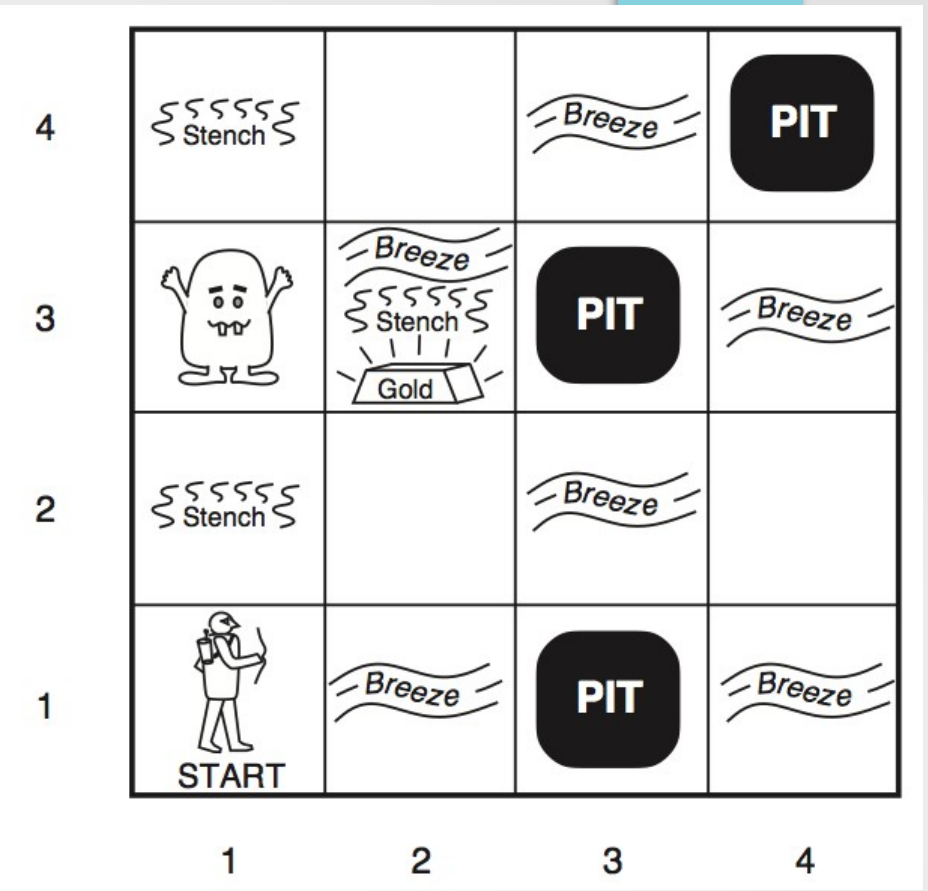
Wumpus World

- Wumpus world is a cave consisting of rooms connected by passageways
- Somewhere in the cave is a terrible beast (wumpus) that eats anyone who enters its room
- The wumpus can be shot by an agent but the agent only has 1 arrow
- Some rooms contain bottomless pits that will trap anyone who enters the room (except the wumpus)
- There is gold hidden somewhere in the environment

Wumpus World



Wumpus World



- Performance measure: Gold +1000;
  Death -1000; -1 per step;
  -10 for arrow

- Environment:
  Squares adj. to wumpus smelly
  Squares adj. to pit are breezy
  Glitter if gold on same square
  Shooting kills wumpus if facing it
  Shooting uses up only arrow
  Grabbing picks up gold if in same square
  Releasing drops gold in same square
  Dead if eaten by wumpus or fallen into pit

- Actions: Turn left, right; Forward; Grab; Release; Shoot

- Sensors: Stench; Breeze; Glitter; Bump(hit a wall); Scream (kill wumpus)

Wumpus World Properties

- Observable?
  - No (partially observable)
- Deterministic?
  - Yes
- Episodic?
  - No (sequential)
- Static?
  - Yes (wumpus doesn't move)
- Discrete?
  - Yes
- Single agent?
  - Yes (wumpus is a feature of environment)

Wumpus World – where do we start?

- Main challenge here is our initial ignorance of the configuration of the cave
- We have to make inferences based on what we know and the rules of the environment
- We know that the first square [1,1] is safe
- First, percept is [none, none, none, none, none]

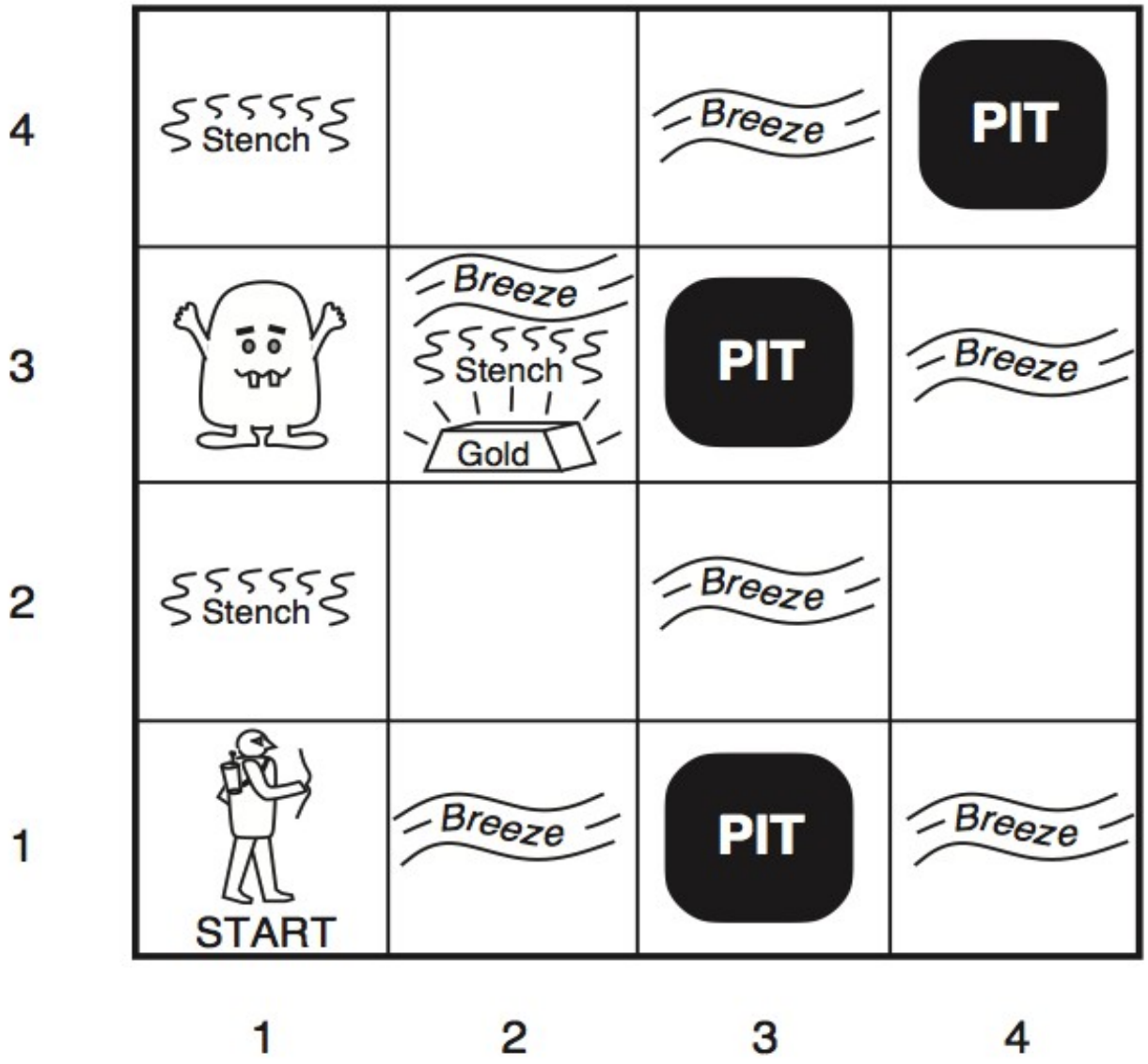
[stench, breeze, glitter, bump, scream]

- From this we can conclude that squares [1,2] and [2,1] are free of danger
- We can cautiously proceed to one of these squares.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 **OK**	2,2	3,2	4,2
1,1 **A** **OK**	2,1 **OK**	3,1	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

Wumpus World

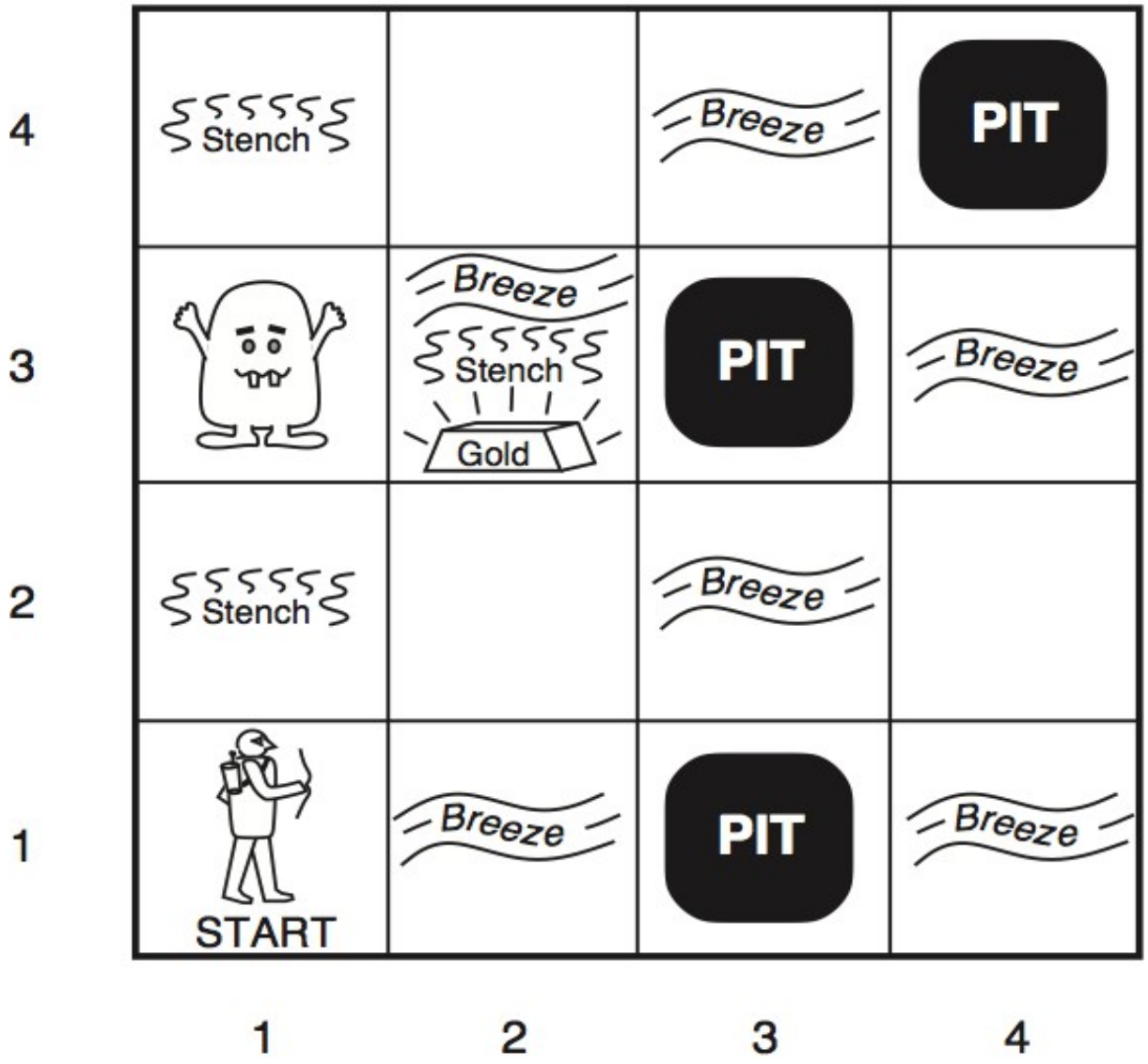


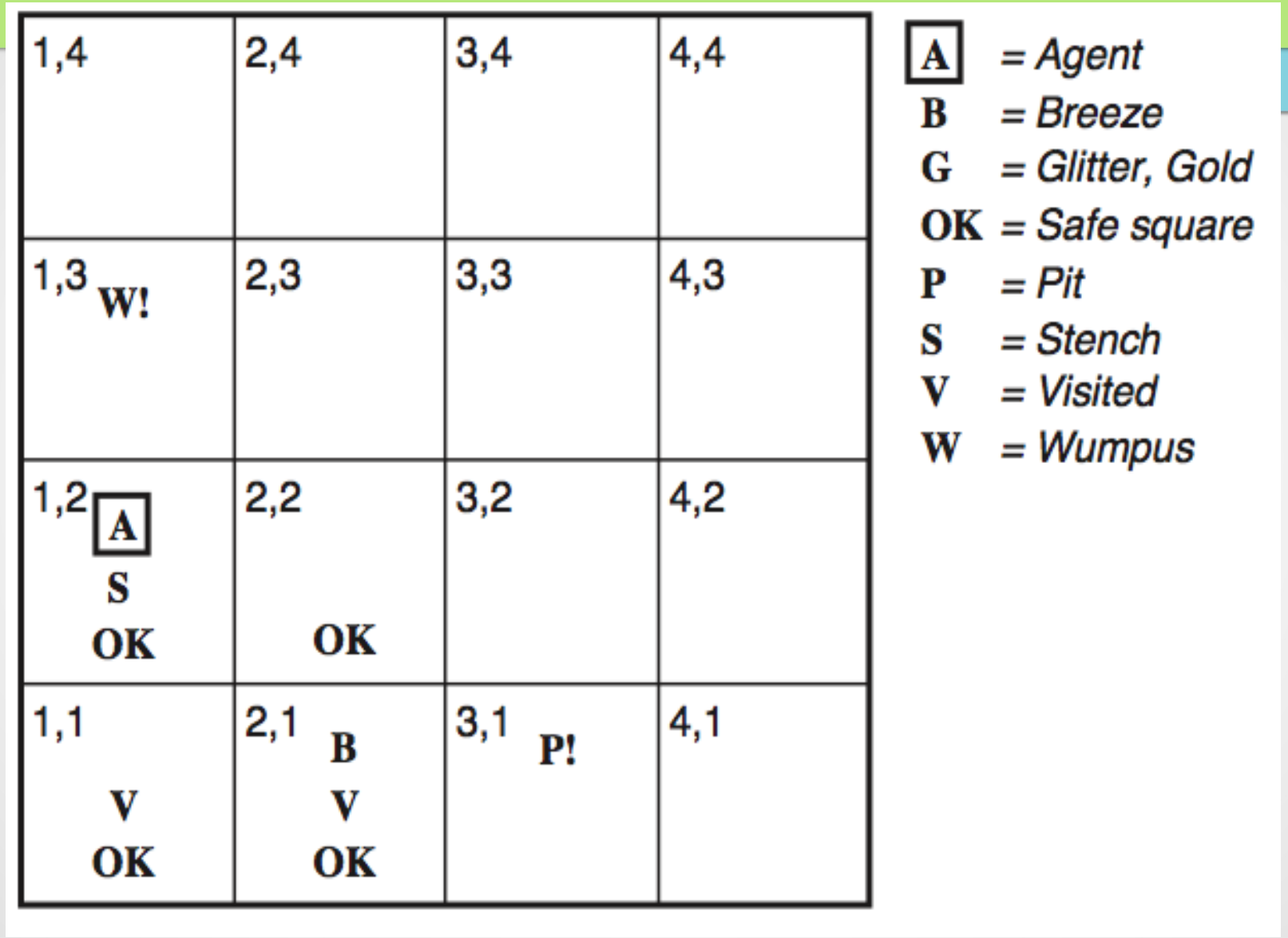
A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

[None, Breeze, None, None, None]

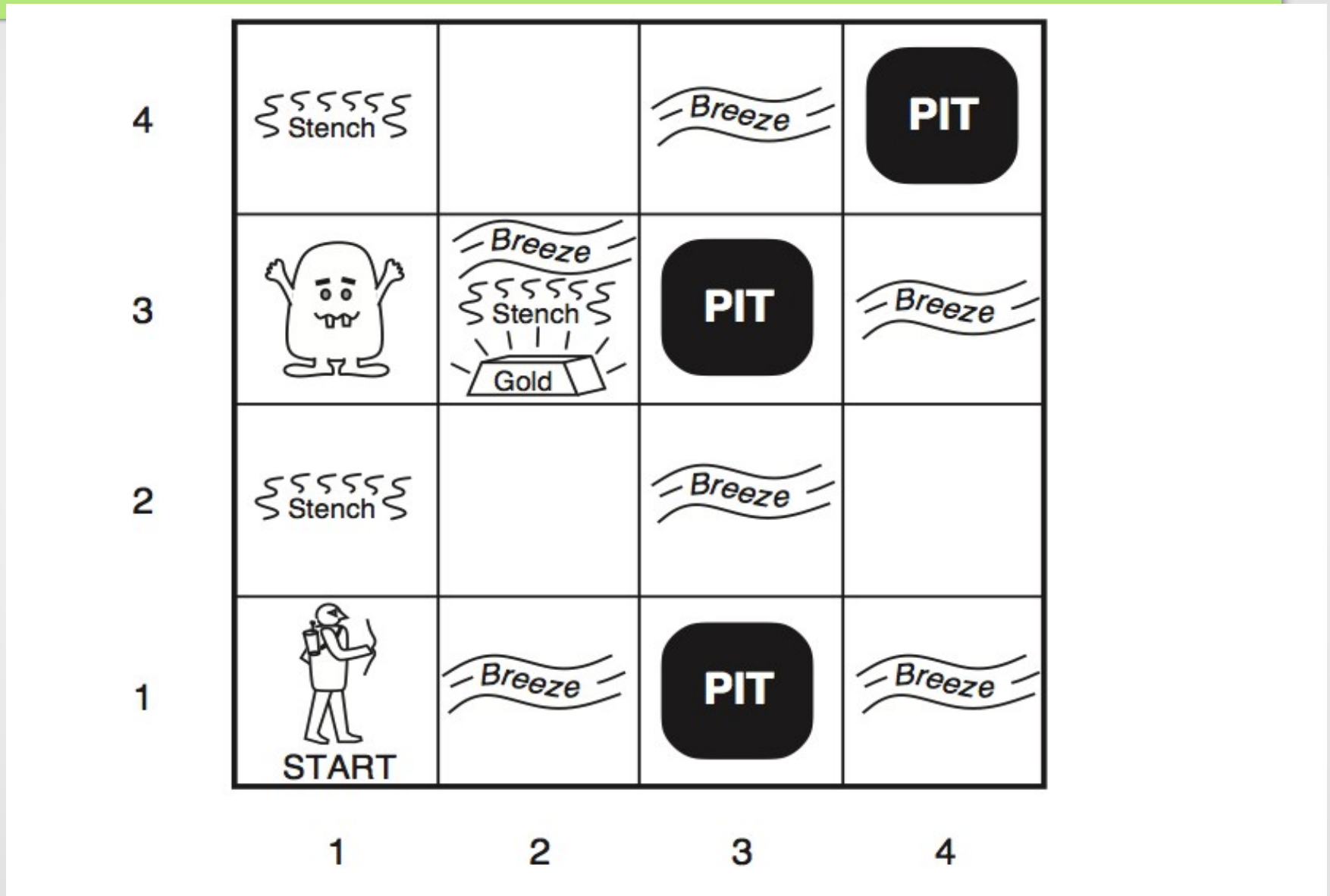
Wumpus World





[Stench, None, None, None, None]

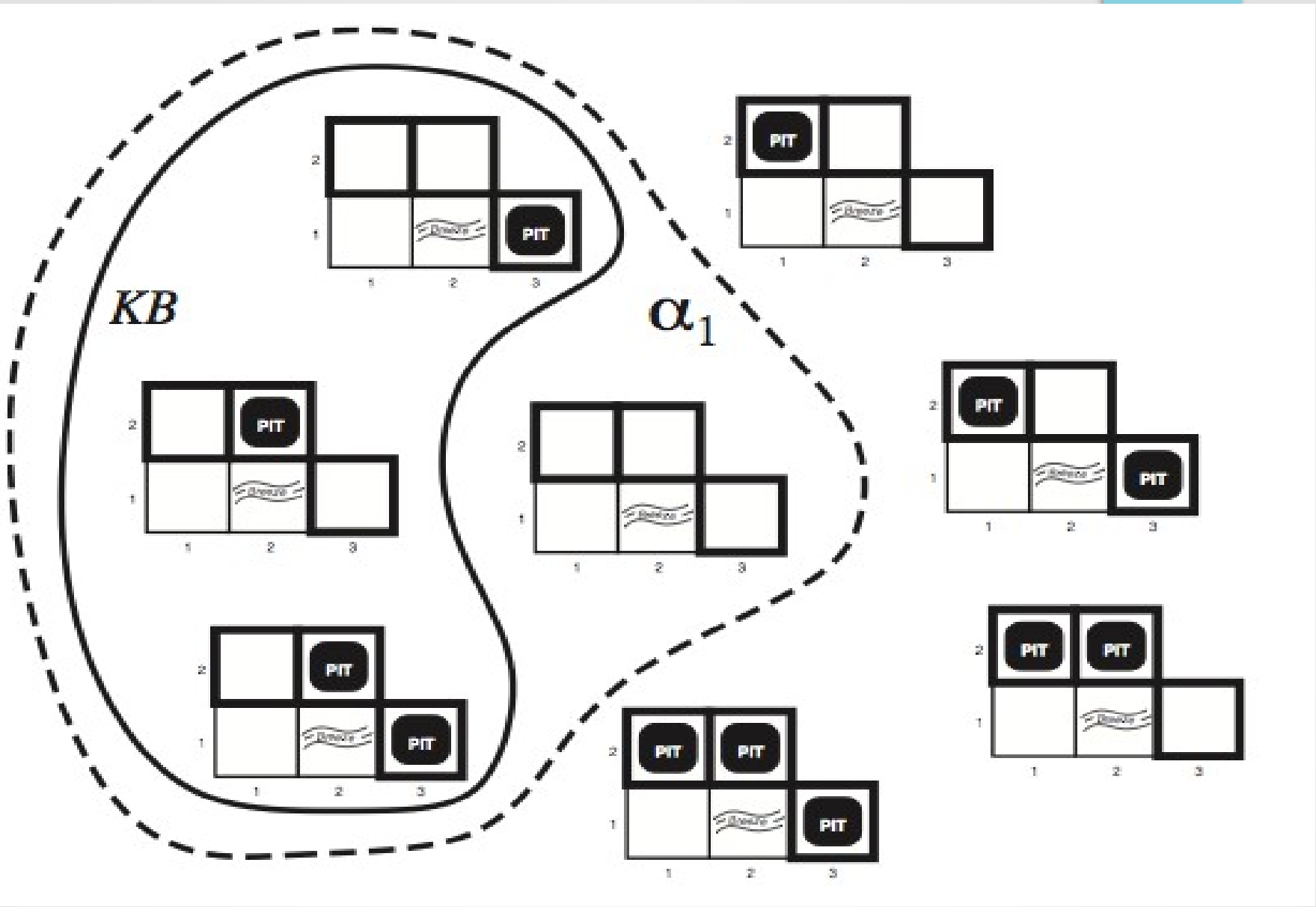
Wumpus World



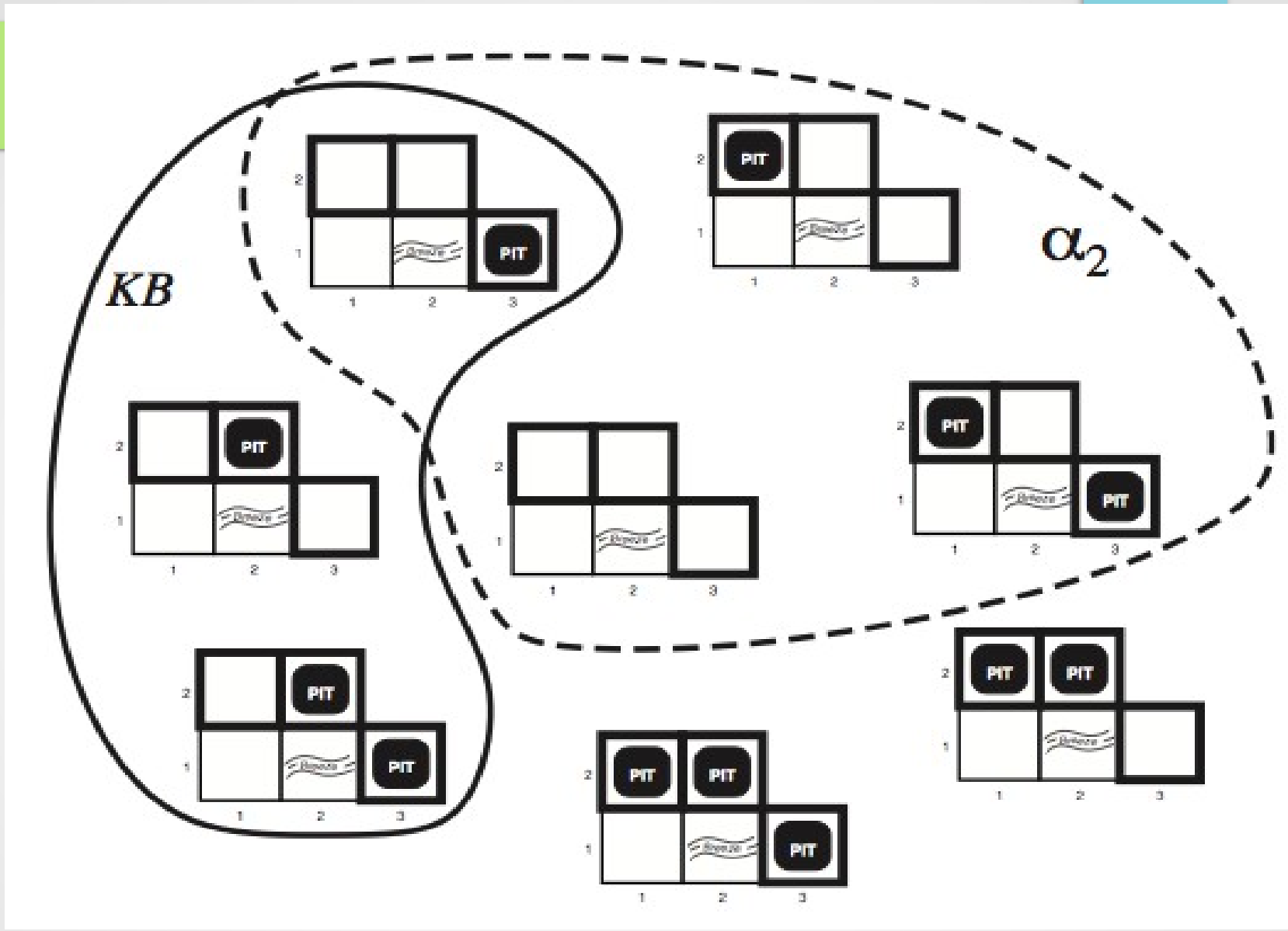
A	= *Agent*
B	= *Breeze*
G	= *Glitter, Gold*
OK	= *Safe square*
P	= *Pit*
S	= *Stench*
V	= *Visited*
W	= *Wumpus*

1,4	2,4 **P?**	3,4	4,4
1,3 **W!**	2,3 **A** **S G** **B**	3,3 **P?**	4,3
1,2 **S** **V** **OK**	2,2 **V** **OK**	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1 **P!**	4,1

[Stench, Breeze, Glitter, None, None]



(after visiting [1,1] and [2,1] only)
Dotted line shows α_1 no pit in [1,2]



Dotted line shows α_2 no pit in [2,2]

KB and entailment

- α_1 = “There is no pit in [1,2]”
- α_2 = “there is no pit in [2,2]”

- In every model in which KB is true α_1 is also true

Thus, KB $\models \alpha_1$: There is no pit in [1,2]

Logic

- $P_{x,y}$ is true if there is a pit in [x,y]
- $W_{x,y}$ is true if there is a wumpus in [x,y] dead or alive
- $B_{x,y}$ is true if the agent perceives a breeze in [x,y]
- $S_{x,y}$ is true if the agent perceives a stench in [x,y]

Wumpus world Rules

- Rules: No pit in starting position:
- R_1 : $\neg P_{1,1}$

- A square is breezy iff there is a pit in a neighboring square:
- R_2 : $B_{1,1} <=> (P_{1,2} \vee P_{2,1})$
- R_3: $B_{2,1} <=> (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

- From the first two observations:
- R_4: $\neg B_{1,1}$
- R_5: $B_{2,1}$

From these we can infer other rules . . .

- By rule R_2 and bidirectional elimination
- R_6 : $B_{1,1} \rightarrow (P_{1,2} \text{ V } P_{2,1})$ ^ $(P_{1,2} \text{ V } P_{2,1}) \rightarrow B_{1,1}$
- . . .

Conjunctive Normal Form

- Every sentence in propositional logic is logically equivalent to a conjunction of clauses
- A sentence is in conjunctive normal form (CNF) if it is expressed as a conjunction of clauses
- Steps to convert to CNF

1) Eliminate <=>
2) Eliminate → replace with negation ¬α V β
3) Move negations inward so only literals contain them
4) Apply distributive law (distribute V over ^)

331 – Intro to Intelligent Systems
Week 08
Machine Learning
Decision Trees

T.J. Borrelli

Machine Learning

“Natural Selection is the blind watchmaker, blind because it does not see ahead, does not plan consequences, has no purpose in view. Yet the living results of natural selection overwhelmingly impress us with the appearance of design as if by a master watchmaker, impress us with the illusion of design and planning.”

- Richard Dawkins, “The Blind Watchmaker”

Machine Learning

- The goal of machine learning is to develop algorithms that will automate the process of decision-making based on data
  - This will allow the machine to change its *behavior* based on the data it receives by recognizing complex patterns in the data and extrapolating to new situations

Machine Learning

- An agent is **learning** if it improves its performance on future tasks after making observations about the world
- From a collection of input/output pairs, learn a function that predicts output for new inputs
- Why do this?
  - Designers cannot anticipate all possible situations
  - Designers cannot anticipate all changes over time
  - Designers may not know how to solve the problem themselves

Machine Learning

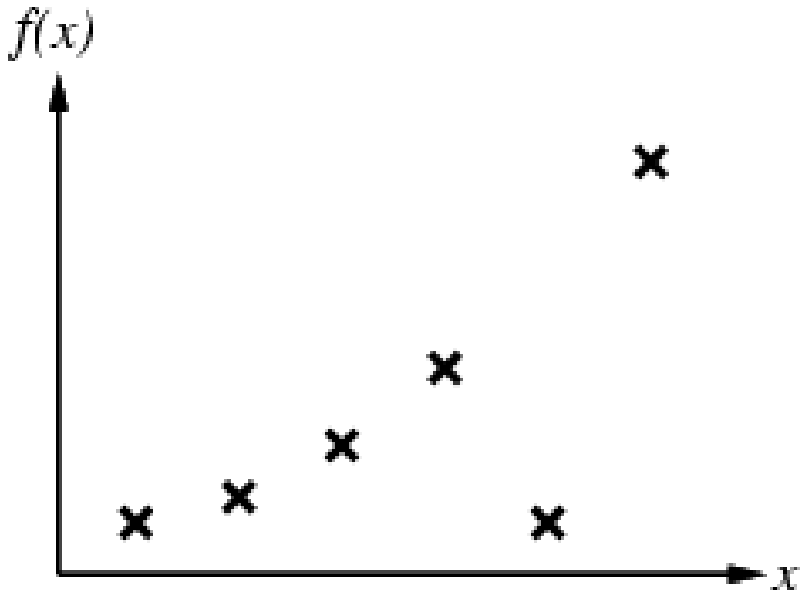
- Machine learning includes:
  - Neural networks
  - Genetic algorithms
  - Bayesian networks
  - Fuzzy Logic
  - Data mining
  - Support vector machines
  - etc.

Learning in General

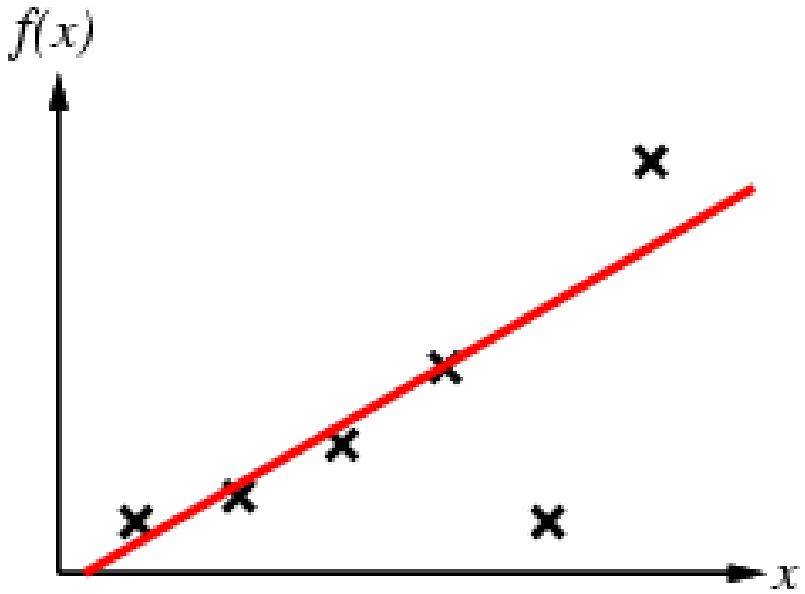
- The goal of learning is to be able to improve performance on future tasks after making observations about the world
- *Deductive learning* attempts to deduce new knowledge from rules and facts using logical inference – going from a known general rule to a new rule that is logically entailed
- *Inductive learning* attempts to learn new knowledge from examples – learning a general function or rule from specific input/output pairs

Inductive Learning Method

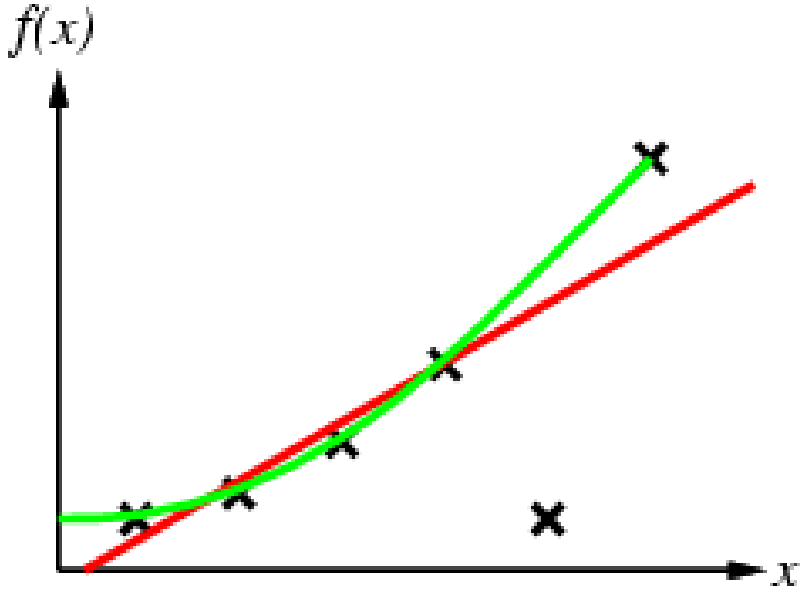
- Construct or adjust an hypothesis *h* to agree with a function *f* given a training set
- *h* is consistent if it agrees with *f* on all examples
- Consider curve fitting:



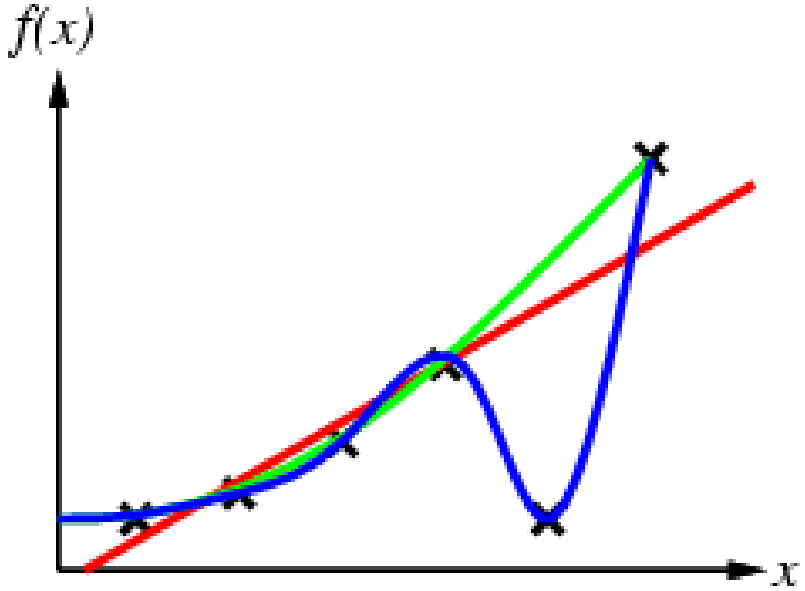
Inductive Learning Method



Inductive Learning Method



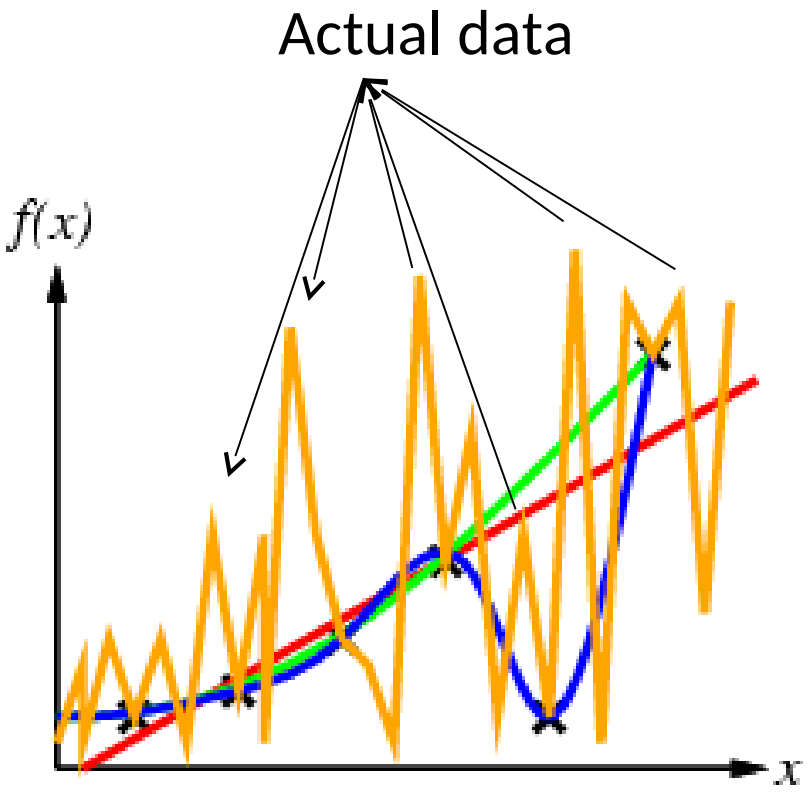
Inductive Learning Method



Inductive Learning Method

Occam's (or Ockham's) razor: prefer the simplest hypothesis that is consistent with *all* of the data. (consistent if it agrees with all data).

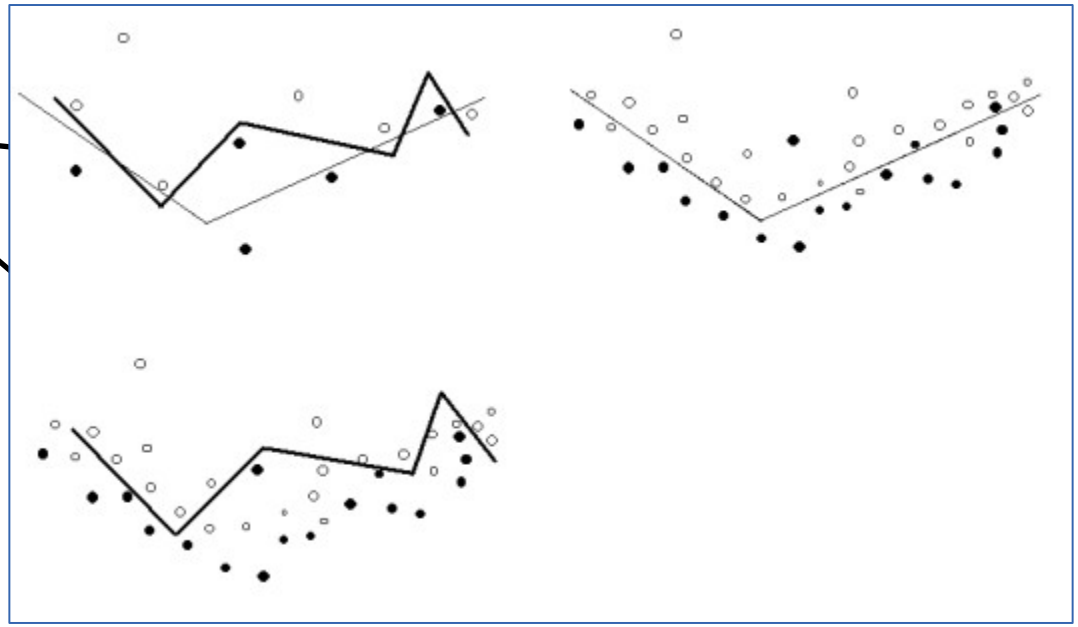
But beware of under-fitting.



The Problem of Over-fitting

The black dots represent positive examples, the gray dots represent negative examples. Goal: find the line that clearly separates the two sets of data. The two lines represent two different hypotheses.

Complex hypothesis

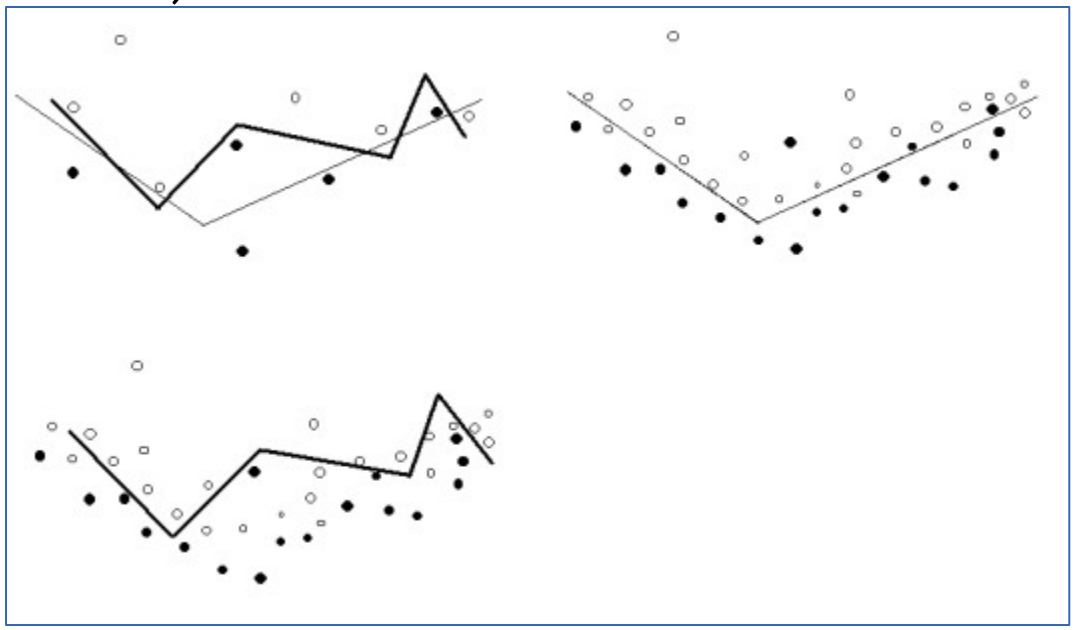


In the upper left diagram, there are just a few items of training data. These are correctly classified by the complex hypothesis. In the upper right diagram we see the complete set of data. The complex hypothesis was obviously incorrect, as shown at the lower left.

In general there is a tradeoff Between a complex hypothesis that fits the training data well and a simpler hypothesis that may generalize better.

The Problem of Over-fitting

- The simpler hypothesis, which matches the training data less well, matches the rest of the data better than the more complex hypothesis, which over-fits



Types of Inductive Learning

- Inductive learning can be *supervised* (a "teacher" is involved) or *unsupervised* (the "student" is on his or her own)
- Supervised learning is called *classification*
- Unsupervised learning is called *clustering*
- Reinforcement learning is another type of learning where there are "rewards" or "punishments" for answers that are correct or incorrect

Supervised Learning

- Also known as classification
- A classifier is designed by using a *training set* of patterns of known class for the purpose of determining the class of future sample patterns of unknown class
- A *test set* of patterns is also provided for which the true class is known, for the purpose of evaluating the effectiveness of the classifier

Non-Parametric Classification

- We do not have enough knowledge or data to be able to assume the general form of the probability model, or to estimate the relevant parameters
  - Look directly at the data instead of a summary of the data (decision trees, etc.)
  - Look at histograms, scatter plots, tables of the data (nearest-neighbor, etc.)

Nearest Neighbor Classification

- Each sample represents a “point” in feature space
- Classify unknown sample as belonging to the same class as the most similar or “nearest” sample point in the training set
- By “nearest” we usually mean the smallest distance in an n-dimensional feature space

Nearest Neighbor Classification

- Euclidean distance formula:

$$d(a,b) = \text{sqrt}\left(\sum (b_i - a_i)^2 \right)$$

- Square root is optional, to save computation time (relative distances remain the same)
- Euclidean distance emphasizes large distances
  - May want to use absolute differences in each feature instead of squared differences

Nearest Neighbor Classification

- Absolute distance formula:

$$d(a,b) = \sum | b_i - a_i |$$

  - Also called “city block distance” or Manhattan distance
- Maximum distance metric

$$d(a,b) = \max | b_i - a_i |$$

  - Finds only the most dissimilar pair of features

Nearest Neighbor Classification

- Minkowski distance formula:

$$d(a,b) = [\sum (b_i - a_i)^r]^{1/r}$$

  - "r" is an adjustable parameter
  - Generalization of the three previously defined distance metrics

Nearest Neighbor Classification

- Problem of scale:
  - An arbitrary change in the unit of measurement of one of the features could easily affect the decision
  - For example, measuring a length in millimeters rather than in meters would increase the relative contribution of this feature by a factor of 1000 compared to the other features if city block distance is used, and by 1,000,000 if Euclidean distance is used!

Nearest Neighbor Classification

- Problem of scale:
  - If one feature has a very wide range of possible values compared to the other features, it will have a very large effect on the total dissimilarity, and the decisions will be based primarily upon this single feature
  - To overcome this, it is necessary to apply scale factors to the features before computing the distances

Nearest Neighbor Classification

- Problem of scale:
  - If we want the potential influence of each of the features to be about equal, the features should be scaled so that each of them has the same standard deviation, range, or other measure of spread for the entire data set

Nearest Neighbor Classification

- Problem of scale:
  - Normalize each feature, x_i, to have a mean of 0 and a standard deviation of 1 for the entire data set:
  - i.e., replace each feature x_i with:

$$z_i = (x_i - \mu_i) / \sigma_i$$

Nearest Neighbor Classification

- Problem of scale:
  - If some prior knowledge of the relative importance of the features is available, the features could be scaled according to their importance or desired contribution to the decision making
  - Let the range, standard deviation, or weight of each normalized feature be proportional to the "accuracy" of that feature (where "accuracy" is defined as the probability of a correct decision when that feature is used alone)

K-Nearest Neighbor (k-NN)

- Base the classification of a sample on the number of “votes” of *k* of its nearest neighbors, rather than on only its single nearest neighbor - denoted *k*-NN
- Choosing *k* too large tends to suppress the fine structure of the underlying density, while choosing *k* too small puts too much emphasis on the chance locations of a few sample points
- In practice, various values of *k* are tried on the training set, and the one that gives the best result is chosen

K-Nearest Neighbor (k-NN)

- k-NN suffers from “the curse of dimensionality”
- In low-dimensional spaces with plenty of data nearest neighbor works well
- As the number of dimensions rises the nearest neighbors are usually not very near
  - Most neighbors are “outliers”!
- In addition, search can be difficult if there are a large number of samples – $O(N^2)$
- We would like sub-linear time complexity

Application: Pandora

- Pandora is an Internet music radio service that allows users to build customized “stations” that play music similar to a song or artist that is specified
- Pandora uses a k-NN style process called *The Music Genome Project* to locate new songs or artists that are similar to the user-specified song or artist

Application: Pandora

The Pandora process in general:

Hundreds of variables are created on which a song can be measured on a scale from 0 to 5 (e.g., acid-rock, accordion playing, etc.)

Pandora pays musicians to analyze songs and rate each one on each variable

The user specifies a song he or she likes (must be in Pandora's database)

The distance between user's choice and every song in the database is calculated

User has option to say "I like this song", "I don't like this song", or nothing

If "like" is chosen, the song and closest song are merged into a cluster

Classification Errors

- Classification errors arise when an observation that is known to belong to a certain class is classified as belonging to a different class
- A very simple rule for classification is to classify the observation as belonging to the most prevalent class (i.e., ignore any predictor results)
- This is known as the *naïve rule*, and is used as a baseline or benchmark for evaluating the performance of more complicated classifiers
    - A classifier that uses predictor results should outperform this

Estimation of Error Rate

- After developing a classification procedure, we must evaluate its performance
  - Model-based error estimation (assumes distributions are known)
  - Counting-based error estimation (assumes true classification or "ground-truth" is known for a sample test set)
  - "Confusion matrix" (also known as a contingency table)

Learning Decision Trees

- Decision tree induction is one of the simplest and most successful forms of machine learning
- A decision tree represents a function that takes as input a vector of attribute values and returns a decision (single output value)
- A decision tree reaches its decision by performing a sequence of tests on each (non-terminal) node in the tree
- Restaurant example

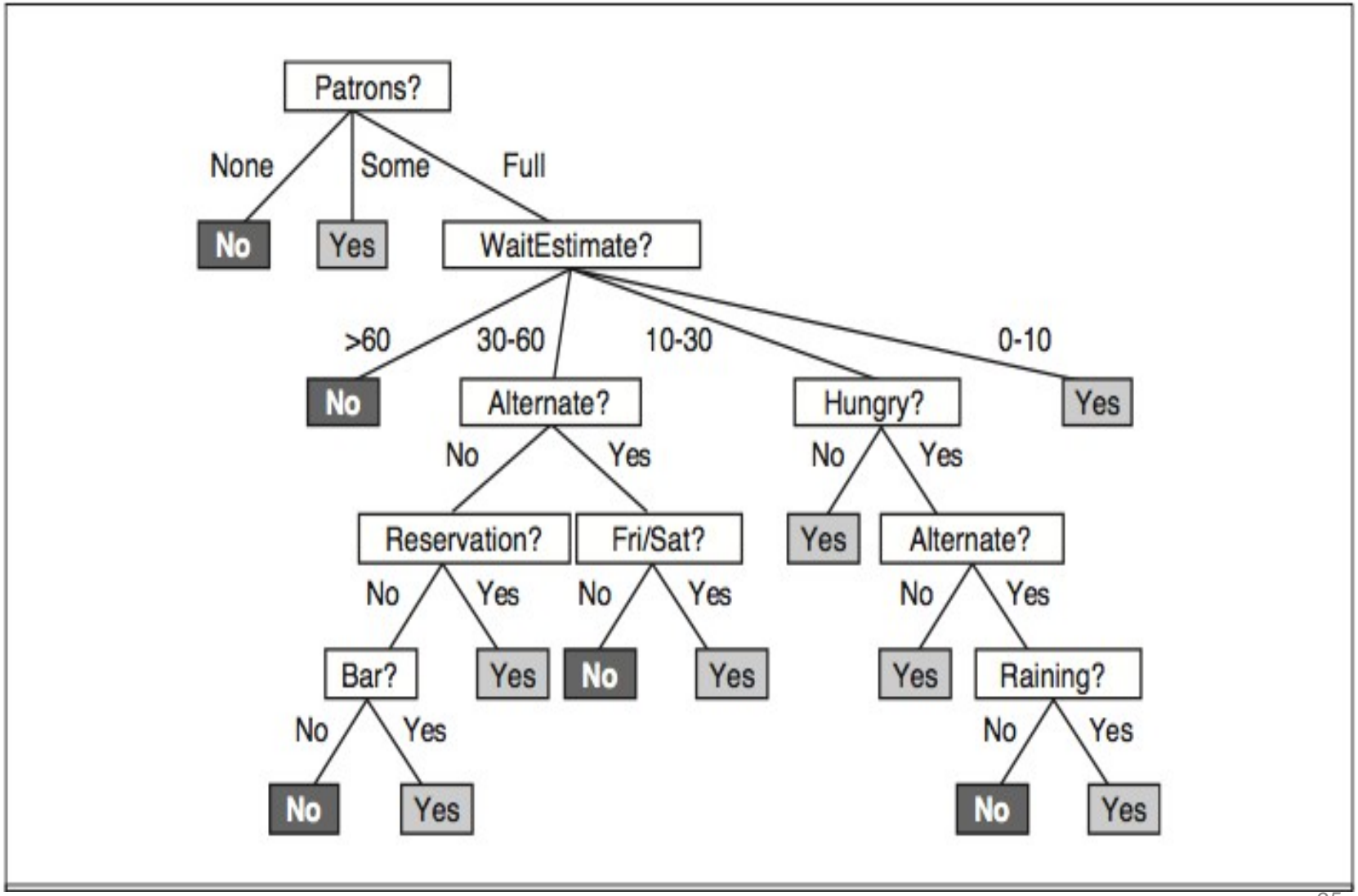
Restaurant Example

- We wish to build a decision tree to decide whether to wait for a table at a restaurant
- Goal predicate: *WillWait*
- List of attributes to consider
  - Alternate, Bar, Fri/Sat, Hungry, Patrons, Price, Raining, Reservation, Type, WaitEstimate

Restaurant Example

- List of attributes to consider
  - Alternate: is there another suitable restaurant nearby
  - Bar: does the restaurant have a comfy bar area
  - Fri/Sat: is it friday or saturday
  - Hungry: are we *really* hungry
  - Patrons: how many people are here (None, Some, Full)
  - Price: 3 categories ($, $$, $$$)
  - Raining: whether it is raining outside
  - Reservation: whether we made a reservation
  - Type: kind of restaurant (French, Italian, Thai, burger)
  - WaitEstimate: wait time estimated by host (<10min, 10-30min, 31-60min, >60min)

Decision Tree – deciding whether to wait for a table



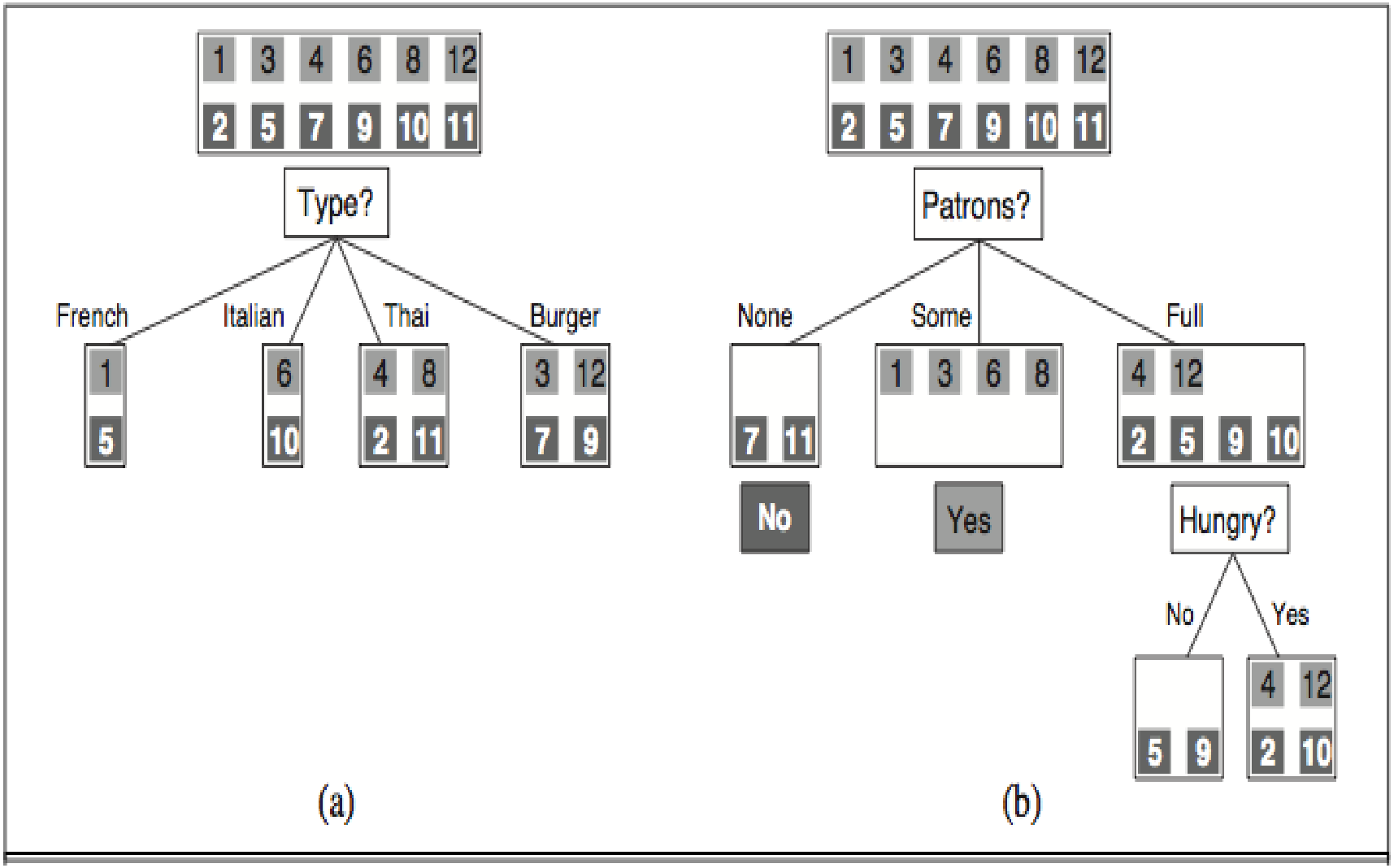
Decision Trees

- For a wide variety of problems, the decision tree format yields a nice, concise result
- Some problems cannot be represented concisely (explodes into exponentially many nodes)
- Our goal with the decision tree is to create one that is consistent with the input/output pairs and is as small as possible
- Often finding the *smallest* tree is also in intractable so we may need to use heuristics

Non-exhaustive list of inputs and outputs

Num	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	Wait
x_1	yes	no	no	yes	some	$$$	no	yes	French	0-10	yes
x_2	yes	no	no	yes	full	$	no	no	Thai	30-60	no
x_3	no	yes	no	no	some	$	no	no	Burger	0-10	yes
x_4	yes	no	yes	yes	full	$	yes	no	Thai	10-30	yes
x_5	yes	no	yes	no	full	$$$	no	yes	French	>60	no
x_6	no	yes	no	yes	some	$$	yes	yes	Italian	0-10	yes
x_7	no	yes	no	no	none	$	yes	no	Burger	0-10	no
x_8	no	no	no	yes	some	$$	yes	yes	Thai	0-10	yes
x_9	no	yes	yes	no	full	$	yes	no	Burger	>60	no
x_{10}	yes	yes	yes	yes	full	$$$	no	yes	Italian	10-30	no
x_{11}	no	no	no	no	none	$	no	no	Thai	0-10	no
x_{12}	yes	yes	yes	yes	full	$	no	no	Burger	30-60	yes

Possible Decision Trees



Decision Trees

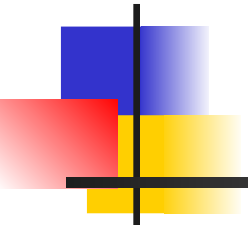
- Splitting on *Type* (on last slide) brings us no closer to being able to distinguish between “Yes” and “No” answers
- Splitting on *Patrons* however does a good job of separating “Yes” from “No” selections
- We still have some work to do, but we found some leaf nodes early on

Decision Trees

- When creating our decision tree the goal is to approximately minimize its depth
- Therefore we should pick attributes in such a way that we can classify inputs as soon as possible
- A “perfect” attribute is one that divides the examples into sets, each of which are all positive or all negative (thus terminal)
- A really useless attribute such as Type on the previous slide, leaves the example sets with roughly the same proportion of positive and negatives examples as in the original set

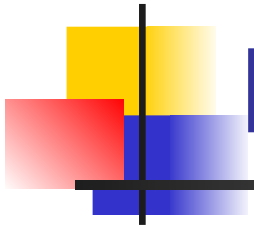
Decision Trees and Entropy

- How do we know which attributes to pick so that we pick good ones?
- (Shannon) Entropy – from information theory – a measure of the uncertainty of a random variable
- As information goes up (i.e. we make new observations) – entropy goes down
- Consider a (fair) coin flip – equally likely to come up heads or tails (0 or 1) – this corresponds to "1 bit" of entropy
- Roll of a 4-sided die has 2 bits of entropy – takes 2 bits to describe each of the 4 outcomes



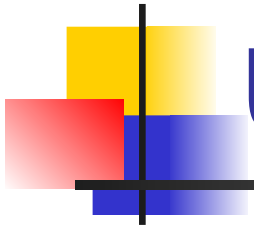
331 – Intro to Intelligent Systems
Week09
Bayesian Belief Networks

T.J. Borrelli



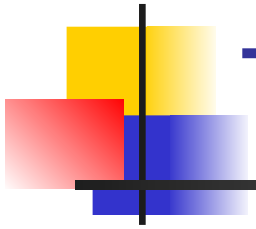
Bayesian Network

- Causality – if *this*, then *that*
- But we do not know the whole picture
- Describe events in terms of probabilities
- Allows us to reverse direction of probabilistic statement



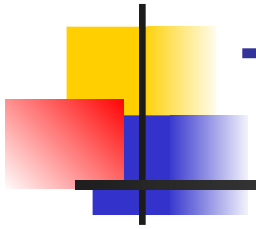
Uses of BBNs

- Predict what will happen
- Or to infer causes from observed events
- Allow calculation of conditional probabilities of the nodes in the network, given that the values of some of the nodes have been observed



Types of Inference for BBNs

- Causal Inference (abduction, prediction) - Given that A implies B, and we know the value of A, we can compute the value of B
- Diagnostic Inference (induction) - Given that A implies B, and we know the value of B, we can compute the probability of A



Types of Inference

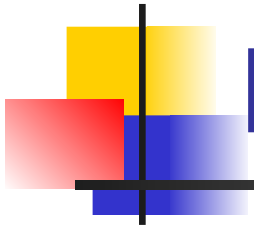
- Intercausal Inference ("explaining away") - Given that both A and B imply C, and we know C, we can compute how a change in the probability of A will affect the probability of B, even though A and B were originally independent

Bayesian Belief Network (BBN)

- Directed Acyclic Graph
- Causal relations between variables
- Models cause-and-effect relations
- Allows us to make inferences based on limited knowledge of environment and known probabilities
- Allows us to reverse direction of probabilistic statement

BBN

- Certain Independence assumptions must hold between random variables
- To specify the probability distribution of a Bayesian network
  - Need to know the *a priori* probabilities of all root nodes (nodes with no predecessors)
  - Conditional probabilities of all non-root nodes



Logically

- Only part of the story is considered in the causal (cause-and-effect) relationship between phenomena
- It is $P \rightarrow Q$ not $P \leftrightarrow Q$
- Meaning there could be other potential causes of Q

Directed and Acyclic Graph

- The directions show the causal relations between events – thus somewhat important
- If cycles are present we have a Causal Loop

Bayes' paper

- Bayes described in his famous paper (in 1763) “degree-of-belief” as opposed to classical probability (or fuzzy logic which has partial set membership)
- As new information is added to our defined system of beliefs, the probabilities of events occurring should be updated
- Also known as “Bayesian Inference”

Bayes says a lot of things ...

- However in his original paper, didn't explicitly state the rule attributed to him (conveniently called Bayes' Rule or Bayes' Theorem). So I will state it here:

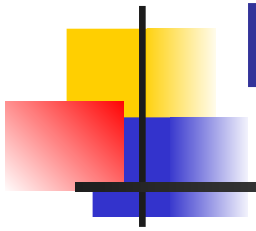
- $P(A \mid B) = (P(B \mid A) * P(A)) / P(B)$

Explanation of Bayes' Rule

- P(A | B) = (P(B | A) * P(A)) / P (B)
- Means that the Probability of event A occurring given that event B has occurred is equal to the probability of event B occurring given that A has occurred times the independent probability of A divided by the independent probability of B

Derivation of Bayes' Rule

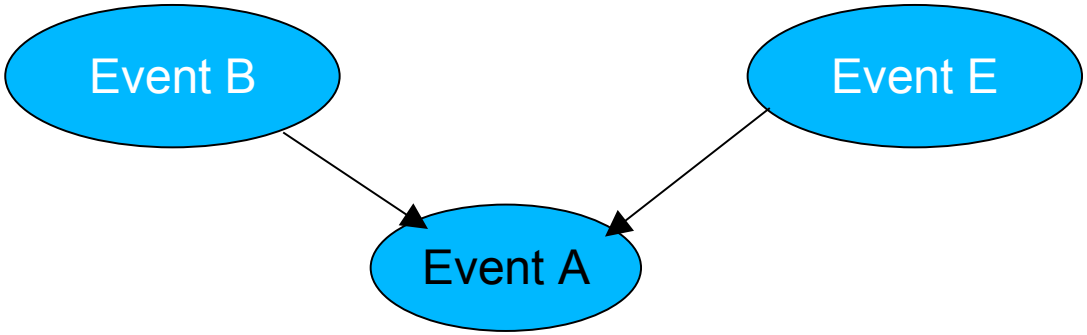
- By definition of Conditional Probabilities:
- $P(A \mid B) = P(A \cap B) / P(B)$
- By the same note: $P(B \mid A) = P(B \cap A) / P(A)$
- By the commutative law of probability and set theory: $P(A \cap B) \equiv P(B \cap A)$
- $P(A \cap B) = P(A \mid B) * P(B) = P(B \mid A) * P(A)$
- Thus: $P(A \mid B) = (P(B \mid A) * P(A)) / P(B)$



Definitions

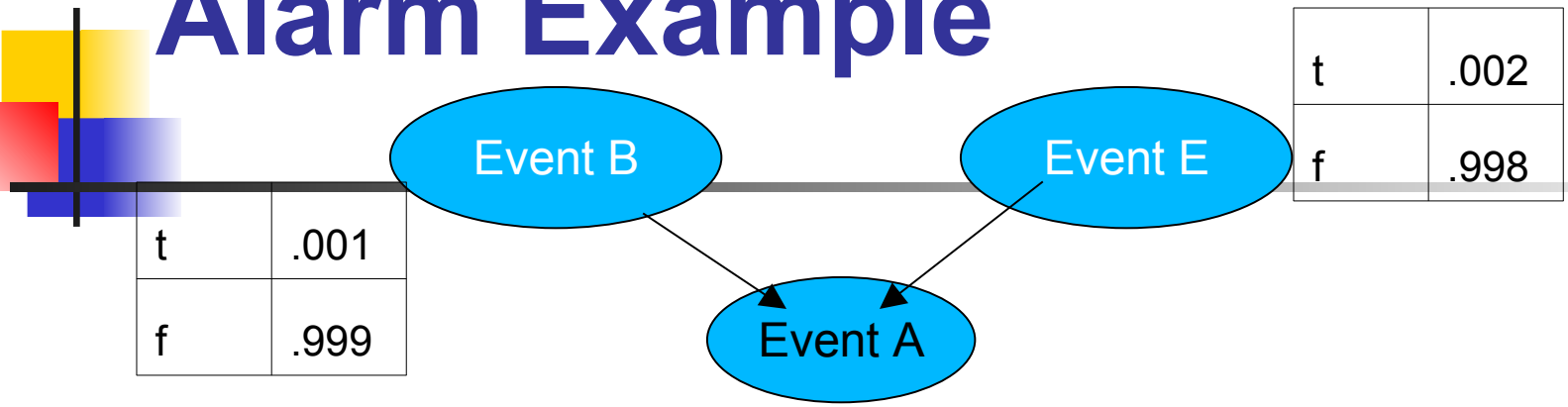
- Bayes' Rule: P(A | B) = (P(B | A) * P(A)) / P (B)
- P(A) is "prior probability" or "marginal probability" of A
- P(A|B) is "conditional probability of A, given B. Also called "posterior probability"
- P(B) is also a "prior" or "marginal" probability and acts as a normalizing constant so event probabilities add to 1

Belief Network



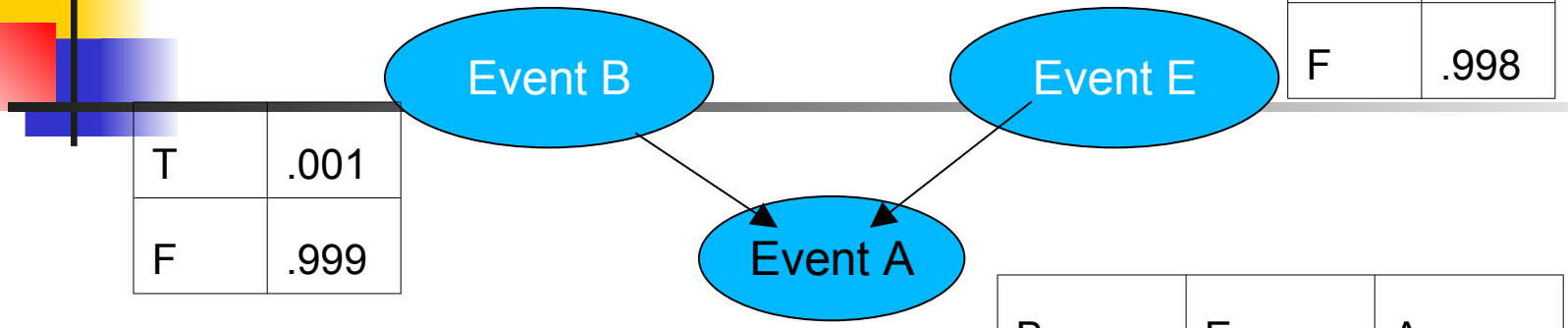
- Event A is “Alarm”
- Event B is “Burglary”
- Event E is “Earthquake”

Alarm Example



- Given
  - Initial Truth Tables
  - Causal relationships

Alarm Example-2



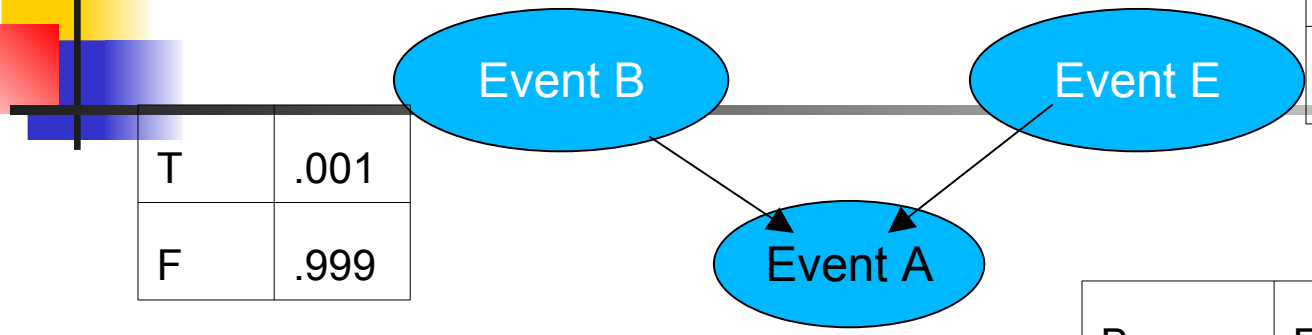
T	.001
F	.999

T	.002
F	.998

B	E	A
T	T	.95
T	F	.94
F	T	.29
F	F	.001

- Given
  - Initial Truth Tables
  - Causal relationships
  - Conditional Probabilities
- Can calculate conditional probabilities based on new information

Alarm Example-3



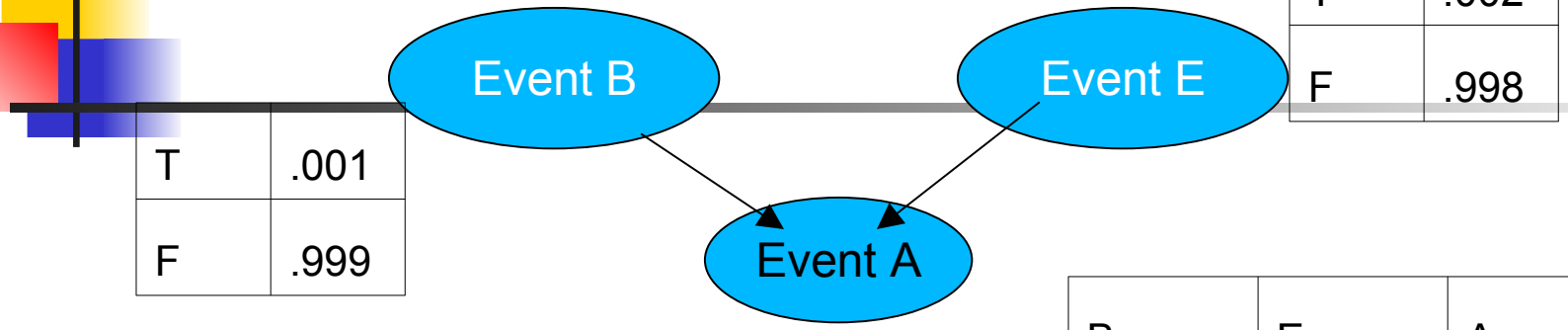
T	.001
F	.999

T	.002
F	.998

B	E	A
T	T	.95
T	F	.94
F	T	.29
F	F	.001

- P(A) = P(A|B,E)*P(B)*P(E) + P(A|¬B,E)*P(¬B)*P(E) + P(A|B,¬E)*P(B)*P(¬E)+ P(A|¬B,¬E)*P(¬B)*P(¬E) = .002516

Alarm Example-4



T	.001
F	.999

T	.002
F	.998

B	E	A
T	T	.95
T	F	.94
F	T	.29
F	F	.001

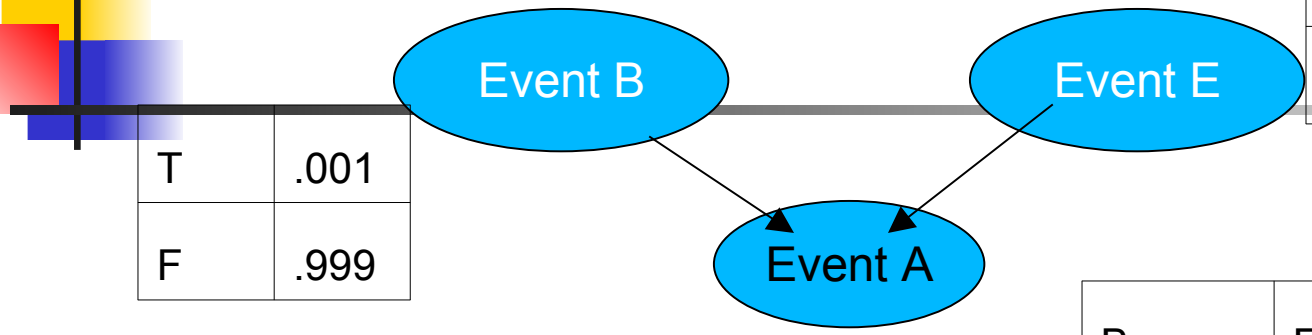
- Now Suppose We Observe

That E is True!

….

What does this change in the previous equation?

Alarm Example-5



T	.001
F	.999

T	.002
F	.998

B	E	A
T	T	.95
T	F	.94
F	T	.29
F	F	.001

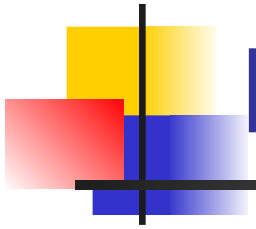
- E is True!
- What does this change in the previous equation?
- P(A) = P(A|B,E)*P(B)*P(E) +

P(A|¬B,E)*P(¬B)*P(E) +

P(A|B,¬E)*P(B)*P(¬E)+

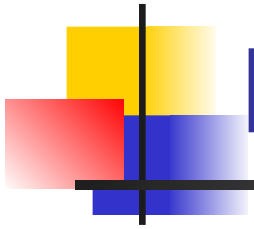
P(A|¬B,¬E)*P(¬B)*P(~E) =

P(A|B,E)*P(B) + P(A|¬B,E)*P(¬B) = .95 * .001 + .29 * .999 = .291



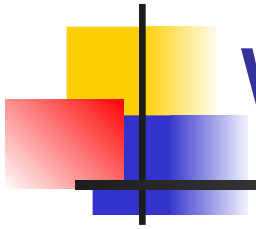
Example

- Suppose we want to know the chance that it rained yesterday if we suddenly find out that the lawn is wet
- By Bayes' rule we can calculate this probability from it's inverse: the probability that the lawn would be wet if it *had* rained yesterday



P(A|B) = (P(B|A) * P(A)) / P(B)

- P(A|B) = “the probability of event A, given that we know B” - for example, the probability that it rained yesterday given that the lawn is wet”
- P(A) = the probability of rain, all other things being equal
- P(B) = the probability of the lawn being wet, all other things being equal

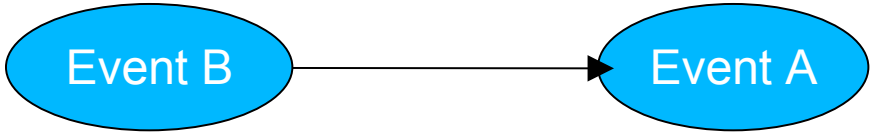


What did we just do?

- We were able to rephrase the probability in terms of the probability of B given A

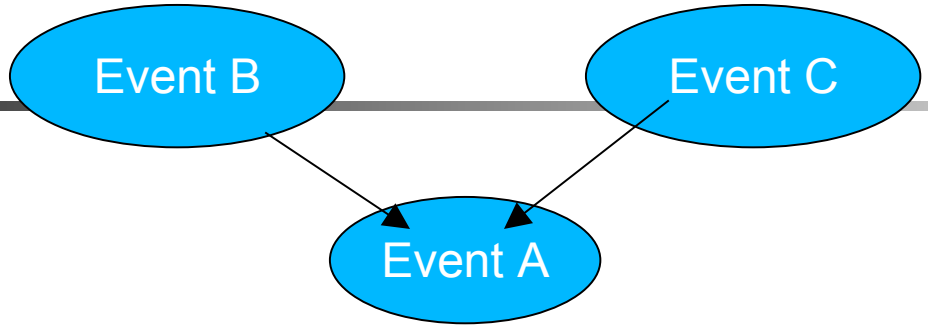
(P(B|A)) and independent probabilities P(A) and P(B)

More Probability

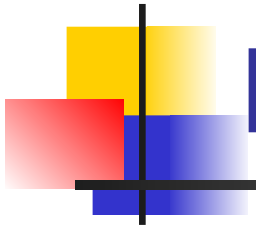


- As an extension of Bayes' Theorem:
- Given that event A is dependent upon event B
- $P(A) = P(A|B) * P(B) +$
  $P(A|\neg B) * P(\neg B)$

Similarly

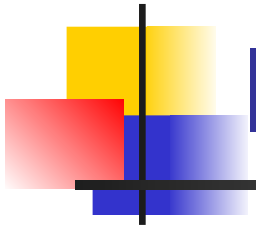


- Event A is dependent upon event B and C
- P(A) = P(A|B,C) * P(B) * P(C) +
  P(A|B, ¬C) * P(B) * P(¬C) +
  P(A|¬B, C) * P(¬B) * P(C) +
  P(A| ¬B, ¬C) * P(¬B) * P(¬C)



Problems of Inconsistency

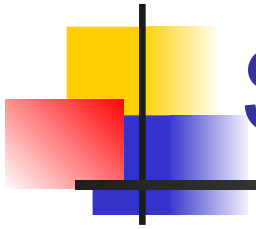
- If we look at the following “reasonable-looking” situation:
  - P(A|B) = .8
  - P(B|A) = .2
  - P(B) = .6
- Bayes’ Rule: P(A|B) = (P(B|A)*P(A))/ P(B)



Inconsistency

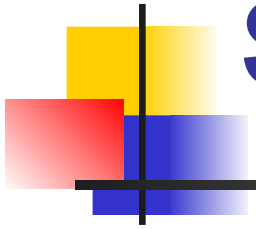
- P(A|B) = (P(B|A)*P(A))/ P(B)
- .8 = .2 * P(A) / .6
- .48 = .2 * P(A)
- Thus, P(A) = 2.4

- So, What's Wrong With This?



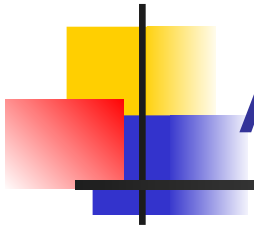
Skunk in the Woodpile

- You can’t see it, but you know it’s there
- The probability for an event cannot exceed 1
- Problem: failure to keep probabilities consistent
- This is sometimes difficult to see at first
- Need to check for this



Some Uses

- Used in medical problem diagnosis (PATHFINDER) when only limited information and events is known
- Used in map learning and language processing and understanding
- Used in Games to provide human-like reasoning based on incomplete information. (First Person Sneaker)



A Good Paper to Read

- Charniak, E. *Bayesian Networks without Tears.* American Association for Artificial Intelligence. AI Magazine. 1991.