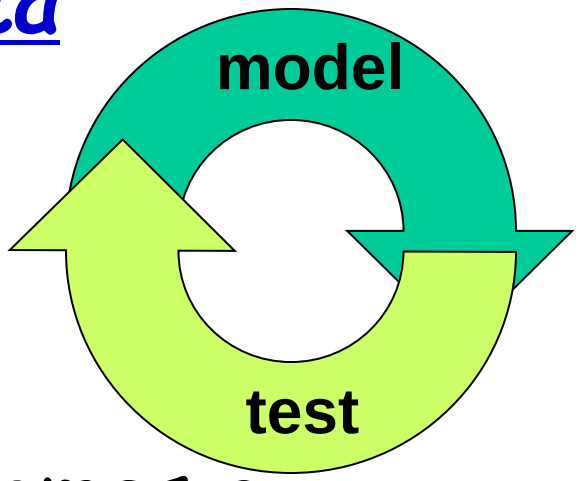
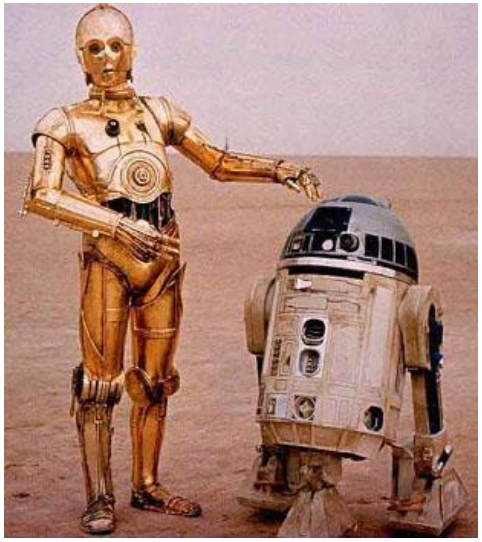
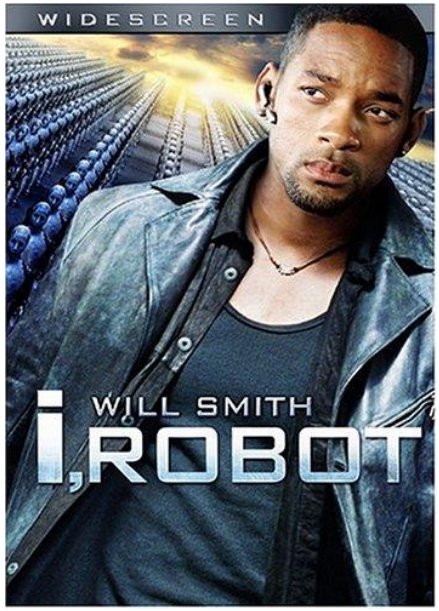
Scientific Method

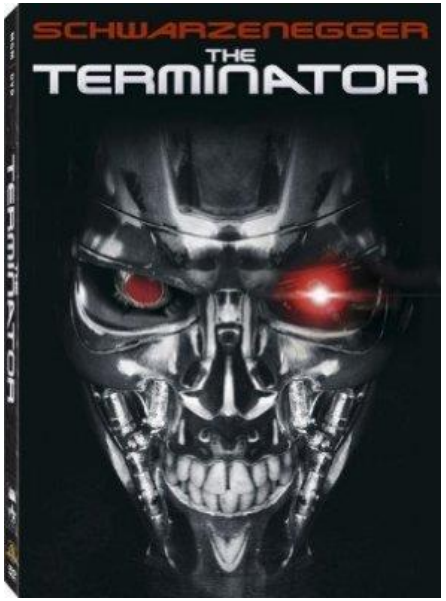
- **Observe** an event(s).
- Develop a **model** (or **hypothesis**) which makes a **prediction** to explain the event
- **Test** the prediction with ***data***
- **Observe** the result.
- **Revise** the hypothesis.
- **Repeat** as needed.
- *A* successful hypothesis becomes a **Scientific Theory**.



Artificial Intelligence in the Movies



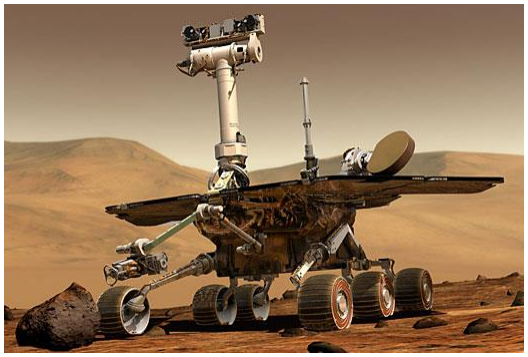


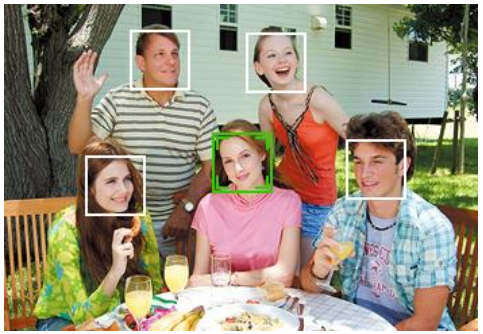


Artificial Intelligence in Real Life

A young *science* (≈ 50 years old)

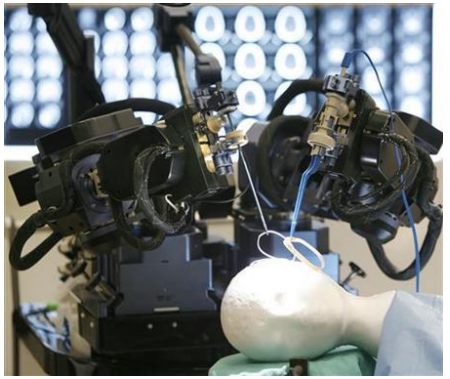
- Exciting and dynamic field, lots of uncharted territory left
- Impressive success stories
- "Intelligent" in specialized domains
- Many application areas





Face detection

Formal verification



What is artificial intelligence?

- There is no clear consensus on the definition of AI
- John McCarthy coined the phrase AI in 1956

http://www.formal.stanford.edu/jmc/whatisai/whatisai.html

Q. What is artificial intelligence?

A. It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human or *other* intelligence, but AI does not have to confine itself to methods that are biologically observable.

Q. Yes, but what is intelligence?

A. Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines.

What is AI? (Cont'd)

"The exciting new effort to make computers think ... *machines with minds*, in the full and literal sense" (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Bellman, 1978)	"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)	"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

Figure 1.1 Some definitions of AI. They are organized into four categories:

Systems that think like humans.	Systems that think rationally.
Systems that act like humans.	Systems that act rationally.

Other possible AI definitions

- AI is a collection of hard problems which can be solved by humans and other living things, but for which we don't have good algorithms for solving.
  - e. g., understanding spoken natural language, medical diagnosis, circuit design, learning, self-adaptation, reasoning, chess playing, proving math theories, etc.
- Russsell & Norvig: a program that
  - Acts like human (Turing test)
  - Thinks like human (human-like patterns of thinking steps)
  - Acts or thinks rationally (logically, correctly)
- Some problems used to be thought of as AI but are now considered not
  - e. g., compiling Fortran in 1955, symbolic mathematics in 1965, pattern recognition in 1970, what for the future?

What is the scientific method hypothesis behind AI?

History of AI

- AI has roots in a number of scientific disciplines
  - computer science and engineering (hardware and software)
  - philosophy (rules of reasoning)
  - mathematics (logic, algorithms, optimization)
  - cognitive science and psychology (modeling high level human/animal thinking)
  - neural science (model low level human/animal brain activity)
  - linguistics
- The birth of AI (1943 – 1956)
  - McCulloch and Pitts (1943): simplified mathematical model of neurons (resting/firing states) can realize all propositional logic primitives (can compute all Turing computable functions)
  - Alan Turing: Turing machine and Turing test (1950)
  - Claude Shannon: information theory; possibility of chess playing computers
  - Boole, Aristotle, Euclid (logics, syllogisms)

- Early enthusiasm (1952 – 1969)
  - 1956 Dartmouth conference
    John McCarthy (Lisp);
    Marvin Minsky (first neural network machine);
    Alan Newell and Herbert Simon (GPS);
  - Emphasis on intelligent general problem solving
    GSP (means-ends analysis);
    Lisp (AI programming language);
    Resolution by John Robinson (basis for automatic theorem proving);
    heuristic search (A*, AO*, game tree search)
- Emphasis on knowledge (1966 – 1974)
  - domain specific knowledge is the key to overcome existing difficulties
  - knowledge representation (KR) paradigms
  - declarative vs. procedural representation

- Knowledge-based systems (1969 – 1999)
  - DENDRAL: the first knowledge intensive system (determining 3D structures of complex chemical compounds)
  - MYCIN: first rule-based expert system (containing 450 rules for diagnosing blood infectious diseases)
    EMYCIN: an ES shell
  - PROSPECTOR: first knowledge-based system that made significant profit (geological ES for mineral deposits)
- AI became an industry (1980 – 1989)
  - wide applications in various domains
  - commercially available tools
  - AI winter
- Current trends (1990 – present)
  - more realistic goals
  - more practical (application oriented)
  - distributed AI and intelligent software agents
  - resurgence of natural computation - neural networks and emergence of genetic algorithms – many applications
  - dominance of machine learning (big apps)

Thinking Rationally: Laws of Thought

- Aristotle (~ 450 B.C.) attempted to codify "right thinking" What are correct arguments/thought processes?
- E.g., "Socrates is a man, all men are mortal; therefore Socrates is mortal"

- Several Greek schools developed various forms of logic: notation plus rules of derivation for thoughts.

- Problems:
  1) Uncertainty: Not all facts are certain (e.g., *the flight might be delayed).*
  2) Resource limitations: There is a difference between solving a problem in principle and solving it in practice under various resource limitations such as time, computation, accuracy etc. (e.g., *purchasing a car)*

Weak and Strong AI Claims

- Weak AI:
  - Machines can be made to act *as if* they were intelligent.
- Strong AI:
  - Machines that act intelligently have real, conscious minds.

Eliza

- Called after Eliza Doolittle of Pygmalion fame.
- Developed in 1964-1966 by Joseph Weizenbaum in MIT
- Models (parodies) the rôle of a Rogerian psychotherapist engaged in an initial interview with a patient. Much or the technique of the Rogerian psychotherapist involves drawing the patient out by reflecting the patient's statements back at him.

Eliza

Men are all alike
IN WHAT WAY
They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE
Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE
He says I am depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED
It's true. I am unhappy.
DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
I need some help, that much seems certain.
WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
Perhaps I could learn to get along with my mother.

Eliza

TELL ME MORE ABOUT YOUR FAMILY
My mother takes care of me.
WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU
My father.
YOUR FATHER
You are like my father in some ways
WHAT RESEMBLANCE DO YOU SEE
You are not very aggressive but I think you don't want me to notice that.
WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE
You don't argue with me
WHY DO YOU THINK I DON'T ARGUE WITH YOU

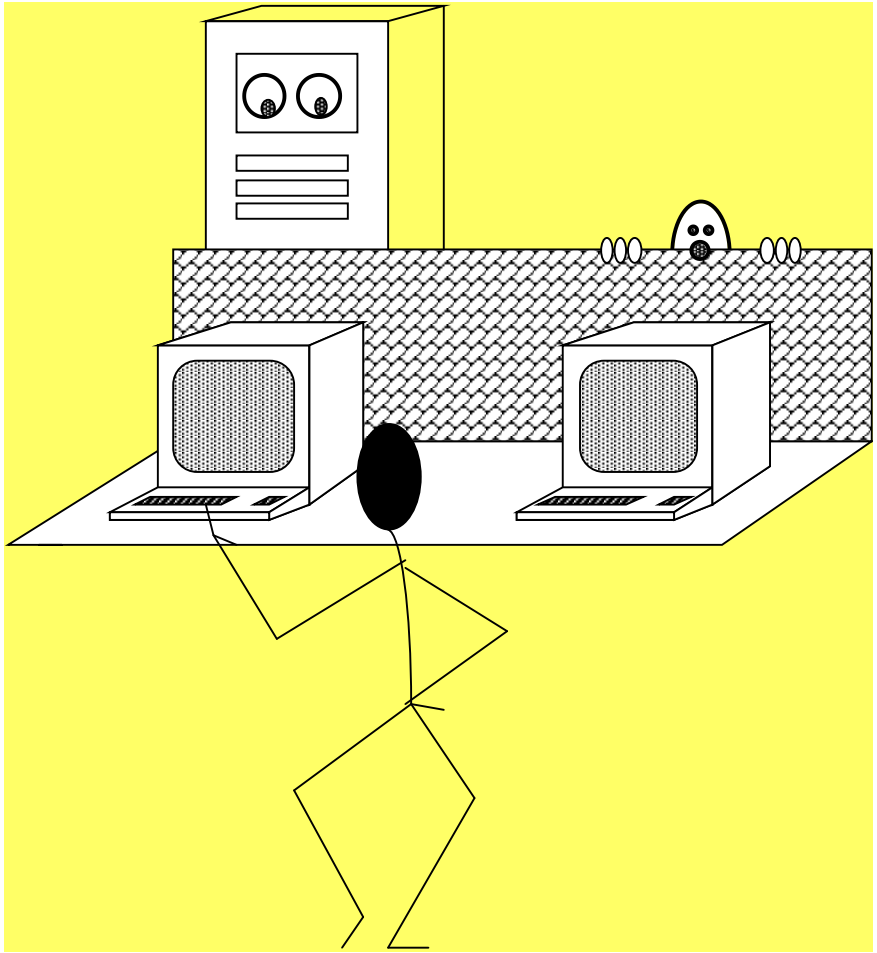
What is/was Eliza?

- A chatterbot
- Turing test competitor
- A friend
- A model for other bots

What is a chatterbot?

- A chatter robot, chatterbot, chatbot, or chat bot is a computer program designed to simulate an intelligent conversation with one or more human users via auditory or textual methods, primarily for engaging in small talk.
  - The primary aim of such simulation has been to fool the user into thinking that the program's output has been produced by a human (the Turing test).
  - Programs playing this role are sometimes referred to as Artificial Conversational Entities, talk bots or chatterboxes.
  - Uses:
    - chatterbots are often integrated into dialog systems for various practical purposes such as online help, personalised service, or information acquisition.
    - Spam in chatrooms
  - Some chatterbots use sophisticated natural language processing systems, but many simply scan for keywords within the input and pull a reply with the most matching keywords, or the most similar wording pattern, from a textual database.

What is Intelligence?
The Turing Test



A machine can be described as a thinking machine if it passes the Turing Test. i.e. If a human agent is engaged in two isolated dialogues (connected by teletype say); one with a computer, and the other with another human and the human agent cannot reliably identify which dialogue is with the computer.

Intelligence

- *Turing Test*: A human communicates with a computer via a teletype. If the human can't tell he is talking to a computer or another human, it passes.
  - Natural language processing
  - knowledge representation
  - automated reasoning
  - machine learning
- Add vision and robotics to get the total Turing test.

Turing Test – Loebner prize

Loebner Prize Judges Could Easily Identify Chatbots

Written by Sue Gee

Friday, 18 May 2012 08:36

The 2012 Loebner Prize for the best chatbot has been awarded to Chip Vivant, created by Mohan Embar, a software consultant based in Milwaukee.

There were four contestants in the final round of this competition which took place at Bletchley Park on May 15, 2012 and none of them were likely to fool the judges into mistaking them for a human as required to win the Loebner Prize Gold Medal. So instead the Bronze Medal and $5,000 was awarded to the chatbot with the most impressive conversational abilities.

Alan Turing depicted on the Loebner Prize Gold Medal

http://www.loebner.net/Prizef/loebner-prize.html

Home Page of The Loebner Prize--"The First Turing Test"

Loebner Prize Gold Medal

(Solid 18 carat, not *gold-plated* like the Olympic *"Gold"* medals)

What is the Loebner Prize?

The Loebner Prize is the first formal instantiation of a Turing Test. The test is named after Alan Turing the brilliant British mathematician. Among his many accomplishments was basic research in computing science. In 1950, in the article *Computing Machinery and Intelligence* which appeared in the philosophical journal Mind, Alan Turing asked the question "Can a Machine Think?"He answered in the affirmative, but a central question was: "If a computer could think, how could we tell?" Turing's suggestion was, that if the responses from the computer were indistinguishable from that of a human,the computer could be said to be thinking.

In 1990 Hugh Loebner agreed with The Cambridge Center for Behavioral Studies to underwrite a contest designed to implement the Turing Test. Dr. Loebner pledged a Grand Prize of $100,000 and a Gold Medal (pictured above) for the first computer whose responses were indistinguishable from a human's. Each year an annual prize of $2000 and a bronze medal is awarded to the **most** human computer. The winner of the annual contest is the best entry relative to other entries that year, irrespective of how good it is in an absolute sense.

Branches of AI

- **Logical AI**
- **Search**
- **Natural language processing**
- **Computer vision**
- **Pattern recognition**
- **Knowledge representation**
- **Inference** From some facts, others can be inferred.
- **Reasoning**
- **Learning**
- **Planning** To generate a strategy for achieving some goal
- **Epistemology** This is a study of the kinds of knowledge that are required for solving problems in the world.
- **Ontology** Ontology is the study of the kinds of things that exist.
- **Agents**
- **Games**
- **Artificial life / worlds?**
- **Emotions?**
- **Knowledge Management?**
- **Socialization/communication?**
- …

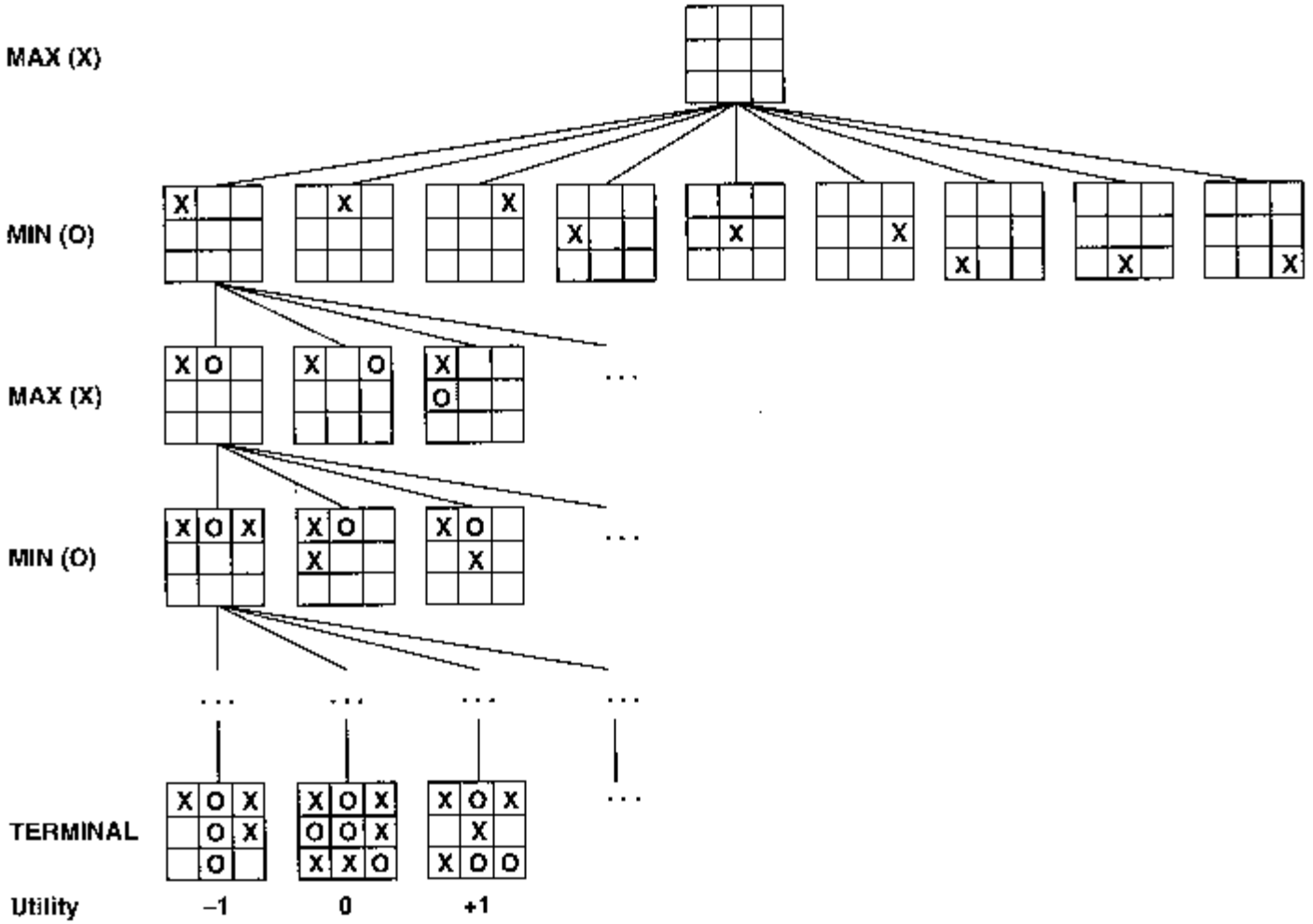
Approaches to AI

- Searching
- Learning
- From Natural to Artificial Systems
- Knowledge Representation and Reasoning
- Expert Systems and Planning
- Communication, Perception, Action

Search

- "All AI is search"
  - Game theory
  - Problem spaces
- Every problem is a feature space of all possible (successful or unsuccessful) solutions.
- The trick is to find an efficient search strategy.

Search: Game Theory



9!+1 = 362,880

Approaches to AI

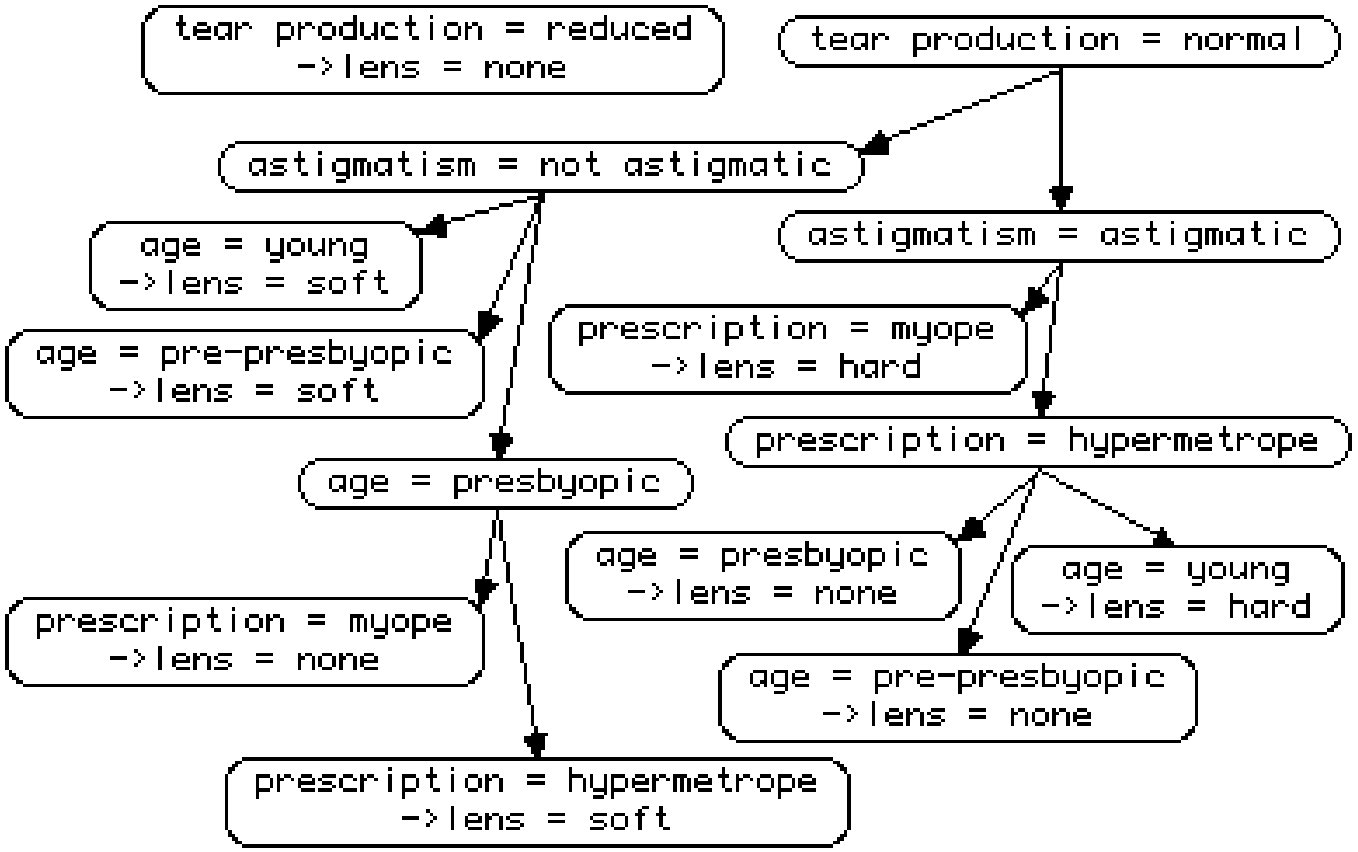
- Searching
- Learning
- From Natural to Artificial Systems
- Knowledge Representation and Reasoning
- Expert Systems and Planning
- Communication, Perception, Action

Learning

- Discovery
- Data Mining
- Neural Nets
- Case Based Reasoning

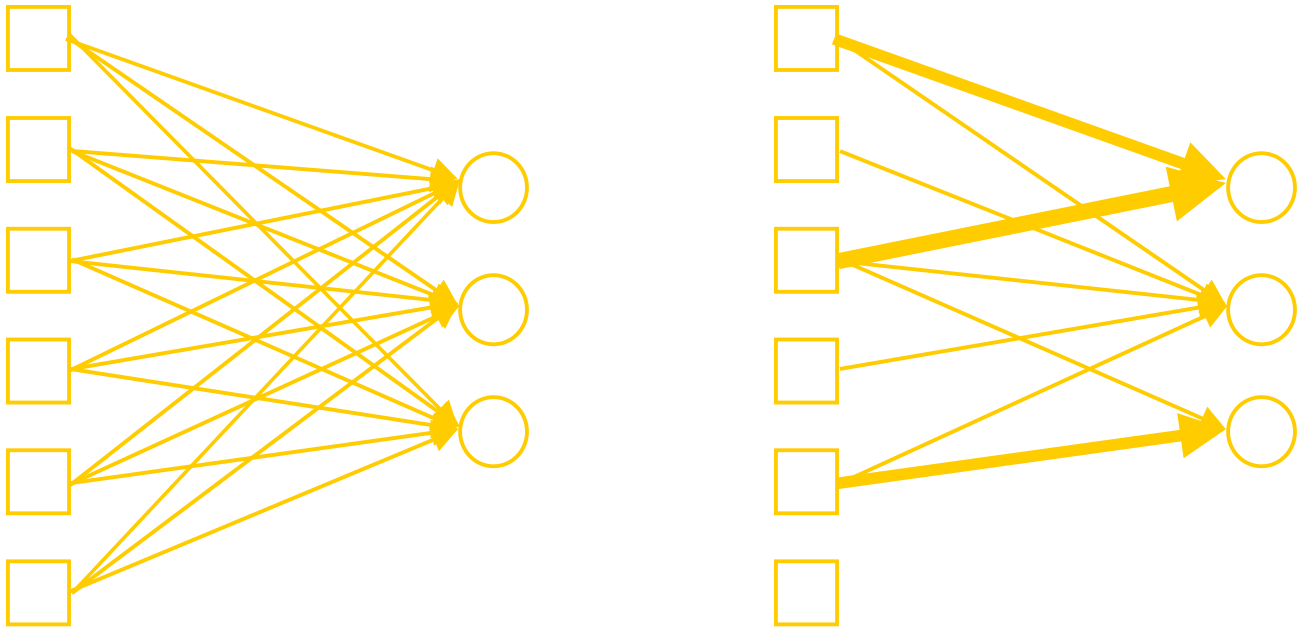
Learning

- Cases to rules



Learning

- Neural nets

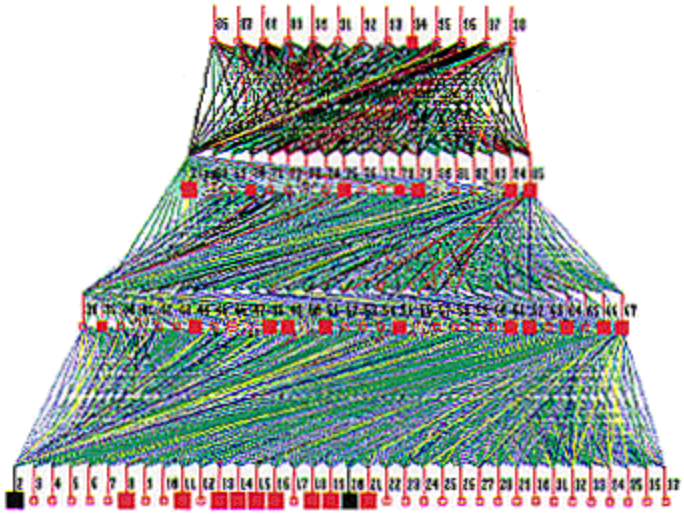


Approaches to AI

- Searching
- Learning
- From Natural to Artificial Systems
- Knowledge Representation and Reasoning
- Expert Systems and Planning
- Communication, Perception, Action

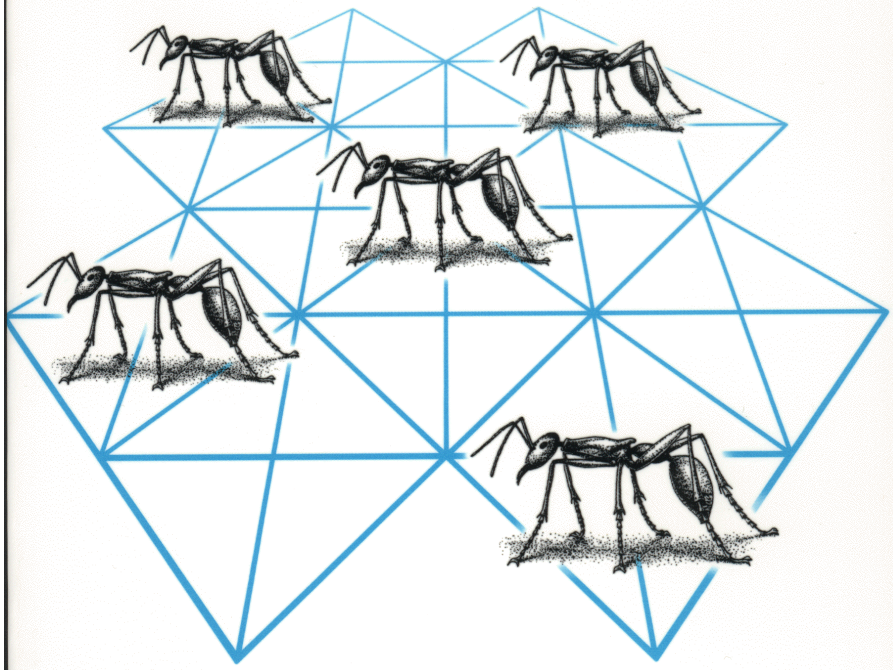
Neural Networks

→



Swarm Intelligence

From Natural to Artificial Systems



Eric Bonabeau
Marco Dorigo
Guy Theraulaz

A VOLUME IN THE
SANTA FE INSTITUTE STUDIES IN THE SCIENCES OF COMPLEXITY

Approaches to AI

- Searching
- Learning
- From Natural to Artificial Systems
- Knowledge Representation and Reasoning
- Expert Systems and Planning
- Communication, Perception, Action

Rule-Based Systems

- Logic Languages
  - Prolog, Lisp
- Knowledge bases
- Inference engines

Rule-Based Languages: Prolog

Father(abraham, isaac).
Father(haran, lot).
Father(haran, milcah).
Father(haran, yiscah).

Male(isaac).
Male(lot).
Female(milcah).
Female(yiscah).

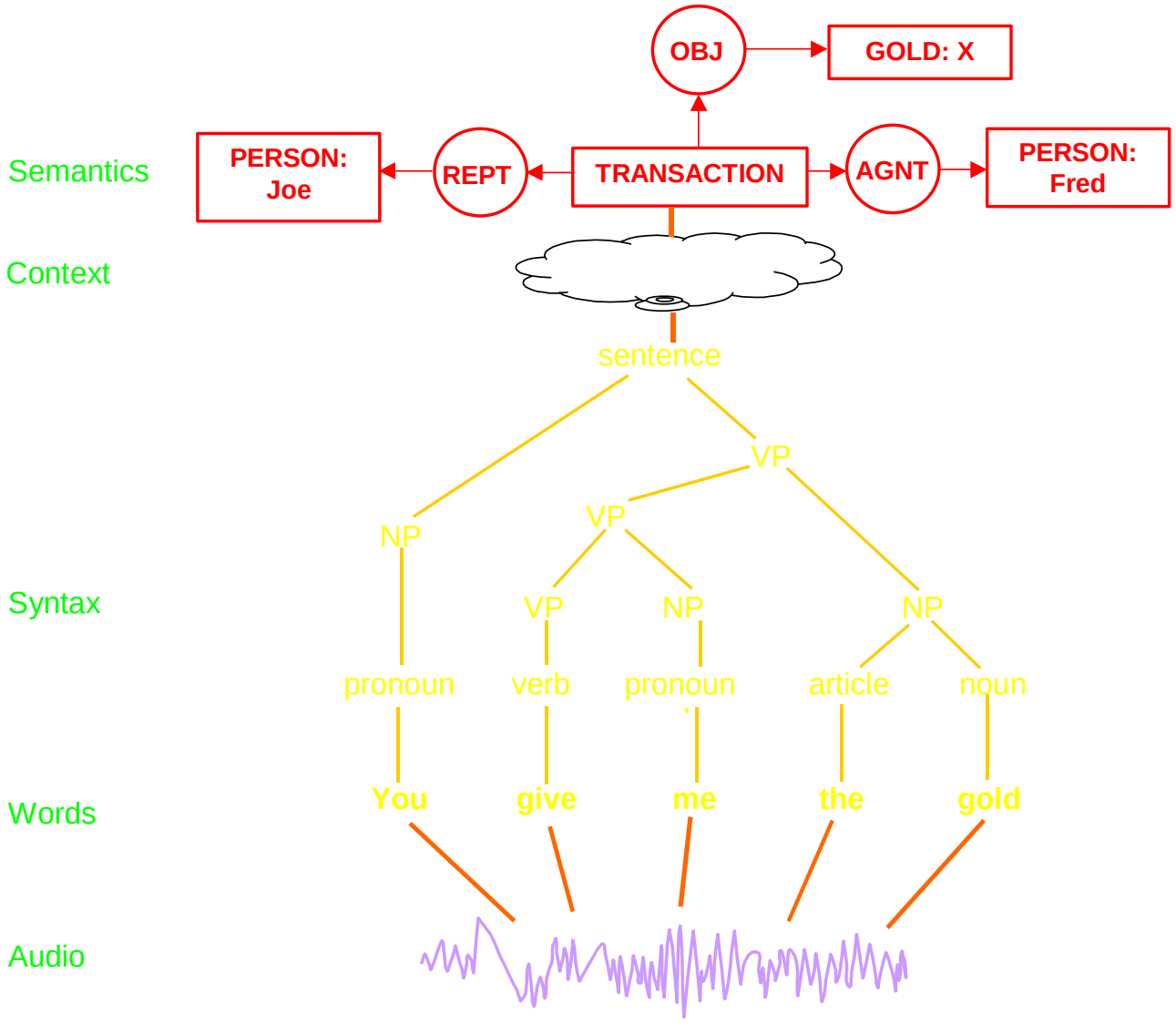
Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

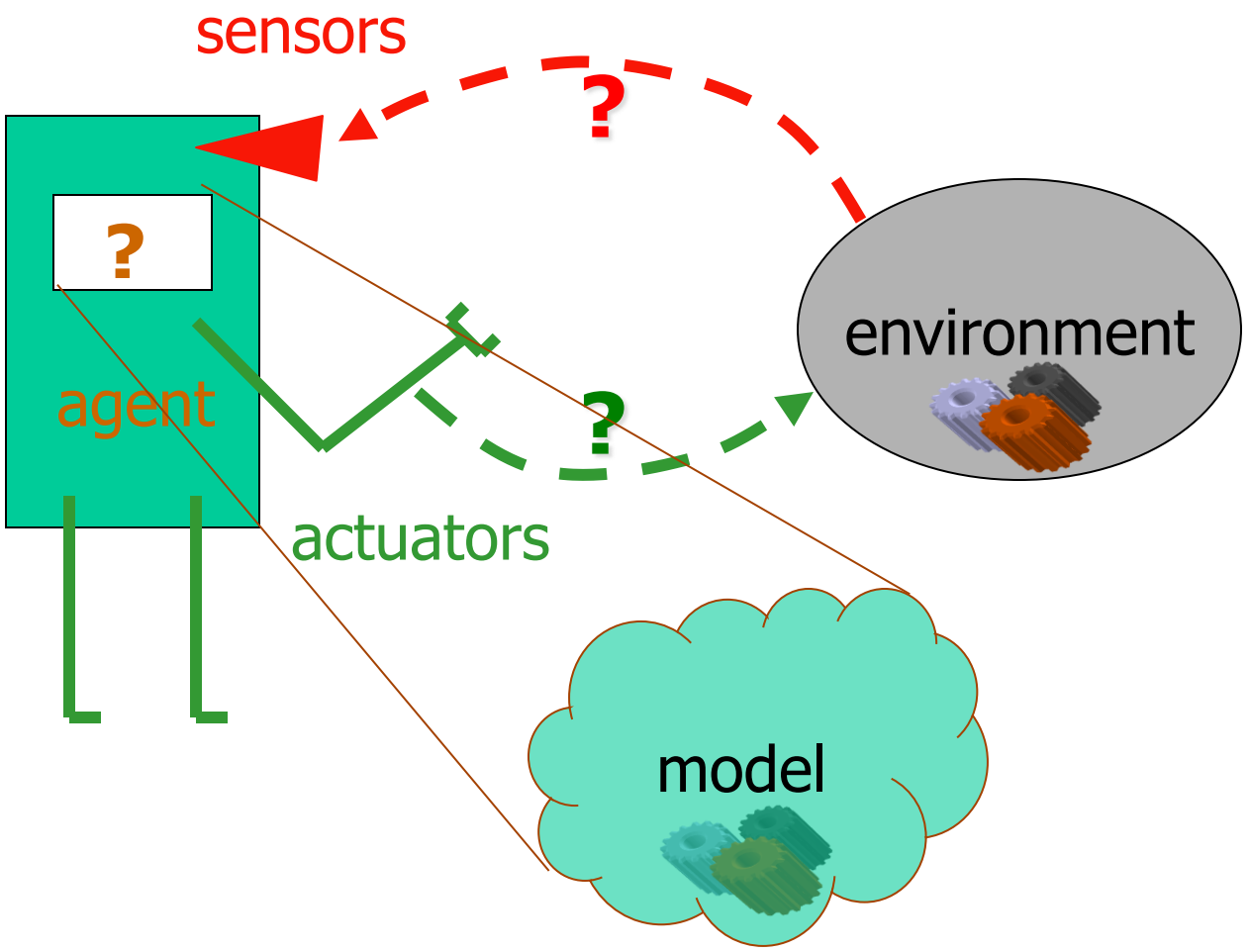
Ability-Based Areas

- Computer vision
- Natural language recognition
- Natural language generation
- Speech recognition
- Speech generation
- Robotics
- Games/entertainment

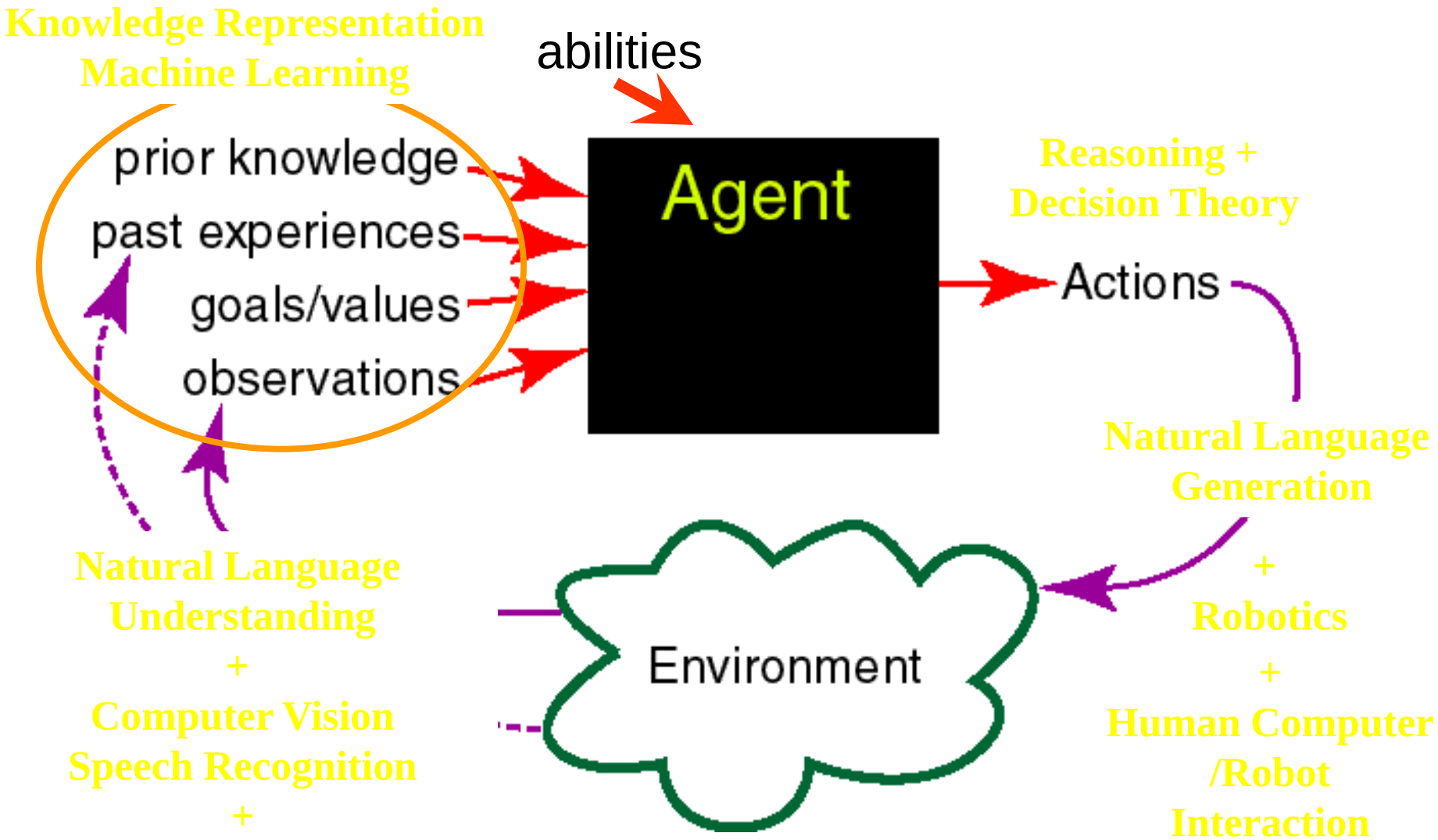
Natural Language Recognition



AI as an Agent



Intelligent Agents in the World



Thank you

- Credits Reference [1]: *Dr. C. Lee Giles, David Reese Professor, College of Information Sciences and Technology, Professor of Computer Science and Engineering, Professor of Supply Chain and Information Systems, The Pennsylvania State University, University Park, PA, USA. http://clgiles.ist.psu.edu*

Uninformed Search

- Breadth-first search
- Depth-first search
- Uniform cost search
- Depth-first iterative deepening

Key Procedures to be Defined

- EXPAND
  - Generate all successor nodes of a given node
- GOAL-TEST
  - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
  - Used to maintain a ranked list of nodes that are candidates for expansion

- Typical node data structure includes:
  - State at this node
  - Parent node
  - Operator applied to get to this node
  - Depth of this node (number of operator applications since initial state)
  - Cost of the path (sum of each operator application so far)

- Search process constructs a search tree, where
  - **root** is the initial state and
  - **leaf nodes** are nodes
    - not yet expanded (i.e., they are in the list “nodes”) or
    - having no successors (i.e., they’re “deadends” because no operators were applicable and yet they are not goals)

Search tree may be infinite because of loops even if state space is small

Evaluating Search Strategies

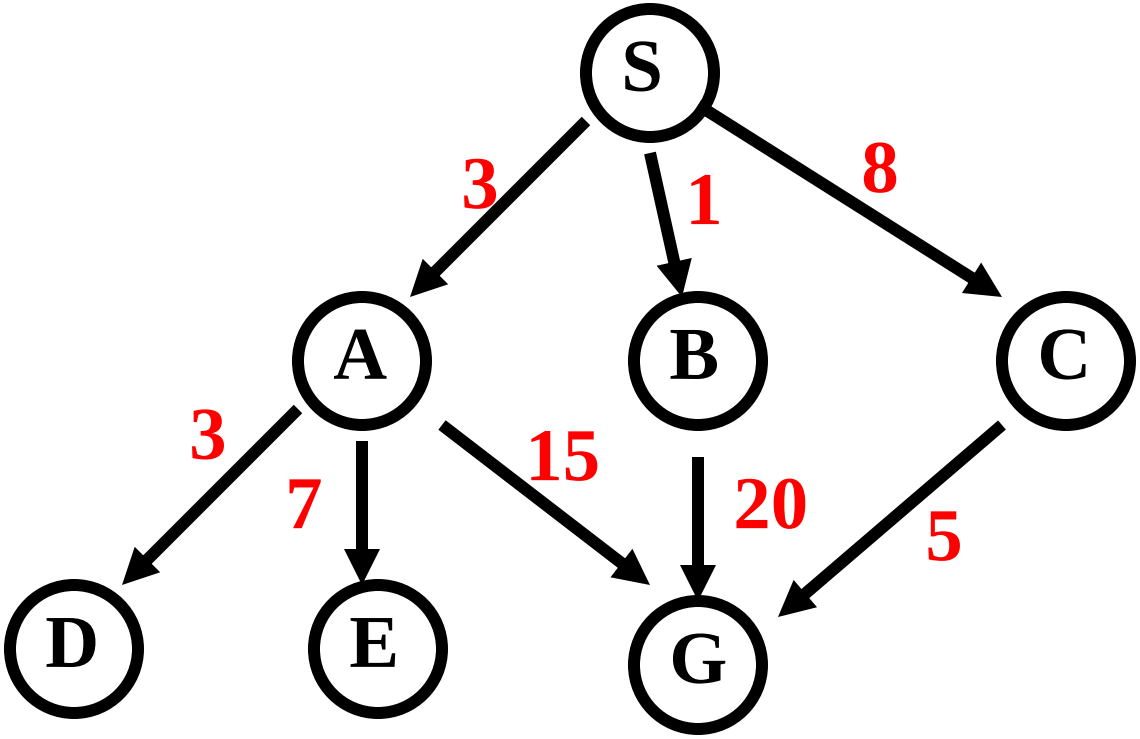
- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity**
  - How long (worst or average case) does it take to find a solution? Usually measured in terms of the number of nodes expanded
- **Space complexity**
  - How much space is used by the algorithm? Usually measured in terms of the maximum size of the “nodes” list during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one? That is, is it the one with minimum cost?

Uninformed vs. Informed Search

- **Uninformed search strategies**
  - Also known as “blind search,” uninformed search strategies use no information about the likely “direction” of the goal node(s)
  - Uninformed search methods: Breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional
- **Informed search strategies**
  - Also known as “heuristic search,” informed search strategies use information about the domain to (try to) (usually) head in the general direction of the goal node(s)
  - Informed search methods: Hill climbing, best-first, greedy search, beam search, A, A*

Uninformed Search Methods

Example for Illustrating Search Strategies



Depth-First Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }

Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Breadth-First Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ B^1 C^8 D^6 E^{10} G^{18} }
B^1	{ C^8 D^6 E^{10} G^{18} G^{21} }
C^8	{ D^6 E^{10} G^{18} G^{21} G^{13} }
D^6	{ E^{10} G^{18} G^{21} G^{13} }
E^{10}	{ G^{18} G^{21} G^{13} }
G^{18}	{ G^{21} G^{13} }

Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7

Uniform-Cost Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ B^1 A^3 C^8 }
B^1	{ A^3 C^8 G^{21} }
A^3	{ D^6 C^8 E^{10} G^{18} G^{21} }
D^6	{ C^8 E^{10} G^{18} G^1 }
C^8	{ E^{10} G^{13} G^{18} G^{21} }
E^{10}	{ G^{13} G^{18} G^{21} }
G^{13}	{ G^{18} G^{21} }

Solution path found is S B G, cost 13

Number of nodes expanded (including goal node) = 7

Breadth-First

- Enqueue nodes on nodes in **FIFO** (first-in, first-out) order.
- **Complete**
- **Optimal** (i.e., admissible) if all operators have the same cost. Otherwise, not optimal but finds solution with shortest path length.
- **Exponential time and space complexity**, $O(b^d)$, where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node
- Will take a **long time to find solutions** with a large number of steps because must look at all shorter length possibilities first
  - A complete search tree of depth d where each non-leaf node has b children, has a total of $1 + b + b^2 + ... + b^d = (b^{(d+1)} - 1)/(b-1)$ nodes
  - For a complete search tree of depth 12, where every node at depths 0, ..., 11 has 10 children and every node at depth 12 has 0 children, there are $1 + 10 + 100 + 1000 + ... + 10^{12} = (10^{13} - 1)/9 = O(10^{12})$ nodes in the complete search tree. If BFS expands 1000 nodes/sec and each node uses 100 bytes of storage, then BFS will take 35 years to run in the worst case, and it will use 111 terabytes of memory!

Depth-First (DFS)

- Enqueue nodes on nodes in **LIFO** (last-in, first-out) order. That is, nodes used as a stack data structure to order nodes.
- **May not terminate** without a "depth bound," i.e., cutting off search below a fixed depth D ("depth-limited search")
- **Not complete** (with or without cycle detection, and with or without a cutoff depth)
- **Exponential time**, $O(b^d)$, but only **linear space**, $O(bd)$
- Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)
- When search hits a deadend, can only back up one level at a time even if the "problem" occurs because of a bad operator choice near the top of the tree. Hence, only does "chronological backtracking"

Uniform-Cost (UCS)

- Enqueue nodes by **path cost**. That is, let g(n) = cost of the path from the start node to the current node n. Sort nodes by increasing value of g.
  - Identical to breadth-first search if all operators have equal cost
- Called *"Dijkstra's Algorithm"* in the algorithms literature and similar to *"Branch and Bound Algorithm"* in operations research literature
- **Exponential time and space complexity**, $O(b^d)$

Depth-First Iterative Deepening (DFID)

- First do DFS to depth 0 (i.e., treat start node as having no successors), then, if no solution found, do DFS to depth 1, etc.

  until solution found do

  DFS with depth cutoff c

  c = c+1

- **Complete**
- **Optimal/Admissible** if all operators have the same cost. Otherwise, not optimal but guarantees finding solution of shortest length (like BFS).
- Time complexity is a little worse than BFS or DFS because nodes near the top of the search tree are generated multiple times, but because almost all of the nodes are near the bottom of a tree, the worst case time complexity is still exponential, $O(b^d)$

Depth-First Iterative Deepening

- If branching factor is b and solution is at depth d, then nodes at depth d are generated once, nodes at depth d-1 are generated twice, etc.
  - Hence $b^d + 2b^{(d-1)} + ... + db <= b^d / (1 - 1/b)^2 = O(b^d)$.
  - If b=4, then worst case is $1.78 * 4^d$, i.e., 78% more nodes searched than exist at depth d (in the worst case).
- **Linear space complexity**, O(bd), like DFS
- Has advantage of BFS (i.e., completeness) and also advantages of DFS (i.e., limited space and finds longer paths more quickly)
- Generally preferred for **large state spaces** where **solution depth is unknown**

How they Perform

- **Depth-First Search:**
  - Expanded nodes: S A D E G
  - Solution found: S A G (cost 18)
- **Breadth-First Search**:
  - Expanded nodes: S A B C D E G
  - Solution found: S A G (cost 18)
- **Uniform-Cost Search**:
  - Expanded nodes: S A D B C E G
  - Solution found: S B G (cost 13)

  This is the only uninformed search that worries about costs.
- **Iterative-Deepening Search**:
  - nodes expanded: S S A B C S A D E G
  - Solution found: S A G (cost 18)

Thank you

- Credits Reference [1]: *https://www.cs.swarthmore.edu/~eeaton/teaching/cs63*

Shortest path using $\mathcal{A}^*$ Algorithm

Phaneendhar Reddy Vanam
Computer Science
Indiana State University
Terre Haute, IN, USA

December 13, 2011

Abstract

The main aim of this project is to find the shortest path using $\mathcal{A}^*$ **Algorithm.**

Contents

1 Introduction

The $\mathcal{A}^*$ Algorithm is a best-first search algorithm that finds the least cost path from an initial configuration to a final configuration.The most essential part of the $\mathcal{A}^*$ Algorithm is a good heuristic estimate function. This can improve the efficiency and performance of the algorithm.It is an extension of Dijkstra.s algorithm.$\mathcal{A}^*$ algorithm uses the function $f(n) = g(n) + h(n)$.

* $f(n) = g(n) + h(n)$.
* $g(n)$ is the path-cost function, which is the cost from the starting node to the current node.
* $h(n)$ is the heuristic estimate of the distance to the goal.

$\mathcal{A}^*$ Algorithm guides an optimal path to a goal if the heuristic function $h(n)$ is admissible.In this projct I am going to explain the algorithm and how this algorithm is going to implemented.

1.1 Applications

The Real time applications of $\mathcal{A}^*$ Algorithm are:

- $\mathcal{A}^*$ mainly used in Computer Gaming, Robotics and Google maps.
- The $\mathcal{A}^*$ for hueristic search is applied to construct a Neural Network structure(NS).

2 Statement of the problem

$\mathcal{A}^*$ depends on the heuristic.$\mathcal{A}^*$ is faster and give good results if we have a good heuristic.In this project I am going to path finding problems, that is, planning routes from a start node to some goal nodes in a graph.Such problems arise in many fields of technology, for example, production planning, message routing in large networks, resource allocation and vehicle navigation systems.I concentrate mostly on planning a minimum cost path using the $\mathcal{A}^*$ algorithm.

In some cases, $\mathcal{A}^*$ is an optimal method in a large class of algorithms.This means, roughly speaking that $\mathcal{A}^*$ explores a smaller region of the search space than the other algorithms in the given class.

A heuristic controls the search of $\mathcal{A}^*$ so that unnecessary branches of the tree of nodes that $\mathcal{A}^*$ visits are pruned.The new method also finds an optimal path to any node it visits for the first time so that every node will be visited only once.The later is an important property considering the efficiency of the search.

In some cases, the $\mathcal{A}^*$ is an optimal resource allocation method, which means that the number of the nodes the path finding algorithms together visit is minimized.

$\mathcal{A}^*$ gives the same results as Dijkstra, but faster when we use a good heuristic.$\mathcal{A}^*$ Algorithm has some conditions for to work correctly such as the estimated distance between current node and the final node should be lower than the real distance.$\mathcal{A}^*$ is guaranteed to give the shortest path when the heuristic is admissible.

3 History of $\mathcal{A}^*$ Algorithm

1. In 1964 Nils Nilsson invented a heuristic based approach to increase the speed of Dijkstra's algorithm.This algorithm was called A1.
2. In 1967 Bertram Raphael made dramatic improvements upon this algorithm, but failed to show optimality.He called this algorithm A2.
3. Then in 1968 Peter E.Hart introduced an argument that proved A2 was optimal when using a consistent heuristic with only minor changes.His proof of the algorithm also included a section that showed that the new A2 algorithm was the best algorithm possible given the conditions.
4. He thus named the new algorithm in Kleene star syntax to be the algorithm that starts with A and includes all possible version numbers or $\mathcal{A}^*$.

4 Steps involved in $\mathcal{A}^*$ algorithm

1. Let's characterize a class of admissible heuristic search strategies, using the evaluation function:$f(n) = g(n) + h(n)$.
2. $\mathcal{A}^*$ can be implemented more efficiently.roughly speaking, no node needs to be processed more than once.
3. As $\mathcal{A}^*$ traverses the graph, it follows a path of the lowest known cost, keeping a sorted priority queue of alternate path segments along the way.
4. If, at any point, a segment of the path being traversed has a higher cost than another encountered path segment, it abandons the higher-cost path segment and traverses the lower-cost path segment.
5. Starting with the initial node, it maintains a priority queue of nodes to be traversed, known as the open set.
6. The lower f(x) for a given node x, the higher its priority.

7. At each step of the $\mathcal{A}^*$ algorithm, the node with the lowest f(x) value is removed from the queue, the f and h values of its neighbors are updated accordingly, and these neighbors are added to the queue.

8. The $\mathcal{A}^*$ algorithm continues until a goal node has a lower f value than any node in the queue.

9. The f value of the goal is then the length of the shortest path, since h at the goal is zero in an admissible heuristic.If the actual shortest path is desired, the algorithm may also update each neighbor with its immediate predecessor in the best path found.

10. A closed set of nodes already traversed may be used to make the search more efficient.This process continues until the goal is reached.

Values of Heuristic i.e straight line distance to Bucharest

This is the example for A* search, I am going to consider the map of Romania and going to give step by step expalnation.

These are the hueristic values i.e straight line distance to Bucharest.

City	Hueristic value	City	Heuristic value
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Eforie	161	Pitesti	100
Fagaras	176	Rimnicu Vilcea	193
Dobreta	242	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

5 Diagrams representing stages in $\mathcal{A}^*$ search [1]

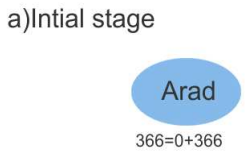


Figure 1: Example

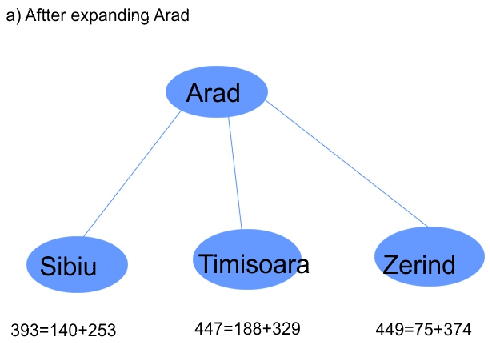


Figure 2: Example2

6 Algorithm

1. Put the start node **bs** in to **OPEN**
2. IF **OPEN** is empty THEN exit with failure.
3. Remove from **OPEN** and place in **CLOSED** a node **n** for which **f** is mimimum.
4. IF **n** is a goal node THEN exit sucessfully with the solution obtained by tracking back the pointers from **n** to **s**.
5. ElSE expand **n**, generating all its succesors,and attach them pointers back to **n**.

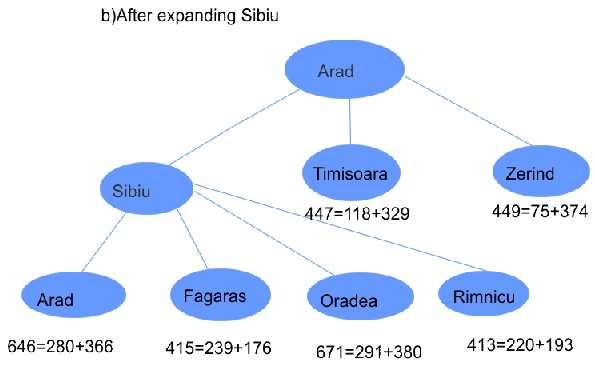


Figure 3: Example3

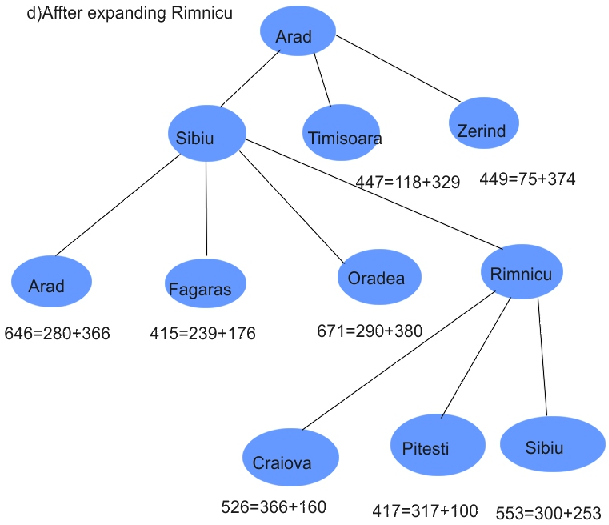


Figure 4: Example4

6. FOR every succesor **n'** of **n** DO IF **n** is not already in **OPEN** or **CLOSED** THEN estimate

$$h(n'),$$

and calculate

$$f(n') = g(n') + h(n')$$

where

$$g(n') = g(n) + c(n, n')$$

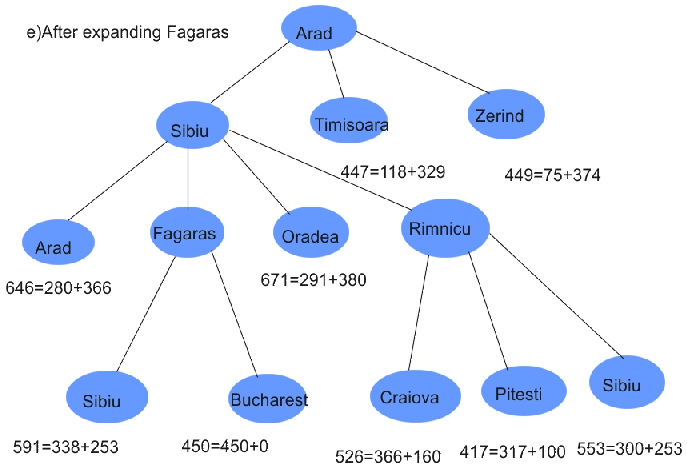


Figure 5: Example5

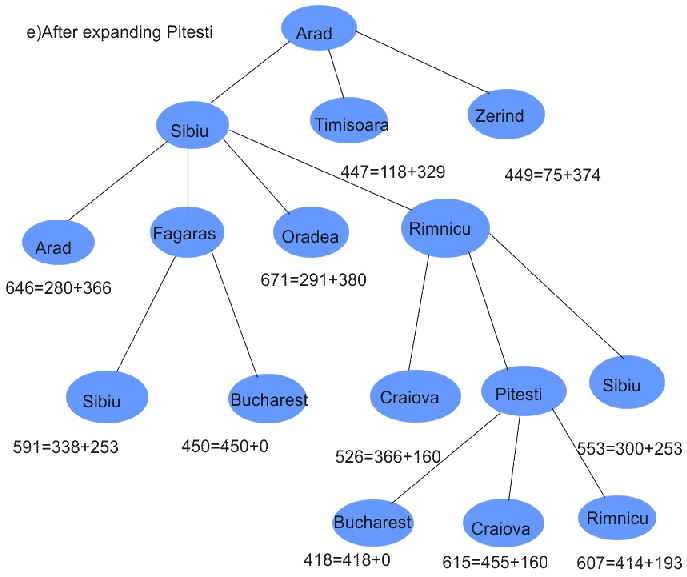


Figure 6: Example6

and put **n'** in to **OPEN**

7. IF **n** is already in **OPEN** or **CLOSED** THEN direct its pointer along the path yielding the lowest

$$g(n)$$

END FOR

8. GO TO step 2

7 Diagrams representing shortest path in Map of Romania [1]

$\mathcal{A}^*$ search in map of Romania

- This figure represents the intial map of Romania.The values representing in red colour are heuristic values(i.e h(n)).
- The values representing in silver colour are path cost values(i.e g(n)).
- The values representing in blue colour are f(n) values i.e

$$f(n) = g(n) + h(n).$$

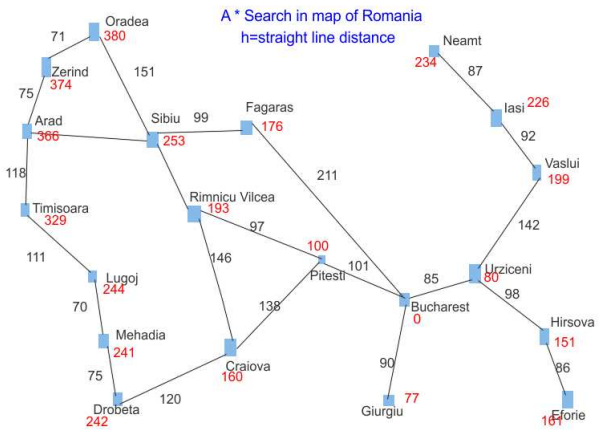


Figure 7: Intial map of Romania

7.1 After expanding Arad

- We have three nodes i.e Zerind, Sibiu and Timisoara.
- As we know f(n)=g(n)+h(n).
- f(Sibiu)=f(n)=140+253=393(g(n)=140 and h(n)=253).
- f(Zerind)=f(n)=75+374=449(g(n)=75 and h(n)=374).
- f(Timisoara)=f(n)=118+329=447(g(n)=118 and h(n)=329).
- From these nodes we have to choose the least f(n) value, so f(Sibiu) is least among these nodes.(Refer example 2.)

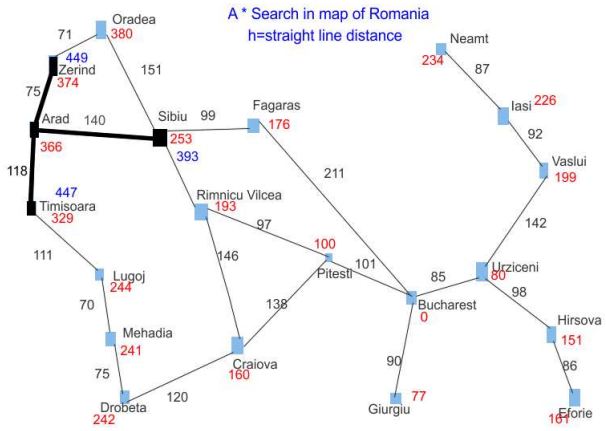


Figure 8: After expanding Arad

7.2 After expanding Sibiu

- After expanding Sibiu we have four nodes i.e Arad, Oradea, Fagaras and Rimnicu Vilcea.
- As we know f(n)=g(n)+h(n).
- f(Arad)=f(n)=280+366=646(g(n)=280 and h(n)=366).
- f(Oradea)=f(n)=291+380=671(g(n)=291 and h(n)=380).
- f(Fagaras)=f(n)=239+176=415(g(n)=239 and h(n)=176).
- f(Rimnicu)=f(n)=220+193=413(g(n)=220 and h(n)=193).
- From these nodes we have to choose the least f(n) value, so f(Rimnicu) is least among these nodes, f(Rimnicu)=413.(Refer example 3).

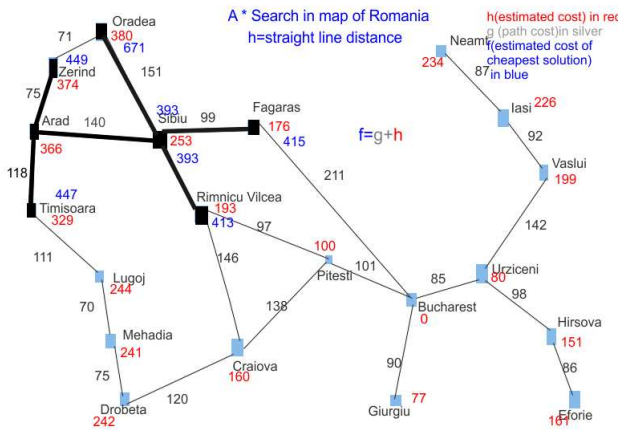


Figure 9: After expanding Sibiu

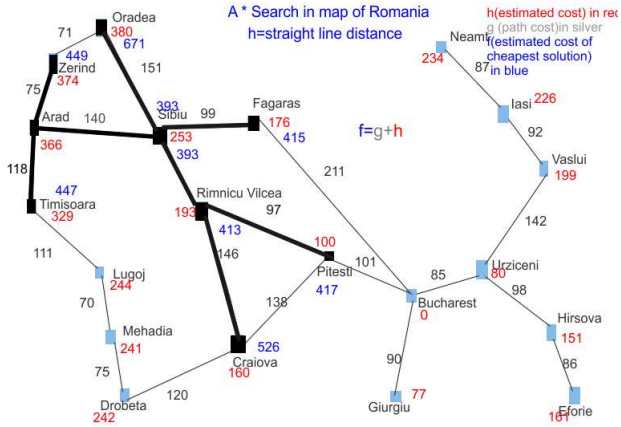


Figure 10: After expanding Rimnicu Vilcea

7.3 After expanding Rimnicu

- After expanding Rimnicu node we have three nodes i.e Craiova, Pitesti and Sibiu.
- As we know f(n)=g(n)+h(n).
- f(Craiova)=f(n)=366+160=526(g(n)=366 and h(n)=160).
- f(Pitesti)=f(n)=317+100=417(g(n)=317 and h(n)=100).
- f(Sibiu)=f(n)=300+253=553(g(n)=300 and h(n)=253).

- From these three nodes we have to choose the least f(n) value, so f(Fagaras) is least among the nodes(Refer exapmle 4.)

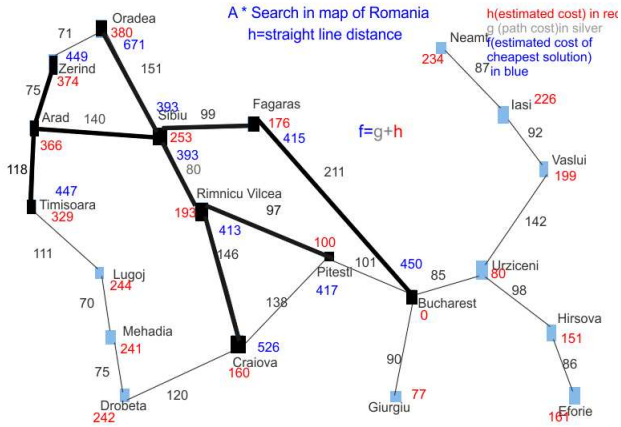


Figure 11: After expanding Fagaras

7.4 After expanding Fagaras

- After expanding Fagaras node we have two nodes i.e Sibiu and Bucharest.
- As we know f(n)=g(n)+h(n)..
- f(Sibiu)=f(n)=338+253=591(g(n)=338 and h(n)=253).
- f(Buchrest)=f(n)=450+0=450(g(n)=450 and h(n)=0).
- From these three nodes we have to choose the least f(n) value, so f(Fagaras) is least among the nodes(Refer exapmle 5.)

7.5 After expanding Pitesti

- After expanding Pitesti node we have three nodes i.e Bucharest, Craiova and Rimnicu.
- As we know f(n)=g(n)+h(n).
- f(Craiova)=f(n)=455+160=615(g(n)=455 and h(n)=160).
- f(Buchrest)=f(n)=418+0=418(g(n)=418 and h(n)=0).
- f(Rimnicu)=f(n)=414+193=607(g(n)=414 and h(n)=193).

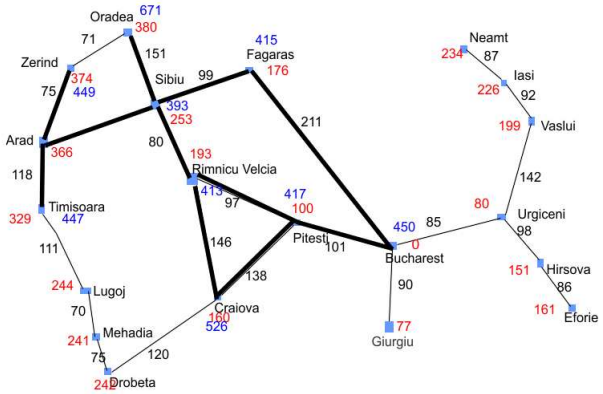


Figure 12: After expanding Pitesti

- From these three nodes we have to choose the least f(n) value, so f(Bucharest) is least among the nodes(Refer exapmle 6.)
- Since Bucharest is the goal node, so h(n)=0 at Bucharest.

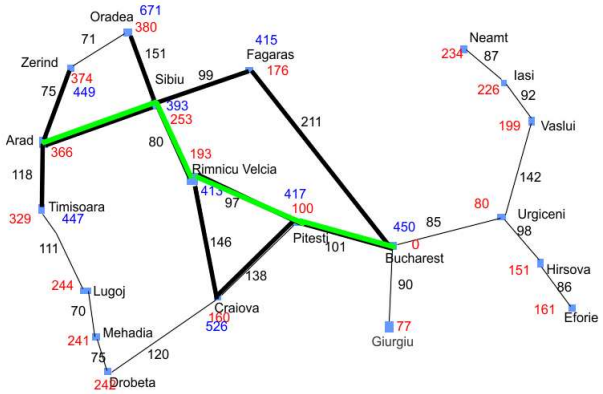


Figure 13: Shortest path from Arad to Bucharest

8 Time complexity

The time complexity of $\mathcal{A}^*$ depends on the heuristic.In the worst case, the number of nodes expanded is exponential in the length of the solution.

$$|h(x) - h^*(x)| = O(log(h)^*(x))$$

where h^* is the optimal heuristic and it is also defined as true cost of getting from n to the goal.For almost all heuristics in practical use, the error is at least proportional to the path cost, and the resulting exponential growth eventually.

8.1 Proving $\mathcal{A}^*$ is optimality

$\mathcal{A}^*$ is the name given to the algorithm where h(node) function is admissible.In the words it is guaranteed to provide underestimate of the true cost to the goal. $\mathcal{A}^*$ is optimal and complete.In the other words, It is guaranteed to find a solution, and the solution is guaranteed to be the best solution. $\mathcal{A}^*$ is complete if the graph it is searching is locally finite.

Proof of $\mathcal{A}^*$ is complete:

Conside two Goals G1 and G2.

The path cost of G1 is f1.

The path cost of G2 is f2, where

$$f2 > f1$$

G1 is the goal with lower cost.Let us assume A* algorithm reached G2 without exploring G1.Let us conside node n, that is an optimal path from root node to G1, the h is admissible heuristics.

$$f1 >= f1(n).$$

the only reason algorithm would not choose to expand n before it reaches G2 would be

$$f(n) >= f(G2).$$

By combining above 2 equations we get

$$f1 >= f(G2)$$

since G2 is goal state

$$h(G2) = 0$$

and thus

$$f(G2) = g(G2)$$

Thus we have

$$f1 >= g(G2)$$

. This, there fore contradicts our orginal assumption that G2 had a higher path cost than G1, Which proves that $\mathcal{A}^*$ can only choose least cost path to a goal.

8.2 Proving $\mathcal{A}^*$ is complete

As said before $\mathcal{A}^*$ expands in incresong f, it must expand to reach a goal state.This is true, of course, unless there are infinetely many nodes with

$$f(n) < f*$$

the only way there could be infinite number of nodes is either there is a node with an infinite branching factor or there is a path with finite path cost but an infinite number of nodes along it.Thus the correct statement is that $\mathcal{A}^*$ is complete on locally finite graphs.

d	IDS	A*(h1)	A*(h2)
2	2.45	1.79	1.79
4	2.87	1.48	1.45
6	2.73	1.34	1.30
8	2.80	1.33	1.24
10	2.79	1.38	1.22
12	2.78	1.42	1.24

Comparing effective branching factor of $\mathcal{A}^*$ algorithm with h1/h1.

8.3 Heuristic accuracy on perfromance

Consider the 8-puzzle problem.let h1=number of tiles that are in wrong position.Generally it has none of the tiles in the goal position, so the start state would be have h1=8, h2=the sum of the distances of tiles from their goal positions.Because tiles cannot move along diagonals, the distance will count is the sum of the horizontal and vertical distances.This is called manhattan distance.

d	IDS	A*(h1)	A*(h2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73

Comparing search cost of $\mathcal{A}^*$ algorithm with h1/h2

we can charcterize the quality of a heuristic is the effective branching factor b*.if the total nodes expanded by $\mathcal{A}^*$ for a particular problem is N and the solution depth is d, the b* is the branching factor that a uniform tree of depth d would have in order to contain N nodes.

$$N = 1 + b^* + (b^*)^2 + (b^*)^3 + ...(b*)^d.$$

A well designed heuristic would have a value of

$$b^* = 1$$

To test the heuristic functions h1 and h2, I randomly generated 100 problems each with solution lenth 2,4.....12 and solved them using $\mathcal{A}^*$ search with h1 and h2.This shows how h2 is better than h1, The above statement is always true for any node n.

$$h2(n) > h1(n)$$

$\mathcal{A}^*$ using h2 will expands fewer nodes tha A* using h1.

References

[1] Stuart Russell,Peter Norvig *Artificial intelligence a modern approach, Second Edition*

[2] David L.Poole and Alan K.Mackworth *Foundations of computational agents* Cambridge University Press, 2010

[3] Ben Coppin *Artificial Intelligence Illuminated*

[4] wikipedia A* search algorithm http://en.wikipedia.org/wiki/A*_algorithm

[5] A* algorithm for beginners http://www.policyalmanac.org/games/aStarTutorial.htm

State Space Search

- Nodes correspond to problem states
- Arcs correspond to steps in a solution process
- One node corresponds to an initial state
- One node corresponds to a goal state

Solution Path

An ordered sequence of nodes from the initial state to the goal state

Search Algorithm

Finds a solution path through a state space

Suppose we have

- An empty 4 gallon jug
- An empty 3 gallon jug
- A source of water
- A task: put 2 gallons of water in the 4 gallon jug

Representation

- State Space
- Node on the graph is an ordered pair (x,y)
  - X is the contents of the 4 gallon jug
  - Y is the contents of the 3 gallon jug
- Intitial State: (0,0)
- Goal State: (2,N) N ε {0, 1, 2, 3}

Rules

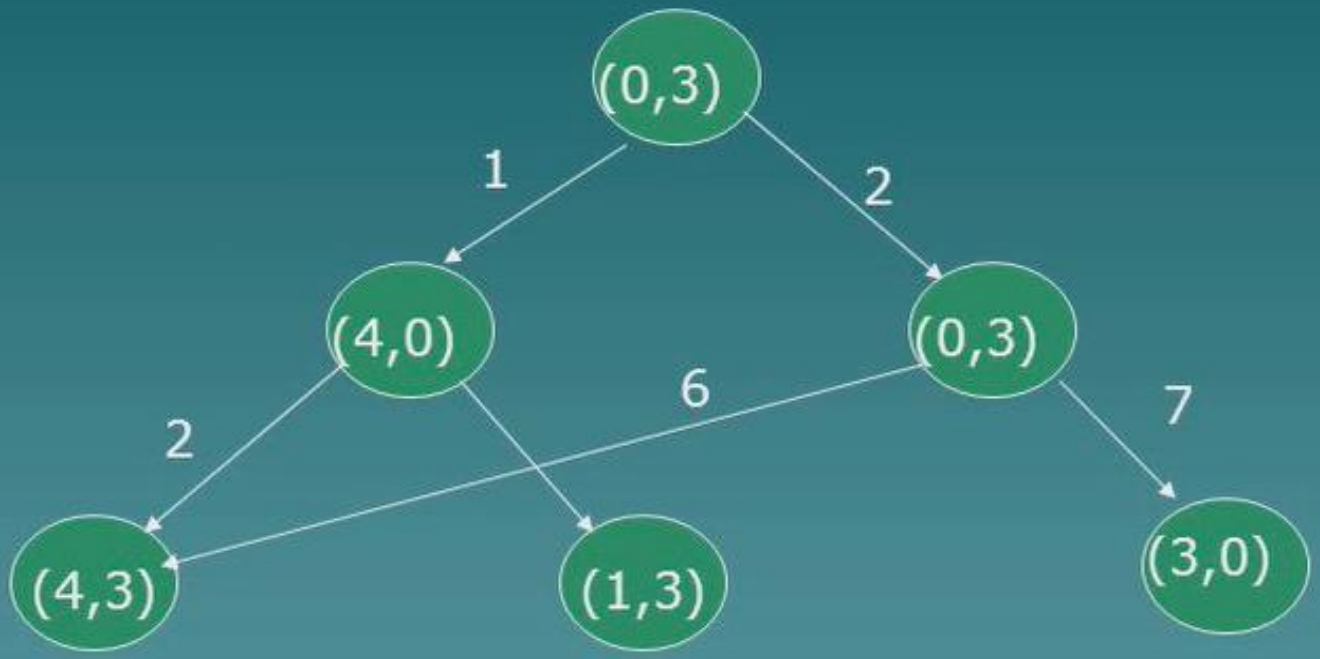
1. if x < 4, fill x : (x,y) → (4,y)
2. if y < 3, fill y : (x,y) → (x,3)
3. if x > 0, empty x : (x,y) → (0,y)
4. if y > 0, empty y : (x,y) → (x,0)
5. if (x+y) >= 4 and y > 0
   fill the 4 gallon jug from the 3 gallon jug
   (x,y) → (4, y – (4 – x))
6. if (x+y) >= 3 and x > 0
   Fill the 3 gallon jug from the 4 gallon jug
   (x,y) → (x –(3 – y), 3))
7. if (x+y) <= 4 and y > 0
   Pour the 3 gallon jug into the 4 gallon jug: (x,y) → (x+y), 0)
8. if (x+y) <= 3 and x > 0
   pour the 4 gallon jug into the 3 gallon jug: (x,y) → (0, x + y)

Is there a solution path?

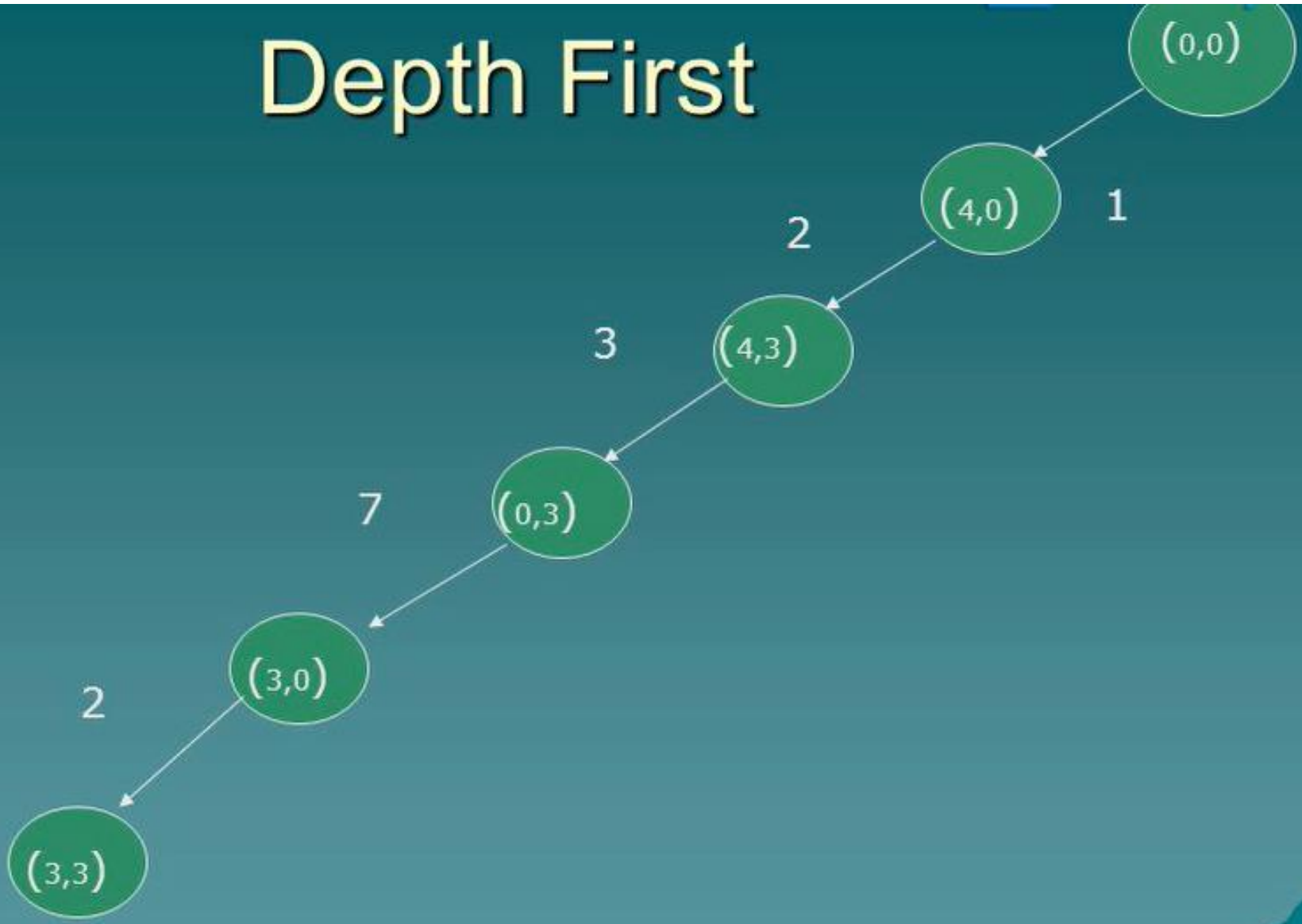
Initial State: (0,0)

Goal State: (2,N)

Breadth First Search



Depth First



Informed Search

- **Hill Climbing**
- **Best-First Search**
- **A***

Cost and Cost Estimation

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

- g(n) the cost (so far) to reach the node n
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal

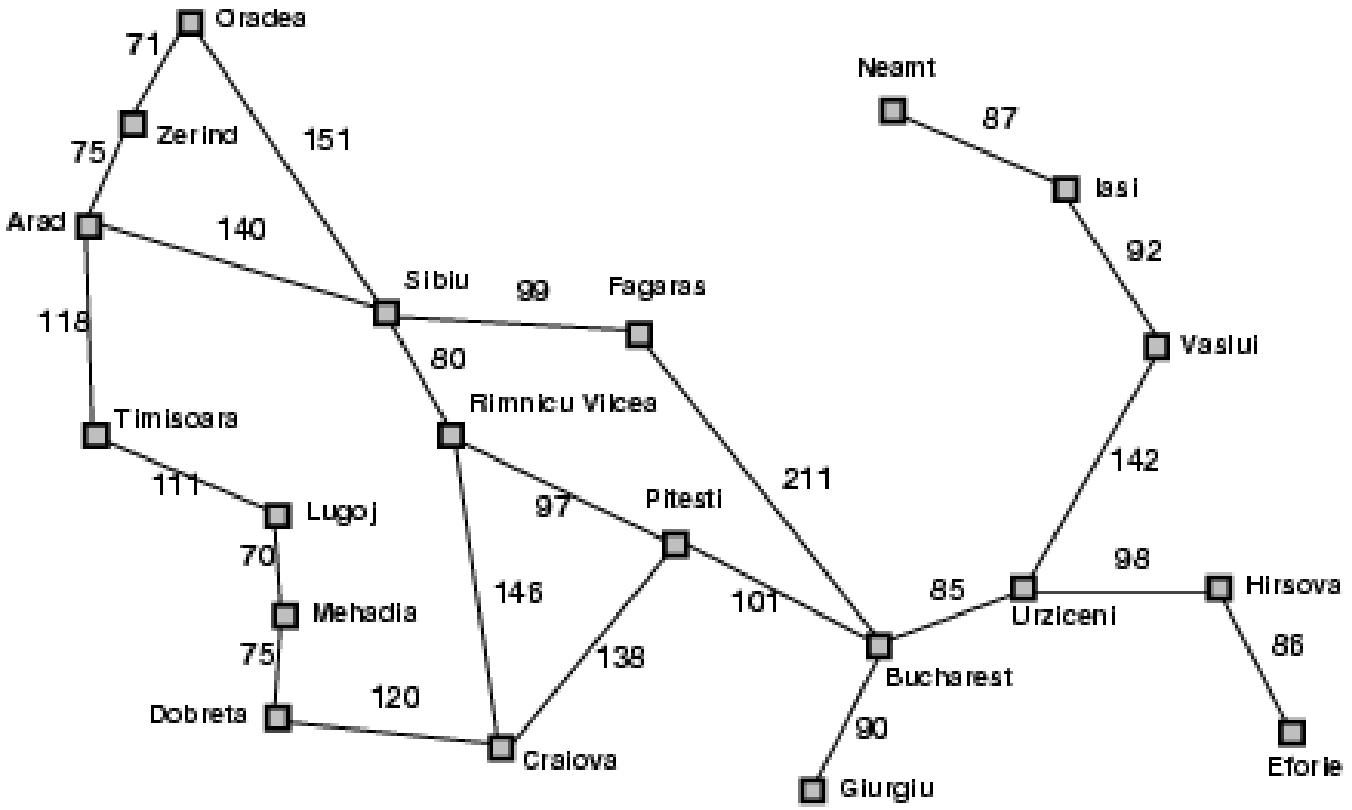
Best-First Search

- Special case of breadth-first search
- Uses h(n) = heuristic function as its evaluation function
- Ignores cost so far to get to that node (g(n))
- Expand the node that *appears* closest to goal

- Best First Search is complete
- Best First Search is not optimal
  - A solution can be found in a longer path (higher h(n) with a lower g(n) value)

- Special cases:
  - uniform cost search: f(n) = g(n) = path to n
  - A^* search

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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

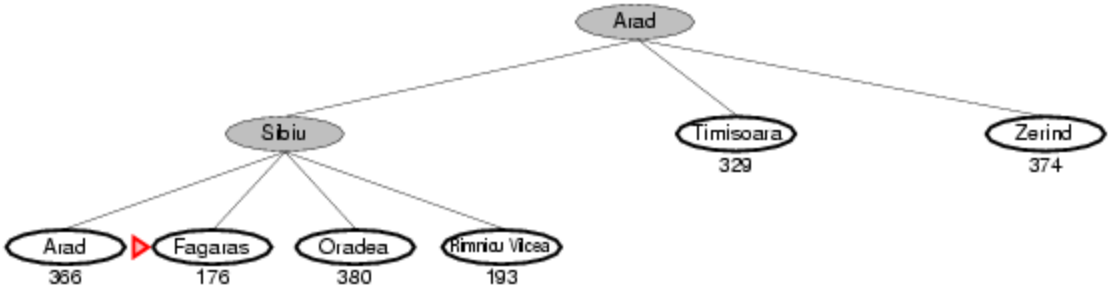
Greedy best-first search

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

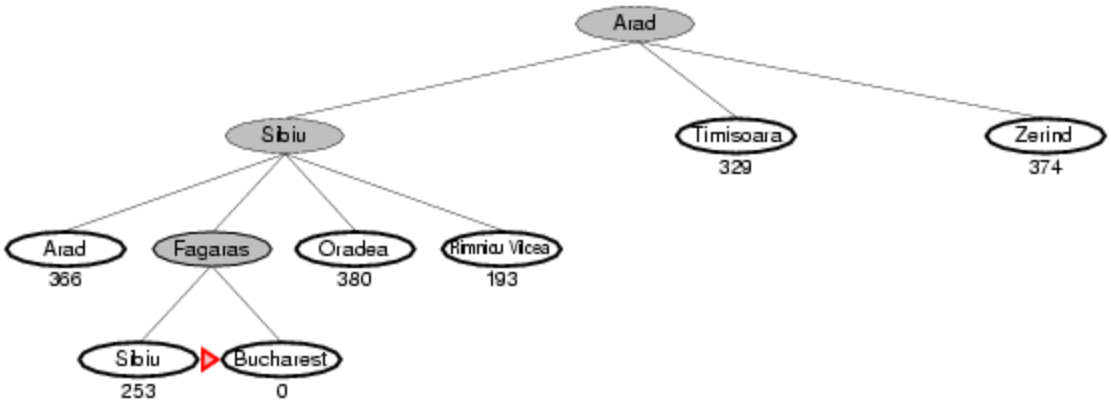
Greedy best-first search example

Greedy best-first search example

Greedy best-first search example



Greedy best-first search example



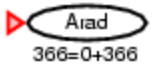
Properties of greedy best-first search

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

A* search

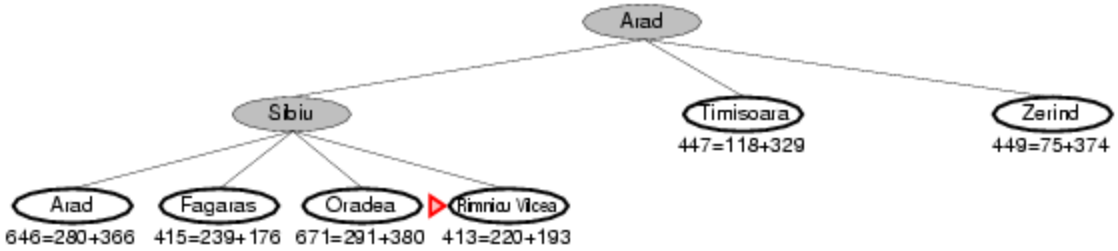
- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

A* search example

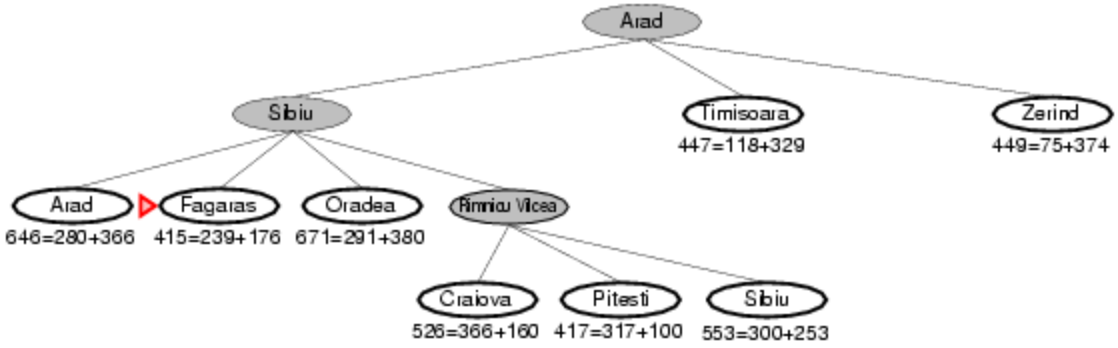


A* search example

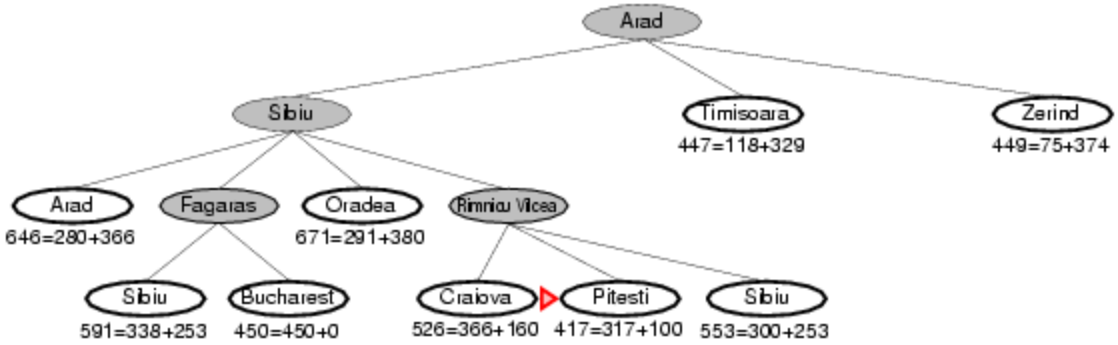
A* search example



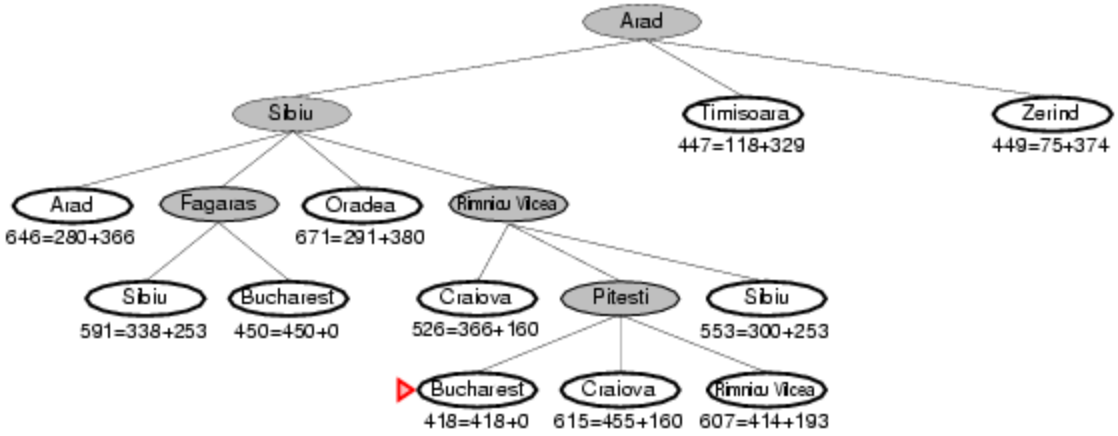
A* search example



A* search example



A* search example



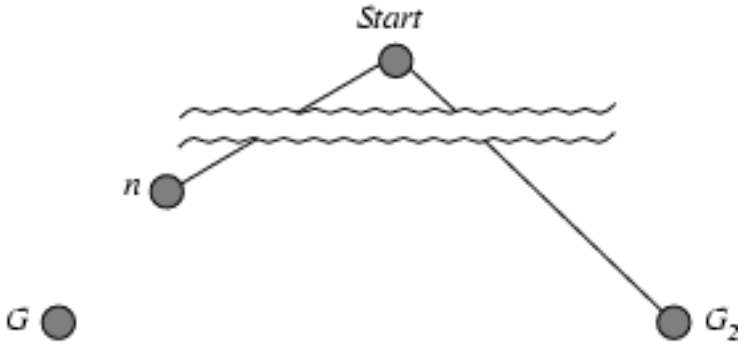
Admissible heuristics

- A heuristic *h(n)* is admissible if for every node *n*,
  h(n) ≤ $h^*(n)$, 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: $h_{SLD}(n)$ (never overestimates the actual road distance)

- Theorem: If *h(n)* is admissible, A* using `TREE-SEARCH` is optimal

Optimality of A* (proof)

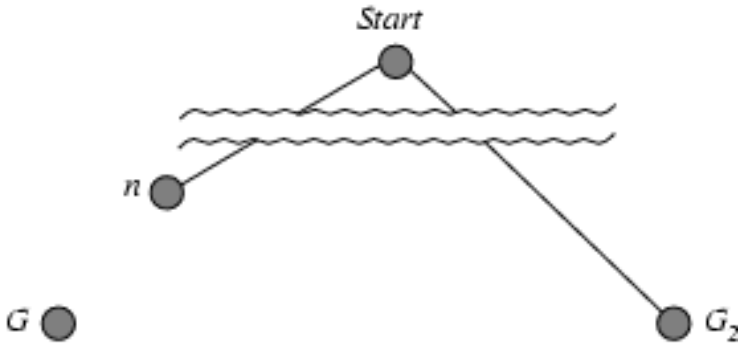
- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.
-



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.
-



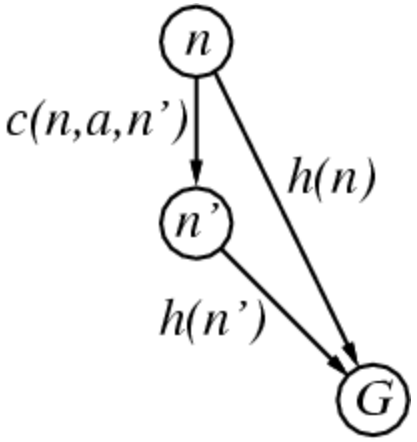
- $f(G_2)$ > $f(G)$ from above
- $h(n)$ ≤ $h^*(n)$ since h is admissible
- $g(n) + h(n)$ ≤ $g(n) + h^*(n)$
- $f(n)$ ≤ $f(G)$

Hence $f(G_2) > f(n)$, and A^* will never select G_2 for expansion

Consistent heuristics

- A heuristic 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, we have

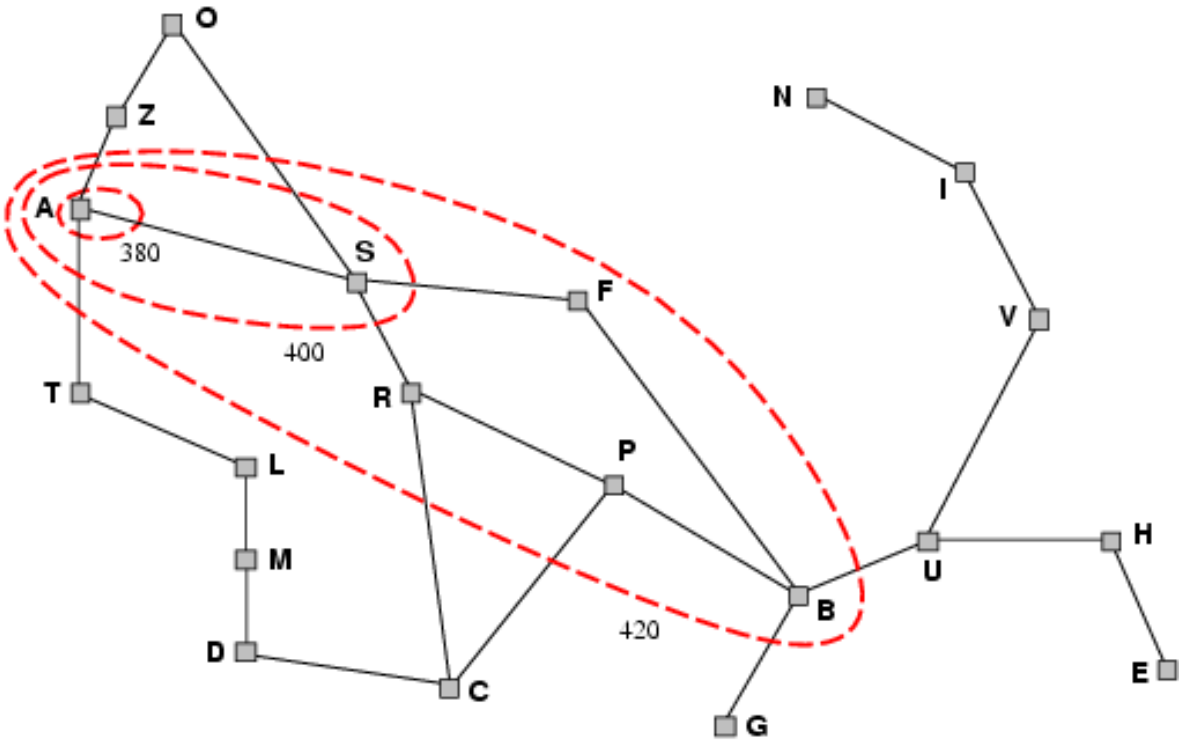
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

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

- Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal

Optimality of A*

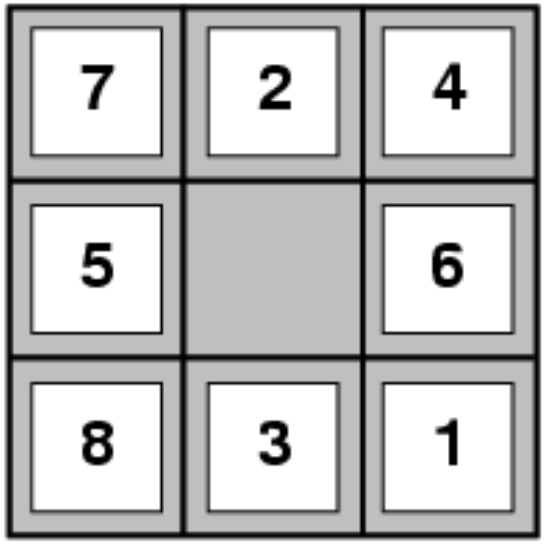
- A* expands nodes in order of increasing *f* value
- Gradually adds "*f*-contours" of nodes
- Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$



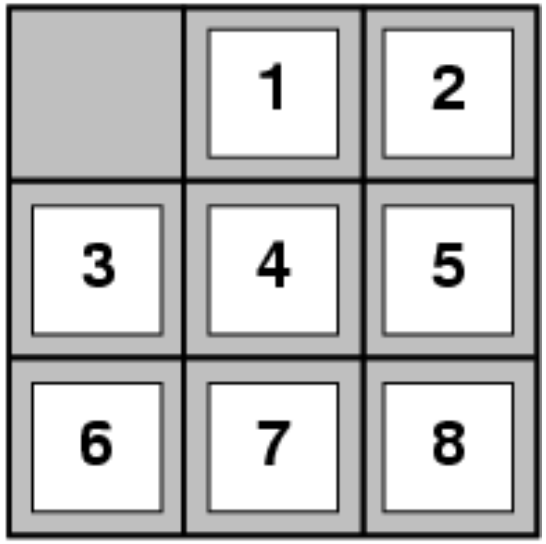
Properties of A*

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

8-Puzzle

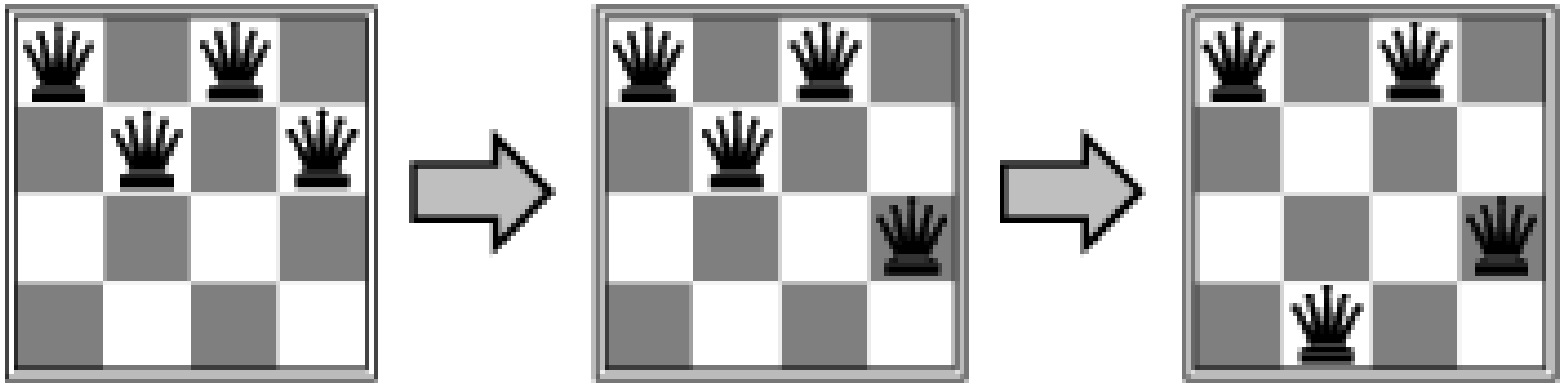


Start State



Goal State

n-queens

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal
- 

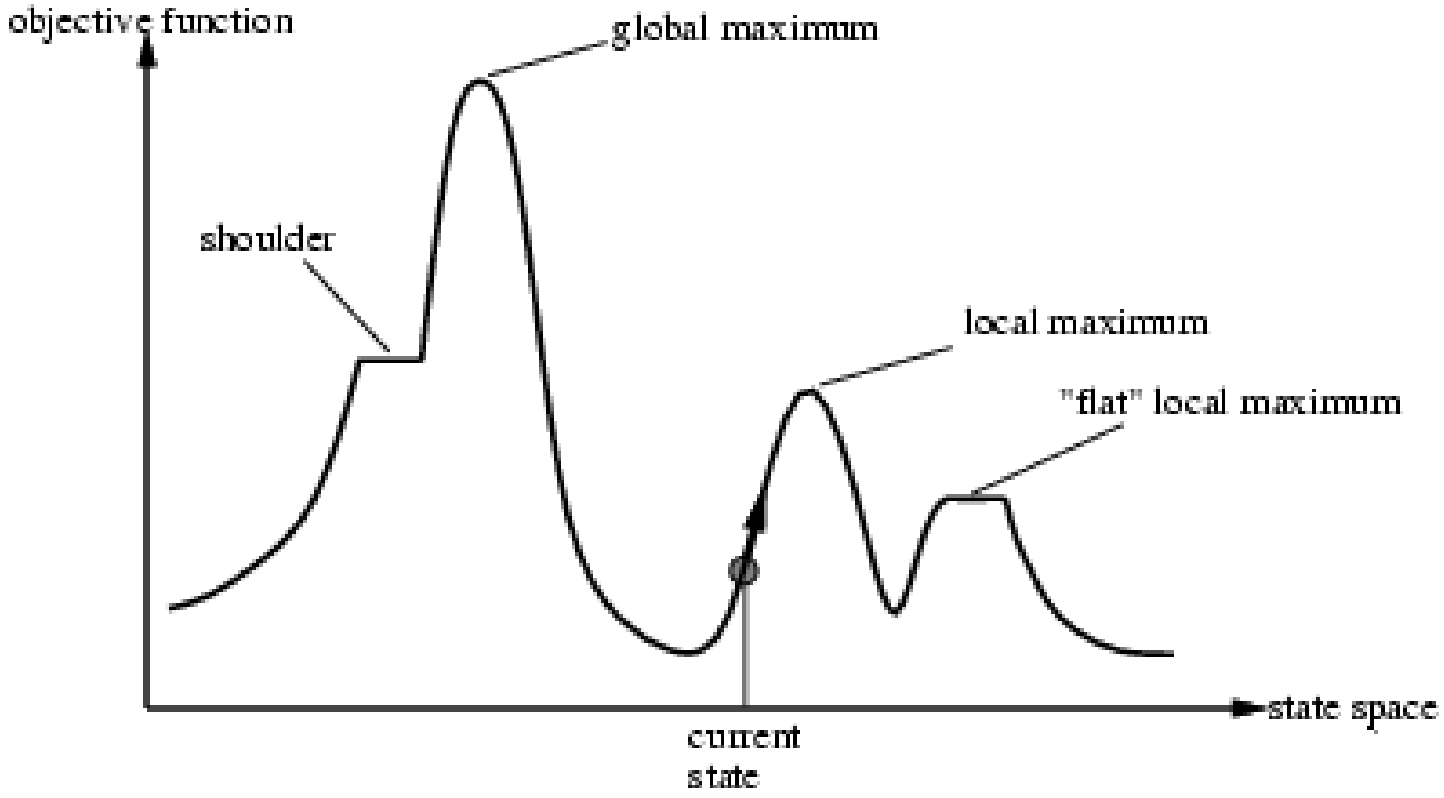
Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"
-

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

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

Hill-climbing search

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

Thank you

- Credits Reference [1]: *http://aima.eecs.berkeley.edu/2nd-ed/*

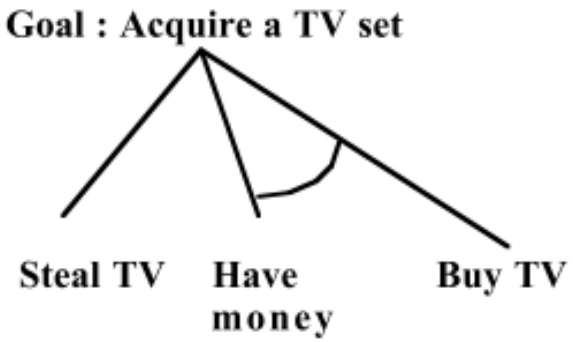
AND OR GRAPH

AND OR Graph

- The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph.

- The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do.

AND OR Graph



- OR graphs : generally used for data driven approach
- AND OR graphs: used for Goal driven approach
  - Problems solvable by decomposing into sub problems some of which is to be solved.
- Graph consisting of OR arcs and AND arcs
  - OR : the node has to be solved.
  - AND : all the nodes in the arc has to be solved

Problem Reduction with AO* Algorithm

- When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, AND-OR graphs or AND - OR trees are used for representing the solution.

- The decomposition of the problem or problem reduction generates AND arcs.

- One AND are may point to any number of successor nodes. All of these must be solved so that the arc will rise to many arcs, indicating several possible solutions. Hence the graph is known as AND - OR instead of AND.

Problem Reduction with AO* Algorithm

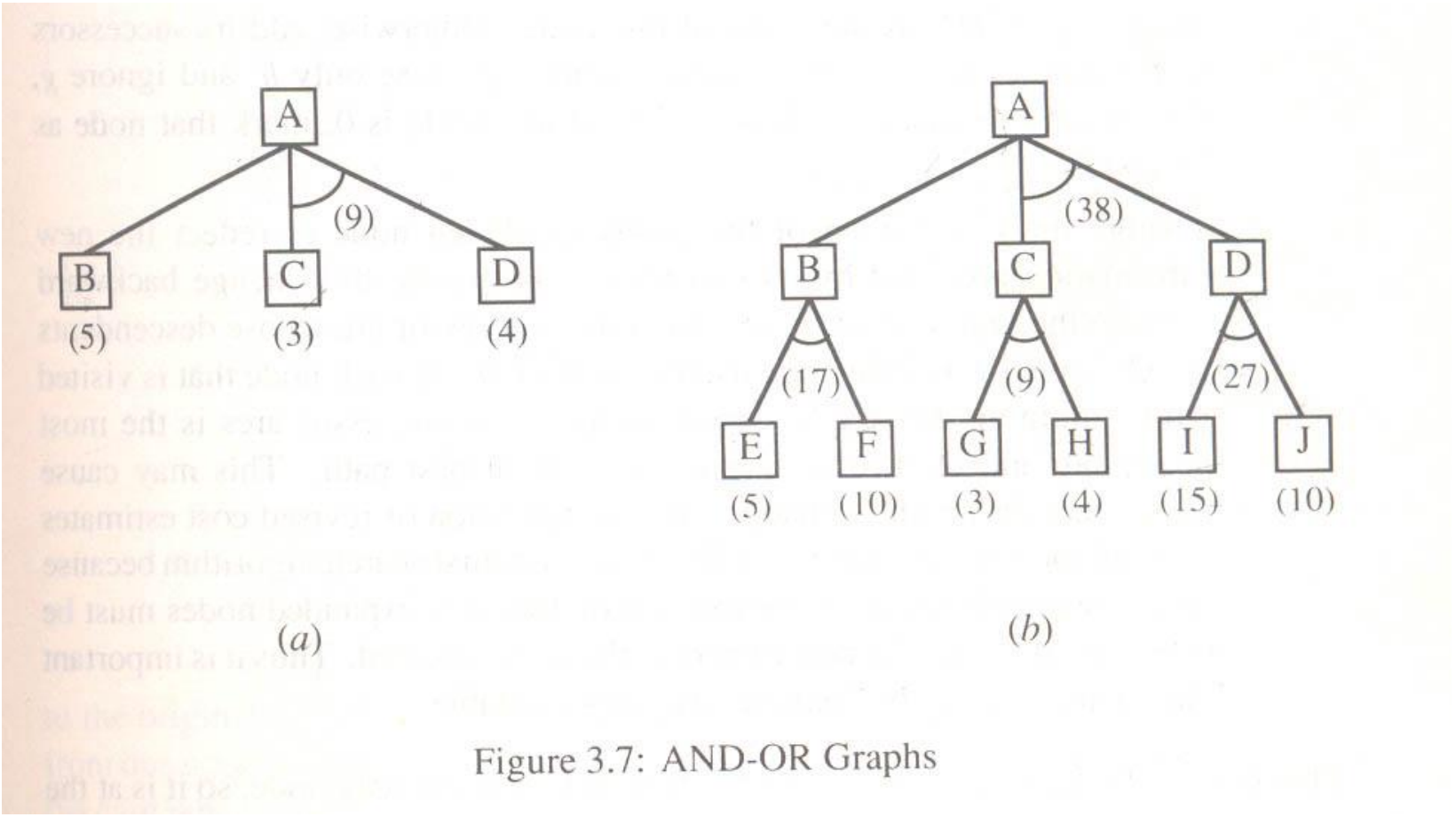
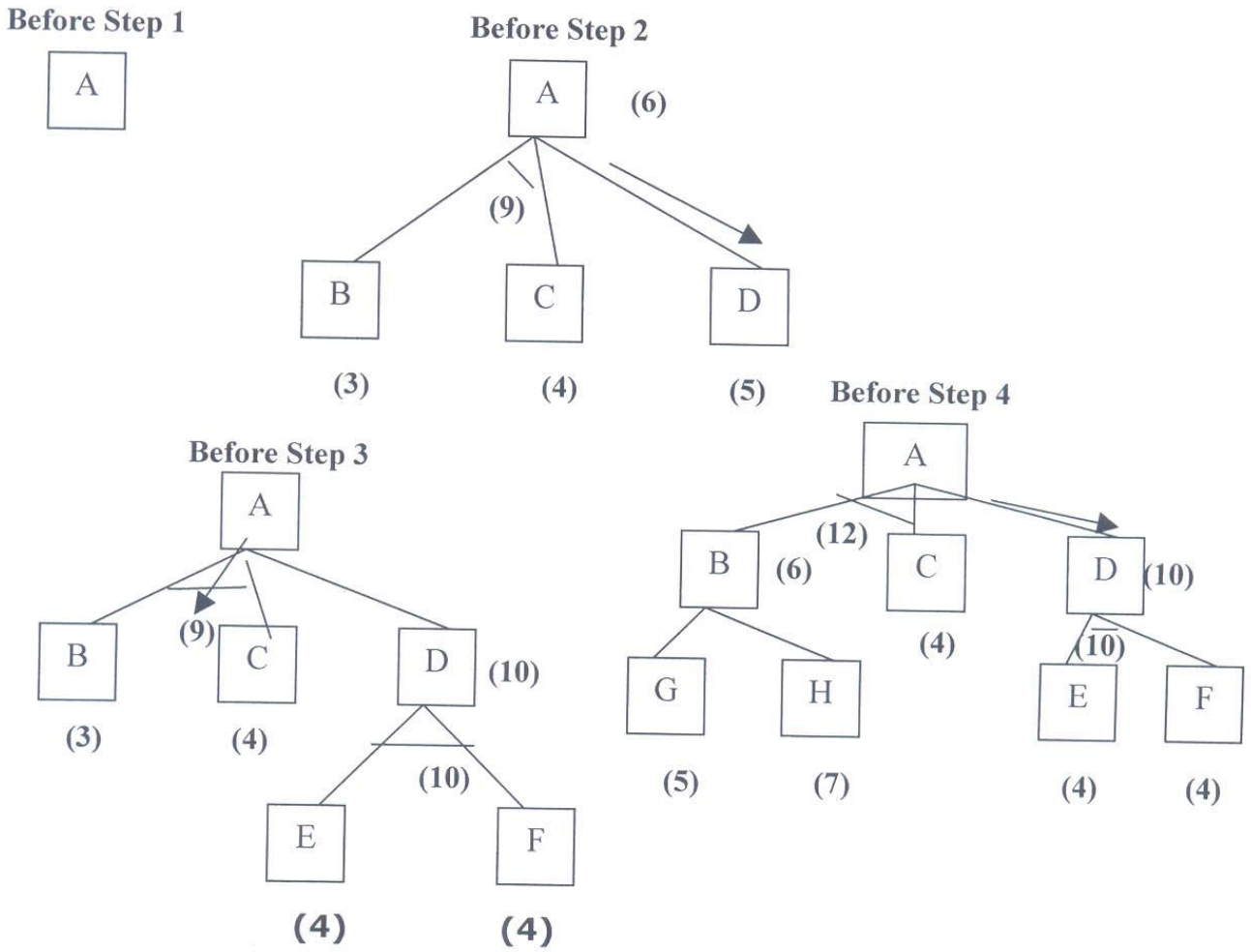


Figure 3.7: AND-OR Graphs

Problem Reduction with AO* Algorithm



How to explore

- Expand nodes
- Propagate values to ancestors

- Futility
  - If the estimated cost of a solution becomes greater than futility then abandon the search
  - A threshold such that any solution with higher cost is too expensive to be practical

AO* (high level view)

1. Given the Goal node, find its possible off-springs.

2. Estimate the h values at the leaves. The cost of the parent of the leaf (leaves) is the minimum of the cost of the OR clauses plus one or the cost of the AND clauses plus the number of AND clauses. After the children with minimum h are estimated, a pointer is attached to point from the parent node to its promising children.

3. One of the unexpanded OR clauses / the set of unexpanded AND clauses, where the pointer points from its parent, is now expanded and the h of the newly generated children are estimated. The effect of this h has to be propagated up to the root by re-calculating the f of the parent or the parent of the parents of the newly created child /children clauses through a least cost path. Thus the pointers may be modified depending on the revised cost of the existing clauses.

Game Playing

Why do AI researchers study game playing?

1. It's a good reasoning problem, formal and nontrivial.

2. Direct comparison with humans and other computer programs is easy.

What Kinds of Games?

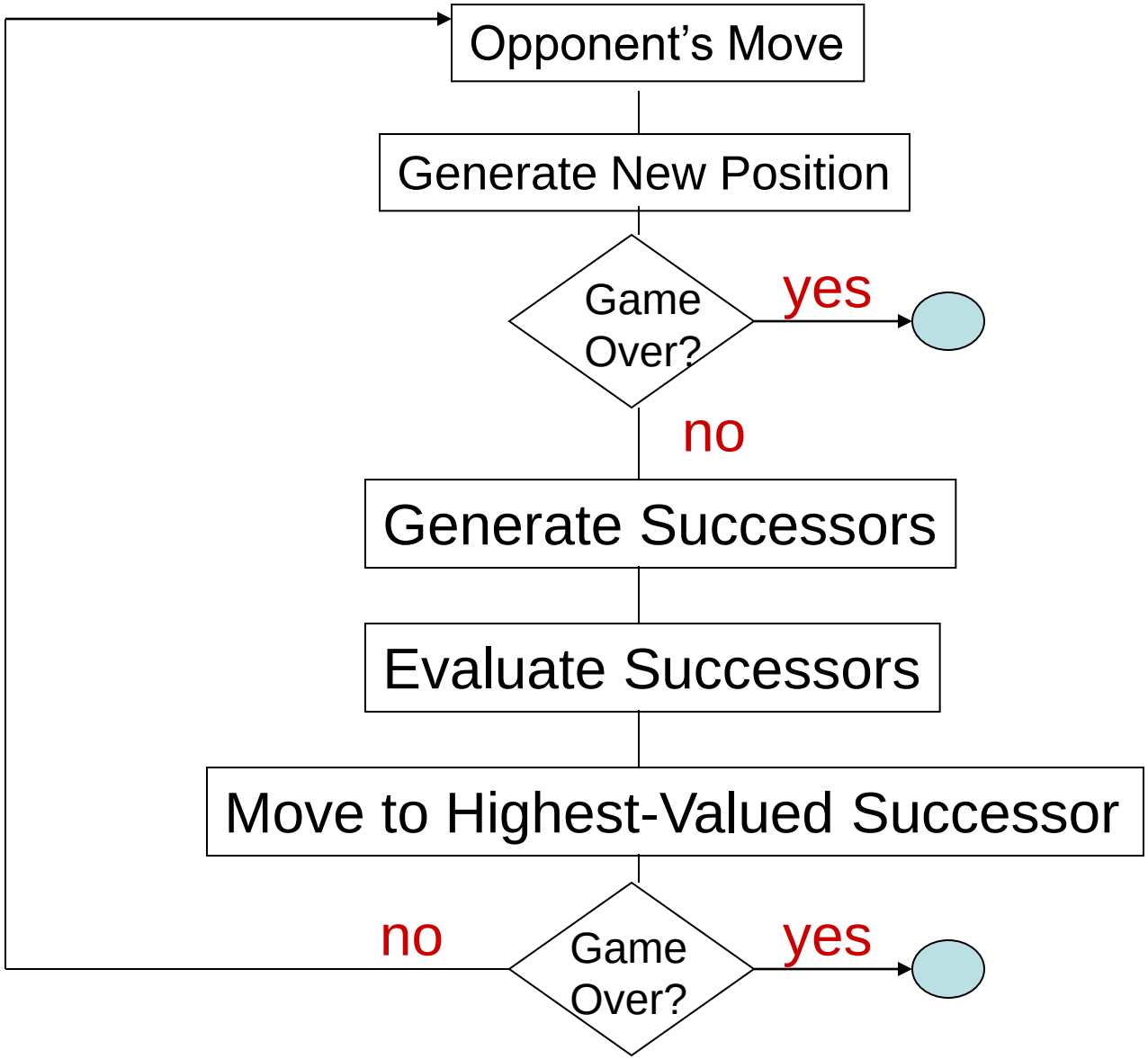
Mainly games of strategy with the following characteristics:

1. Sequence of moves to play
2. Rules that specify possible moves
3. Rules that specify a payment for each move
4. Objective is to optimize your payment

Games vs. Search Problems

- Unpredictable opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate

Two-Player Game

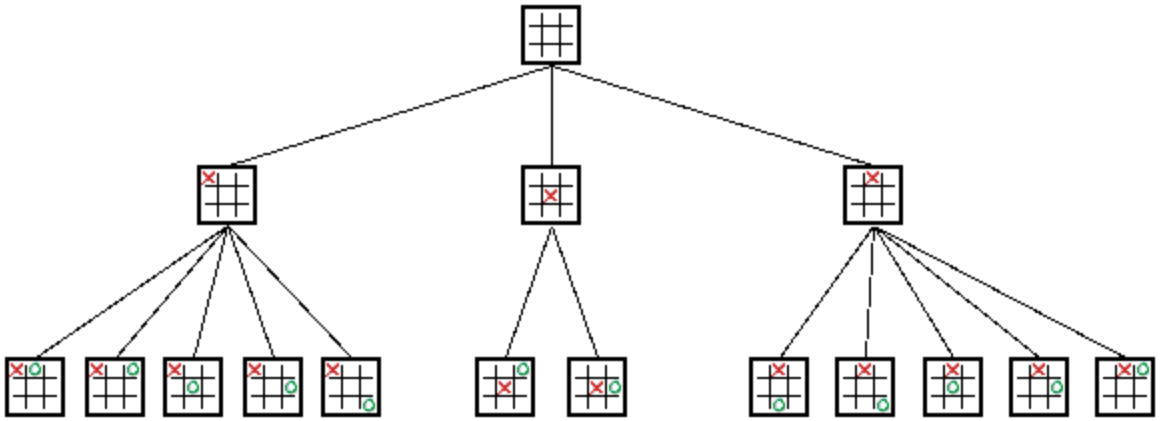


Minimax

- Minimax is a method used to evaluate game trees.
- A static evaluator is applied to leaf nodes, and values are passed back up the tree to determine the best score the computer can obtain against a rational opponent.

MinMax - Overview

- Search tree
  - *Squares* represent decision states (ie- after a move)
  - *Branches* are decisions (ie- the move)
  - Start at root
  - Nodes at end are leaf nodes
  - Ex: Tic-Tac-Toe (symmetrical positions removed)



- Unlike binary trees can have any number of children
  - Depends on the game situation
- Levels usually called *plies* (a *ply* is one level)
  - Each ply is where "turn" switches to other player
- Players called *Min* and *Max*

MaxMin - Algorithm

- Named *MinMax* because of algorithm behind data structure
- Assign points to the outcome of a game
  - Ex: Tic-Tac-Toe: *X* wins, value of 1. *O* wins, value -1.
- Max (X) tries to maximize point value, while Min (O) tries to minimize point value
- Assume both players play to best of their ability
  - Always make a move to minimize or maximize points
- So, in choosing, Max will choose best move to get highest points, assuming Min will choose best move to get lowest points

Game Tree (2-player, Deterministic, Turns)

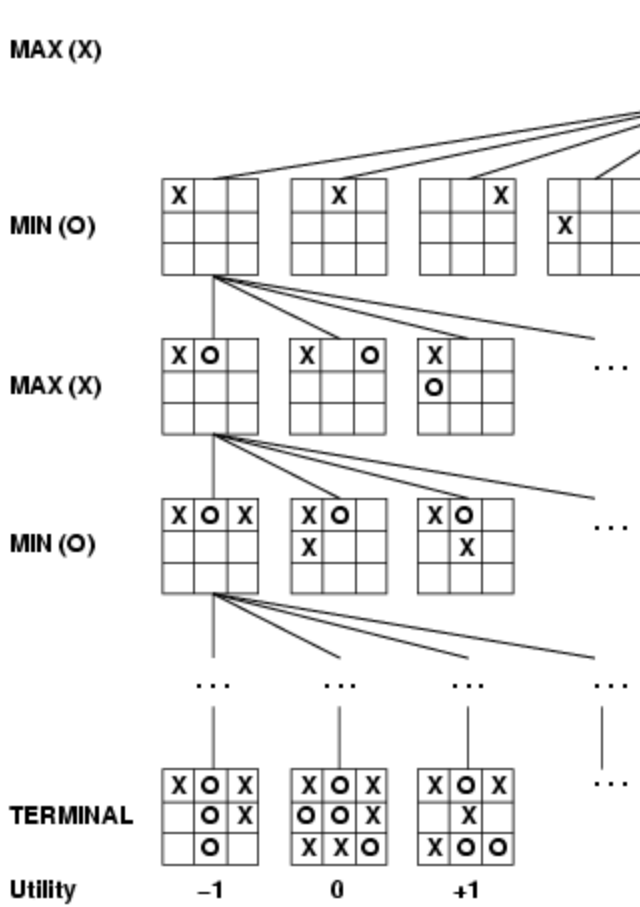
computer's turn

opponent's turn

computer's turn

opponent's turn

leaf nodes are evaluated



The computer is **Max**.
The opponent is **Min**.

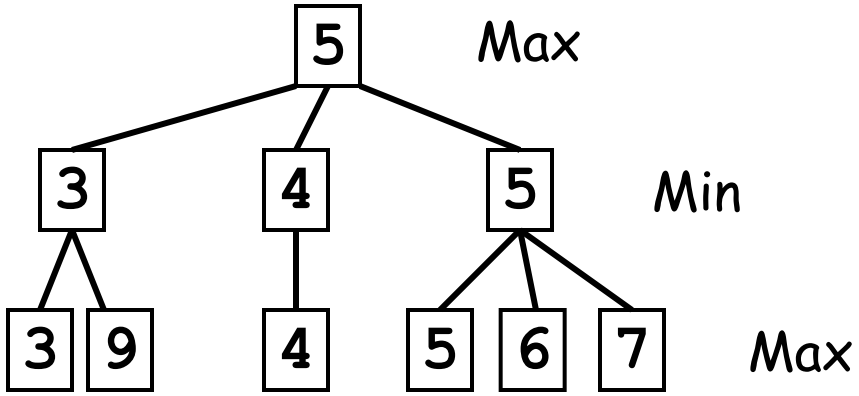
At the leaf nodes, the **utility function** is employed. Big value means good, small is bad.

Mini-Max Terminology

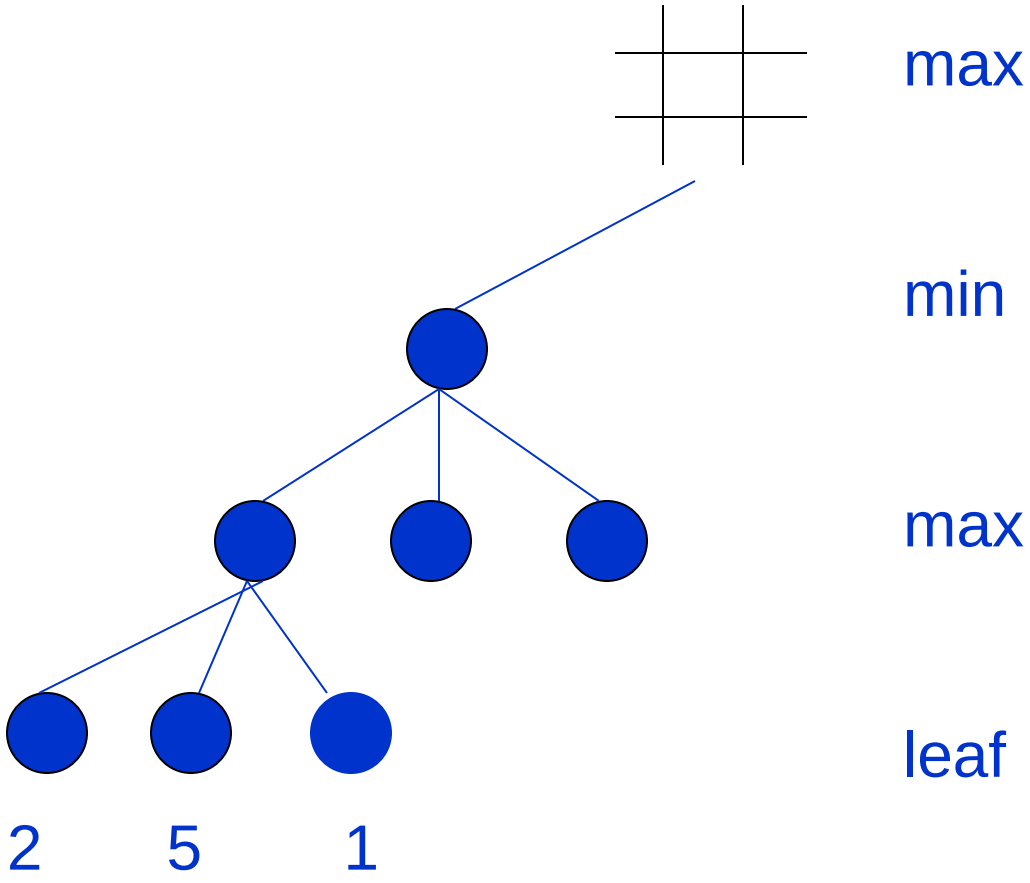
- utility function: the function applied to leaf nodes
- backed-up value
  - of a max-position: the value of its largest successor
  - of a min-position: the value of its smallest successor
- minimax procedure: search down several levels; at the bottom level apply the utility function, back-up values all the way up to the root node, and that node selects the move.

MinMax – First Example

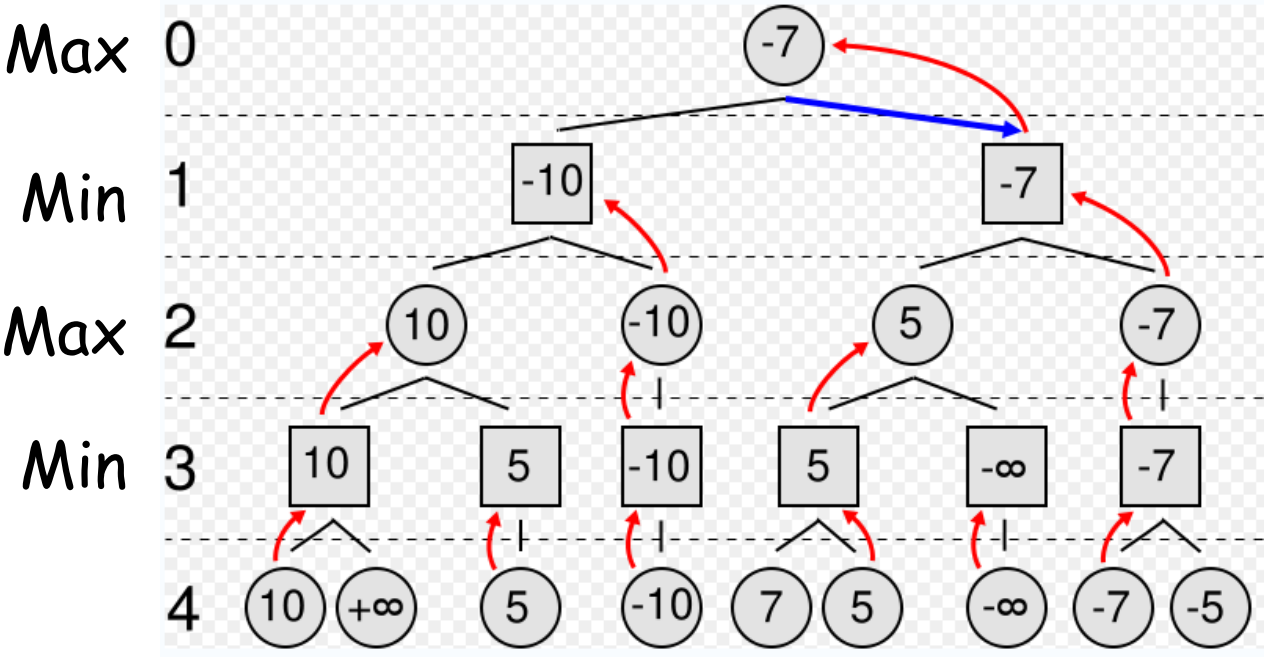
- Max's turn
- Would like the "9" points (the maximum)
- But if choose left branch, Min will choose move to get 3
  → left branch has a value of 3
- If choose right, Min can choose any one of 5, 6 or 7 (will choose 5, the minimum)
  → right branch has a value of 5
- Right branch is largest (the maximum) so choose that move



Minimax is done depth-first



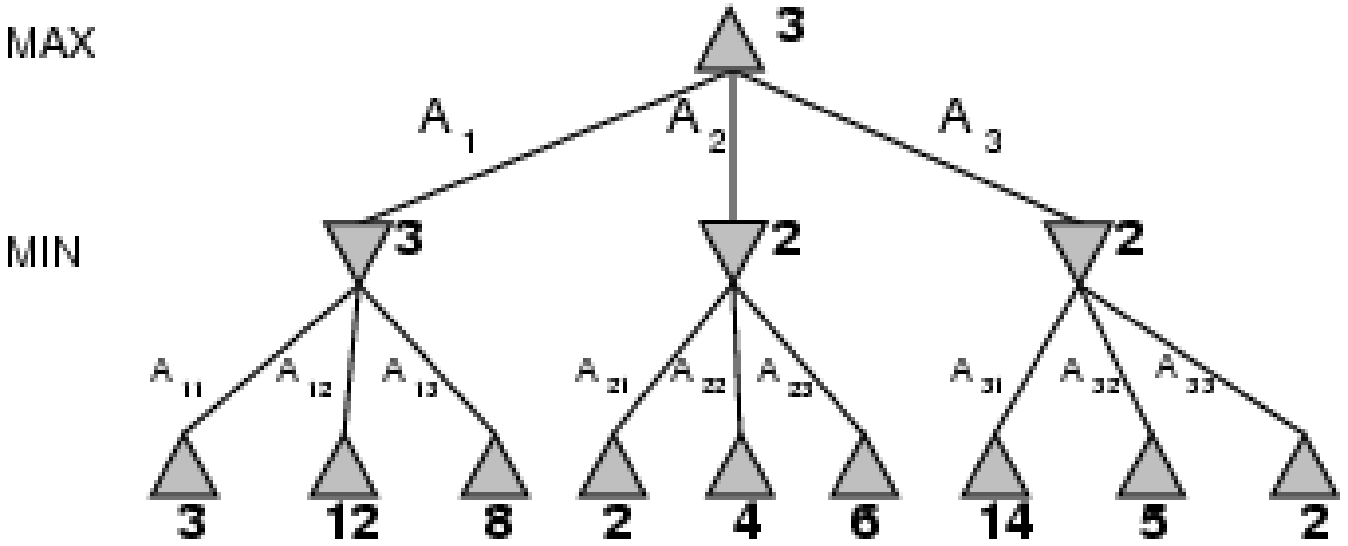
MinMax – Second Example



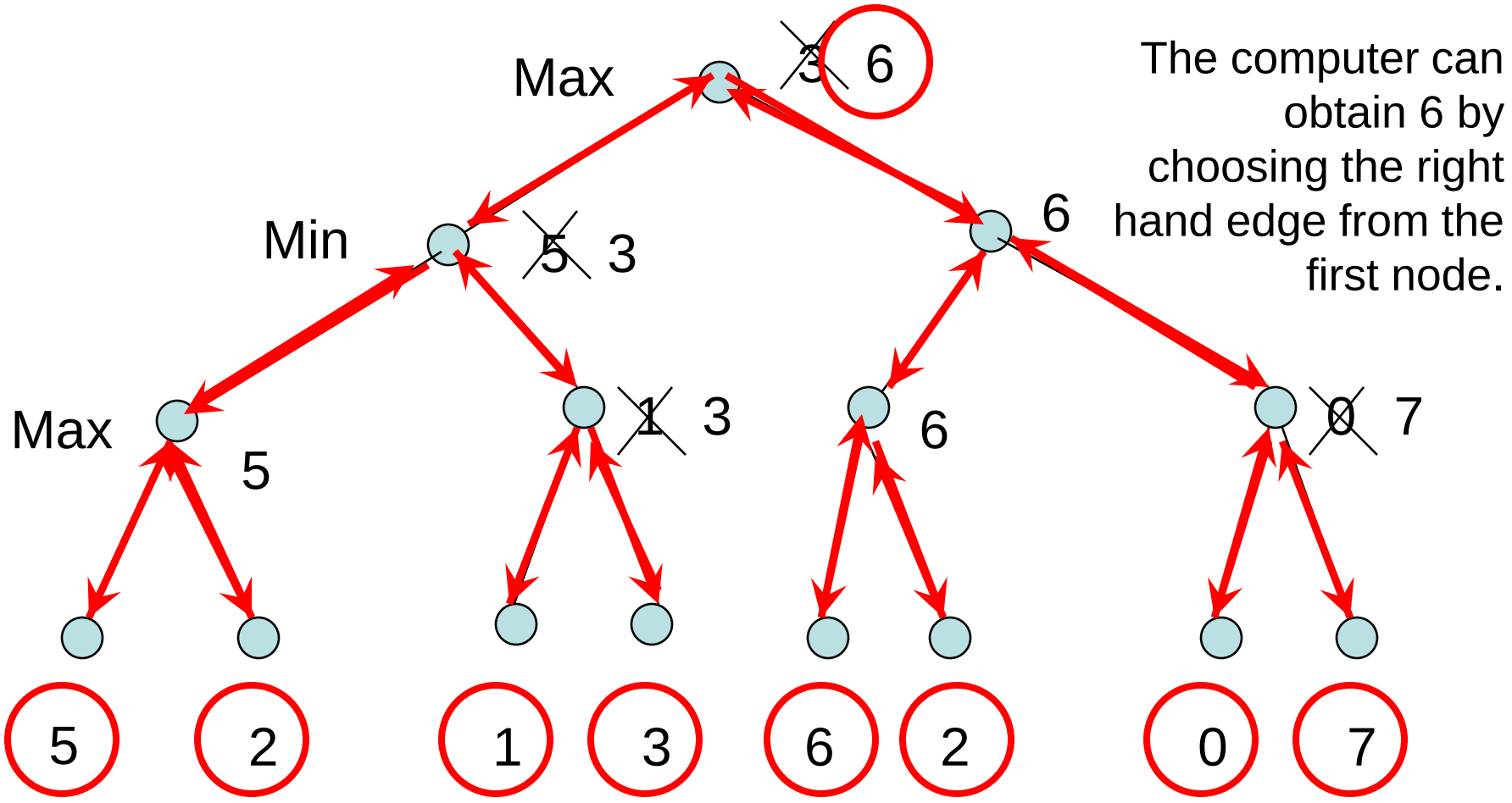
- Max's turn
- Circles represent Max, Squares represent Min
- Values inside represent the value the MinMax algorithm
- Red arrows represent the chosen move
- Numbers on left represent tree depth
- Blue arrow is the chosen move

MinMax – Third Example

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:



Minimax – Animated Example



The computer can obtain 6 by choosing the right hand edge from the first node.

Minimax Strategy

- Why do we take the min value every other level of the tree?
- These nodes represent the opponent's choice of move.
- The computer assumes that the human will choose that move that is of least value to the computer.

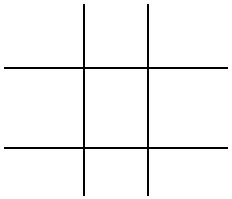
Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

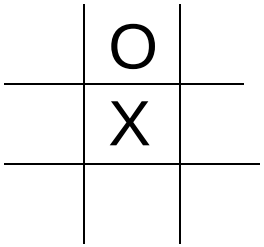
Tic Tac Toe

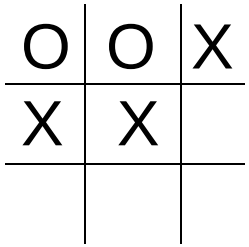


- Let p be a position in the game
- Define the utility function f(p) by
  - f(p) =
    - largest positive number if p is a win for computer
    - smallest negative number if p is a win for opponent
    - RCDC – RCDO
  - where RCDC is number of rows, columns and diagonals in which computer could still win
  - and RCDO is number of rows, columns and diagonals in which opponent could still win.

Sample Evaluations

- X = Computer; O = Opponent





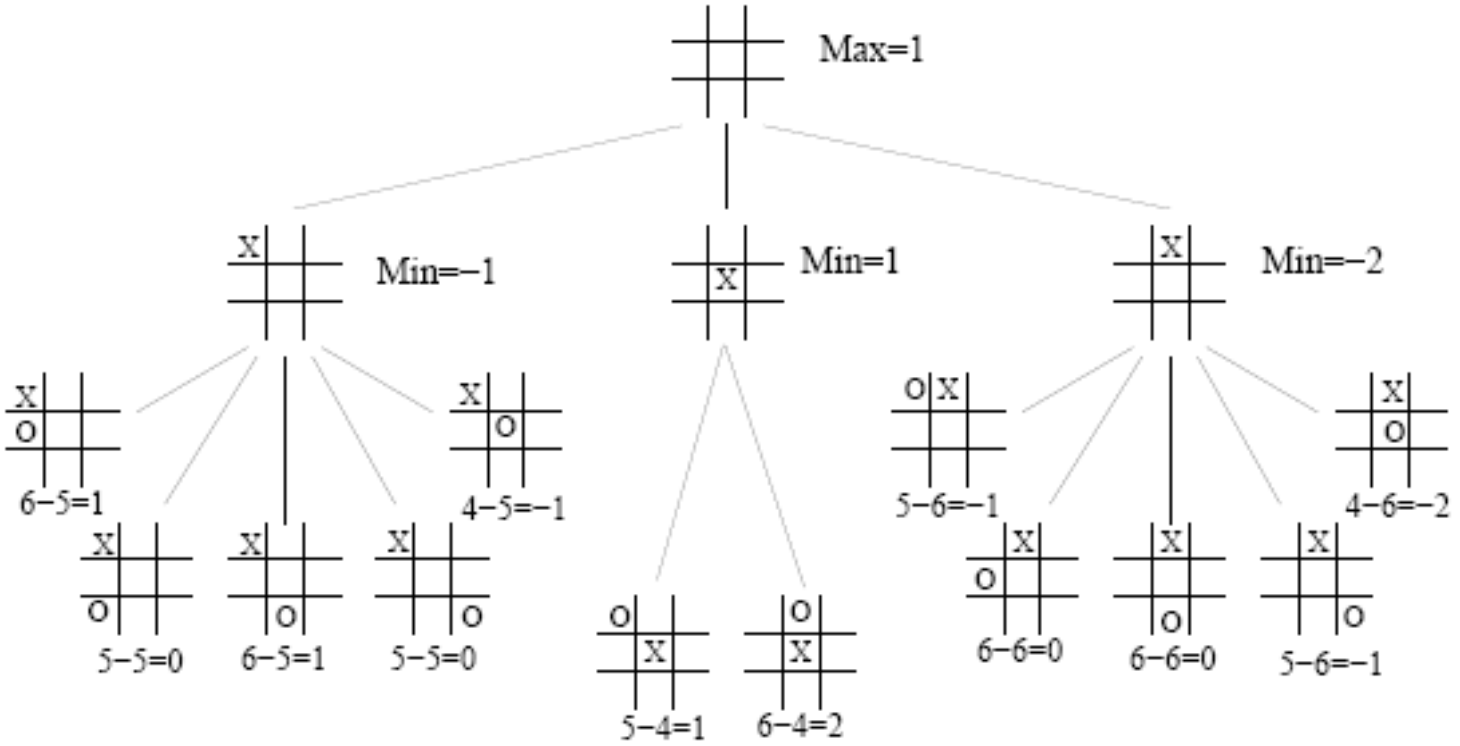
	X	O
rows		
cols		
diags		

	X	O
rows		
cols		
diags		

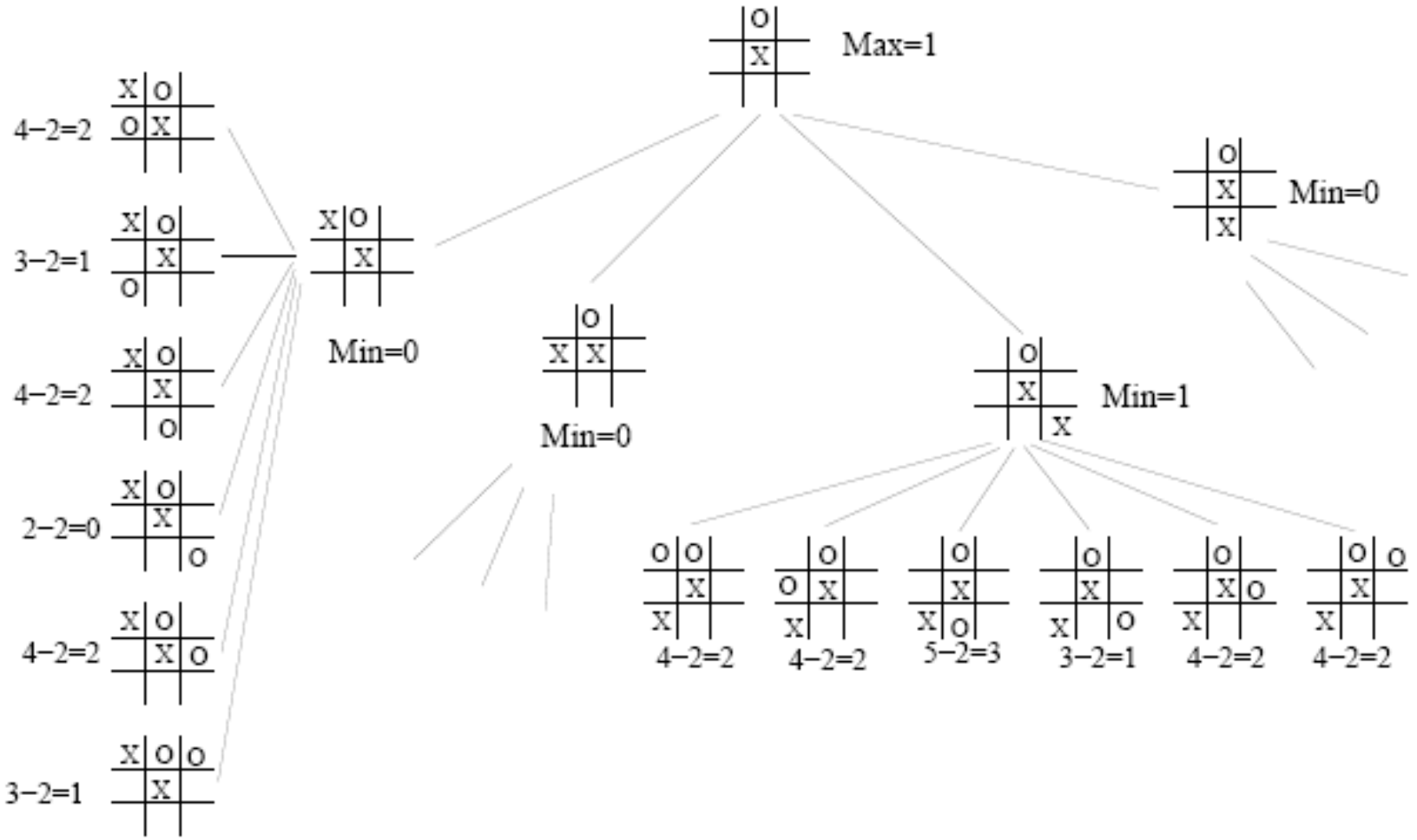
Example: tic-tac-toe

- e (evaluation function → integer) = number of available rows, columns, diagonals for MAX - number of available rows, columns, diagonals for MIN
- MAX plays with “X” and desires maximizing e.
- MIN plays with “0” and desires minimizing e.
- Symmetries are taken into account.
- A depth limit is used (2, in the example).

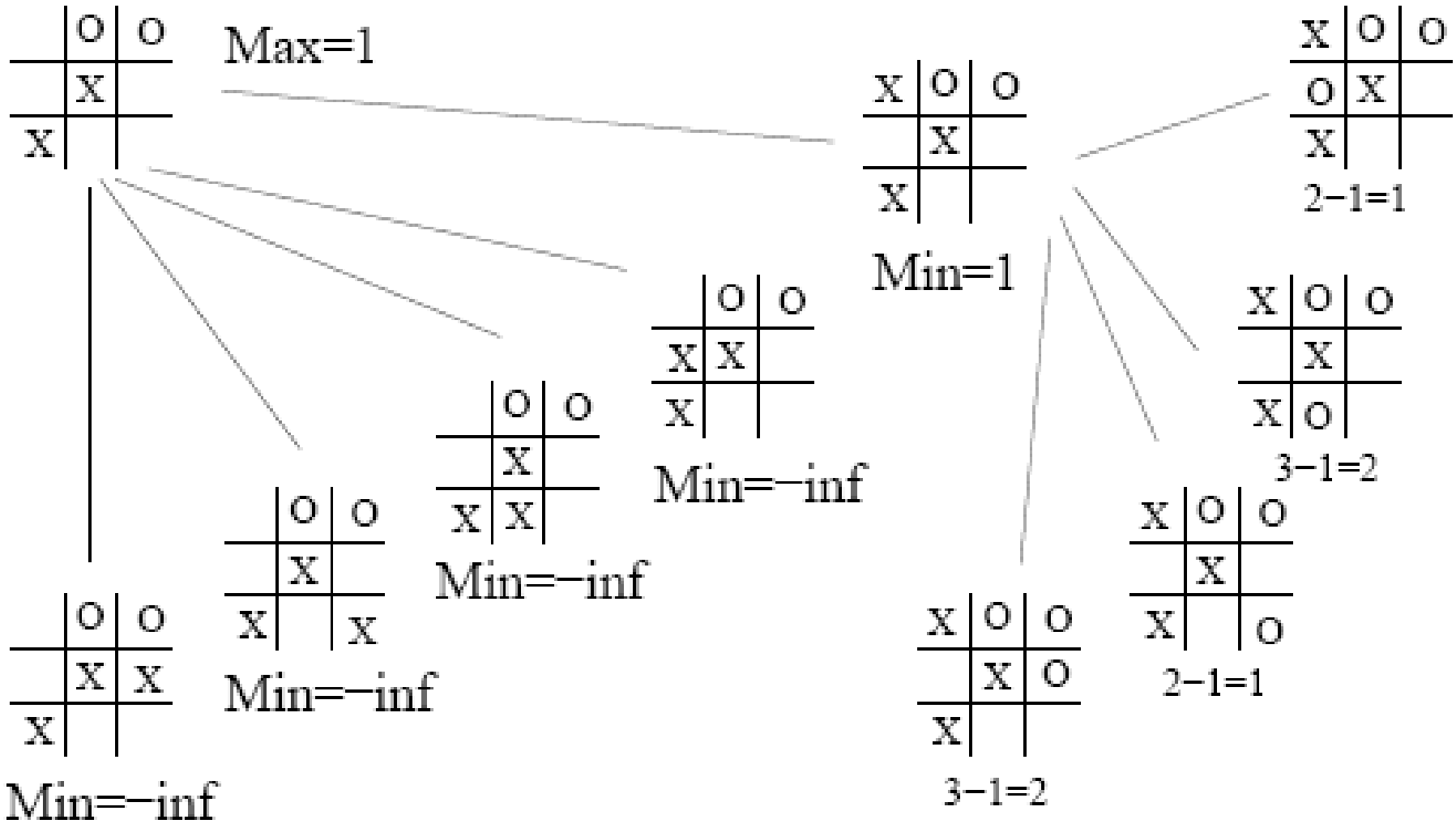
Example: tic-tac-toe



Example: tic-tac-toe



Example: tic-tac-toe



The minimax algorithm

- The **minimax algorithm** computes the minimax decision from the current state.
- It uses a simple recursive computation of the minimax values of each successor state:
  - directly implementing the defining equations.
- The recursion proceeds all the way down to the leaves of the tree.
- Then the minimax values are **backed up** through the tree as the recursion unwinds.

Properties of Minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first exploration)

- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible

Need to speed it up.

Alpha-Beta Procedure

- The alpha-beta procedure can speed up a depth-first minimax search.
- Alpha: a lower bound on the value that a max node may ultimately be assigned

$$v \geq \alpha$$

- Beta: an upper bound on the value that a minimizing node may ultimately be assigned

$$v \leq \beta$$

Alpha-beta pruning

- It is possible to compute the correct minimax decision without looking at every node in the game tree.
- Alpha-beta pruning allows to eliminate large parts of the tree from consideration, without influencing the final decision.

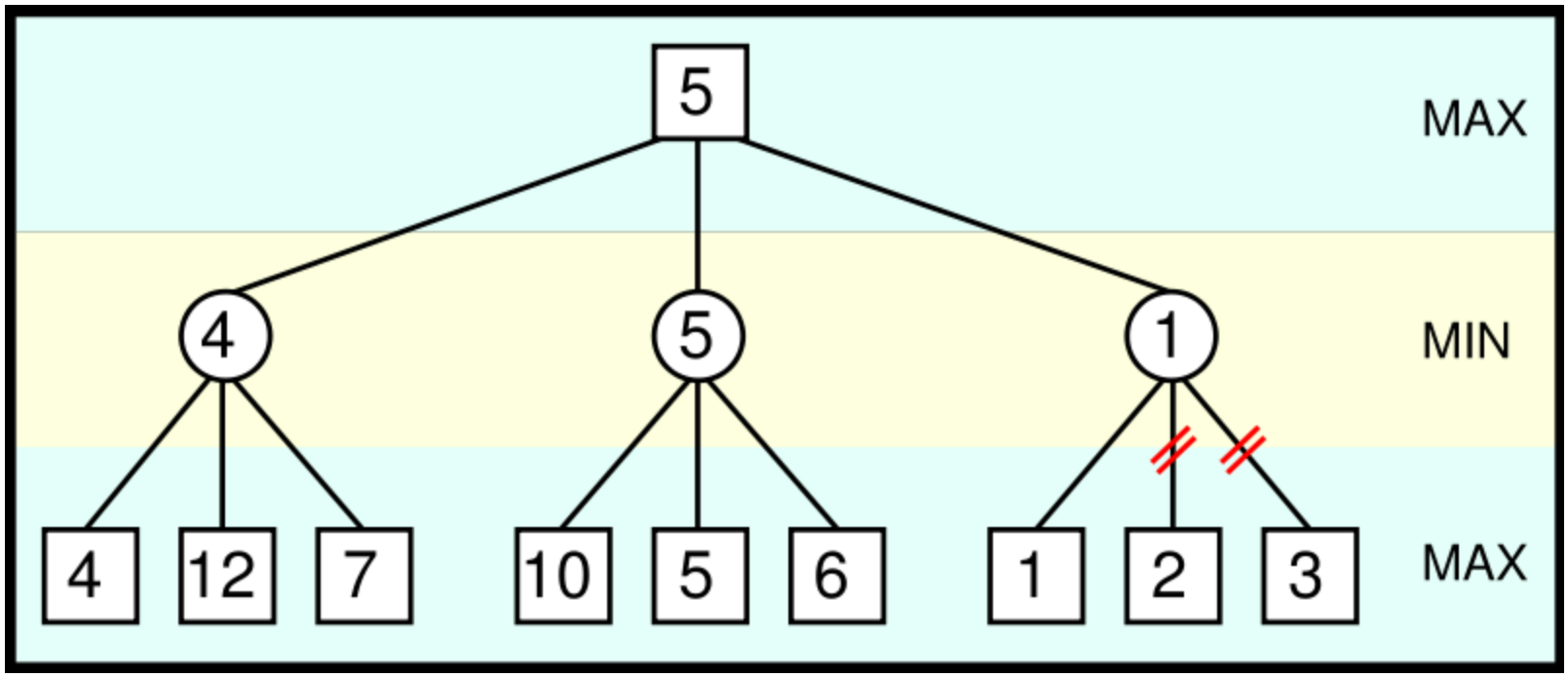
Alpha-beta pruning

- Alpha-beta pruning gets its name from two parameters.
  - They describe bounds on the values that appear anywhere along the path under consideration:
    - α = the value of the best (i.e., highest value) choice found so far along the path for MAX
    - β = the value of the best (i.e., lowest value) choice found so far along the path for MIN

Alpha-beta pruning

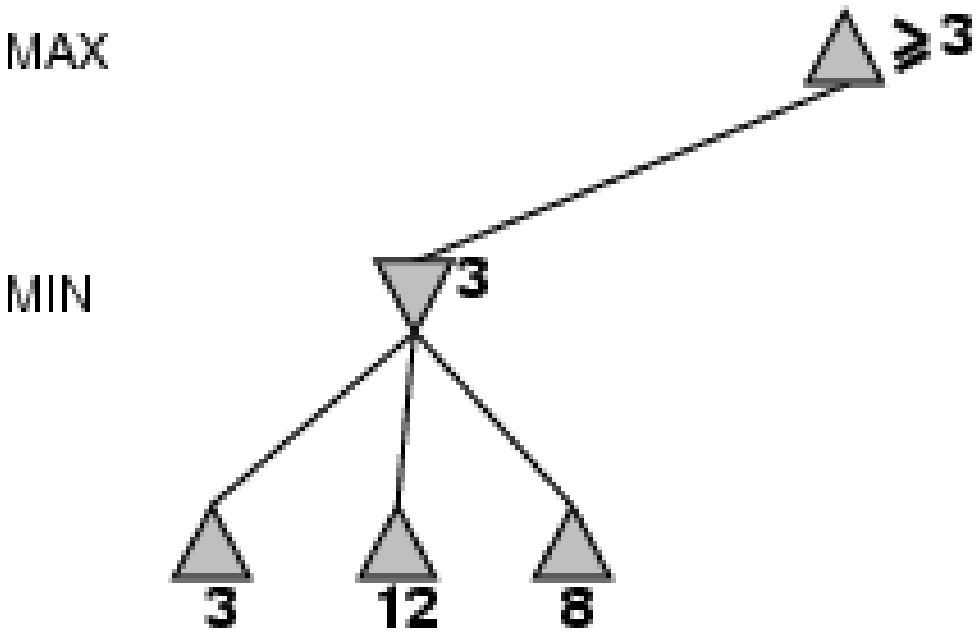
- Alpha-beta search updates the values of α and β as it goes along.
- It prunes the remaining branches at a node (i.e., terminates the recursive call)
  - as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively.

MinMax – AlphaBeta Pruning Example 1

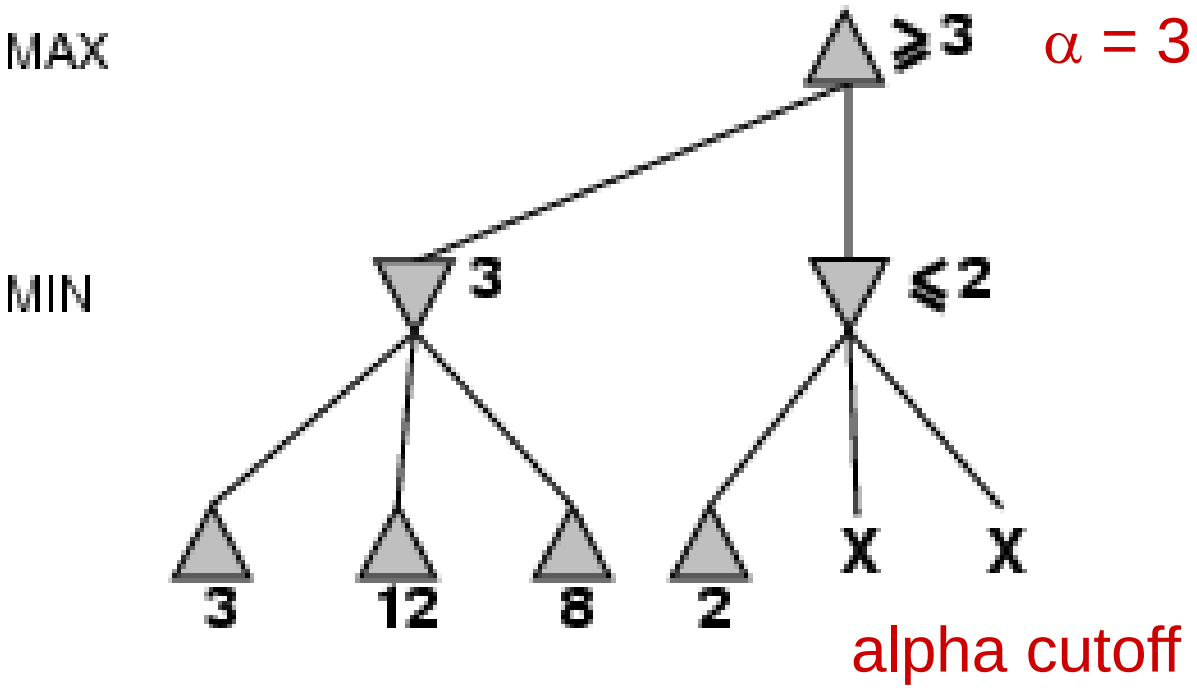


- From Max point of view, 1 is already lower than 4 or 5, so no need to evaluate 2 and 3 (bottom right) → Prune

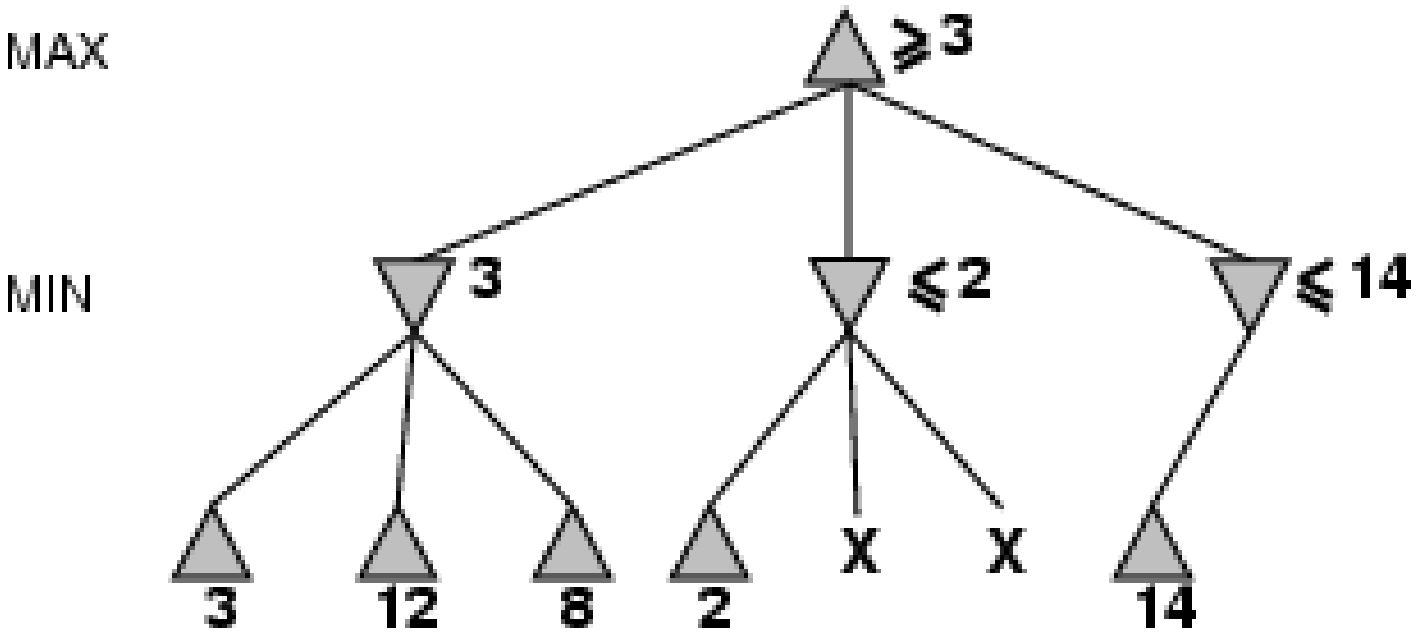
α-β pruning Example 2



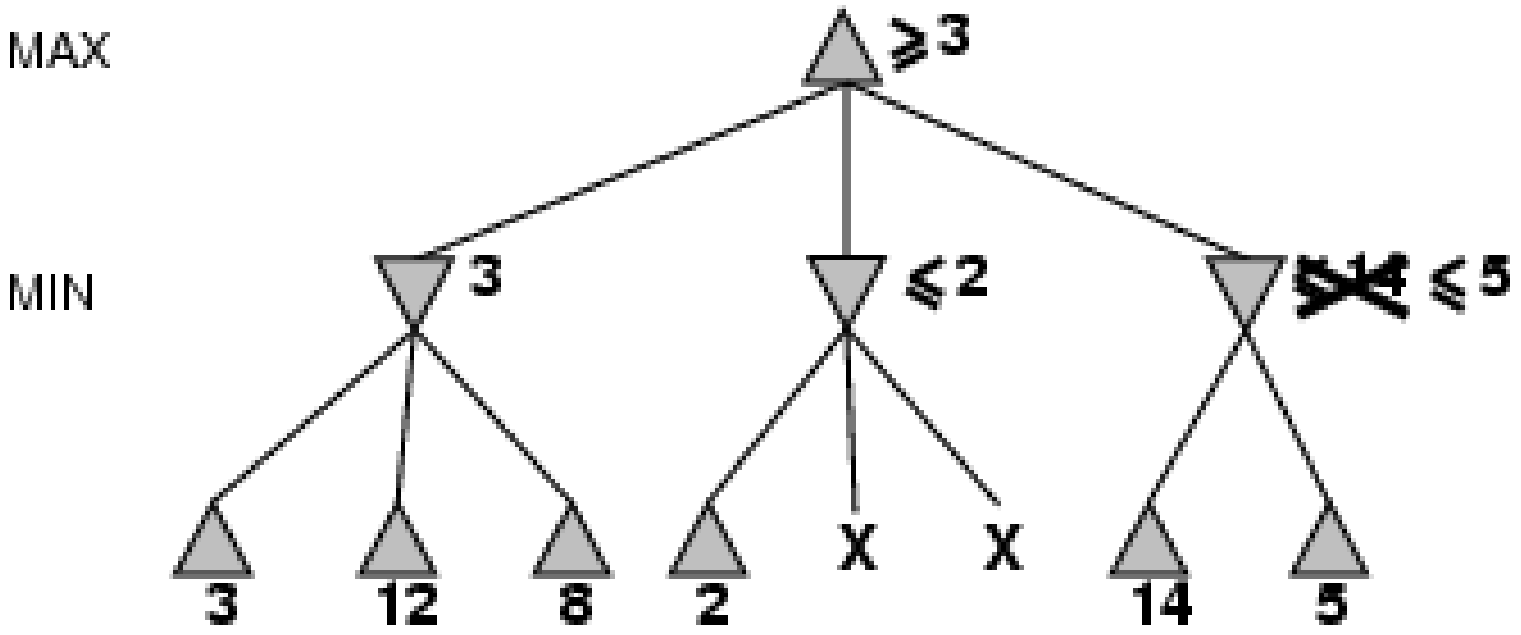
α-β pruning Example 2



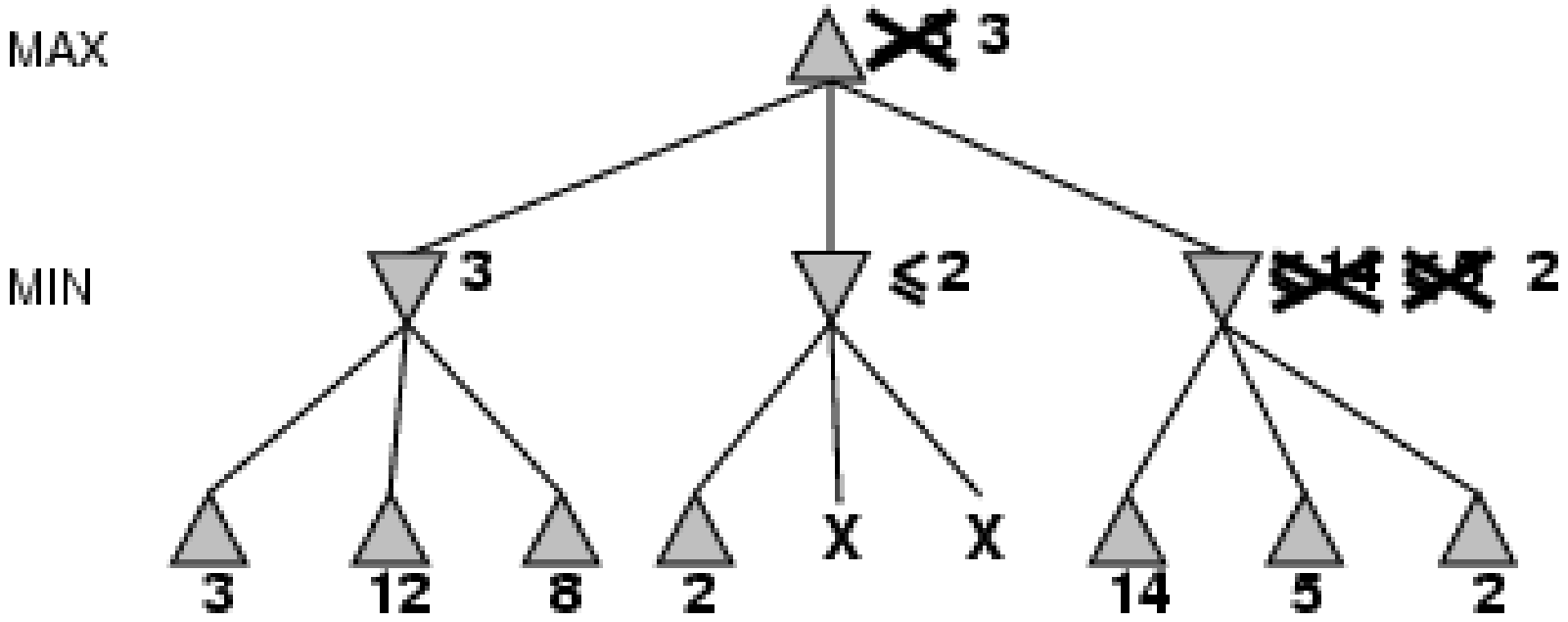
α-β pruning Example 2



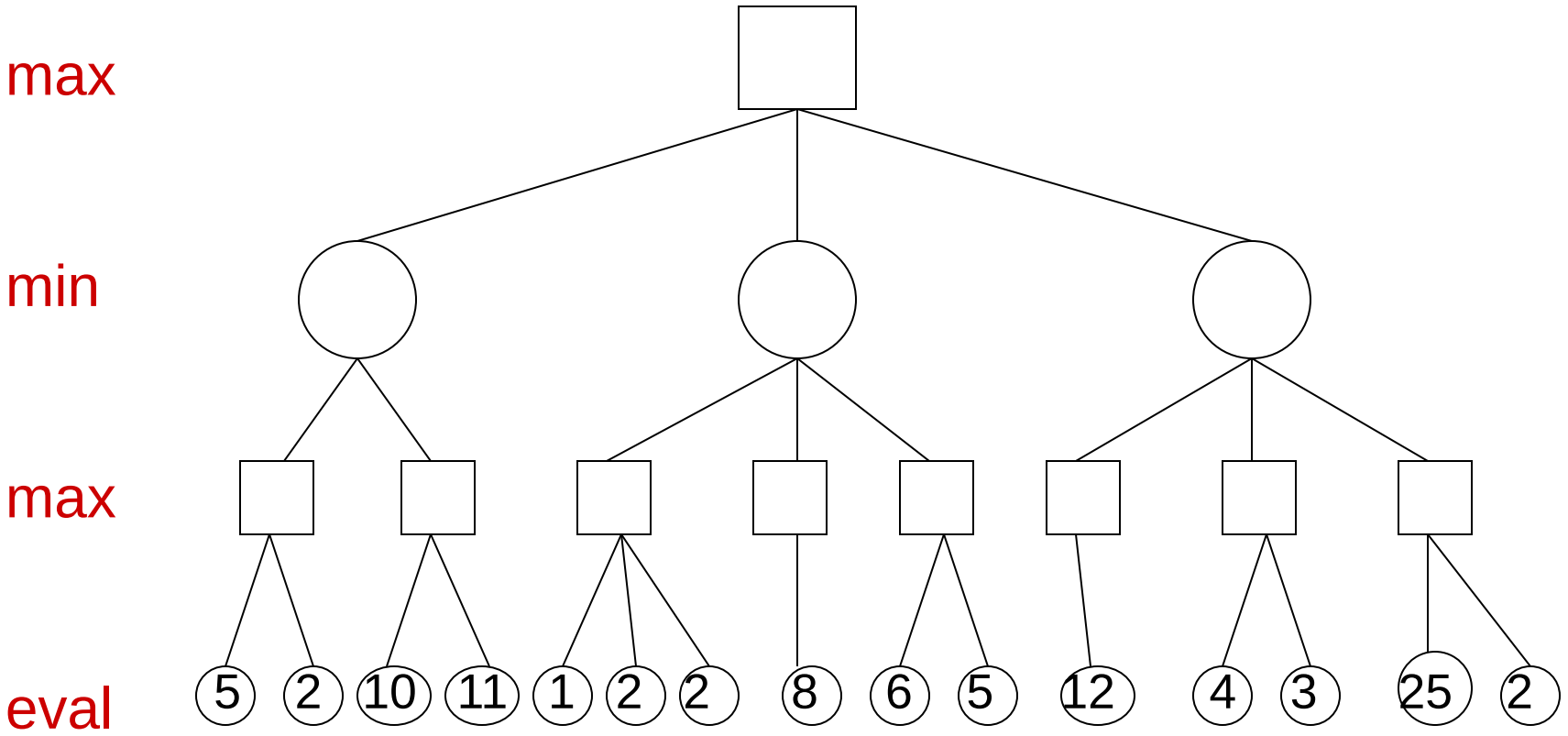
α-β pruning Example 2



α-β pruning Example 2



Alpha-Beta Pruning



Properties of α-β

- Pruning does not affect final result. This means that it gets the exact same result as does full minimax.
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  → doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

The α-β algorithm

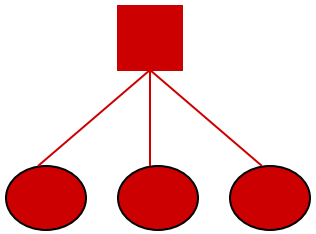
function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v cutoff
    $\alpha \leftarrow$ MAX(α, v)
  return v

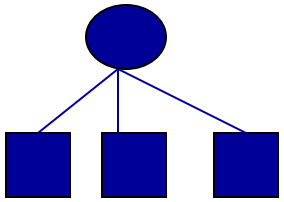


The α-β algorithm

```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

cutoff



Thank you

- Credits Reference [1]: *https://courses.cs.washington.edu/courses/cse473/*

Means-Ends Analysis in AI

- We have studied the strategies which can reason either in forward or backward, but a mixture of the two directions is appropriate for solving a complex and large problem. Such a mixed strategy, make it possible that first to solve the major part of a problem and then go back and solve the small problems arise during combining the big parts of the problem. Such a technique is called **Means-Ends Analysis**.
- Means-Ends Analysis is problem-solving techniques used in Artificial intelligence for limiting search in AI programs.
- It is a mixture of Backward and forward search technique.
- The MEA technique was first introduced in 1961 by Allen Newell, and Herbert A. Simon in their problem-solving computer program, which was named as General Problem Solver (GPS).
- The MEA analysis process centered on the evaluation of the difference between the current state and goal state.

How means-ends analysis Works:

The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving. Following are the main Steps which describes the working of MEA technique for solving a problem.

a. First, evaluate the difference between Initial State and final State.

b. Select the various operators which can be applied for each difference.

c. Apply the operator at each difference, which reduces the difference between the current state and goal state.

Operator Subgoaling

In the MEA process, we detect the differences between the current state and goal state. Once these differences occur, then we can apply an operator to reduce the differences. But sometimes it is possible that an operator cannot be applied to the current state. So we create the subproblem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called **Operator Subgoaling**.

Algorithm for Means-Ends Analysis:

Let's we take Current state as CURRENT and Goal State as GOAL, then following are the steps for the MEA algorithm.

- **Step 1:** Compare CURRENT to GOAL, if there are no differences between both then return Success and Exit.

- **Step 2:** Else, select the most significant difference and reduce it by doing the following steps until the success or failure occurs.

a. Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.

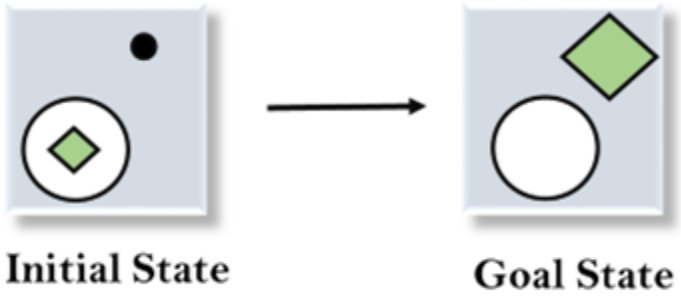
   b. Attempt to apply operator O to CURRENT. Make a description of two states.
   i) O-Start, a state in which O?s preconditions are satisfied.
   ii) O-Result, the state that would result if O were applied In O-start.

   c. If
   (First-Part <------ MEA (CURRENT, O-START)
   And
   (LAST-Part <----- MEA (O-Result, GOAL), are successful, then signal Success and return the result of combining FIRST-PART, O, and LAST-PART.

The above-discussed algorithm is more suitable for a simple problem and not adequate for solving complex problems.

Example of Mean-Ends Analysis:

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.



Solution:

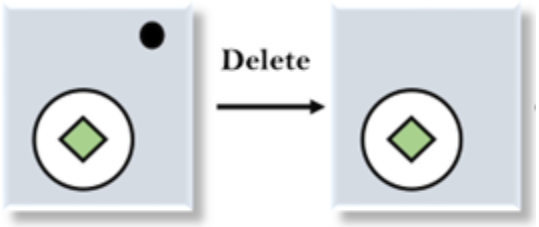
To solve the above problem, we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:

- **Move**
- **Delete**
- **Expand**

1. Evaluating the initial state: In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.

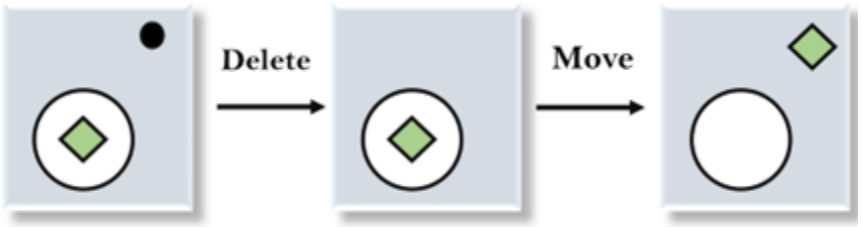
Initial state

2. Applying Delete operator: As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.



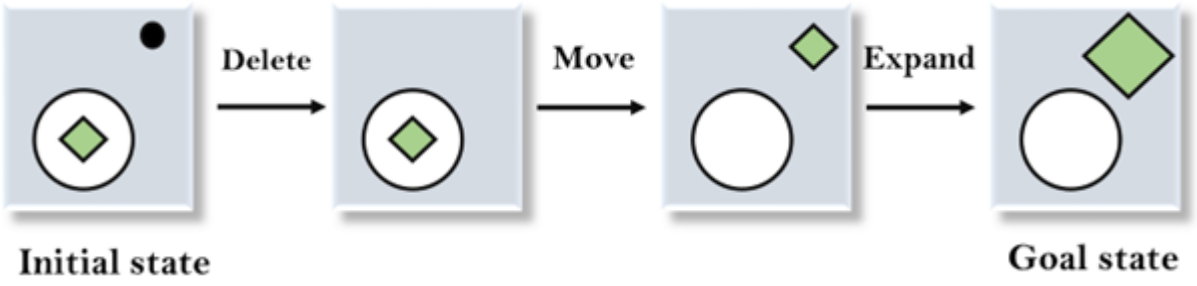
Initial state

3. Applying Move Operator: After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the **Move Operator**.



Initial state

4. Applying Expand Operator: Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply **Expand operator**, and finally, it will generate the goal state.


Delete
Move
Expand
Initial state
Goal state

Hill climbing

This is a *greedy* algorithm: go as high up as possible as fast as possible, without looking around too much.

The algorithm

select a heuristic function;

set C, the current node, to the highest-valued initial node;

loop

select N, the highest-valued child of C;

return C if its value is better than the value of N;
otherwise set C to N;

Problems with hill climbing

1. Local maximum, or the foothill problem: there is a peak, but it is lower than the highest peak in the whole space.
2. The plateau problem: all local moves are equally unpromising, and all peaks seem far away.
3. The ridge problem: almost every move takes us down.

   Random-restart hill climbing is a series of hill-climbing searches with a randomly selected start node whenever the current search gets stuck.

Means-ends analysis

The idea is due to Newell and Simon (1957): work by reducing the difference between states, and so approaching the goal state. There are procedures, indexed by differences, that change states.

The General Problem Solver algorithm

set C, the current node, to any initial node;

loop

succeed if the goal state has been reached, otherwise find the difference between C and the goal state;

choose a procedure that reduces this difference, apply it to C to produce the new current state;

Means-ends analysis

An example: planning a trip.

Distance \ Procedure	Plane	Train	Car	Taxi	Walk
miles					
> 1000	X				
100-1000		X	X		
10-100			X	X	
< 1					X
Prerequisite	At plane	At train	At car	At taxi	

Problem Characteristics

1. Is the problem decomposable?
2. Can solution steps be ignored or undone?
3. Is the universe predictable?
4. Is a good solution absolute or relative?
5. Is the solution a state or a path?
6. What is the role of knowledge?
7. Does the task require interaction with a person?

Production system characteristics

- Monotonic: execution of a rule never prevents the execution of other applicable rules
- Non-monotonic: without this property

- Partially commutative: if application of a particular sequence of rules transforms state x into state y, then any permutation of those rules that are allowable, transforms x into y
- Commutative: monotonic + partially commutative

Problems and production system

	monotonic	Non-monotonic
Partially commutative	Theorem proving	Robot navigation
Non-partially commutative	Chemical synthesis	backgammon

Problems and production system

	monotonic	Non-monotonic
Partially commutative	Ignorable	Recoverable
Non-partially commutative	Irrecoverable	Irrecoverable Unpredictable

Sample problems - 8-puzzle

The 8-puzzle is a small single board player game:

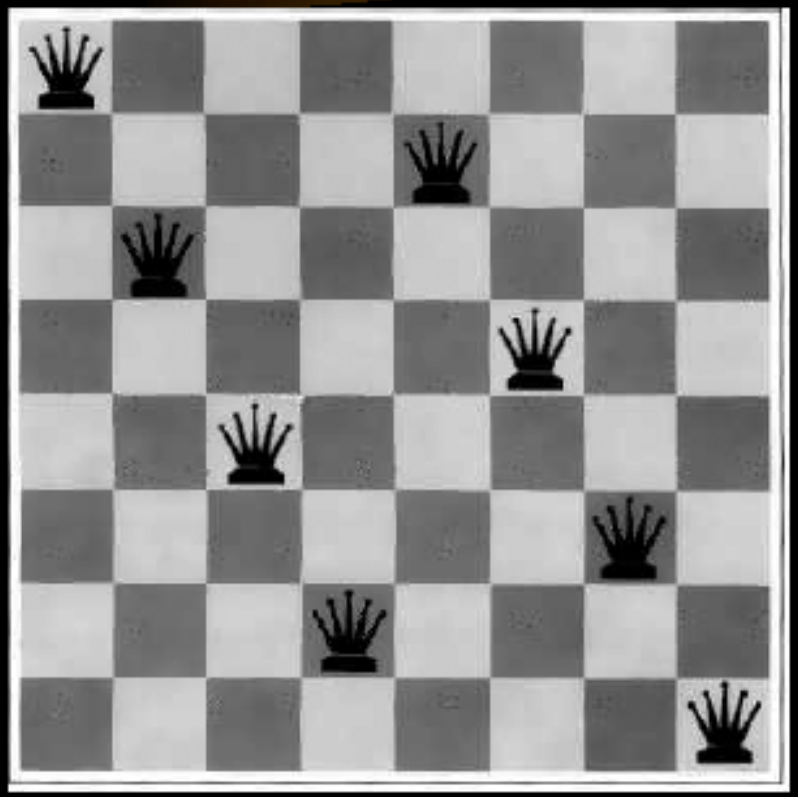
- Tiles are numbered 1 through 8 and one blank space on a 3 x 3 board.
- A 15-puzzle, using a 4 x 4 board, is commonly sold as a child's puzzle.
- Possible moves of the puzzle are made by sliding an adjacent tile into the position occupied by the blank space, this will exchanging the positions of the tile and blank space.
- Only tiles that are horizontally or vertically adjacent (not diagonally adjacent) may be moved into the blank space.

Sample problems - 8-puzzle

- **Goal test:** have the tiles in ascending order.
- **Path cost:** each move is a cost of one.
- **States:** the location of the tiles + blank in the n x n matrix.
- **Operators:** blank moves left, right, up or down.

Sample problems - 8 queens

- In this problem,we need to place eight queens on the chess board so that they do not check each other. This problem is probably as old as the chess game itself, and thus its origin is not known, but it is known that Gauss studied this problem.

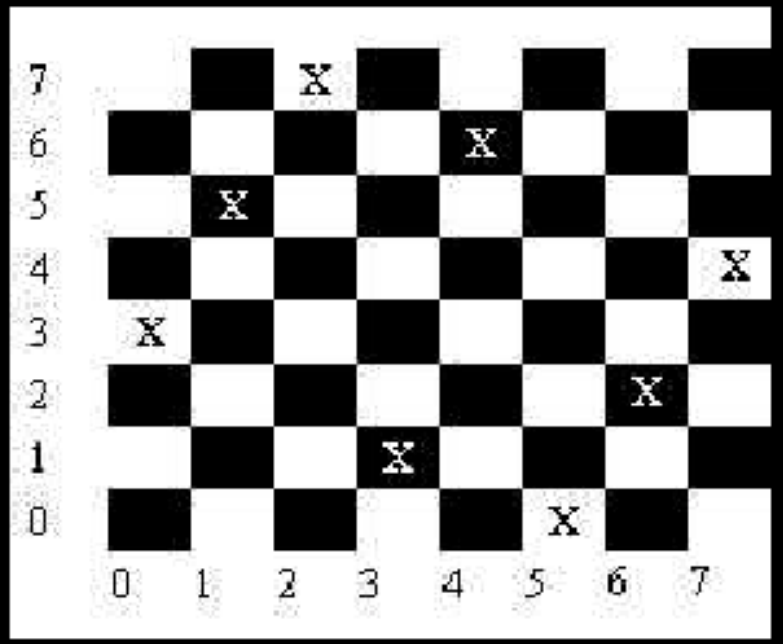


Sample problems - 8 queens

- **Goal test:** eight queens on board, none attacked
- **Path cost:** zero.
- **States:** any arrangement of zero to eight queens on board.
- **Operators:** add a queen to any square

Sample problems - 8 queens

- If we want to find a single solution, it is not hard. If we want to find all possible solutions, the problem becomes increasingly difficult and the backtrack method is the only known method. For 8-queen, we have 96 solutions. If we exclude symmetry, there are 12 solutions.



Sample problems - Cryptarithmetic

- In 1924 Henry Dudeney published a popular number puzzle of the type known as a cryptarithm, in which letters are replaced with numbers.
- Dudeney's puzzle reads: SEND + MORE = MONEY.
- Cryptarithms are solved by deducing numerical values from the mathematical relationships indicated by the letter arrangements (i.e.) . S=9, E=5, N=6, M=1, O=0,….
- The only solution to Dudeney's problem: 9567 + 1085 = 10,652.

Sample problems - Cryptarithmetic

- **Goal test:** puzzle contains only digits and represents a correct sum.
- **Path cost:** zero. All solutions equally valid
- **States:** a cryptarithmetic puzzle with some letters replaced by digits.
- **Operators:** replace all occurrences of a letter with a digit not already appearing in the puzzle.