# ECE 457 Applied Artificial Intelligence

## Calendar Description:

Artificial intelligence (AI) is a broad term that refers to a collection of problem solving techniques.

These techniques differ from traditional techniques in that they make use of both a knowledge base and an inference mechanism by which to apply knowledge.

We will discuss various concepts and techniques relevant to artificial intelligence.

Topics to be discussed

include search and game playing, logic and reasoning, rule-based systems and learning.

## Course Outline

The following list is an approximate guide for the material to be covered in the course.

1. Introduction: goals and definitions of artificial intelligence; intelligent agents and their environment.
2. Search: state space problem formulation and representation; uninformed search; heuristic search; iterative improvement; constraint satisfaction; game-playing.
3. Logic: propositional and first-order logic; unification and resolution; forward and backward chaining.
4. Uncertainty: probabilistic reasoning; decision-making.
5. Introduction to advanced topics: learning; soft computing; ontologies.

## Schedule, Textbook, and Marking Scheme

**Schedule**

The course consists of scheduled weekly lectures and tutorials. New material will be presented during the lectures. The tutorials will provide help with assignments, problems, and other course-related questions. The course has projects, which the students are expected to do outside of scheduled lecture and tutorial hours.

Lectures: Thursdays 11:30-02:20; DWE 3522
Tutorials: Thursdays 7:00PM to 7:50PM – DWE 3522

**Required Textbook**

Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach", Second edition, Prentice Hall, 2003.

**Marking Scheme**

Project 1 (10%)
Project 2 (10%)
Project 3 (10%)
Project 4 (10%)
Final Exam (60%)
To pass the course, you need a total mark of 50% or more. You do not necessarily need to pass the exam.

## Course Webpage

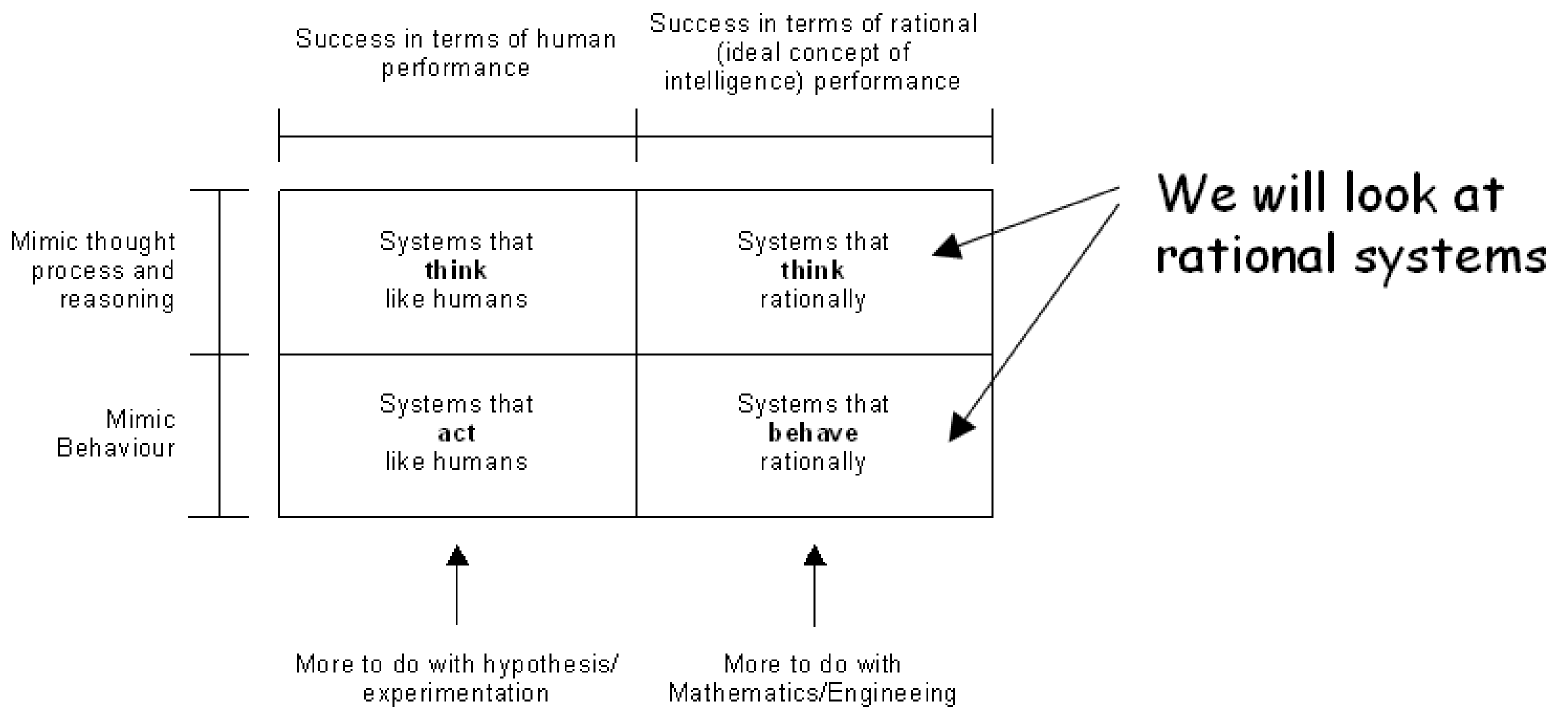
http://pami.uwaterloo.ca/~basir/ECE457/

## Artificial intelligence is all around us

- Computer players in video games
- Robotics
  - Assembly-line robots, auto-pilot, Mars exploration robots, RoboCup, etc.
- Expert systems
  - Medical diagnostics, business advice, technical help, etc.
- Natural language
  - Spam filtering, translation, document summarization, etc.

## Definition and Categorization

- How should we define Artificial Intelligence (AI) ?
- Can organize it into categories based on **goals** and **methodologies**.



Success in terms of human performance
Success in terms of rational (ideal concept of intelligence) performance
Mimic thought process and reasoning
Mimic Behaviour
Systems that think like humans
Systems that think rationally
Systems that act like humans
Systems that behave rationally
More to do with hypothesis/ experimentation
More to do with Mathematics/Engineeing
We will look at rational systems

## Goals

- There is a scientific vs. engineering goal for AI.
  - Scientific goal for AI is to develop concepts and mechanisms to **understand** biological intelligent behavior.
  - Engineering goal for AI is to develop concepts, theory and practice of **building** intelligent machines.

- Viewed differently, our goals might include…
  - Replicate human intelligence and reasoning.
  - Solve hard and knowledge intensive problems.
  - Develop a connection between perceptions and actions.
  - Enhance human-human, human-machine, iteractions.
  - And so forth…

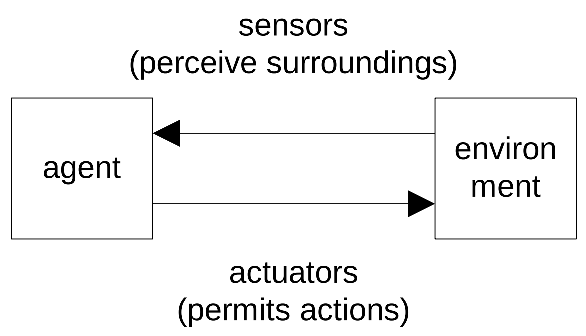
## Rational Systems

- Mentioned that in engineering, we are interested in **rational** systems.

- **Thinking** rationally means using logic to achieve goals via logical inferencing.
  - Hard to represent informal knowledge
  - Not all problems solvable in this manner (e.g., uncertainty).

- **Behaving** rationally means perceiving the “world” and acting to achieve some goals given a set of beliefs.
  - Amenable to computation.
  - More general that inferencing (but can use inferencing).
  - Actions taken to achieve a goal are not necessarily “correct”, but accomplish task at hand.

## Intelligent Agents



- Text introduces the concept of an “agent” and “agent-based systems”.
  - An agent is something that **senses** its environment and **acts** on it.

- We would like to design **rational agents**.
  - A **rational agent** is one that for each perceived sequence of events, it does what is expected to maximize performance on the basis of perceptual history and built-in knowledge (leads to concept of **autonomy**).

## Types of Agents

- Different categories of agents (see text):
  - **Simple Reflex Agents**
    - Table-lookup approach; needs fully-observable environment.
  - **Model-Based Reflex Agents**
    - Adds state information to handle partially observable environments.
  - **Goal-Based Agents**
    - Adds concept of goals to augment knowledge to help choose best actions.
  - **Utility-Based Agents**
    - Adds utility to decide "good" and "bad" with conflicting goals.
  - **Learning Agents**
    - Adds ability to learn situations that affect performance; decides how to change things to improve.

- Essentially, agent design is more or less complex depending on the particular characteristics of its **environment**.

## Environments

- Agents work within an **environment** with certain characteristics.
- It's useful to identify and understand a small number of possible dimensions for an environment that can influence the design of an agent (as previously mentioned).
- E.g., is an environment …
  - **Fully Observable vs. Partially Observable**
  - **Deterministic vs. Stochastic**
  - **Episodic vs. Sequential**
  - **Static vs. Dynamic**
  - **Discrete vs. Continuous**
  - **Competitive vs. Co-operative**

  - See Ch. 2 in text for more on environments.

## Possible Applications of AI Concepts

- **Game playing**:
  - Have well-defined rules with moves that can be searched intelligently.
- **Theorem proving**:
  - Proving correctness of this or that (e.g., circuits).
- **Diagnostic and Expert Systems**:
- **Robotics:**
  - Intelligent responses to some environment such as obstacle avoidance.
- **Pattern recognition**,
- **Language processing**,
- And so forth…

## Search

- Can often solve a problem using **search**.

Two requirements to use search:

- **Goal Formulation**.
  - Need goals to **limit search** and **allow termination**.
- **Problem formulation**.
  - Compact representation of problem space (**states**).
  - Define **actions** valid for a given state.
  - Define **cost** of actions.
- Search then involves moving from state-to-state in the problem space to find a goal (or to terminate without finding a goal).

## Outcome of Search

- Possible outcomes:
  - **Goal** itself (i.e., does problem have solution?)
  - **Path** from initial state to goal state (i.e., sequence of steps to achieve something?)

- Search assumes that an environment is:
  - Static,
  - Observable,
  - Discrete, and
  - Deterministic.

## Performance of Search

- Need to measure the performance of a search; i.e., given a search **strategy**, how good is it?

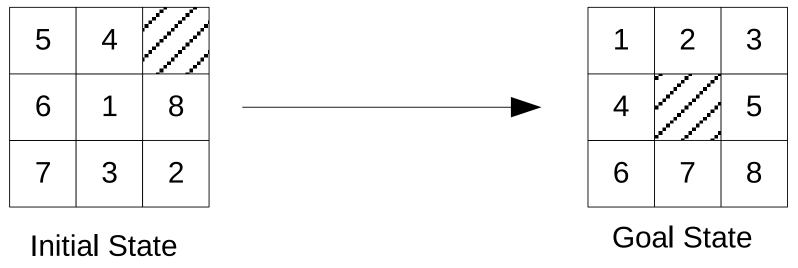
Four criteria:

- **Completeness**
  - Is the search strategy guaranteed to find a solution?
- **Optimality**
  - Is the solution found the best possible?
- **Time Complexity**
  - How long does the search strategy take to run?
- **Space Complexity**
  - How much memory does the search strategy require?

## Problem Formulation

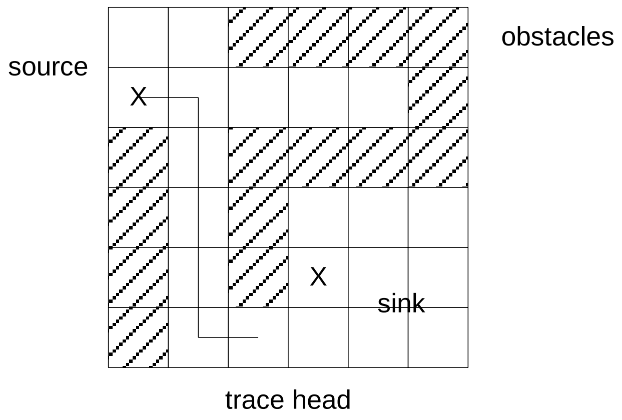
- Search requires a well-defined problem space including:
  - **Initial state**
    - A search starts from here.
  - **Goal state and goal test**
    - A search terminates here.
  - **Sets of actions**
    - This allows movement between states (successor function).
  - **Concept of cost** (action and path cost)
    - This allows costing a solution.
- The above defines a **well-defined state-space formulation** of a problem.
- Note: A state can represent either a complete configuration of a problem (e.g., 8-puzzle) or a partial configuration of a problem (e.g., routing).

## Problem Formulation Example – 8 Puzzle



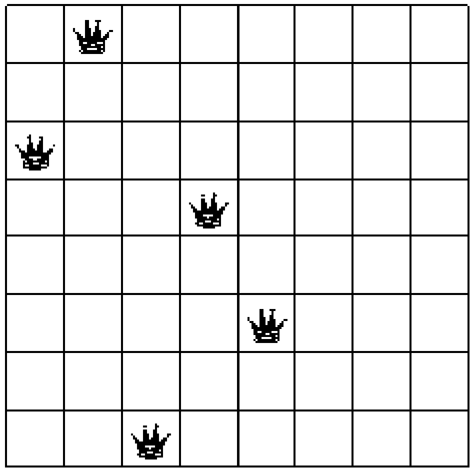
- Want to take an initial arrangement of tiles and get them into a desired arrangement.
  - States: encode location of each of 8 tiles and the blank.
  - Initial State: any arrangement of tiles.
  - Goal State: predefined arrangement of tiles.
  - Actions: slide blank UP, DOWN, LEFT or RIGHT (without moving off the grid).
  - Goal Test: position of tiles and blank match goal state.
  - Cost: sliding the blank is 1 move (cost of 1). Path cost will equal the number of moves from initial state to goal state.
- Solution is a path of moves to get to the goal state. Fewest moves is best.

# Problem Formulation Example – Route Finding



- Problem is to connect a source to a sink while avoiding obstacles on 2-D grid.
  - States: ordered pair (x,y) of the trace head.
  - Initial State: trace at location of source.
  - Goal State: trace at location of sink.
  - Actions: move trace head UP, DOWN, LEFT or RIGHT (without moving off the grid and avoiding obstacles).
  - Goal Test: trace at location of sink.
  - Cost: moving trace head costs 1. Path cost is length of trace.
- Solution might be (i) trace with shortest path or (ii) a path if one even exists.

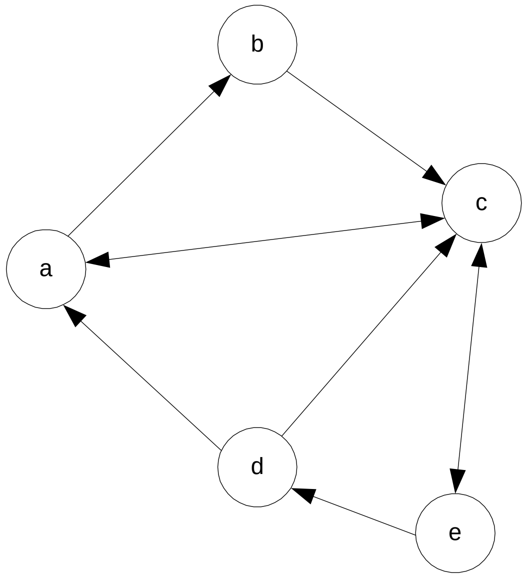
## Problem Formulation Example – 8 Queens Problem



- Position 8 queens on a chess board such that no 2 queens attack each other.
  - States: placement of 0 to 8 queens on board such that no attacks.
  - Initial State: no queens placed.
  - Goal State: position of 8 queens in non-attacking arrangement.
  - Actions: place next queen into next column in a non-attacking position.
  - Goal Test: position of queens in non-attacking position.
  - Cost: placing next queen costs 0. Solution is simply one of many goal states.
- Solution is any goal state (no concept of path…)

## Problem Formulations, Graphs and Search Trees

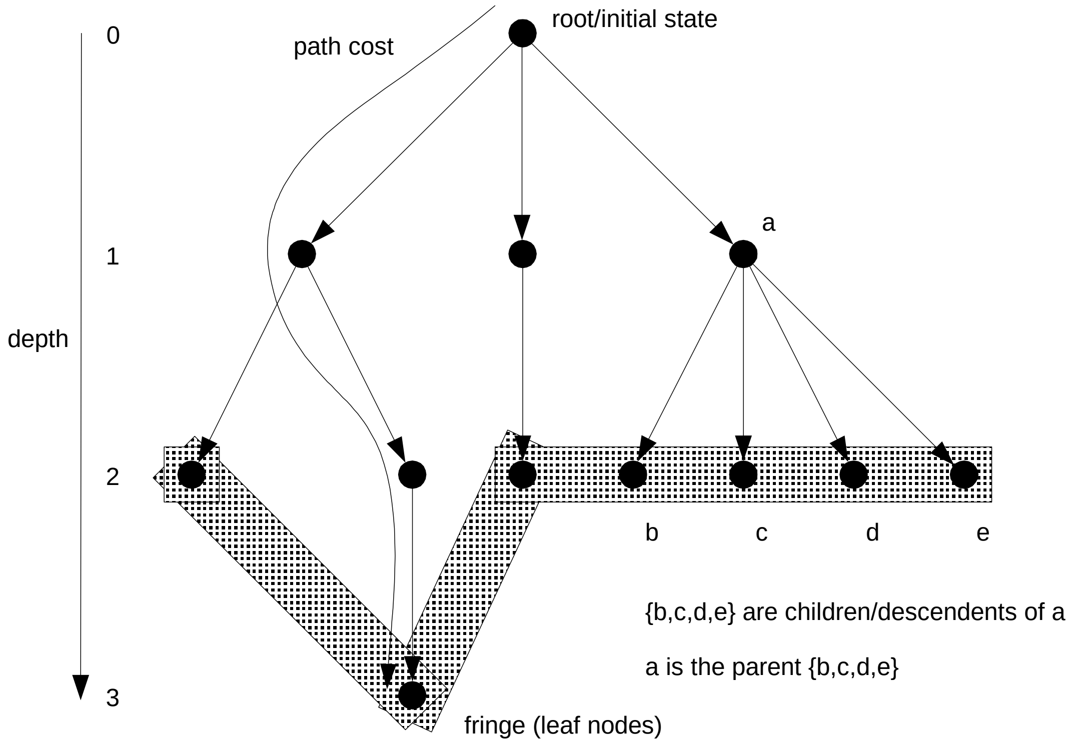
- Problems formulated for search has an analogy with directed graphs:
  - Nodes represent states (configurations or partial configurations).
  - Edges represent actions (directional)
- Output of search is either a path in the graph or a node that passes the goal test.



## Problem Formulations, Graphs and Search Trees

- When we search in a graph, we build a **search tree**.
- Search trees also have nodes and edges:
  - Nodes are states (root is the initial state).
  - Nodes have a parent and descendants (neighboring states reachable via an action).
  - Nodes at the bottom of the tree are **leaf nodes** and define the **fringe**.
    - We will find a goal state in the fringe.
  - Edges represent actions and have costs.
- Each tree node has a path from the root and a path cost from the root.
- Nodes in a search tree are more than just states since they hold state information, reference to actions, path costs, etc.

# Illustration of Search Tree



{b,c,d,e} are children/descendents of a

a is the parent {b,c,d,e}

## Comments on Search Trees

- Search trees are superimposed over top of the graph representation of a problem.
- While the graph might be finite, the search tree can be either finite or infinite.
  - Infinite if we allow repeated states due to reversible actions and/or cycles of actions.
- Some useful terminology:
  - The maximum number of children possible for a node, b, in a search tree is called the **branching factor**.
  - A finite tree has a **maximum depth**, d.
  - Any node in the search tree occurs at a **level**, l, in the tree ($l \leq d$).

## Template of Generic Search

- Given concept of graph and search tree, generic search is a repetition of **choose, test, and expand**.

- A particular search strategy (uninformed or informed, more in a minute …) influences how we choose the next node to consider in the search! (this is where types of searches differ).

- Generally use a **queue** to store nodes on the fringe to be expanded. Different search strategies use different queue structures.

## Template of Generic Search

```
open_queue.insert(init_state);
while (open_queue.size() != 0) {
    curr_state = open_queue.remove_front();
    if (is_goal(curr_state)) {
            return success; // and solution.
    }
    closed_queue.insert(curr_state); // state visited.

    child_state = expand(curr_state); // other states reachable via an action.
    for (i = 1 ; i <= child_state.size() ; i++) {
            if (open_queue.find(child_state[i]) || closed_queue.find(child_state[i])) {
                    ; // child state already expanded or in fringe.
            } else {
                    open_queue.insert(child_state[i]);
            }
            // nb: what if better path to child states?????
    }
};
return failure; // no solution.
```

## Repeated States

- During search, desirable to avoid repeated states (i.e., revisiting the same state again and again).
  - This is possible due to reversible actions and/or cycles of actions.

- We can keep track of visited states and ignore repetition via a closed queue.
  - However, what happens if, upon revisiting a state, we find a that we have a better path/solution to the revisited state?
  - We need to deal with this (not shown in the generic template!).

## Types of Search

Two main types of search:

- Uninformed search:
  - Has only the knowledge provided in the problem formulation (e.g., actions, costs, goal or not goal, etc.)
- Informed search:
  - Has additional knowledge in order to better judge the overall promise of an action in reaching a goal; e.g., has an estimate of cost to goal from current location).