CSC384

Introduction to Artificial Intelligence

Sinan Li

2024

CONTENTS

I Notes **5**

1 **Chapter 1**
Introduction to Artificial Intelligence

1.1 Artificial Intelligence (AI) *7*
1.1.1 Intelligence *7*
1.1.2 The Turing Test *7*
1.1.3 Computational Intelligence *8*
1.2 Rationality *8*
1.3 Subareas of AI *9*
1.3.1 Further Courses in AI *9*
1.4 A Brief History of AI *10*

2 **Chapter 2**
Search

2.1 Search Problems *11*
2.1.1 Formalizing a Problem as a Search Problem *11*
2.1.2 Graphical Representation *14*
2.1.3 Algorithms for Search *15*

3 **Chapter 3**
Game Tree Search

4

Chapter 4
Constraint Satisfaction Problems

5 Chapter 5
Representing and Reasoning under Uncertainty

6 Chapter 6
Symbolic Knowledge Representation and Reasoning

II Appendices **25**

Bibliography **27**

Part I

Notes

CHAPTER

INTRODUCTION TO ARTIFICIAL INTELLIGENCE 1

1.1 Artificial Intelligence (AI)

Artificial Intelligence is a branch of computer science utilizing *computational* ideas and examining how we can achieve *intelligent* behavior through computation.

1.1.1 Intelligence

The ability to apply knowledge to manipulate one's environment or to think abstractly as measured by objective criteria (such as tests)

— *Merriam-Webster Dictionary*

Human features that are considered intelligent includes the ability to learn, understand, reason, plan, communicate, and perceive. For example, a human can learn to play a game, understand the rules of the game, reason about the best moves to make, plan a sequence of moves, communicate with other players, and perceive the state of the game.

1.1.2 The Turing Test

The Turing Test is a test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human. The test was introduced by Alan Turing in his 1950 paper, *Computing Machinery and Intelligence*, while working at the University of Manchester [1]. Turing proposed that a human evaluator would judge natural language conversations between a human and a machine designed to generate human-like responses. The evaluator would be aware that one of the two partners in conversation is a machine, and all participants would be separated

from one another. The conversation would be limited to a text-only channel such as a computer keyboard and screen so that the result would not be dependent on the machine's ability to render words as speech. If the evaluator cannot reliably tell the machine from the human, the machine is said to have passed the test. The test does not check the ability to give the correct answer to questions; it checks how closely the answer resembles typical human answers. The conversation is limited to a single topic chosen by the examiner.

- Turing provided some very persuasive arguments that a system passing the Turing test is *intelligent*.
  - We can only really say it *behaves* like a human
  - **No guarantee** that it *thinks* like a human
- The Turing test does not provide much traction on the question of **how to build** an intelligent system.

Why not just simulate human's brain?

- Brains are very good at making rational decisions, but **not perfect**.
- Brains **aren't as modular** as software, so hard to reverse engineer!
- Computers and Humans have quite **different abilities**.
  - *Memory* and *simulation* are key to decision making.
  - *Perceptual tasks* (vision, sound, etc.) are effectively accomplished by architectures related to the way the brain works (deep neural networks).

1.1.3 Computational Intelligence

Artificial Intelligence tries to understand and model *intelligence* as a **computational process**. Thus we try to construct systems whose **computation** achieves or approximates the desired notion of intelligence.

Other areas interested in the study of intelligence lie in other areas or study, e.g., cognitive science which focuses on human intelligence. Such areas are very related, but their central focus tends to be different.

1.2 Rationality

Formally, we can define an **agent** as something that **perceives** and **acts** in an *environment*. An agent can be a *human*, a *robot*, or a *software agent*.

A **rational agent** is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome. Rationality is distinct from omniscience (knowing everything) and omnipotence (being able to do anything). Rationality maximizes **expected utility**, which is the sum of the utility of each possible outcome of an action weighted by its probability of occurring.

Rationality is measured by the *outcome*, not the *action* itself. It is a precise *mathematical* notion of of what it means to do *the right thing* in any particular circumstance. Provides

- A **precise mechanism** for analyzing and understanding the properties of this ideal behaviour we are trying to achieve.
- A **precise benchmark** against which we can measure the behaviour the systems we build.

Trying/Expectation

Rational action is not always equal to rational decision.

1. We often don't have **full control** or **knowledge** of the world we are interacting with.
2. We usually don't know **precisely** what the **effects** of our actions will be.

In some contexts we can *simplify* the computational task by assuming that we do have full knowledge/control.

1.3 Subareas of AI

A common misconception is to equate AI with Machine Learning. But AI is much more than that. This course will not focus on Machine Learning, but rather on the other subareas of AI. What we cover is not an exhaustive list of all subareas of AI, but rather a starting point for further exploration.

- **Perception**: vision, speech understanding, etc.
- **Machine Learning**, **Neural Networks**
- **Robotics**
- **Natural Language Processing**
- **Reasoning and Decision Making**
  - **Symbolic Knowledge Representation**
  - **Reasoning** (logical, probabilistic)
  - **Decision Making** (search, planning, decision theory)

1.3.1 Further Courses in AI

- Perception: vision, speech understanding, etc.
  - CSC487H1 "Computational Vision"
  - CSC420H1 "Introduction to Image Understanding"
- Machine Learning, Neural networks
  - CSC311H "Introduction to Machine Learning"
  - CSC412H "Probabilistic Learning and Reasoning"

    - CSC413H1 "Neural Networks and Deep Learning"
- Robotics
    - Engineering courses
- Natural language processing
    - CSC401H1 "Natural Language Computing"
    - CSC485H1 "Computational Linguistics"
- Reasoning and decision making
    - CSC486H1 "Knowledge Representation and Reasoning"

1.4 A Brief History of AI

- 1940-1950: Early days
    - 1943: McCulloch & Pitts: Boolean circuit model of brain
    - 1950: Turing's "Computing Machinery and Intelligence"
- 1950—70: Excitement: Look, Ma, no hands!
    - 1950s: Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
    - 1956: Dartmouth meeting: "Artificial Intelligence" adopted
    - 1965: Robinson's complete algorithm for logical reasoning
- 1970—Early 2000: Knowledge-based approaches
    - 1969—79: Early development of knowledge-based systems
    - 1980—88: Expert systems industry booms
    - 1988—93: Expert systems industry busts: "AI Winter"
    - 1997: IBM's Deep Blue beats chess grandmaster Garry Kasparov
- Early 2000— present: Statistical approaches
    - Resurgence of probability, focus on uncertainty
    - Agents and learning systems... "AI Spring"
    - 2007: DARPA Urban Challenge – CMU autonomous vehicle drives 55 miles in an urban environment while adhering to traffic hazards and traffic laws.
    - 2016: AlphaGo beats 9-Dan pro Go player Lee Sedol
    - 2017: AlphaGo Zero – learns by playing with itself
    - 2022: Large Language Models (LLM) Chat Generative Pre-trained Transformer, which has been fueled by advances in Neural Net architecture and access to big data.

There are many **unsolved** problems yet... including lots of **legal/ethical** ones.

CHAPTER

2 SEARCH

2.1 Search Problems

Searching is one of the most *fundemental techniques* in AI, underlying sub-module in many AI systems. Ai can *solve* many problems that homansare not good at, and achieve *super-human performance* in many domains (e.g. chess, go, etc.).

- **Benefits**
  - Useful as ageneral algorithmic technique for solving problems, both in AI and in other areas.
  - Outperform humans in some areas (e.g. games).
  - Practical:
    - Many problems don't have specific algorithms for solving them.
    - Useful in approximation (e.g., local search in optimization problems).
  - Some critical aspects of intelligent behaviour, e.g., planning, can be cast as search.
- **Limitations**
  - Only shows how to solve the problem once we have it correctly formulated.

2.1.1 Formalizing a Problem as a Search Problem

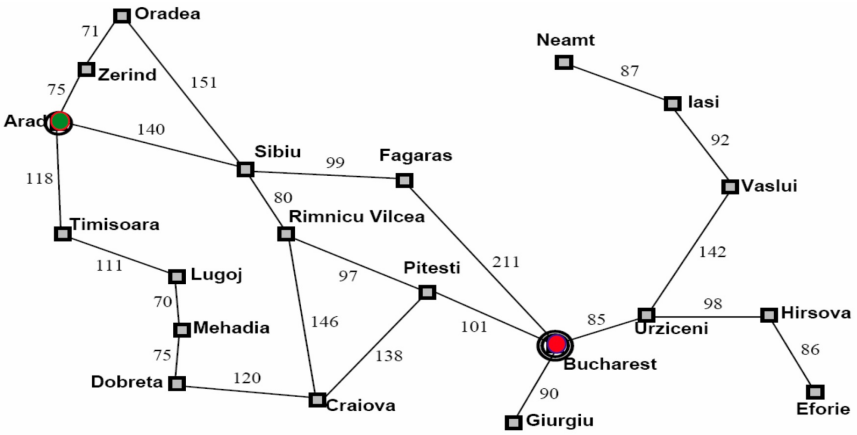
- Necessary components

1. **State Space**: A **state** is a representation of a *configuration* of the problem domain. The state space is the *set of all states* include in our model of the problem.
2. **Initial State**: The starting configuration.
3. **Goal State**: The configuration one wants to achieve.
4. **Actions** (or State Space Transitions): Allowed changed to move from one state to another.

- Optional Ingredients
  - *Costs*: Representing the cost of moving from state to state.
  - *Heuristics*: Help guide the search process.

Once a search problem is formalized, there are a number of algorithms one can use to solve it. A **solution** is a *sequence of actions* or moves that can transform the **initial state** into a **goal state**.

Example (Romania Travel)**.** We want to travel in Romania from Arad to Bucharest as fast as possible.



Each state would be a city.

- **State Space**: The set of all cities on the map.
- **Initial State**: Arad.
- **Goal State**: Bucharest.
- **Actions**: Driving between neighbouring cities.

♢

Example (Water Jugs)**.** We have a 3-liter jug and a 4-liter jug. We can fill either jug to the top from a tap, or we can empty either jug onto the ground. We can also pour the contents of one jug into the other until the receiving jug is full or the pouring jug is empty.

Suppose initially the 4-liter jug is full, we want to have exactly 2 liters in the 3-liter jug.

We can use a pair of numbers to represent the state of the system: the amount of water in the 3-liter jug and the amount of water in the 4-liter jug.

- **State Space**: The set of all pairs of numbers (a, b) where a is the amount of water in the 3-liter jug and b is the amount of water in the 4-liter jug.
- **Initial State**: $(0, 4)$.
- **Goal State**: $(2, 0)$, $(2, 1)$, $(2, 2)$, $(2, 3)$, $(2, 4)$.
- **Actions**:
  - Fill the 3-liter jug from the tap.
  - Fill the 4-liter jug from the tap.
  - Empty the 3-liter jug onto the ground.
  - Empty the 4-liter jug onto the ground.
  - Pour the contents of the 3-liter jug into the 4-liter jug.
  - Pour the contents of the 4-liter jug into the 3-liter jug.

> **Remark**
>
> When formalizing a search problem, always consider these questions:
>
> 1. Can we reach all states fron any given start state?
> 2. Will all actions result in a change of state?
>
>   **No!** Imagine you have $(3, 4)$ and you try to fill the 3-liter jug from the tap. You will still have $(3, 4)$.

♢

More Complex Situations

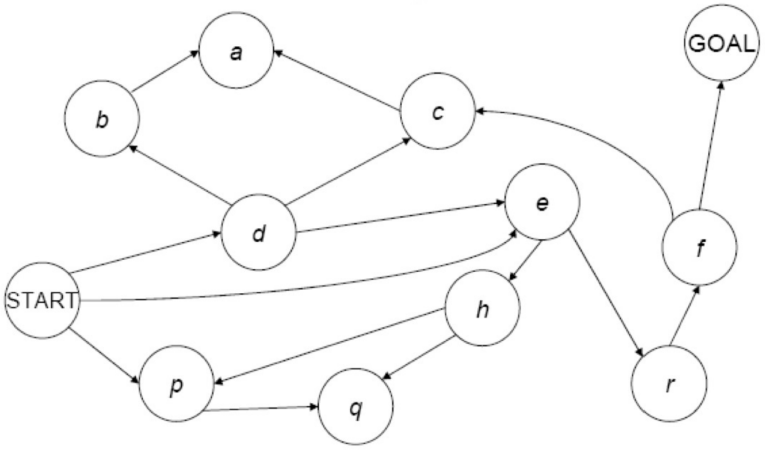
In more complex situations,

- Actions may lead to **multiple states**.

  For example, filpping a coin may lead to heads or tails.
- We may not be **sure of a given state**

  For example, when prize is behind door 1, 2, or 3.
- Such situations require techniques for reasoning under uncertainty: assign probabilities to given outcomes.

2.1.2 Graphical Representation



Assuming a finite search space, the

- **vertices** represent states in the search space; and the
- **edges** represent transitions resulting from actions (or successor functions).

Search Tree

Definition 2.1.1 Search Tree

A **search tree** is a *directed graph* where

- Each node represents a state.
- Each edge represents an action.
- The root node represents the initial state.
- The leaf nodes represent goal states.

A search tree reflects the behaviour of an algorithm as it walks through a search problem. It has two important properties:

- **Solution depth**, denoted d, the depth of the shallowest goal node in the tree.
- **Maximum branching factor**, denoted b, the maximum number of children of any node in the tree.

Remark

Note that the **same** state may appear **multiple times** in a search tree.

Remark

It is important to distinguish between **states** from **nodes**.

- A **state** represents a possible configuration of the world.
- A *node* is a data structure constituting part of a search tree. It includes
  - a **state** and
  - the **parent node**,
  - the **action** that led to this node, and
  - the **cost** of the path from the initial node to this node.
- Intuitively speaking, each node corresponds with a path from the initial state to the node's state.
- Two **different nodes** are allowed to contain the **same world state**.

2.1.3 Algorithms for Search

- **Input**
  - **Initial node**
  - **Successor Function** $S(x)$
    returns the set of nodes that can be reached from node c via a single action.
  - **Goal Test Function** $G(x)$
    returns true if node c satisfies the goal condition.
  - **Action Cost Function** $C(x, a, y)$
    returns the cost of moving from node x to node y using action a.
    Note that $C(x, a, y) = \infty$ if y is not reachable from a via a.
- **Output**
  - A *sequence of actions* that transforms the initial node satisfying the goal test.
  - The sequence might be, **optimal in cost** for some algorithms, **optimal in length** for some algorithms, come with **no optimality** guarantees from other algorithms.
- **Procedure**
  - Put nodes have not yet expanded in a list called the `Frontier` (or **Open**).
  - Initially, only the *initial node* is in the `Frontier`.
  - At each iteration, pull a node from the `Frontier`, apply $S(x)$, and insert the children back into the `Frontier`.
  - Repeat until pulling a goal node.

Algorithm 1 Tree Search Algorithm

```
1: function TreeSearch(Frontier, Successors, Goal?)
2:     if Frontier is empty then
3:         return failure
4:     end if
5:     Curr ← select state from frontier
6:     if Goal?(Curr) then
7:         return Curr
8:     end if
9:     Frontier' ← Frontier − { Curr } ∪ Successors(Curr)
10:    return TreeSearch(Frontier', Successors, Goal?)
11: end function
```

Note that the search terminates only when a goal node is expanded into the `Frontier`.

CHAPTER

3 GAME TREE SEARCH

CHAPTER

CONSTRAINT SATISFACTION PROBLEMS 4

CHAPTER

REPRESENTING AND REASONING UNDER UNCERTAINTY

5

CHAPTER

SYMBOLIC KNOWLEDGE REPRESENTATION AND REASONING

6

Part II

Appendices

BIBLIOGRAPHY

[TUR50] A. M. TURING. "I.—COMPUTING MACHINERY AND INTELLIGENCE". In: *Mind* LIX.236 (Oct. 1950), pages 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433 (cited on page 7).