University of West Attica (UniWA)
Dept. Of Informatics and Computer Engineering
Athens, Greece

The differences between problem solving approaches in AI

Course: Artificial Intelligence
Professor: Katerina Georgouli
Author: Nikola Micic

9.12.2019.

Contents

Introduction

In computer science, artificial intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans. Leading AI textbooks define the field as the study of „intelligent agents“: any device that perceives its environment and takes actions that maximize its change of successfully achieving its goals. The term „artificial intelligence“ is often used to describe machines (or computers) that mimic „cognitive“ functions that humans associate with the human mind, such as „learning“ and „problem solving“.
In computer science, problem solving encompasses a number of techniques known as algorithms, heuristics, etc.
Problems are the issues which comes across any system. A solution is needed to solve that problem.
When we have some particular problem we can build an artifically intelligent system to solve that problem. To do this, we need to define the problem statements first and then generating the solution by keeping the conditions in mind.

The process of solving a problem consists of five steps. There are:

1. Defining the problem – The definition of the problem must be included precisely. It should contain the possible initial as well as final situations which should result in acceptable solution.
2. Analyzing the problem – Analyzing the problem and its requirement must be done as few features can have immense impact on the resulting solution.
3. Identification of solutions – This phase generates reasonable amount of solutions to the fiven problem in a particular range.
4. Choosing a solution – From all the identified solutions, the best solution is chosen basis on the results produced by respetive
5. Implementation – After choosing the best solution, its implementation is done.

References:

1. https://www.minigranth.com/artificial-intelligence/problem-solving-in-artificial-intelligence
2. https://en.wikipedia.org/wiki/Artificial_intelligence#cite_note-Definition_of_AI-1

Search algorithms insisting at heuristic algorithms

In computer science, artificial intelligence and mathematical optimization, a **heuristic** (from Greek “eurisko” “I find, discover”) is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed.
The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand. This solution may not be the best of all the solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time.
Heuristics may produce results by themselves, or they may be used in conjunction with optimization algorithms to improve their efficiency.

The trade-off criteria for deciding whether to use a heuristic for solving a given problem include the following:

- **Optimality***:* When several solutions exist for a given problem, does the heuristic guarantee that the best solution will be found? Is it actually necessary to find the best solution?
- **Completeness***:* When several solutions exist for a given problem, can the heuristic find them all? Do we actually need all solutions? Many heuristics are only meant to find one solution.
- **Accuracy and precision***:* Can the heuristic provide a confidence interval for the purported solution? Is the error bar on the solution unreasonably large?
- **Execution time**: Is this the best known heuristic for solving this type of problem? Some heuristics converge faster than others. Some heuristics are only marginally quicker than classic methods.

In some cases, it may be difficult to decide whether the solution found by the heuristic is good enough, because the theory underlying heuristics is not very elaborate.
Example of heuristic making an algorithm faster occurs in certain search problems. Initially, the heuristic tries every possibility at each step, like the full-space search algorithm. But it can stop the search at any time if the current possibility is already worse than the best solution already found.
Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

- A search problem consists of:
    - **A State Space**: Set of all possible states where can you be.
    - **A Start State**: The state from where the search begins.

    - **A Goal Test**: A function that looks at the current state returns whether or not it is the goal state.
- The **Solution** to a search problem is a sequence of actions, called the **plan** that transforms the start state to the goal state.
- This plan is achieved through search algorithms.

The search algorithms are divided in two categories:

- **Uninformed Search Algorithm**
- **Informed Search Algorithm**

The Uninformed Search Algorithms have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called Blind search.

Examples of Uninformed Search Algorithms are Depth First Search, Breadth First Search and Uniform Cost Search.

Each of these algorithms will have:

- A problem **graph**, containing the start node S and the goal node G.
- A **strategy**, describing the manner in which the graph will be traversed to get to G .
- A **fringe**, which is a data structure used to store all the possible states (nodes) that you can go from the current states.
- A **tree**, that results while traversing to the goal node.
- A solution **plan**, which the sequence of nodes from S to G.

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a ‘search key’), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

Uniform Cost Search (UCS) is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is least.

The Informed Search Algorithms have information on the goal state, which helps in more efficient searching.

Examples of Informed Search Algorithms are Greedy Search and A* Tree Search.

Search Heuristics: In an informed search, a heuristic is a function that estimates how close a state is to the goal state. Different heuristics are used in different informed algorithms discussed below.

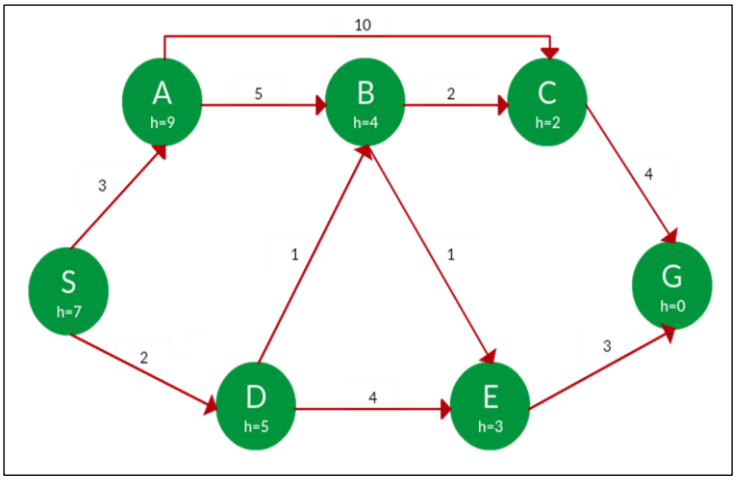
In **Greedy Search** we expand the node closest to the goal node. The “closeness” is estimated by a heuristic h(x). A heuristic h is defined as h(x) = Estimate of distance of node x from the goal node. Lower the value of h(x), close is the node from the goal.

A* Tree Search, or simply known as A* Search, is a search strategy used for finding an efficient path between two points (represented as nodes in the tree structure). A* Search combines the strengths of uniform-cost search and greedy search. In this search, the heuristic is the summation of the cost in UCS, denoted by g(x), and the cost in greedy search, denoted by h(x). The summed cost is denoted by f(x).
Heuristic: $f(x) = g(x) + h(x)$.

- h(x) is called the forward cost, and is an estimate of the distance of the current node from the goal node.
- g(x) is called the backward cost, and is the cumulative cost of a node from the root node.
- A* search is optimal only when for all nodes, the forward cost for a node h(x underestimates the actual cost h*(x) to reach the goal. This propery of A* heuristic is called admissibility ($0 \leq h(x) \leq h^*(x)$).
- Chose the node with the lowest f(x) value.

A* Tree search is complete – it will always find a solution if one exists. For A* search to be optimal it must be used with an admissibile heuristic. An admissibile heuristic, also nown as an optimistic heuristic, never overestimates the cost of reaching the goal.
When A* search reaches a goal state it has found a solution with a total cost less than or equal to the estimated cost of any unsearched paths. If the estimated cost are optimistic then the true cost of any solutions discovered by traversing the unsearched paths are guaranteed to be no better than solution already found.

Example: Find the path to reach from S to G using A* search.



Solution: Starting from S, the algorithm computes g(x) + h(x) for all nodes in the fringe at each step, choosing the node with the lowest sum. The entire working is shown in the table below.

Note that in the fourth set of iteration, we get two paths with equal summed cost f(x), so we expand them both in the next set. The path with lower cost on further expansion is the chosen path.

Path	H(x)	G(x)	F(x)
S	7	0	7
S -> A S -> D	9 5	3 2	12 7
S -> D -> B S -> D -> E	4 3	2 + 1 = 3 2 + 4 = 6	7 9
S -> D -> B -> C S -> D -> B -> E	2 3	3 + 2 = 5 3 + 1 = 4	7 7
S -> D -> B -> C -> G ***S -> D -> B -> E -> G***	0 ***0***	5 + 4 = 9 ***4 + 3 = 7***	9 ***7***

Path: S -> D -> B -> E -> G
Cost: 7

References:

1. https://en.wikipedia.org/wiki/Heuristic_(computer_science)
2. https://www.geeksforgeeks.org/search-algorithms-in-ai/
3. http://how2examples.com/artificial-intelligence/tree-search
4. A.R. Greef and R. Reinecke, “Problem solving using Artifical Intelligence Techniques”

Knowledge-based systems

A knowledge-based system (KBS) is a computer program that captures and uses knowledge base to solve complex problems. A KBS assists with solving problems, particularly complex issues, by artificial intelligence. These systems are primarily used to support human decision making, learning, and other activities. A knowledge-based system is a major area of artificial intelligence. These systems can make decisions based on the data and information that resides in their database. In addition, they can comprehend the context of the data being processed.

The typical architecture of a knowledge-based system, which informs its problem-solving method, includes a knowledge base and an inference engine. The knowledge base contains a collection of information in a given field - medical diagnosis, for example. The inference engine deduces insights from the information housed in the knowledge base. Knowledge-based systems also include an interface through which users query the system and interact with it.

Over the years, KBS have been developed for a number of applications. MYCIN, for example, was an early KBS created to help doctors diagnose diseases. Knowledge-based systems have also been employed in applications as diverse as avalanche path analysis, industrial equipment fault diagnosis and cash management.

While the earliest knowledge-based systems were almost all expert systems, the same tools and architectures can and have since been used for a whole host of other types of systems. Virtually all expert systems are knowledge-based systems, but many knowledge-based systems are not expert systems.
In contrast to conventional computer-based information systems, a KBS has several advantages. They provide excellent documentation while handling large quantities of unstructured data in an intelligent way. A KBS helps improve decision making and enables users to work at greater levels of expertise, productivity, and consistency. In addition, a KBS is useful when expertise is not available, or when information must be stored effectively for future use. It also provides a common platform for integrating knowledge on a large scale. Finally, a KBS is capable of generating new knowledge by using the stored data.
The limitations of knowledge-based systems are the abstract nature of the concerned knowledge, acquiring and manipulating large volumes of information or data, and the limitations of cognitive and other scientific techniques.

Knowledge Based Systems (KBSs) are computer systems that contains stored knowledge and solve problems like humans.
Their core components are:

- Knowledge-base (rules, facts, meta-knowledge)
- Inference engine (reasoning and search strategy for solution, other services)

Characteristics of KBSs:

- Intelligent information processing systems
- Representation of domain of interest -> symbolic representation
- Problem slving -> by symbol – manipulation

Structure of KBS:

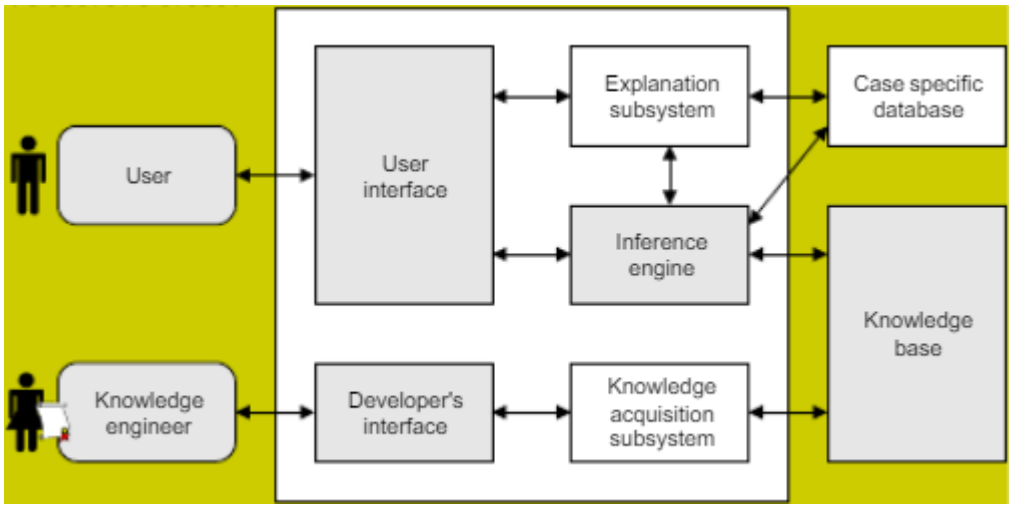


Figure 1: Structure of KBS

Knowledge engineer selects the knowledge representation method and reasoning strategy. Knowledge engineer is also responsible for the implementation of knowledge-based system. Other tasks are:

- knowledge acquisition and design of KBS: determination, classification, refinement and formalization of methods, thumb-rules and procedures
- verification and validation of KB
- KB maintenance

Knowledge acquisition subsystem, its main tasks are:

- checking the syntax of knowledge elements
- checking the consistency of KB (verification, validation)
- knowledge extraction, building KB
- automatic logging and book-keeping of the changes of KB
- tracing facilities (handling breakpoints, automatic monitoring and reporting the values of knowledge elements)

Knowledge-base (KB): It contains knowledge about the field of interest (in natural languagelike formalism). It is a symbolically described system-specification. Knowledge-representation method!

Inference engine: It is an engine of problem solving (general problem solving knowledge) and supports the operation of the other components. Problem solving method!

Case-specific database: It is an auxiliary component. It comprises of specific information (information from outside, initial data of the concrete problem) and information obtained during reasoning.
Explanation subsystem: It contains the explanation of system's actions in case of user's request.
User interface (→ user): dialogue on natural language (consultation/ suggestion).

References:

1. https://searchcio.techtarget.com/definition/knowledge-based-systems-KBS
2. https://www.scribd.com/document/23282044/knowledge-based-system
3. https://en.wikipedia.org/wiki/Knowledge-based_systems

Machine learning techniques

Machine Learning is about building programs with adaptable parameters that are adjusted automatically based on the data they receive. This allows them to improve their behavior by independently adapting to the previously seen data.

Machine Learning is a subfield of Artificial Intelligence: The algorithms that make up Machine Learning are like building blocks that make computers learn to behave more intelligently. They do this by generalizing data, meaning that they are able to accurately perform functions on previously unseen datasets, whereas a non-intelligent database system just stores and retrieves data items.

Machine learning algorithms fall into two categories - supervised and unsupervised. Supervised algorithms use training data that is comprised of input vectors as well as their corresponding target vectors, or expected output. Unsupervised algorithms use training data that only consist of input vectors; the algorithm creates groups and subgroups within the data, a process known as clustering.

The most well-known machine learning techniques are decision trees, neural networks and genetic algorithms.

Decision Trees (DS) is a supervised predictive model that can learn to predict discrete or continuous outputs by answering a set of simple questions based on the values of the input features it receives.
A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.
A decision tree is drawn upside down with its root at the top.

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails), each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules.
Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.

Tree models where the target variable can take a discrete set of values are called classification trees. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Classification And Regression Tree (CART) is general term for this.

Advantage of Decision Tree

- Easy to use and understand.
- Can handle both categorical and numerical data.
- Resistant to outliers, hence require little data preprocessing.

Disadvantage of Decision Tree

- Prone to overfitting.
- Require some kind of measurement as to how well they are doing.
- Need to be careful with parameter tuning.
- Can create biased learned trees if some classes dominate.

Neural Networks are a class of models within the general machine learning literature. Neural networks are a specific set of algorithms that have revolutionized machine learning. They are inspired by biological neural networks and the current so-called deep neural networks have proven to work quite well. Neural Networks are themselves general function approximations, which is why they can be applied to almost any machine learning problem about learning a complex mapping from the input to the output space.

Studying neural computation will help us to understand how the brain actually works, to understand a style of a parallel computation inspired by neurons and their adaptive connection and to solve practical problems by using novel learning algorithms inspired by the brain.

Neural networks are one of the most beautiful programming paradigms ever invented. In the conventional approach to programming, we tell the computer what to do and break big problems up into many small, precisely defined tasks that the computer can easily perform. In contrast, we don't tell the computer how to solve our problems for a neural network. Instead, it learns from observational data and figures out its own solution to the problem.

Today, deep neural networks and deep learning achieve outstanding performance for many important problems in computer vision, speech recognition, and natural language processing. They're being deployed on a large scale by companies such as Google, Microsoft, and Facebook.

Genetic Algorithms (GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance

in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate “survival of the fittest” among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

Genetic algorithms are based on an analogy with genetic structure and behavior of chromosome of the population. Following is the foundation of GAs based on this analogy:

- Individual in population compete for resources and mate
- Those individuals who are successful (fittest) then mate to create more offspring than others
- Genes from “fittest” parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
- Thus each successive generation is more suited for their environment.

The population of individuals are maintained within search space. Each individual represent a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus a chromosome (individual) is composed of several genes (variable components).

References:

1. James Le, “A Gentle Introduction to Neural Networks for Machine Learning”,2018.
2. Jonathan Shapiro, “Genetic Algorithms in Machine Learning”, 2001.
3. https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

Conclusion about solving problem based on its nature

One of the most important fields of AI is problem solving. In many instances the knowledge or information about a problem domain is difficult or unnatural to represent in data structures such as arrays or sets of numbers. Often it is more "natural" to represent a problem as a set of English language sentences or statements describing the problem domain. In these instances AI programming techniques are useful.

There are two difficulties associated with the algorithmic (step-by-step) approach to solving a class of problems:

- there are some classes of problems for which there is no algorithm that will solve every problem in a particular class
- even if there is an algorithm that will solve every problem in a particular class, the algorithm may be so inefficient as to be unsuitable for practical problems

There are 5 parts in the problem solving process:

- Part 1: illustrates methods for representing a problem for coding into a computer.
- Part 2: gives an overview of the "weak" methods that can be employed for finding the solution to a problem.
- Part 3: describes the production system architecture that most AI programmes adhere to.
- Part 4: presents methods of representing knowledge to enrich the problem representations outlined in Part 1.
- Part 5: contains an overview of planning, an advanced problem solving technique.

Part 1: **Problem Representation**

There are two well-used methods for representing a problem. The first method treats the problem as a network of inherently ordered states. The second sees the problem as a series of sub-problems which in turn have sub-sub problems and so on until a problem with an immediate solution is reached. Formally these two problem representation methods are known as:

- **the state-space representation**
- **the problem-reduction representation**

Part 2: **“Weak” Problem Solving Techniques**

Having represented a problem using one of the methods in Part 1, its solution can then be formulated as a search for a sequence of nodes in a graph that connects the root node with a terminal node. This sequence of nodes is the solution to the problem.

If a tree search is conducted by starting with the initial problem configuration at the root of the tree and continues by tracing the solution path to the final problem configuration at a terminal node, then the system conducts forward reasoning (forward chaining). If the search begins at the final problem configuration and ends at the initial problem configuration, the system conducts backwards reasoning (backward chaining).

Two exhaustive search methods are:

- **Depth First Search**
- **Breadth First Search**

Part 3: **Production Systems**

Most AI systems have a clear separation between the standard computational components of data, operations and control. Various generalizations of this computational formalism are known as production systems. The major elements of an AI production system are:

- a global data base
- a set of production rules
- a control system

Part 4: **Knowledge Representation and reasoning**

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. But how machines do all these things comes under knowledge representation and reasoning. Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

Part 5: **Advanced Problem Solving - Planning**

In order to solve most nontrivial problems, it is necessary to combine some of the "weak" search methods with one or more knowledge representation techniques. It is also necessary to divide the large problem into smaller problems that are solved separately. The separate solutions are then combined to form the full problem's solution. The methods used to decompose the original problem into sub-parts, and the ways of recording and handling interactions between the sub-parts as they are detected, are often collected under the heading of planning.

Reference:

1. A.R. Greef and R. Reinecke, “Problem solving using Artifical Intelligence Techniques”
2. https://en.wikipedia.org/wiki/Knowledge_representation_and_reasoning

References

3. A.R. Greef and R. Reinecke, “Problem solving using Artifical Intelligence Techniques”
4. A.Barr, E.A. Feigenbaum, “The Handbook of Artificial Intelligence”, 1981.
5. M.Boden, “Artifical Intelligence and Natural Man”, USA, 1977.
6. P.H. Winston, “Artificial Intelligence”, USA, 1984.
7. James Le, “A Gentle Introduction to Neural Networks for Machine Learning”,2018.
8. Jonathan Shapiro, “Genetic Algorithms in Machine Learning”, 2001.
9. https://en.wikipedia.org/wiki/Heuristic_(computer_science)
10. https://www.geeksforgeeks.org/search-algorithms-in-ai/
11. https://en.wikipedia.org/wiki/Knowledge-based_systems
12. https://en.wikipedia.org/wiki/Knowledge_representation_and_reasoning
13. https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052