Logo, icon

Description automatically generated**MINISTRY OF EDUCATION AND TRAINING  
 HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY AND EDUCATION  
FACULTY FOR HIGH QUALITY TRAINING**--------------- ---------------

**ARTIFICIAL INTELLIGENCE**

**TOPIC:**

**CLASS ID: ARIN330585E\_22\_1\_01CLC**

**GROUP: 3**

**INSTRUCTOR: TRAN NHAT QUANG**

**STUDENTS:**

|  |  |
| --- | --- |
| Name | ID |
| Nguyen Huynh Thanh Toan | 20110420 |
| Nguyen Hoang Nhat Nam | khung doi20110383 |
| Bui Ngoc Anh | 20110354 |
| Vu Trong Nhan | 20110390 |

Hồ Chí Minh,…../ ……/ 2022

**GRADE**

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | CONTENT | PERFORMANCE | TOTAL |
| GRADE |  |  |  |

**COMMENT**

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

............................................................................................................................

**Lecturer**

**Tran Nhat Quang**

**Contents**

[**I. Plagiarism 5**](#_Toc121567014)

[**II. Introduce Artificial Intelligence 6**](#_Toc121567018)

[**1. What is Artificial Intelligence? 6**](#_Toc121567019)

[**2. History of AI 6**](#_Toc121567020)

[**3. Applications of Artificial Intelligence 7**](#_Toc121567021)

[**III. Ethics And Law In Data And Analytics 8**](#_Toc121567022)

[**1. Ethics 8**](#_Toc121567023)

[**2. Virtua ethics: 8**](#_Toc121567024)

[**3. Deontology: 8**](#_Toc121567025)

[**4. Consequentialism: 9**](#_Toc121567026)

[**5. Beneficial and questionable of Big data 9**](#_Toc121567027)

[**6. Facial recognition technology 9**](#_Toc121567028)

[**7. Data analytics and law 9**](#_Toc121567029)

[**8. Explainable AI (XAI) 10**](#_Toc121567030)

[**IV. Search Algorithms 11**](#_Toc121567031)

[**1. Basic Steps Of A Search Algorithm 11**](#_Toc121567032)

[**2. Algorithm Evaluation Criteria 11**](#_Toc121567033)

[**3. Conceptual algorithm 12**](#_Toc121567034)

[**3.1 Conceptual algorithm #1 Tree search 12**](#_Toc121567035)

[**3.2 Conceptual algorithm #2 Graph search (search with memory) 13**](#_Toc121567036)

[**3.3 Two types of search algorithms 14**](#_Toc121567037)

[**4. Data structures 14**](#_Toc121567038)

[**5. Uninformed Search Algorithms 15**](#_Toc121567039)

[**5.1. Breadth-First-Search (BFS) 15**](#_Toc121567040)

[**6. Uniform-Cost Search Algorithm 16**](#_Toc121567041)

[**6.1. Depth-First Search Algorithm (DFS). 17**](#_Toc121567042)

[**6.2. Depth-Limited Search Algorithm 17**](#_Toc121567043)

[**6.3. Iterative deepning search Algorithm 18**](#_Toc121567044)

[**7. Informed Search Algorithms 19**](#_Toc121567045)

[**7.1. Distance estiminate function 19**](#_Toc121567046)

[**7.2 General Idea of Informed Search 20**](#_Toc121567047)

[**7.3 Best First Search Algorithm (Greedy search) 20**](#_Toc121567051)

[**7.4 A\* search Algorithm 20**](#_Toc121567052)

[**7.5 How to create heuristic functions 21**](#_Toc121567053)

[**V. Local Search – Searching For Goal State 21**](#_Toc121567054)

[**1. Compare Algorithms 21**](#_Toc121567055)

[**2. State space landscape 22**](#_Toc121567056)

[**3. Local Search 23**](#_Toc121567057)

[**4. Hill-climbing search: 24**](#_Toc121567058)

[**4.1 The idea: 24**](#_Toc121567059)

[**4.2 Algorithm description: 24**](#_Toc121567060)

[**4.3. Completeness: 25**](#_Toc121567061)

[**4.4. Optimal: 25**](#_Toc121567062)

[**4.5. Cost: 26**](#_Toc121567063)

[**5. Issues of Hill-climbing search: 26**](#_Toc121567064)

[**5.1. Too many successors: 26**](#_Toc121567065)

[**5.2. Local optimum: 27**](#_Toc121567066)

[**6. Simulated annealing 28**](#_Toc121567067)

[**7. Searching in NONDETERMINISTIC environments 29**](#_Toc121567068)

[**7.1 Search in unknown environments 29**](#_Toc121567069)

[**7.2 And-Or-Graph Search 30**](#_Toc121567070)

[**8. Searching in Partially Observable Enviroments 33**](#_Toc121567071)

[**8.1. No observable 33**](#_Toc121567072)

[**8.2. Partially observable 34**](#_Toc121567073)

[**9. Online Search 35**](#_Toc121567074)

[**9.1 Comparison of Offline Search and Online Search: 35**](#_Toc121567075)

[**9.2 Competitive ratio: 36**](#_Toc121567076)

[**9.3 Online Depth-first search agent: 36**](#_Toc121567077)

[**9.4 Online A\* Search: Learning Real – Time A\* agent 38**](#_Toc121567078)

[**10. Constraint Satisfaction Problems 40**](#_Toc121567079)

[**10.1 Constraint satisfaction problems (CSP) 40**](#_Toc121567080)

[**10.2 Constraint graph: 41**](#_Toc121567081)

[**10.3 Constraint propagation: 42**](#_Toc121567082)

[**10.4 Local consistency (ý tưởng chính của Constraint propagation): 43**](#_Toc121567083)

[**10.5 AC–3 algorithm (arc consistency) 44**](#_Toc121567084)

[**10.6. Backtracking search algorithm 46**](#_Toc121567085)

**CONTENT**

# Plagiarism

**What is plagiarism?**

Plagiarism is the act of arbitrarily copying or taking the ideas, thoughts, images or expressions of another person and treating it as one's own work. Besides, it is a serious ethical violation and sometimes can violate copyright and it is also a taboo in study and work.

Manifestations of plagiarism:

+ Copy ideas, and source a lot of other people's content (eg copying more than 30%) into your own.

Full use of other people's ideas and styles without crediting or citing sources but using more than 80%

+ Use other people's ideas and intentionally modify them to create your own products

**Things not to do**

- Arbitrarily using, stealing, copying other people's ideas as if they were their own products

- Using other people's ideas, products, or articles without clearly stating the source.

- Although the source is clearly and fully stated, it is not recommended to copy more than 80%, but should learn and write the article in the way you find out.

**Things to do**

# - Understand what is the concept of plagiarism.

# - When referencing documents should cite the source clearly and completely and should only refer to less than 30%, so based on that, develop your own ideas.

# Commit:

We hereby declare that this project is done by team members. We do not copy or use any other people's materials or source code without specifying the source. If we commit plagiarism errors (such as: using other people's documents, code without specifying the source, copying over 30% of the report), we will take full responsibility.

|  |  |
| --- | --- |
| **Full Name** | **Student ID** |
| Bùi Ngọc Ánh | 20110354 |
| Nguyễn Hoàng Nhật Nam | 20110383 |
| Nguyễn Huỳnh Thanh Toàn | 20110420 |
| Vũ Trọng Nhân | 20110390 |

# Introduce Artificial Intelligence

## What is Artificial Intelligence?

• Intelligence is understood as the ability to think, reason, perceive and learn

• Man-made is anything created by man, which is not found in nature

• Understandably, Artificial Intelligence is a technology that simulates human perception, thinking, reasoning and learning processes for machines, especially computer systems.

1. History of AI

- In 1956 (AI field) at a conference held at Dartmouth College by a group of scientists about 10 people. At the conference they talked about the development vision of AI. Since then, AI has been treated as an independent industry.

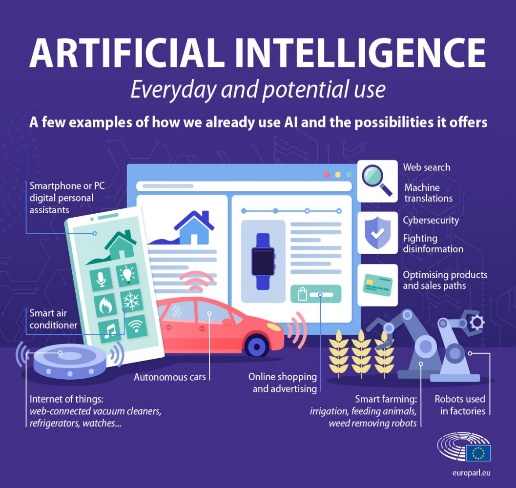
- In the conference statement they hoped to develop machines that could use language and manipulate abstract concepts, solve problems in 1956 that only humans could do, and even be able to do so. do better than humans.

- During that time, the AI ​​industry had to go through many ups and downs, focusing on or being abandoned. Until 1980 (AI industry), due to the development of a number of research works (such as Artificial Neural Networks, Hidden Network Model, Bayesian Networks)

- In 1995 with the explosion of the internet, people's demand for intelligent agents was increasing day by day, which is an evolving form of AI (Chatbot, search bot, game bot). As people's digital data warehouse gets bigger and bigger, we gradually realize the dream of creating very intelligent AIs (Human-level AI), which can then surpass humans to reach the next level. General AI (a true intelligence)

## Applications of Artificial Intelligence

Artificial intelligence is applied in many practical fields:



# Ethics And Law In Data And Analytics

## Ethics

Ethics is a large field of study. There are 3 main schools.

## Virtua ethics:

The school of virtue. Virtue is the most important thing of man, the source of other virtues. If you are a person, you must cultivate virtue first.

## Deontology:

The school follows the rules. For this school, people are considered good if they act according to the laws, and respect those laws.

## Consequentialism:

School of results. Actions that are considered good or bad will depend on the results they bring. If it is good, the action is right, if not, it will be eliminated.

## Beneficial and questionable of Big data

For example:

## Facial recognition technology

Beneficial: Use to detect people who can break into your home, phone,...

Questionable : Used to recognize faces in videos to help police find criminals

## Data analytics and law

In the US: Privacy laws

+ HIPAA 1996: The HIPAA privacy rule establishes national standards for protecting individuals' medical records and personal health information. It then applies to health plans and healthcare providers that conduct some healthcare transactions electronically.

+ COPPA 1998: A United States federal law designed to limit the collection and use of personal information about children by Internet and website operators.

+ FACTA 2003: Law designed to promote consumer protection. Facta is mainly known for its anti-theft provisions

In Europe:

+ There is a large set of laws called the General Data Protection Regulation (GDPR), with some key rules such as: mandating the analysis to show fairness (fairness in terms of ethnicity, religion, gender); Use of data requires the consent of the data owner;

+ Persons who own or relate to data must have a right of access to their personal data; Be responsible for your data use

## Explainable AI (XAI)

Four approaches to AI

Human performance approach

+ Thinking humanly (thinking like a human): concerned about the thought process. There are three ways in which AI learns how humans think, and how to mimic and program machines to do it: observing their own thought processes, performing conduct experiments in psychology, using technology that measures brain waves.

+ Acting humanly: care about the output and behavior of that AI. It is one of the early ones and has a famous test called Taring Test. The Taring Test is about how to tell if an AI is smart or not. The way to do it is to give a question and the machine answers in human language. If the examiner reads the machine's answer and asks that question with a real person and compares the two answers, if he can't distinguish which is the machine's answer, the conclusion that the machine has intelligence.

Ideal Performance Approach

+ Thinking rationally (thinking in rational ways): Making logical conclusions (using logic)

+ Acting rationally (Doing things better than humans): is the best form, a combination of Human performance and Thinking rationally. It creates Agents that when it does something, it gets the best results

# Search Algorithms

## Basic Steps Of A Search Algorithm

• Step 1: Depending on the search strategy, choose a node to expand

• Step 2: Expanding the selected node

Repeat the above two steps until a solution is found or there are no more nodes to expand . All algorithms include the above two steps, the only difference is the node selection in step 1

## Algorithm Evaluation Criteria

Before we get into the design of specific search algorithms, we need to consider the criteria that can be used to select an algorithm out of many different algorithms. We can evaluate the performance of the algorithm by four criteria:

• Completeness: Does the algorithm guarantee to find a solution (solution) if any for the problem posed or not?

• Optimality: The solution found is the most optimal solution or not (optimal solution)

• Time complexity: How long does the algorithm take to find the solution?

• Space complexity: How much memory is needed during the search?

Time complexity and space complexity are always considered for several measures of different problems. In theoretical computer science, the typical measure is the size of the state space graph, | V | + | E |, where V is the set of vertices (vertices - nodes) of the graph and E is the set of edges (edges, links). This makes perfect sense when the input is a graph with a clear data structure (Example: a road map between provinces/cities)

In Artificial Intelligence, graphs are usually implicitly represented by initial state, action, and transition model. Therefore, complexity is expressed in terms of three quantities:

• b, the branching coefficient or the maximum number of successors (child nodes) of any node

• d, depth (number of steps along the path from the origin) of the shallowest goal state

• m, the maximum length of any path in state space

Time is usually measured in terms of the number of nodes created during the search, and space in terms of the maximum number of nodes stored in memory.

# **Conceptual algorithm**

### **Conceptual algorithm #1 Tree search**

- As a general model for building other search algorithms

Text

Description automatically generated

**frontier:** set of all leaf frontier nodes

**Initialize frontier:** 1 node then start the loop

**Loop:**

+ Select 1 leaf node from frontier

+ Repeat until the frontier is empty, no results are found, or if the solution is run, it will return to exit the loop

Problem: re-expanding previously expanded nodes leads to an endless loop, use graph search

## Conceptual algorithm #2 Graph search (search with memory)

Similar to tree search, but adding an explored set containing expanding nodes, avoiding endless loops

Text, letter

Description automatically generated

- First add to the explored set to contain the passed points to avoid repeating that point

- Each time a leaf node is selected to expand, it will be added to the explored set

- After expanding the node, there will be resulting nodes (child nodes, not in the frontier or explored set) added to the frontier.

- Tree search algorithms are divided into two types, Uninformed Search and Informed Search.

### **Two types of search algorithms**

Uninformed search (blind search)

* Only know information about the problem to be solved.
* Randomly select nodes to expand.
* Example: In a game of chess, only knowing the starting position, not knowing who the opponent is, the opponent's strengths and weaknesses.

Informed search (informed search) based on available information, guessing f(n) to choose the node with the smallest f(n) value to expand.

* In addition to information about the problem to be solved, there is some other information that helps in solving the problem.
* Based on the available information, guess f(n) and choose the node with the smallest f(n) value to expand.
* For example: In addition to the above issues, there is also information on how to find out how the car goes, how the queen goes.

## Data structures

**Components of node:**

For each node node n :

* n.STATE: state (position, state at that time)
* n.PARENT: parent node creates node
* n.ACTION: n.PARENT executes n.ACTION
* n.PATH-COST: from init state to goal state (plus all costs)

**Queue**

* Queue operator
* Empty? (queue)
* Pop (queue)
* nsert (element, queue)

Types of queues

* FIFO Queue
* LIFO Queue (stack)
* Priority queues

## Uninformed Search Algorithms

## Breadth-First-Search (BFS)

**Idea**

The algorithm uses a list to contain vertices waiting to be considered.

At each step, root node (Initial-state) is expanded first and exclude to consider the next vertices (successors) in the list then continue to expand the successors of the successors until a solution is found.

The condition for the algorithm to terminate is that the list must be empty.

Input: Problem

Output: Solution

Graphical user interface, application

Description automatically generated with medium confidence

**Algorithm Description**

Text

Description automatically generated

Breadth-first search implemented using FIFO queue data structure.

**Completeness:** Yes. Which means if the shallowest goal node is at some finnite depth.

**Optimality:** the path cost is a non-decreasing fucnton of the depth of node so can find solution

**Time complexity:** Cost time,

**Space complexity:** memory consuming, memory size of frontier is

## Uniform-Cost Search Algorithm

**Idea**

This algorithm is used to traverse a weighted tree or graph. Based on the breadth-first search algorithm, the node with the smallest path cost is chosen first.

**Algorithm Description**

**Completeness:** Yes. If there is a solution, UCS will resolve it.

**Optimal:** Yes. Because it has only one path with the lowest path cost.

**Time Complexity:** the worst case .

**Space Complexity:** the worst case .

With C\* is path cost of optimal solution và the minimum step cost is

## Depth-First Search Algorithm (DFS).

**Idea**

Expand the nodes with the greatest depth in the frontier before find solution in next path.

Input***:*** problem

Output: solution

**Algorithm Description**

Depth-first search implemented using LIFO queue data structure (stack).

**Completeness:** No

**Optimality:** No. Because it can generate more steps or high path cost to find goal node.

**Time complexity:** Depend on the node traversed by the algorithm , better than BFS if m < d.

**Space complexity:**, less memory consuming than BFS.

## Depth-Limited Search Algorithm

**Idea**

Similar to DFS but the depth limit, if selected node has same level with limit, will rollback and choose another node to expand.

Input: Problem

Output: Solution

**Algorithm Description**

Text

Description automatically generated

**Completeness:** No, but can complete when above the depth limit

**Optimality:** No even if l > d

**Time complexity:** , better than DFS

**Space complexity:** , less memory consuming than DFS

Với l m là depth-limited

## Iterative deepning search Algorithm

**Idea**

It combines DLS and BFS algorithm. This search algorithm finds the answer to the best depth limit and does it by moderately increasing the limit until a goal is resolved

**Input:** problem

**Output:** solution

**Alogorithm Description**

Text

Description automatically generated

**Completeness:** Yes if the branching factor is finite.

**Optimality:** Yes if if path cost is a non- decreasing function of the depth of the node.

**Time complexity:**

**Space complexity:**

When the depth-limited increases until it is equal to or greater than m, the time and space complexity is equal to the time and space complexity of DFS.

# Informed Search Algorithms

## Distance estiminate function

**Evaluation function f(n):** is a function that uses information about the environment to estimate the cost of the path from initial state to goal state.

Example: BFS

**Heuristic functions h(n):** is a function that estimates the distance from node n to the goal state.

Example: A\* use heuristic function of each node to find a best estimate cost for next move

## General Idea of Informed Search

* The informed search algorithm contains a bunch of knowledge like how far we are from the target, the path cost, the approach to the target node, etc.
* This knowledge helps agents explore less of the search space and find the target node more efficiently.
* The informed search algorithm is more useful for large search spaces.

## Best First Search Algorithm (Greedy search)

**Evaluation function**: f(n) = h(n).

**Input:** problem

**Output:** solution

**Completeness**: Yes

**Optimality**: Due to the use of f(n) = h(n), the Best-first search skipped the path from the initial state to node n (path cost), leading to the selection of node expand not being optimal, not finding the optimal solution.

## A\* search Algorithm

**Evalution function**: f(n) = h(n) + path cost

**Input:** problem

**Output:** solution

**Completeness:** Yes

**Optimality:**  Yes

Heuristic functions is consistent heuristic, h(n) <= cost(n, n’) + h(n’) with n’ is succesor of node n

If h(n) = 0 -> f(n) = path cost -> A\* = Uniform-cost search (optimal)

**Time complexity:** Time complexity depends on the accuracy of the heuristic function, if the value of the heuristic function is close to the actual path, the faster the solution will be found.. The worst case , h(n) = 0, time complexity of A\* search equal to Uniform-cost search is

***Space complexity:***

+ Based on Uniform-cost search, but Uniform-cost search is based on Breadth-first search, so it can still overflow.

+ To solve this problem, the Simplified Memory Bounded A\* (SMA) algorithm can be used. This algorithm removes nodes whose f(n) is too large.

## How to create heuristic functions

• Method 1: Mitigate the problem by removing restrictions for each action

• Method 2: Use pattern databases

o Select subproblem (subproblem), keep the same restrictions on each action

o Organize pattern databases by storing solution costs for each instance of the subproblem.

Choose the largest heuristic function among the newly created heuristic functions to get a consistent heuristic

# Local Search – Searching For Goal State

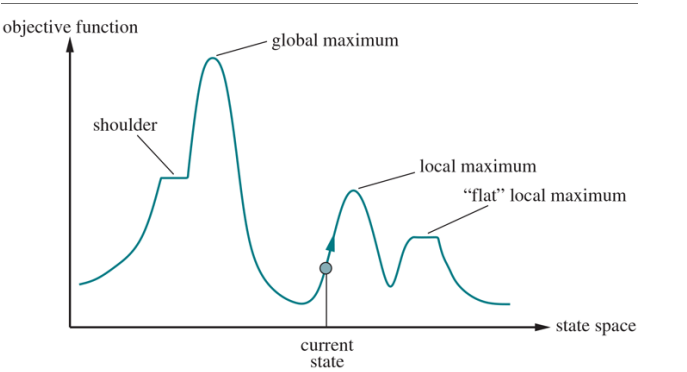
## Compare Algorithms

- Learned search algorithms (Uninformed (blind) or Informed) survey the search space systematically according to certain rules:  
+ Need to save information about your status and your way out.  
+ it's not appropriate for the math to have a large state of space.  
- It's different from search algorithms before, looking for locator at a point.  
Considering the current state and the current state of vicinity:  
+ Do not save information about your status and your survey route.  
+ Save time and memory  
+ can apply to all the major space-space cards.  
+ Don't give me the best answer.  
- It's different than the usual search for the local field is important to the destination state.  
- Best conditions, no matter the road  
+ every status corresponds with a positive explanation.

## State space landscape

Each point (state) in the landscape has an "elevation", which is determined by the value of the target function.

* Global maximum - the state of maximizing the target function across the entire landscape.
* Local peak - the state of maximizing the target function in a small area around it.
* Highland - a state such that the target function remains constant in an area around it.
* Shoulder - a plateau with a steep edge.
* Flat - plateaus with downhill edges.
* Ridge - sequence of local maximums



## Local Search

**Idea:**

The local search algorithm starts from a candidate solution and then iterates to a neighboring solution; A neighborhood is a collection of all potential solutions that differ from the current one to the minimum possible. This requires a neighborly relation to be defined on the search space.

* They mainly apply to problems where we do not need to know the path to the solution, but only know the solution itself.
* They work by using a state or a handful of states and exploring the neighboring states of that state. They usually don't store paths. 🡺 save memory
* A specific case is the optimization problems where we look for the best solution according to a target function. They can often find plausible solutions in **large or infinite** state spaces for which system algorithms are not suitable.
* The distribution of the values of the target function in the state space is called landscape.

## Hill-climbing search:

## The idea:

- The term "hill-climbing" comes from the rectification mechanism: at each step of the search, we will choose a transition that improves the value of the objective function.

- Method: From the current state, consider the set of neighbors, move to the better state (best neighbor)

- It is a continuous loop moving in the direction of increasing value and it ends when it reaches the top

- State destination: Algorithm stops when there is no better neighbor

- Algorithm can find local extreme or extreme

**The relevant definition:**

* Objective function: function about the purpose you want. I want to reduce the code, or reward
* Neighbor: is successor
* Global maximum: It's the most valuable thing I've ever wanted.
* Local maximum: It is the largest value in a small area

## Algorithm description:

- Algorithm to receive input state, target state, state transition function, ...

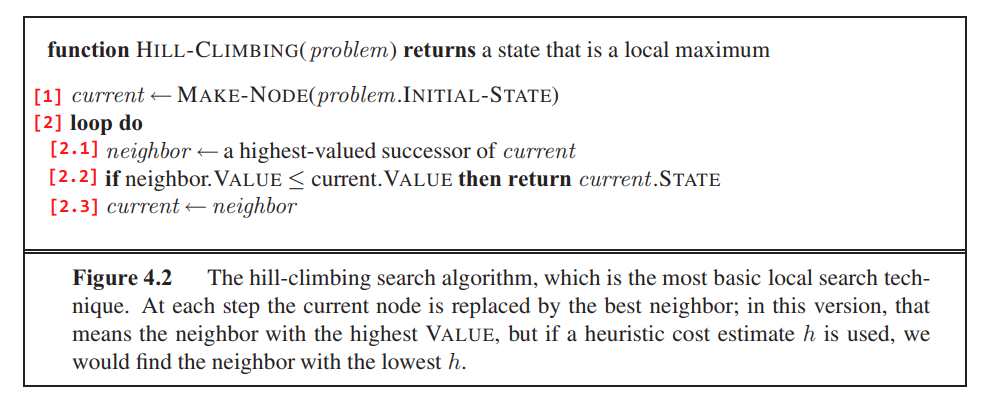
+ Step 1 Initialize a node with INITIAL-STATE to put in the state whose current node is current

+ Step 2 Make an endless loop, until the local maximum is found, then it's over

+ Choose the neighbor as the successor that has the highest-value (which is a better child state than the current state) of all the successors of the current state. (neighbor value will be returned to Objective function)

+ Check neighbor's value against current state

- If neightbor is equal to or less than current, the current state is returned, we can't move any other point "higher" (this time has reached the local maximum).

- If that fails, the neighbor becomes current, we continue to find the neighbor with the highest value around, and then continue the loop****

## 4.3. Completeness:

- Incomplete because the hill-climbing algorithm never makes "downhill" move to state is a lower value or higher costs, guaranteed not to be set up, because it could be stuck at the most impotent at maximum of the most extreme.

## 4.4. Optimal:

- Not optimal because it will easily fall into the local optimal region, that's when we climb to a point where we can't find a better neighbor but this vertex is not the only one (not the global maximum)

Diagram

Description automatically generated

## 4.5. Cost:

- Space cost: only takes memory to store the value of the current state, no need to save all the traversed values like the search algorithm

# 5. Issues of Hill-climbing search:

## 5.1. Too many successors:

- When there are too many successors, it will take a long time to choose the neighbor with the highest value and it is difficult to find the optimal solution for the algorithm.

* The solution is to use Stochastic hill-climbing to randomly generate succesors until a better successor is found.

## 5.2. Local optimum:

Graphical user interface, text, application, email

Description automatically generated

Looking at the figure we can see that on the if line from the current state, there is no way to select a neighbor with a higher VALUE value than the current value – we cannot climb any higher (now reached local maximum).

Diagram

Description automatically generated

But the local maximum here can be shouder, "flat" it can't find a higher neighbor, so it doesn't continue climbing it returns and ends the process.

* From that we can see that this algorithm is not really optimal (no local optimum).
* To solve the local optimum: we use random-restart hill-climbing. That is, in the case of falling into the local optimum, we will run the hill-climbing algorithm again with the INITIAL\_STATE value being a randomly selected state (different from the Initial State in the initial case) it will not care about any the starting state

## Simulated annealing

**Idea:**

At one point, we are in a state A0, the next we want to find a state A1 so that state A1 is best for us (for example, when we play chess, we are in the nth state, in the next turn, at state n +1 we want to choose the optimal move to move, For humans it may be simple, but it is not easy for a computer to find it).

**Graphical user interface, text, application

Description automatically generated**

**Explain:**

**[1]** Initializing the *current* node is the root node of the problem, based on the initialization state corresponding to that problem (problem. INITIAL-STATE).

**[2]** Perform an infinite loop with an initialization value of t of 1:

**[2.1]** Calculate temperature value T = schedule(t)

**[2.2]** If T=0, it returns the current state of *current*.

**[2.3]**  Conversely, choosing the *next* value at random, is a successor of *the current.*

**[2.4]**  Calculate the difference .

**[2.5]**  Check to see if the loop continues to follow *next* . Continue only if (the next state selected carries a better VALUE than the current state), otherwise, select only with probability

**Cost:**

Space: only store the current node and its neighbors, not all search trees as the previous algorithms (Use only one current node and its neighbors).

🡺 Save memory

## Searching in NONDETERMINISTIC environments

### **Search in unknown environments**

* An unknown algorithm is an algorithm that can exhibit different behaviors on different runs, as opposed to a deterministic algorithm that will always produce the same output for a given input that passes through the same outputs. The concurrency algorithm can work differently on different runs due to a status race.
* Results (State, action) 🡪 States{s1, s2, s3,…}

Example:

* The robot performs a leftward move (The current position is B moving to A)
* Self-driving cars: the action of self-driving cars is not unique, it changes over time.
* Results (B, Left) à States {s1 = B, s2 = A}Diagram

  Description automatically generated

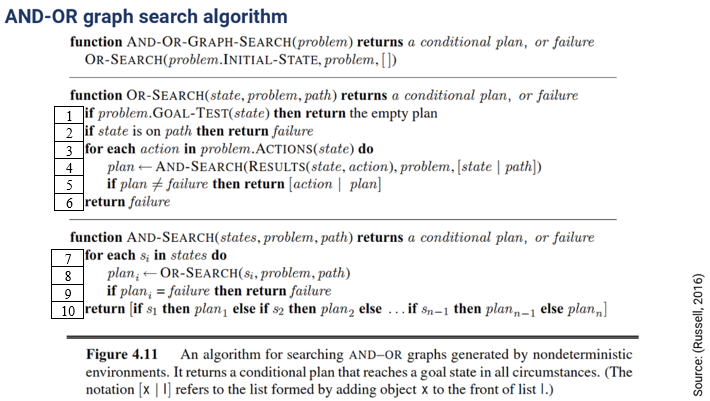
### **And-Or-Graph Search**

**Idea**

* From an existing state it is possible to perform one of the actions (Or search)
* Performing an action will give one or more a new state (and search) all new states
* The solution is a subtree with all leaf nodes being the state to be achieved (at the and node, it must include all cases)

**Inputs, outputs**

* The Input is the problem.
* The output is a solution or can return Failure when a solution is not found.

**Algorithm**

**Variable**:

* state: current state (perform or search).
* problem: the problem to be solved.
* path: the array containing the states that have appeared.
* plan: contains the returned result (including the action and the state achieved when performing that action).
* states: contains multiple states (perform and search).

**Functions:**

* ACTIONS (state) returns all actions while in the current state.
* RESULTS (state, action) returns all new states when performing the action in the current state.

**The running cycle of the algorithm:**

**[1]** Recursive escape condition (exit if goal state)

**[2]** Recursive exit condition (exit if already in path)

**[3]** Get all the actions that can be performed in the current state

Perform each action

**[4]** Get the result when performing the action (states 🡨 RESULT (state, action))

Include the current state in the path

Call And search with states generated

**[5]** Check the results returned by and search

If it fails, add the action to the plan.

**[6]** If a failure is returned, then a return failure

**[7]** Take out each state in the states to do

**[8]** Call the Or search function with the state removed and save the returned result to plani

**[9]** Check plani = failure then return failure (the current branch does not arrive at the result)

**[10]** Returns all results of each state

**Optimals:**

The algorithm does not use the search method as DFS (deep search).

**Cost:**

Time: takes a little time, take advantage of the DFS algorithm

Space: takes a lot of memory, because using Graph search must save the states that have appeared to check.

# 8. Searching in Partially Observable Enviroments

- Partially Observable is when the agent does not know or observe part of the environment. Therefore, the agent cannot determine which state it is in

For example: maze games, climbing mountains, ...

• **Advantages:**

- Save money on purchase and installation of sensors (electronic devices that receive data from the real world to convert it into a base signal on the machine)

- Save time for the sensor to receive data information from the environment

• **Drawbacks**

- Agent of this type, the problem it solves very little or maybe the problem it solves is not effective by equipping it with a sensor.

- Belief states: set of physical states (set the physical state of the agent at the position it is standing in)

### **No observable**

- In fact, this problem is called "Sensor less", which means that there is no information about the environment at all.

Solution idea:

- Build belief state

Initial state: all possible states in reality

Possible actions: ACTIONS(belief state b) b = {s1, s2}

actionsp(s1) = {a, b, c}

actionsp(s2) = {a, c, d}

Can choose actions(b) = {a, b, c, d}

or = {a, b}

Transition model: RESULT (result of an action is the set of all possible results)

(Belief state b, action a) = new belief state b’

b = {s1, s2}

b’ = Resultp (s1, a) fit Resultp (s2, a)

Goal test: all states in belief state to be goal state

b = {s1, s2}

if goal-testp(s1) and goal-testp(s2) both return true then goal-test(b) is true

cost: step-cost(b) and path-cost(b)

- After there's a belief state, it can use fully observable algorithms to run.

### **8.2. Partially observable**

- Unlike no observable, the Partially observable with transition model will be gradually narrowed by putting the belief state in the percept, checking if the condition is met, then that state b is sent back in belief, if not, it is removed from belief state.

- There are a few small sensors.

- Consists of 3 components:

+ State prediction

+ Observation

+ State update

# Online Search

## Comparison of Offline Search and Online Search:

The concept of offline search and online search

**+ Offline search:**

- Offline Search are algorithms that run on the computer first, to find the solution and give the agent to run in practice to find the solution. Each agent's action can know the result based on the transition model.

- Offline Search consists of 5 components: initial state, possible actions, transition model, goal test, cost.

- VD: Find all the ways to put the queen on the chessboard in the N-queens problem, the Hanoi tower problem, the wayfinding problem in the case of a map ...

+ Algorithms used: AND-OR search,LRTA\* algorithm, Reservoir sampling,

**+ Online search:**

- Are algorithms that both run in a real environment and learn to find solutions, with problems without transition models. The more you do it, the smarter the online search.

- Place agents in a real environment, both running and collecting information

- Online search consists of 4 components: initial state, possible actions, goal test, cost.

- VD: the problem of finding odds, robot stocks moving in an area without maps, weather forecast models ...

+ Algorithm of use: BFS, DFS, IDS, DLS, A\*, …

### **Competitive ratio:**

- In online search there are 2 types of path costs which are:

* Actual path cost: the actual cost that the agent performs, including steps to explore the environment and find solutions
* Shortest path cost: the shortest cost to find a solution

- Competitive ratio = actual path cost / shortest path cost

### **Online Depth-first search agent:**

1. **Input:** (s') percept indicates the current state of enviroment.
2. **Output**: for each state that the agent thinks he is in, what action should it have, or in other words, return an action for 1 state.
3. **Text

   Description automatically generatedAlgorithm description**

**[1]** Theentry is the current environment(s') state, which pays out with each state what action the agent should have.

**[2]** Inputs (s': the current state that the agent thinks it is in)

**[3]** Persistent (global variable)

Result: returns a table containing (s, a, s'), which means that the agent is standing at state s and performing action a is stated s'

Untried: an array containing states and actions that I haven't tried before.

Unbacktracked: an array containing states of previous states that it has not considered.

**[4]** Check if the (s') is the target state, if true then stop.

**[5]** If (s') is a new sate (not in untried) then we put actions(s') in untried[s'].

**[6]** If s is not null (it's only null at the start), then we go through step 7.

**[6.1]** We put [s, a]s' in the result table (i.e. scoring a line in state s that performs action a obtains state s')

**[6.2]** Add (s) and unbacktracked[s']

**[7]** If the action of (s') is empty (there is no action to perform anymore), then we go through step 10.

**[7.1]** If unbacktracked[s'] is empty (there's no state to go back to) then we stop.

**[7.2]** Conversely, unbacktracked[s'] is not an empty set, then we perform action b to return to the previous state to perform (backtrack)

**[8]** If in [7] the action of (s') is not an empty set, then we take 1 action from untried[s']

**[9]** s' will now be s.

**[10]** return action a.

• The execution time of online search will be longer than offline search because the algorithm will automatically learn and collect information in the real environment

### **Online A\* Search: Learning Real – Time A\* agent**

* **Idea:** At each step, until the agent moves to the best estimated cost, untried actions in a state are always assumed to lead immediately to the goal at the lowest possible cost.
* H(s') = h(s') still calculates the distance from s' to goal, H is different from h because it updates step by step in the real environment, it will be more accurate
* **Input:** (s') percept indicates the current state of enviroment.
* **Output:** for each state that the agent thinks he is in, what action should it have, or in other words, return an action for 1 state.

Text

Description automatically generated

**[1]** The input is the current state(s'), the result returned will be an action.

**[2]** Inputs (s’, a)

**[3]** Persistent (global variable)

Result: returns a table containing (s, a, s'), which means that the agent is standing at state s and performing action a is stated s'

H : a table that estimates the path from the s state to the destination, initialized as empty

**[4]** If (s') is the target state then we stop.

**[5]** If (s') is a new state, not in H, then we initialize

**[6]** If s is not null (only the starting state will s be assigned as null)

**[6.1]** We put [s, a]s' in the result table (i.e. scoring a line in state s that performs action a obtains state s')

**[6.2]** Update H[s] with the min value in the LRTA\*-COST(s, b, result[s, b], H) function (b belongs to Action(s), the state s has many actions, each action has 1 different cost, we choose the action with the lowest cost.)

**[7]** The input of LRTA\*-COST is, a certain state s, performing action a, obtaining the unknown s' state.

**[8]** If s' is not already in H, initialize h(s)

**[9]** Conversely, if it is in H, then we update c(s, a, s') + H[s']

**[10]** Choose an action that will make the next move the lowest cost

**[11]** s' becomes s.

**[12]** Returns action a.

Completeness: the algorithm will find the solution if the problem has a solution

Implementation time is improved because it uses estimated cost

1. Constraint Satisfaction Problems

### **Constraint satisfaction problems (CSP)**

- Atomic state representations: a state is equivalent to an entity that we cannot see and describe in detail.

- Factored representation: A state is a set of variables

- Constraint satisfaction problems: (this is the problem of using factored representations), which finds the assignment to the state so that all values satisfy the constraint.

- An assignment is a set of one or more assignments to a variable

- If some variables are unassigned, then the opposite incomplete is complete

- If a few variables do not satisfy all constraints, then it is called inconsistent and vice versa consistent

- In CSP there are 3 components:

+ X (set of all variables)

+ D (each variable has a domain or a value assigned to it)

+ C (set of constraints that variables must satisfy)

The number of variables and constraints must not be equal

- The solution of CSP is an assignment that satisfies all constraints (consistent) and assigned variables (complete)

Application of CSP: Map coloring, sodoku,...

Example: n-queens

Chart, box and whisker chart

Description automatically generated

X: 4 variables, each for a queen is Q1, Q2, Q3, Q4

D: each queen can be moved in columns, domain is 1,2,3,4

C: non-attacking (Qi, Qj) means that 4 queens can't attack each other

Solution:

Calendar

Description automatically generated

There may be one or more solutions, but they must all be consistent and complete.

### **Constraint graph:**

The constraint graph is illustrated in the Map coloring problem as follows

Map

Description automatically generated

Constraint graph

Chart, radar chart

Description automatically generated

We can understand as:

1. Nodes: is the variable.
2. Links: are 2 variables in any 1 constraint

### **Constraint propagation:**

- Constraints propagation (binding spread):Constraint propagation is the process of communicating the domain reduction of a decision variable to all constraints stated in this variable. The viral constraint is that this process can lead to a reduction of more domains. Narrow the domains of the surrounding variables by using variables with domains of 1 value. (csp solves problems faster than normal search algorithms such as (BFS, DFS, A\*,...), in many cases the search space is too large, the search algorithms cannot find it but the CSP still finds it.)

- Specifically: using constraints to narrow the domain for variables, when a certain variable has been narrowed the domain, it may affect the domain of the 2nd, then 3rd variable (propagation). By the time we narrow the domain for each variable to only 1 value, we have found a solution.

- If we want to solve the CSP problem in the direction of search, let's define the article with the components Initial State (empty assignments), Successor function (Assign values to variables not marked in proportion without conflict), Goal test, Path cost. 🡪 From there, it can be seen that CSP can be solved according to the search algorithm

### **Local consistency (ý tưởng chính của Constraint propagation):**

- The strengthening of Local consistency in each part of the chart causes inconsistent values to be eliminated throughout the chart

- Local consistency: 1 group of variables located close to each other satisfying the constraint, we call local consistency

- How local consistency works - we make variables that are close to each other satisfy all constraints, and then spread that satisfaction to the surrounding variables, to ensure that the variables around them are consistent. (removing the values that are in the domain makes the variable unconsistent)

- Types of local consistency are

+ Node consistency: a variable is considered a node-consistent if all values in its domain satisfy its first-order constraints (First-order constraints involve a single variable Example LA = 'red')

+ Arc consistency: means that the two variables X and Y must have some constraint so that when X is found, it can help to find Y as a binary constraint (Binary constraints involve pairs of variables For example AG ≠ DT). Arc consistency can help reduce the domains of variables. However, sometimes it doesn't work.

+ Path consistency: The set of two variables {Xi, Xj} is the variable that matches the path Xm if, for every assignment {Xi = a, Xj = b} that matches the constraints on {Xi, Xj}, there is an assignment for Xm that satisfies the constraints on {Xi, Xm} and {Xm, Xj}.

+ K-consistency: A CSP is K-consistency if, for any set of k - 1 variables and for any consistent assignment to those variables, a consistency value can always be assigned to any k-th variable.

* ***Some special cases:***

1.node-consistency

2. arc-consistency

3. path-consistency

+ Global constraints

* The whole is a constraint that involves an arbitrary number of variables. Global constraints define the region of nodes to be searched for the optimal path. Nodes outside this area are not searched. Basically, global constraints determine the overall stretching or compression allowed for the matching procedure.
* A good example is the **alldiff** constraints we see in the *Solving a Sodoku puzzle* example, which executes a set of variables with different values. The problem with the problem is that no digit appears twice in a single line, column, or 3x3 box. To solve it, we can approach it in the following ways: The first can use the AC-3 algorithm to directly reduce the domain of variables by constraints; The second way is to find **triplets**, in one unit, find 3 squares that remove these 3 numbers from domains. The approaches mentioned are applicable to any CSP, not just Sudoku.

### **AC–3 algorithm (arc consistency)**

* Input: a binary csp of a problem with components (X, D, C)
* Output: False if the constraint is not satisfied, True if the constraint is satisfied
* Algorithm description:

Text

Description automatically generated

**[3]** The first step is to start the queue with all or some arcs in the bound chart

**[4]** The while ring handles until the queue is empty:

**[4.1]** Take out 1 constraint 2 variables (X i, X j) so that it is forced, then put (X i, Xj) into the Revise function

**[4.2]** If the Revise function returns true (with the domain contraction of Xi) then we pass [4.2.1]

**[4.2.1]** If the domain set of Di = 0 (meaning there is no value) then we return false (the problem has no solution)

**[4.2.2]** Select the variable Xk adjacent to the variable Xi. Add (Xk , Xi) to the queue and especially have to exclude Xj otherwise there will be endless loops.

double-check the rules of a node if its domain is modified

**[5]** The Revise function takes the input as (csp, X i, X j) and returns true if there is a narrowing of the domain of X i, false if the domain of X i is not narrowed

**[6]** Assign a value to the revised variable as false

**[7]** Take out each value x in the domain set D i of Xi . Considering that there is no value y in the domain set D[ j] of Xj , which makes (x, y) satisfy the constraint, then we pass [7.1].

**[7.1]** Remove x from set Di  to enforce consistency

**[7.2]** Assign revised as true

**[8]** Returns revised

**[9]** If the constraints in the queue have been completed, return true.

## 10.6. Backtracking search algorithm

- Backtrackingsearch to solve [constraint satisfaction problems](https://www.sciencedirect.com/topics/computer-science/constraint-satisfaction-problems) and focus on improvements using forward-looking algorithms. The main variants of forward check, forward check, forward look, partial forward look and arc consistency forward look were introduced, and their use for variable and value selection was discussed.

- Idea: A depth first search selects values for one variable at a time and comes back when a variable no longer has a legal value to assign.

Main steps:

+ Step 1: Select a variable that has not been specified before, assign a test to a value

+ Step 2: If the value inconsistent with assignment is specified, try another value, and if the value selected is consistent with the specified assignment, go back to step 1.

* Recursive search used for CSP components
* Input: csp
* Output: solution or failure
* Text

  Description automatically generated Algorithm description

**[1]** The input is a csp consisting of components (X, D, C) and returns the solution (if all variables have been assigned values) or returns failure

**[2]** return BackTrack ({}, csp)

**[3]** The BackTrack function receives the assignment, csp (for the sodoku problem, the initial assignment is the cells that already have values), (if there is nothing, put the empty set into the assignment)

**[4]** If the assignment is complete, which means that all variables are assigned with a single value, then that assignment is also the solution, we return assignment.

**[5]** If the assignment is not completed, we select a variable that has not been assigned a value.

**[6]** Assign the variable that just selected a value from its domain set.

**[6.a]** If the value just assigned to that variable is consistent with the current assignment, then we go through [6.1]

**[6.1]** Give that value to the variable and add it to the assignment.

**[6.2]** Check if the domain set of surrounding variables decreases.

**[6.3]** If inferences do not encounter a variable whose domain set is empty, then through [6.3.1]

**[6.3.1]** Add inferences to the assignment if it has a domain set of only 1 value

**[6.3.2]** Go back to [5], take the other variable, and continue to assign a value to it.

**[6.3.3]** If the result is different, then the return result

**[6.b]** If in [6.a] inferences there is a failure state, we remove {var = value} and inferences from the assignment and through [7].

**[7]** return false and through other branch performed.

- Select next variable and value

+ Random selection: is a method of describing all variables => inefficient.

+ Minimum remaining-value heuristic: the method of selecting the variable with the smallest domain name.

+ Degree heuristic: assign a value to the variable that is associated with the largest number of constraints on other unassigned variables. Residual minimum value (MRV): select the variable with the least possible.

+ Least-constraining-value heuristic: the way to choose the value for which the value is selected leaves the most choices for other variables.

-Inference

+ Fordward checking: check the domain of neighboring variables

+ Maintaining arc consistency (AC-3): check the domain value of all variables

-Backtracking strategies

+ Backtrack according to Backjumping: create a conflict-set set, containing assignments that are likely to conflict with the current variable you are considering.

+ When encountering failure, immediately jump to the conflict-set episode.

# REFERRENCE

<https://taxplus.vn/tri-tue-nhan-tao-ai-la-gi/>

Trần Nhật Quang (2022). Bài giảng môn Trí tuệ nhân tạo

Stuart J. Russell, Peter Norvig (2016). Artificial intelligence a modern approach

Prof Tom Lenaerts

forums on the internet