1. Problem definition contains: search space, >1 initial state, >1 final state, >1 paths, a set of rules
2. Search space = all possible states and the operators that map them
3. **Uninformed search strategies (USS)**
    a. Not based on problem-specific info; blind strategies; brute force methods
    b. Ex. linear search, binary search, BFS, DFS..
4. Bi-directional search - 2 parallel search strategies: root-leaves and leaves-root; they meet in an already established point
5. **Informed search strategies (ISS)**
    a. Based on specific info about the problem; specific to the problem; uses an **heuristic** to order the nodes
    b. Global search strategies: best-first search, greedy, A*
    c. Local search strategies: tabu search, hill climbing, simulated annealing
6. Evaluation function for a node = cost from initial state to there + estimation from there to final state; f(n) = g(n) + h(n)
7. **Local search strategies (LSS)**
    a. **Simple** = a single neighbour state at a time
        i. Hill climbing - choose best neighbour
        ii. Simulated annealing - probabilistically choose the best neighbour
        iii. Tabu search - most visited solutions
        iv. Apparent SA si TS nu se cer, dar le las aici
    b. Beam = more states
        i. Evolutionary algorithms (EA)
        ii. Particle swarm optimisation (PSO)
        iii. Ant colony optimisation (ACO)
8.

Natural evolution		Problem solving
Individual	↔	Possible solution
Population	↔	Set of possible solutions
Chromosome	↔	Coding of a possible solution
Gene	↔	Part of coding
Fitness	↔	Quality
Crossover and Mutation	↔	Search operators
Environment	↔	Problem

9. **Evolutionary algorithm (EA)**

a. Most complex part = fitness evaluation;
b. Easily adaptable to a lot of problems (just change the representation / fitness function)
c. **Representation** - external = phenotype, internal = genotype
    i. Linear
        1. Discrete
            a. Binary = knapsack problem; there is usually an encoder / decoder that allow the user to understand the information
            b. Integers = random (image processing), permutation (TSP)
            c. Class-based = map colouring problem
        2. Continuous - function optimization
    ii. Tree-based - regression problems
d. **Population** = collection of possible solutions; usually a fixed dimension; diverse
    i. Should be uniformly distributed in the search space
    ii. **Generation gap** = proportion of replaced population; we denote the pop. size by μ
    iii. Generational - each generation is completely replaced by its offspring, individuals survive only 1 generation; gg = 1
    iv. Steady-state - a parent is replaced by offspring if the offspring is better; gg = 1 / μ
e. **Fitness function** - represents the adaptation to the environment
f. **Selection** - based on the fitness; helps to escape local optima (bc weaker individuals also have a chance to survive)
    i. Deterministic - the best always wins
    ii. Stochastic - the best has more chances to win
    iii. Proportional selection, ranking selection, tournament selection
    iv. Survival selection
        1. Based on age - eliminate the oldest
        2. Based on fitness - those from above + elitism (keep the best k), genitor (eliminate the worst k)
g. Variation - generate new possible solutions; not based on fitness; must produce valid individuals
    i. Arity = 1 => mutation; else => recombination / crossover
    ii. Mutation
        1. Bit flipping - flip bits, each with a probability; there is a “strong” and a “weak” version

2. Random resetting - change the value with another from the domain, each with a probability
3. Creep - add a positive / negative value, which follows a 0-symmetric distribution
4. Swap - randomly swap 2 genes
5. Insertion - move a gene next to another
6. Inversion - reverse a section of the chromosome
7. Scramble - shuffle a section…
8. K-opt -
9. Uniform (for real representations) - basically random resetting, the new value is uniformly distributed
10. Non-uniform (real repr) - add a positive / negative value, which belongs to a Cauchy, normal distr

iii. Recombination (crossover)

1. The offspring gets sth from both parents
2. N-cutting point crossover: choose n points, “cut” the parents there, put together the parts, alternatively
3. Uniform crossover, order crossover, partially mapped, cycle crossover, edge-based crossover
4. Discrete crossover
5. Arithmetic crossover
    a. Singular
    b. Simple
    c. Complete
6. Geometric crossover
7. Blend crossover, simulated binary crossover

h. **Stop condition** - optimal solution found, or ran out of resources, or we got bored

10. Particle swarm optimization (PSO) - population of particles that searches for the best solution

a. Based on the behaviour of birds and fish
b. Each particle moves in a search space and has a memory (**cognitive behaviour**) (eg. the place where it got the best results); has neighbours
c. **Inertia** - can be constant or descending, forces the particle to move in the same direction, balances the search between global and local exploration
d. ^There’s also **memory, velocity, position**
e. Particles cooperate; they know their neighbours’ fitness

f. **Social behavior** - rely on the knowledge of other particles
g. Stop condition - reaching a predefined number of iterations or after evaluating the fitness function k times (k is established beforehand)

11. **Ant colony optimization (ACO)** - similar to PSO; best solution is now an *optimal path in an oriented graph*
    a. Particle = ant; they walk through the graph and leave **pheromones** behind; the more pheromones an edge has, the more ants go there
    b. Pheromone trail - collective and distributed memory => indirect communication between ants
    c. Best used for the travelling salesman problem (TSP)
    d. **Ant system** (AS) - all the ants deposit pheromones after a solution is complete
    e. **Ant colony system** (ACO) - all the ants deposit phero at each step of the solution; the best ant only deposits phero after the solution is complete
    f. **MaxMin ant system** (MMAS) - the best ant only deposits phero after a solution is complete; deposited phero is limited to a given range

12. Games
    a. Basically searches in the presence of an adversary; main difficulty = compute the consequences of all possible moves
    b. State = board config; initial state = initial game config; final state = config that wins the game
    c. Operators = allowed moves
    d. Utility function = score function, associate a value to each state
    e. **Strategy** = rules that define the free moves of the game for a given state
        i. **Step by step** - linear (symmetry, pairs, parity, DP), tree-based (AOT, minimax); **complete** (pathfinding, planning)
        ii. **Strategy of symmetry** - player B imitates player A based on a symmetry axis; game ends when A cannot move
        iii. **Strategy of pairs** - generalization of ^; moves are grouped in pairs
        iv. Parity strategy
            1. singular (even) and non singular (odd) positions
            2. each odd is transformed by a move into an even
            3. end position == even => win
            4. ex. NIM

v. **Dynamic programming** - decompose into several subgames, solve each subgame, combine the results

vi. **AndOr trees (AOT)**

1. OR nodes = select a move for A, there is a move s.t. A moves into an AND node
2. AND nodes = all the possible moves of the opponent; by any move, B will move into an OR node
3. The game can be won if starting on an OR node; so the root level is always OR
4. Nodes on a level - possible moves of a player
5. For each node, label it T / F, if A has a winning strategy or node from there
6. Leaves are labeled depending on the initial game config
7. Internal nodes: if we are on an OR node, its value = OR (children values); same for AND

vii. **Minimax** - maximise the position of a player, while minimizing the position of an opponent

1. Search tree - alternates levels of profit maximization for self and minimization for the opponent
2. Profits are propagated from leaves -> root
3. Node on min level => value = min of children; max level…; leaves are evaluated on their own / have an initial value
4. Root level = max
5. **α**-**β** pruning
   a. **α** = value of the best selection done over the MAX path, if a node has a value < **α** => the MAX player will avoid it and its subtree;
   b. **β** is the same but for MIN
6.

	MinMax	MinMax α-β
Time complexity	$O(b^m)$	$O(b^{m/2})$ with sorted nodes
Space complexity	$O(b^m)$	$O(2b^{d/2})$ – best case (when for a MAX node the first generated child is that of largest value / a MIN node the first generated child is that of smallest value) $O(b^d)$ – worst case (without pruning)
Completeness	Da	Yes
Optimality	Da	Yes

viii. **Path-finding** - find the optimal path between 2 locations
ix. **Planning** - identify a sequence of actions that transforms the initial state into the final state

f. **Conflict** = several parts act with contrary purposes
g. Strategic form (matrix) - players / strategies on rows / columns; each cell is the profit for that player / strategy
h. Expanded form (tree) - level = player; node = move
i. **Zero sum** game => profit of a player = loss of others; non zero sum => profit != loss of others
j. **Deterministic** game - free moves, selected from the set of possible moves; non-deterministic - random moves (ex. dice)
k. **Perfect information** games - a player knows the consequences of all the moves of everyone; imperfect - …
l.

	Deterministic	Non deterministic (random)
Perfect information	Chess, go	Backgammon, monopoly
Imperfect information	”Vaporașe / avioane / submarine”	Poker, scrabble

13. KBS = knowledge based systems

14. Knowledge base (KB)

a. Knowledge representation = formal logic + rules + semantic nets
b. Formal logic
   i. Syntax = atomic symbols used by the language
   ii. Semantic = associates a meaning to each symbol and a truth value to each rule
   iii. Syntactic inference - rules for generating new expressions / theorems
c. Rules = special heuristics that generate information; interdependence among rules => inference network
d. Semantic nets
   i. **Meronymy** (A is a meronym of B if A is a part of B) - “finger is a meronym of hand, wheel is a meronym of car”
   ii. **Holonymy** (A is holonym of B dacă B is a part of A) - “tree is a holonym of branch”
   iii. **Hyponymy** (A is hyponym of B if A is a kind of B) - “tractor is a hyponym of vehicle”

iv. **Hypernymy** (A is hypernym of B if A is a generalisation of B) - “fruit is a hypernym of orange”
v. **Synonymy** (A is a synonym of B if A and B have the same meaning) - “run is synonym to jog”
vi. **Antonymy** (A is an antonym of B if A and B reflect opposed things) - “dry is an antonym to wet”

15. **Inference engine (IE)** - determine a conclusion using some premises and applying some inference rules
    a. Forward chaining - start from available info, determine a conclusion; **data driven**
    b. Backward chaining - start from a possible conclusion, identify some explanations for it; **goal driven**
16. **Logic based systems (LBS)** - uses method of formal logic
    a. Ex. automatic theorem proving
    b. Architecture - KB + IE
    c. Based on propositional logic / first-order logic / higher-order logic; temporal systems (“x is sometimes true”), modal systems (“x could be true”)
    d.

Regulă de inferenţă	Premisă	Propoziţia derivată
Modus ponens	$A, A \Rightarrow B$	B
Şi introductiv	A, B	$A \wedge B$
Şi eliminativ	$A \wedge B$	A
Negaţie dublă	$\neg\neg A$	A
Rezoluţie unitară	$A \vee B, \neg B$	A
Rezoluţie	$A \vee B, \neg B \vee C$	$A \vee C$

17. Rule based systems (RBS) - try to simulate human reasoning
    a. Architecture - KB + IE + UI (user interface) + knowledge acquisition facility (automatic way to acquire knowledge) + explanation facility (explain the system reasoning to the user)
    b. KB
        i. Facts = correct affirmations
            1. By persistence, they are static (~permanent) or dynamic (~speficic for an instance / run)
            2. By generation - given / derived
        ii. Rules = special heuristics that generate knowledge
            1. Can be exact / uncertain
            2. Cause-effect: if cond then effect1 else effect2

3. There can be +1 clauses and/or +1 consequences
4. Deduction: cause + rule => effect
5. Abduction: effect + rule => cause
6. Induction: cause + effect => rule

c. Conflict resolution methods

i. First applicable - pick the first rule; only works if the rules are ordered and for small systems
ii. Random
iii. Most specific ("longest matching strategy") - pick the one with the most conditions
iv. Least recently used ("recency") - each rule has a timestamp, earlier rules have priority
v. "Best" rule ("prioritization") - each rule is given a weight by a human expert, pick the best one

18. **Bayes systems** = probabilistic KBS
19. Systems based on certainty factors (CF)

a. Facts and rules have associated a certainty (confidence) factor

Bayes	CF
Theory of probabilities is old and has a mathematical foundation	Theory of CFs is new and without mathematical demonstrations
Require statistical information	Do not require statistical data
Certainty propagation exponentially increases	Information is quickly and efficiently passed
Require to *apriori* compute some probabilities	Do not require to *apriori* compute some probabilities
Hypothesis could be independent or not	Hypothesis are independent

20.
21. **Fuzzy systems** - no longer just T / F, we can have partial truths

a. Conjunction becomes minimum (a ^ b = min(a, b)); disjunction - ..maximum (a v b = max(a, b)), negation - ..difference (!a = 1 - a)
b. A value can belong to several classes at once
c. Define inputs + outputs -> construct rules -> evaluate rules -> aggregate results -> defuzzyfication -> interpret the result
d. Characteristic function for a set - no longer just {0, 1}, but [0, 1]; 1 - totally in a set; 0 - not in a set; (0, 1) - part of a set (fuzzy member)

e. Fuzzification - transform input data into a fuzzy set using membership functions (determine how much a value belongs to each membership set)
f. We need a DB of fuzzy rules (“if temperature is low => it’s cold”) => decision matrix
g. Fuzzy inference - rules evaluated in parallel; defuzzification only after all the rules have been evaluated
    i. Evaluation of causes (“degree of fullfilment”)
    ii. Evaluation of consequences
        1. Mamdani - “output variable belongs to a fuzzy set” - basically the classic model that we had at the lab
h. Defuzzification - fuzzy result => raw value
    i. Centroid area, bisector of area, mean of maximum, smallest of maximum, largest of maximum

22. Machine learning

a. There is training data + test data; data is independent (otherwise => collective learning); test / train should have the same distribution (otherwise => transfer learning / inductive transfer)
b. For prediction / regression / classification / planning
c. With supervised / unsupervised / active / reinforcement learning
d. Supervised learning - provide correct output for a new entry
    i. We provide the input and the expected output, we want to find the function that maps input -> output
    ii. Problem types: regression (predicting output for a new input), classification (classifying new input)
    iii. **Quality of learning** = measure of the algorithms performance
    iv. Evaluation methods - disjoint sets for training / testing, cross-validation with multiple subsets, leave-one-out cross validation
    v. Accuracy = correct classified / total
    vi. Precision (P) = correct classified positive / total classified positive
    vii. Rappel (R) = correct classified positive / total positive
    viii. Score (S) = 2PR / (P + R)
e. Unsupervised learning - find a model / structure in a data set; clustering
    i. Find a function that groups the data set into several classes, according to some criteria

ii. Training = find the clusters; testing = add a new elem in one of the clusters to see if it matches
iii. We want min distance, max similarity
iv. Quality of learning
    1. Internal criteria - high similarity inside a cluster, low between different clusters; David-Bouldin, Dunn
    2. External criteria
v. Clustering determination - hierarchical / non-hierarchical / based on data density / grid based
vi. Agglomerating - initially each elem is a cluster, keep merging the closest clusters
vii. Divisive - determine k random cluster centers, assign the elems to the clusters, recompute cluster centers, repeat
    1. k-means

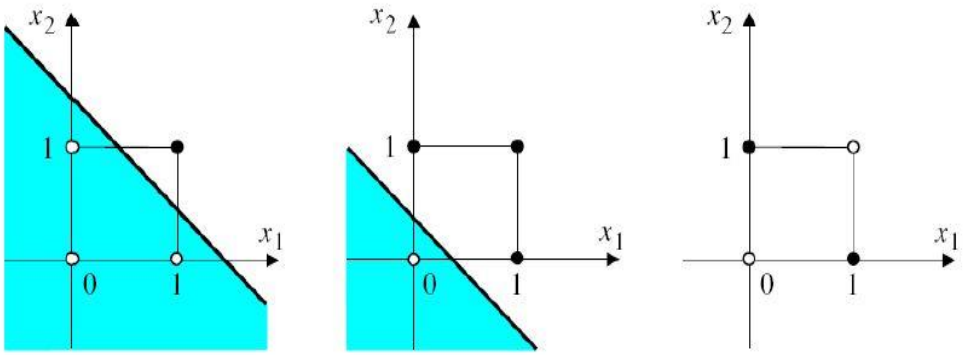
f. Automatic learning
    i. Least squares method - minimise the loss function
    ii. Descending gradients

23. **Support vector machines (SVM)** - linear classifier that identifies the separators between several classes; work well with large data
    a. Solves classification problems
    b. Data might be separable (error = 0) or non-separable

24. **Artificial neural network (ANN)** - binary classification for any input data
    a. A set of nodes (= neurons / units) located in a graph with several layers
    b. Prone to over-fitting and finding just a local optimum
    c. **Neuron** - has inputs / outputs
        i. performs simple computation (througn an **activation function**, which can be **constant, slope, liniar, sigmoid, gaussian**)
        ii. connected by weighted links
        iii. aim is to find the optimal weights s.t. the error is minimised
        iv. Fire process (?) = perform computation + compute the weighted sum of inputs
        v. Weights are usually initialised with a random in [-1, 1]
        vi. Training algorithm: activate neuron -> compute error (difference between actual output and provided output) -> reevaluate weights -> repeat if stop condition is not met
        vii. Training algorithms - perceptron, delta (gradient descent)

viii. A perceptron can learn AND, OR operations, but not XOR, bc it's not linear separable;

ix. Perceptron works with a single input data; delta works with all input data (?)



d. Layers - input (# of data attributes), intermediate (several layers), output (# of outputs)

e.

BNN	ANN
Soma	Node
Dendrite	Input
Axon	Output
Activation	Processing
Synapse	Weighted connection

f. Information is forward propagated, error is backward..

g. Feed-forward ANN - only connections between different layers

h. Recurrent ANN (with feedback) - can have conn between nodes of the same layer, there can be cycles

25. **Deep learning** - subfield of ML, tries to emulate the human brain

a. Deep convolutional neural network (CNN) - used to classify images, object recognition

i. OCR = optical character recognition

b. **Tensor** = generalization of array, matrix (can have any number of dimensions); CNN process images as tensors

c. Convolution operation - operation on the input (x) and kernel (w) functions

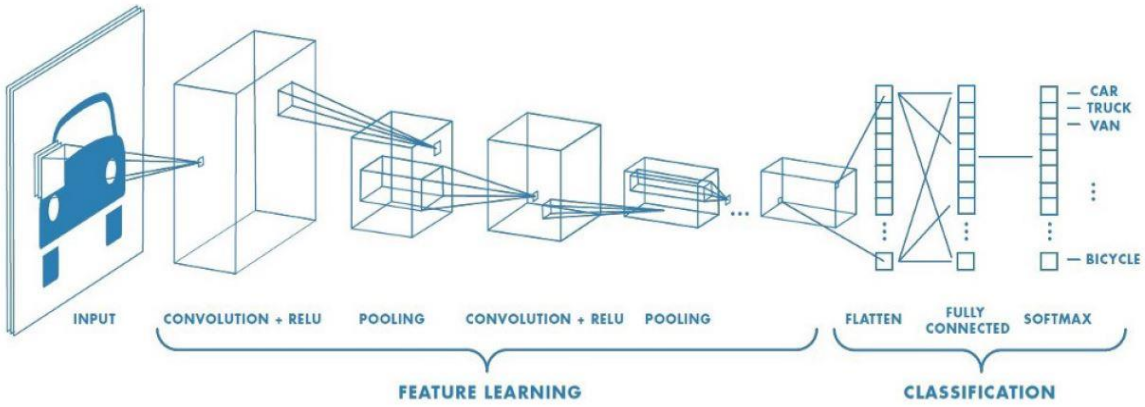
$$s(t) = \int x(a)w(t-a)da$$

$$s(t) = (x \circledast w)(t)$$

d.

e. Aim of the kernel is to extract the high-level features (edges, color etc)

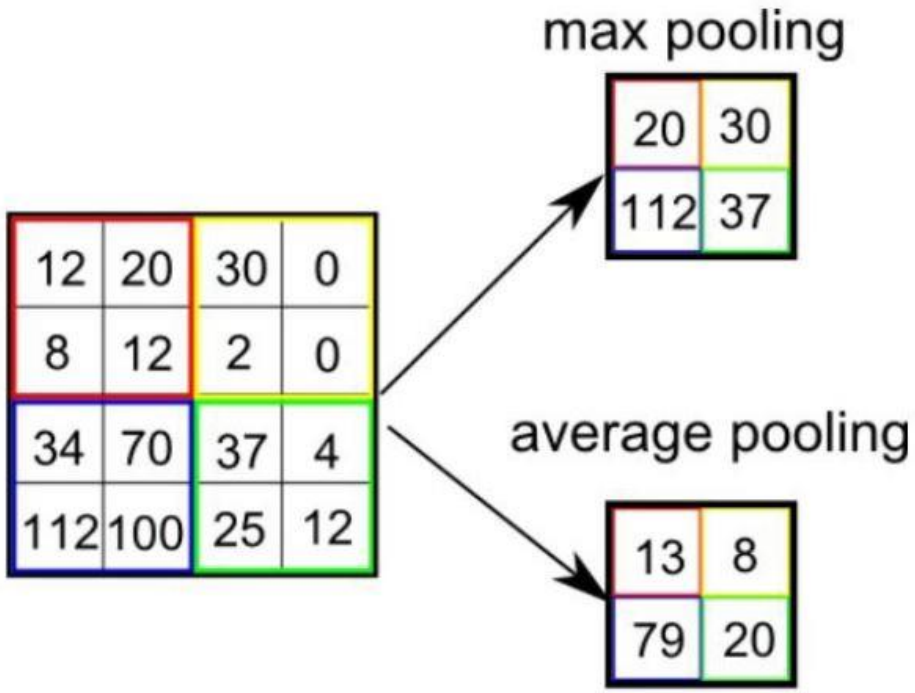
f. CNN layers - several convolutions, nonlinear activation function (**detector stage**), pooling



g.

h. Feature learning - repeatedly apply filters that each detect a line / shape etc; filter values are learned during the network training;

   i. Using gradient descent, the extracted features are those which minimise the loss function



i.

26. Genetic programming (GP)

a. Special case of evolutionary algorithms; each chromosome is a tree that encodes small programs;
   i. Nodes in a chromosome tree are either functions or attributes (on the leaves)
b. Fitness = prediction error
c. Survival selection - can be problematic because of **bloating** (survival of the fattest); possible solution = parsimony pressure (penalty to long programs)
d. Crossover - by cutting point - exchange subtrees
e. Mutation
   i. grow mutation = replace a leaf by a new subtree
   ii. shrink = opposite;
   iii. “Koza” = grow but with any node, not just leaves
   iv. Switch - reorder subtrees of an internal node
   v. Cycle - replace a node with a new node of the same type (function -> function, attr -> attr)
f. Useful for problems with variables that are frequently changing
g. Versions: linear GP, gene expression programming, multi expression programming, grammatical evolution, cartesian genetic programming

27. **Decision trees (DT)** - divide a collection of articles in smaller sets by successively applying some decision rules => more questions

a. Bi-color and oriented tree

b. Decision nodes - possibilities of the decider
c. Hazard nodes - random events (opposite of decision)
d. Result nodes - final states, have utility or a label
e. Internal node -> attribute; branch -> value; leaf -> class
f. Process
    i. Tree construction (induction) - based on training data
    ii. Problem solving - all the decisions from root to leaf = rules; use these rules to classify the test data
    iii. Pruning - find + eliminate branches that reflect noise or exceptions; ∃pre- / post-pruning; this may result in some small errors