What is AI?

Definition (according to Encyclopedia Britannica)

Artificial intelligence (AI) is the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. The term is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalize, or learn from past experience.

Learning

Example:
a simple method by trial and error based on rote learning

- advantages: really easy to implement;
- disadvantages: it may lack generalization;

Generalization

involves applying past experience to *analogous* new situations

Example:

- a program that learns the past tense of regular English verbs by rote will not be able to produce the past tense of a word such as jump unless it previously had been presented with jumped
- a program that is able to generalize can learn the *add ed* rule and so forms the past tense of jump based on experience with similar verbs

Reasoning

Ability to draw **inferences** appropriate to the situation!

- deductive:
  Fred must be in either the museum or the café. He is not in the café; therefore he is in the museum.
  - common in mathematics and logic, where elaborate structures of irrefutable theorems are built up from a small set of basic axioms and rules
- inductive:
  Previous accidents of this sort were caused by instrument failure; therefore this accident was caused by instrument failure.
  - common in science, where data are collected and tentative models are developed to describe and predict future behaviour – until the appearance of anomalous data forces the model to be revised

Problem solving

In AI is a **systematic search** through a range of possible actions in order to reach some predefined goal or solution.

- **special purpose** – tailor-made for a particular problem and often exploits very specific features of the situation in which the problem is embedded
- **general purpose** – is applicable to a wide variety of problems
  Example: means-end analysis: a step-by-step, or incremental, reduction of the difference between the current state and the final goal.
  - for a robot the program selects actions from a list {PICKUP, PUTDOWN, MOVEFORWARD, MOVEBACK, MOVELEFT, MOVERIGHT} until the goal is reached

Examples of problems solved: finding the sequence of moves in a board game, devising mathematical proofs, and manipulating "virtual objects" in a computer-generated world, ...

Perception

The environment is scanned and the scene is decomposed into separate objects in various spatial relationships.

Analysis is complicated because an object may appear different depending on:

- the angle from which it is viewed
- the direction and intensity of illumination in the scene
- how much the object contrasts with the surrounding field

Present: able to identify individuals, autonomous vehicles to drive at moderate speeds on the open road, and robots to roam through buildings collecting empty soda cans.

FREDDY (1966–73) - a stationary robot able to recognize a variety of objects and could be instructed to assemble simple artifacts, such as a toy car, from a random heap of components

Language

A system of signs having meaning by convention!

Not necessary a full language, or s spoken one

- traffic signs
- bird calls

An important characteristic of full - fledged human languages – in contrast to others – is their productivity.

A productive language can formulate an unlimited variety of sentences.

AI methods

a very simple classification:

Two distinct methods:

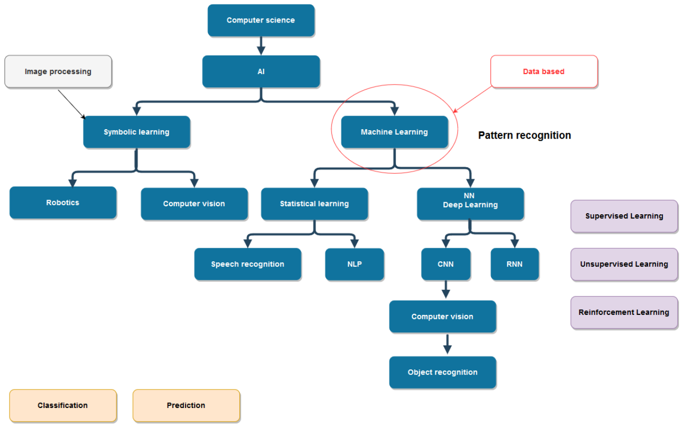
- symbolic ("top-down") approach – seeks to replicate intelligence by analyzing cognition independent of the biological structure of the brain, in terms of the processing of symbols;
- connectionist ("bottom-up") approach – involves creating artificial neural networks in imitation of the brains structure;

Comparison with humans

- humans see $\rightarrow$ computer vision
- humans hear $\rightarrow$ speech recognition
- humans execute tasks, move, ... $\rightarrow$ robotics
- humans recognise patterns $\rightarrow$ Pattern recognition
- humans recognise objects $\rightarrow$ Object recognition
- humans hear and understand language $\rightarrow$ NLP

...

Comparison with humans



Natural computing

algorithms that mimic natural phenomena

- birds (/fish) behaviour → Particles Swarm Optimisation
- life evolution → Evolutionary Computation
- ants behaviour → Ant Colony Optimisation
- annealing → Simulated Annealing

  ...

Solving a problem

Identifying a solution

- in mathematics $\rightarrow$ optimisation process
- in computer science (AI) $\rightarrow$ search process

Optimisation problems

collection of mathematical principles and methods used for solving quantitative problems in many disciplines

- a single numerical quantity, or objective function, that is to be maximized or minimized

  Example: the expected return on a stock portfolio, a company's production costs or profits, the time of arrival of a vehicle at a specified destination, or the vote share of a political candidate

- a collection of variables, which are quantities whose values can be manipulated in order to optimize the objective

  Example: the quantities of stock to be bought or sold, the amounts of various resources to be allocated to different production activities, the route to be followed by a vehicle through a traffic network, or the policies to be advocated by a candidate

- a set of constraints, which are restrictions on the values that the variables can take

  Example: a manufacturing process cannot require more resources than are available, nor can it employ less than zero resources

Search problems

- a single numerical quantity, or objective function
- actions that accomplish the objectives
    - each action changes a state of the problem
- more actions that map the initial state of problem into a final state

problem definition

- search space
  - all possible states
  - representation:
    - explicit - construction of all possible states
    - default - by using some data structures and some functions (operators)
- one or more initial state
- one or more final states
- one or more paths
  - more successive states
- a set of rules (actions)
  - successor functions (operators) - next state after a given one
  - cost functions that evaluate:
    - how a state is mapped into another state
    - an entire path
  - objective functions that check if a state is final or not

solving the problems by search

Aspects to consider:

- Computational complexity (temporal and spatial)
- Completeness $\rightarrow$ the algorithms always ends and finds a solution (if it exists)
- Optimallity $\rightarrow$ the algorithms finds the optimal solution (the optimal cost of the path from the initial state to the final state)

Search strategies

- Search space organises similar with an abstract data type (ADT)
  - ADT list $\rightarrow$ linear structure
  - ADT tree $\rightarrow$ hierarchic structure
  - ADT graph $\rightarrow$ graph-based structure
- aspects to consider:
  - representation
  - domain and operations
- types:
  - informed search strategies (ISS)
  - uninformed search strategies (USS)

Uninformed Search strategies

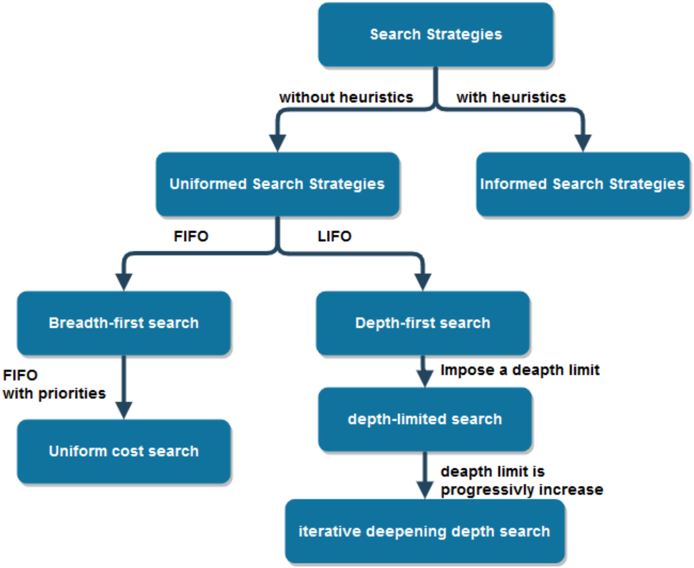
Characteristics:

- are NOT based on problem specific information
- are general
- brute force methods

Topology (based on node exploration):

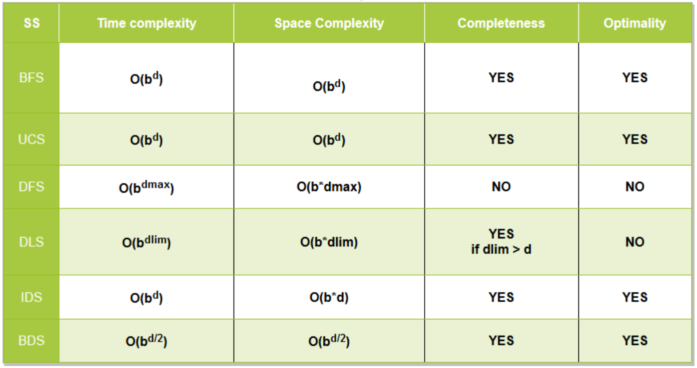
- USS in linear structures
  - linear search
  - binary search
- USS in non-linear structures
  - Breadth-first search
    - Uniform cost search (branch and bound)
  - Depth first search
    - Limited depth first search
    - Iterative deepening depth-first search
  - Bidirectional search

Search strategies



Performances' comparison

for uninformed Search Strategies



SS	Time complexity	Space Complexity	Completeness	Optimality
BFS	$O(b^{d})$	$O(b^{d})$	YES	YES
UCS	$O(b^{d})$	$O(b^{d})$	YES	YES
DFS	$O(b^{dmax})$	O(b*dmax)	NO	NO
DLS	$O(b^{dlim})$	O(b*dlim)	YES if dlim > d	NO
IDS	$O(b^{d})$	O(b*d)	YES	YES
BDS	$O(b^{d/2})$	$O(b^{d/2})$	YES	YES

Informed search strategies (ISS)

classification based on topology

- Global search strategies
  - Best first search
    - Greedy best-first search
    - A^* + versions of A^*
- Local Search strategies
  - Hill Climbing
  - Simulated Annealing
  - Tabu search

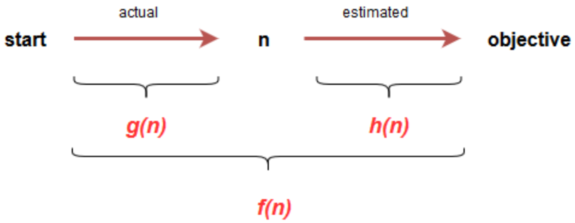
Informed search strategies (ISS)

- Characteristics
  - Based on specific information about the problem (trying to choose 'intelligent' the nodes to be explored)
  - An evaluation (heuristic) function sorts the nodes
  - Specific to the problem

SS in tree-based structures

Basic elements:

- $f(n)$ – evaluation function for estimating the cost of a solution through node (state) n
- $h(n)$ – evaluation function for estimating the cost of a solution path from node (state) n to the final node (state)
- $g(n)$ – evaluation function for estimating the cost of a solution path from the initial node (state) to node (state) n
- $f(n) = g(n) + h(n)$



Heuristic functions

- Etymology: heuriskein (gr)
  - *to find, to discover*
  - *study of methods and rules of discovering and invention*
- Utility
  - Evaluation of the state potential (in the search space)
  - Estimation of paths cost from the current state to the final state
- Characteristics
  - Depends on the problem to be solved
  - New functions for new problems
  - A specific state is evaluated (instead of operators that map a state into another one)
  - Positive functions for each node n
    - $h(n) \geq 0$ *for all states* n
    - $h(n) = 0$ *for final state*
    - $h(n) = \infty$ *for a state that is dead end*

Examples of heuristic functions

- Missionary and cannibal problem
  - * $h(n)$ - no of persons from initial river side
- 8-puzzle
  - * $h(n)$ - no of pieces that are in wrong places
  - * $h(n)$ - sum of Manhattan distance (of each piece relative to the final position)
- Travelling salesman problem
  - * $h(n)$ - nearest neighbour ! ! !
- Pay a sum by using a minimal number of coins
  - * $h(n)$ choose the coin of best (large) value smaller than the sum to be paid

Best First Search (BFS)

Basic Elements

- Best first search = first, the best element is processed
- Each state is evaluated by a function f
- The best evaluated state is explored
- Example of a SS that depends on evaluation function:
  - Uniform cost search (from USS)
    - $f = pathcost$
  - ISSs use heuristic functions
- 2 possible BFS strategies
  - Expand the closest node to the objective state
  - Expand the best evaluated (best cost) node

Best First Search (BFS)

```
function BestFS(elem,list)
    $found = false$
    $visited = \varnothing$
    $toVisit = \{start\}$                ▷ FIFO sorted list (priority queue)
    while $((toVisit! = \varnothing)\&\&(!found))$ do
        if $(toVisit == \varnothing)$ then return $false$
        end if
        $node = pop(toVisit)$
        $visited = visited \cup \{node\}$
        if $(node == elem)$ then $found = true$
        else$aux = \varnothing$
        end if
        for all unvisited children of node do
            $aux = aux \cup \{child\}$
        end for
        $toVisit = toVisit \cup aux$    ▷ adding best ones in the front
    end while
    return $found$
end function
```

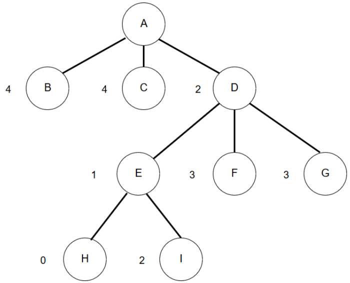
Best First Search (BFS)

- Complexity analyse
  - Time complexity
    - b - ramification factor
    - d - maximal length (depth) of solution
    - $T(n) = 1 + b^2 + ... + b^d \geq O(b^d)$
  - Space complexity
    - $S(n) = T(n)$
  - Completeness
    - No - infinite paths if the heuristic evaluates each node of the path as being the best selection
  - Optimality
    - Possible - depends on heuristic
- Advantages
  - Specific information helps the search
  - Good speed to find the final state
- Disadvantages
  - State evaluation $\rightarrow$ effort (computational, physic, etc)
  - Some 'bad' paths could seem to be good
- Applications Web crawler (automatic indexer); games

Greedy

Evaluation function $f(n) = h(n)$

- cost path estimation from the current state to the final one - $h(n)$
- cost minimization for the path that must be followed



visited	to visit
$\varnothing$	A
A	D, B, C
A, D	E, F, G, B, C
A, D, E	H, I, F, B, C
A, D, E, H	$\varnothing$

Greedy

```
function GREEDY(elem,list)
    $found = false$
    $visited = \varnothing$
    $toVisit = \{start\}$                    ▷ FIFO sorted list (priority queue)
    while $((toVisit! = \varnothing) \&\& (!found))$ do
        if $(toVisit == \varnothing)$ then return $false$
        end if
        $node = pop(toVisit)$
        $visited = visited \cup \{node\}$
        if $(node == elem)$ then $found = true$
        else
            $aux = \varnothing$
        end if
        for first unvisited child of node do
            $aux = aux \cup \{child\}$
        end for
        $toVisit = aux \cup toVisit$             ▷ adding best one in the front
according to $h(n)$
    end while
    return $found$
end function
```

A^*
Basic Elements

- Combination of positive aspects from:
  - Uniform cost search
    - Optimality and completeness
    - sorted queues
  - Greedy Search
    - Speed
    - Sorted based on evaluation
- Evaluation function f(n)
  - Cost estimation of the path that passes though node n
    $f(n) = g(n) + h(n)$
  - $g(n)$ - cost function from the initial state to the current state n
  - $h(n)$ - cost heuristic function from the current state to the final state
- Minimisation of the total cost for a path

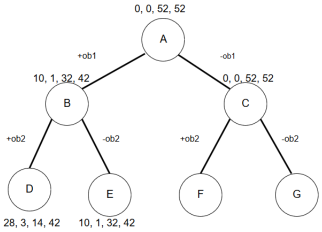
A^*

Example – Knapsack problem

capacity W, n objects $(o_1, o_2, ..., o_n)$ each of then having a profit p_i, $i = 1, 2, ..., n$

	o_1	o_2	o_3	o_4
p_i	10	18	32	14
W_i	1	2	4	3

Solution: for $W = 5 \rightarrow o_1$ and o_3



- $g(n) = \sum p_i$, for selected objects o_i
- $h(n) = \sum p_j$, for not selected objects and $\sum w_j \leq W - \sum w_i$

Fetch node is a tuple (p, w, p^*, f) where:

- p - profit of selected objects (function $g(n)$)
- w - weight of selected objects
- p^* - maximal profit that can be obtained starting from the current state and tacking into account the available space in the knapsack (function $h(n)$)

A^*

```
function BESTFS(elem, list)
    $found = false$
    $visited = \varnothing$
    $toVisit = \{start\}$                ▷ FIFO sorted list (priority queue)
    while $((toVisit! = \varnothing)\&\&(!found))$ do
        if $(toVisit == \varnothing)$ then return false
        end if
        $node = pop(toVisit)$
        $visited = visited \cup \{node\}$
        if $(node == elem)$ then $found = true$
        else
            $aux = \varnothing$
        end if
        for all unvisited children of node do
            $aux = aux \cup \{child\}$
        end for
        $toVisit = toVisit \cup aux$          ▷ adding a node based on its
evaluation $f(n) = g(n) + h(n)$ (best one in the front of list)
    end while
    return $found$
end function
```

A^*

- Complexity analyse
  - Time complexity
    - b - ramification factor
    - d_{max} - maximal length (depth) of an explored tree
    - $T(n) = 1 + b + b^2 + ... + b^{d_{max}} => O(b^{d_{max}})$
  - Space complexity
    - d - length (depth) of solution
    - $T(n) = 1 + b + b^2 + ... + b^d => O(b^d)$
  - Completeness: yes
  - Optimality:yes
- Advantages
  - Expands the fewest nodes of the tree
- Disadvantages
  - Large amount of memory
- Applications:
  - Planning problems
  - Problems of partial sums
  - Puzzles
  - Optimal paths in graphs

A^*

- Versions
  - iterative deepening A* (IDA*)
  - memory-bounded A* (MA*)
  - simplified memory bounded A* (SMA*)
  - recursive best-first search (RBFS)
  - dynamic A* (DA*)
  - real time A*
  - hierarchical A*
- Bibliography
  - $02/A_IDA.pdf$
  - $02/A_IDA_2.pdf$
  - $02/SMA_RTA.pdf$
  - $02/RecursiveBest - FirstSearch.ppt$
  - $02/IDS.pdf$
  - $02/IDA_MA.pdf$
  - *http://en.wikipedia.org/wiki/IDA**
  - *http://en.wikipedia.org/wiki/SMA**

Topology of search strategies

- Solution generation
  - Constructive search: Solution is identified step by step
  - Perturbative search: A possible solution is modified in order to obtain another possible solution
- Search space navigation
  - Systematic search: The entire search space is visited
    - Solution identification (if it exists) $\rightarrow$ complete algorithms
  - Local search
    - Moving from a point of the search space into a neighbour point $\rightarrow$ incomplete algorithm
    - A state can be visited more times
- Certain items of the search
  - Deterministic search: identify exactly the solution
  - Stochastic search: approximate the solution
- Search space exploration
  - Sequential search
  - Parallel search

Local search strategies (LSS)

- Simple local search - a single neighbour state is retained
  - Hill Climbing $\rightarrow$ chooses the best neighbour
  - Simulated Annealing $\rightarrow$ probabilistic-ally chooses the best neighbour
  - Tabu search $\rightarrow$ retains the recent visited solutions
- Beam local search - more states (population) are retained
  - Evolutionary Algorithms
  - Particle swarm optimisation
  - Ant colony optimisation

Hill climbing (HC)

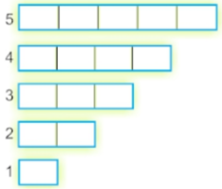
Basic elements

- Climbing a foggy mountain by an amnesiac hiker
- Continuous moving to better values (larger $\rightarrow$ mountain climbing)
- Search advances to improved states until an optimal one is identified
- How a possible solution is accepted
  - Best neighbour of the current solution better than the current solution
- Improvement by:
  - Maximisation of states quality $\rightarrow$ steepest ascent HC
  - Minimisation of states quality $\rightarrow$ gradient descent HC
- HC $\neq$ steepest ascent/gradient descent (SA/GD)
  - HC optimises $f(x)$ with $x \in R^n$ by changing an element of x
  - SA/GD optimises $f(x)$ with $x \in R^n$ by changing all the elements of x

Hill climbing (HC) I

Example - Construct towers from different geometrical shapes

We have n rectangular pieces (of the same width, but different lengths) that are overlapped in a stack. Construct a stable tower from all pieces such that at each move only a piece is moved from the top of the stack (on one of two supplementary stacks).

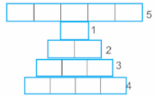


- Solution representation
  - State x - vector of n pairs (i, j), where i is the index of the piece $(i = 1, 2, ..., n)$ and j is the index of the stack $(j = 1, 2, 3)$

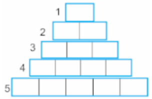
Hill climbing (HC) II

Example - Construct towers from different geometrical shapes

- initial state - vector of initial tower



- final state - vector of the final tower



- State evaluation
  - $f_1 = no.ofcorrectlylocatedpieces \rightarrow$ maximisation
  - $f_2 = no.ofwronglylocatedpieces \rightarrow$ minimisation
  - $f = f_1 f_2 \rightarrow$ maximization
- Neighbourhood
  - Possible moves – Move a piece i from stack j_1 on stack j_2
- How a possible solution is accepted: Best neighbour of the current solution better than the current solution

Hill climbing (HC)
Algorithm

```
function HC(S)
    $x = s_1$
    $x^* = x$
    $k = 0$
    while (not termination criteria) do
        $k = k + 1$
        $N$ = all neighbours of $x$
        $s$ = best solution from $N$
        if ($f(s)$ is better than $f(x)$) then
            $x = s$
        else
            State
        end if
    end while
    $x^* = x$
    return $x^*$
end function
```

Hill climbing (HC)

- Search analyse: convergence to local optima
- Advantages
  - Simple implementation $\rightarrow$ solution approximation (when the real solution is difficult or impossible to find) (TSP with many towns)
  - Does not require memory (does not come back into the previous state)
- Disadvantages
  - Evaluation function is difficult to be approximated
  - If a large number of moves are executed, the algorithm is inefficient
  - If a large number of moves are executed, the algorithm can block
    - in a local optimum
    - On a plateau - evaluation is constant
    - On a peak - a skip of more steps can help the search
- Applications: Cannibal's problem, 8-puzzle, 15-puzzle, TSP, Queens problem, ...

Hill climbing (HC)

Versions

- Stochastic HC
  - The next state is randomly selected
- First-choice HC
  - Randomly generation of successors until a new one is identified
- Random-restart HC $\rightarrow$ beam local search
  - Restart the search from a randomly initial state when the search does not advance

Simulated Annealing

Basic elements

- Inspired by physical process modelling
  - Metropolis et al. 1953, Kirkpatrick et al. 1982
- Successors of the current state are randomly selected
  - if a successor is better than the current state
    - it becomes the new current state
    - otherwise is retained with a given probability
- Weak moves are allowed with a given probability p
  - escape from local optima
- Probability $p = e^{\Delta E/T}$
  - Depends on difference (energy) ΔE
  - Is modelled by a temperature parameter T
- The frequency of weak moves and their size gradually decrease when T is decreasing
  - $T = 0 \rightarrow$ hill climbing
  - $T \rightarrow \infty \rightarrow$ weak moves are frequently performed
- Optimal solution is identified only if the temperature slowly decreases
- How a new possible solution is accepted

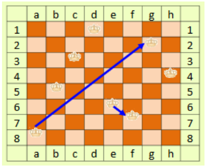
Simulated Annealing

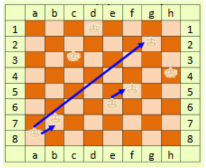
Example: 8-queens problem

Iteration 1 ($k = 1$)

- Current state x =initial state $s_1 = (8, 5, 3, 1, 6, 7, 2, 4)$
  - $f(s_1) = 1 + 1 = 2$
- $x^* = x$
- $T = 100/1 = 100$
- A neighbour of current state $x \rightarrow$ queen of line 5 is swapped with queen of line 7 $\rightarrow$ $s_2 = (8, 7, 3, 1, 6, 5, 2, 4)$
  - $f(s_2) = 1 + 1 + 1 = 3$

- $\Delta E = f(s_2) - f(s_1) = 1$
- $P(\Delta E) = e^{-1/100}$
- $r = rndom(0, 1)$
- if $r < P(\Delta E) \rightarrow x = s_2$





Simulated Annealing

Algorithm

```
function SA(S)
    $x = s_1$                                              ▷ initial state
    $x^* = x$                  ▷ best solution found until a given moment
    $k = 0$                                           ▷ iteration number
    while (not termination criteria) do
        $k = k + 1$
        generate a neighbour $s$ of $x$
        if $f(s)$ is better than $f(x)$ then
            $x = s$
        else
            pick a random number $r \in (0, 1)$
            if $r < P(\Delta E)$ then
                $x = s$
            end if
        end if
    end while
    $x^* = x$ return $x^*$;
end function
```

Simulated Annealing

- **Search analyse**
  - Convergence (complete, optimal) through global optima is slowly
- **Advantages**
  - Statistic-based algorithm $\rightarrow$ it is able to identified the optimal solution, but it requires many iterations
  - Easy to implement
  - Generally, if find a good (global) solution
  - Can solve complex problems (with noise and many constraints)
- **Disadvantages**
  - Slowly algorithm convergence to solution takes a long time
    - Trade-off between the solutions quality and the time required to find it
  - Depends on some parameters (temperature)
  - The provided optimal solution could be local or global
  - The solutions quality depends on the precision of variables involved in the algorithm
- **Applications**
  - Combinatorial optimisation problems $\rightarrow$ knapsack problem
  - Design problems $\rightarrow$ digital circuits design
  - Planning problems $\rightarrow$ production planning, tennis game planning