# ARTIFICIAL INTELLIGENCE

**Solving search problems**

Informed search strategies

Global and local

# Content

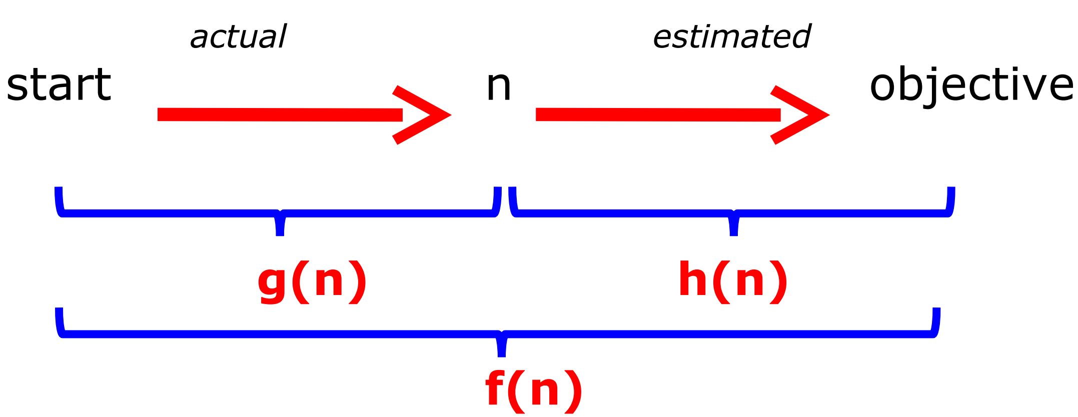
- Solving problem by search
  - Informed search strategies (ISS)
    - Global search strategies
      - Best first search
    - Local search strategies
      - Hill Climbing
      - Simulated Annealing
      - Tabu search

# Informed search strategies (ISS)

- Characteristics
  - Based on specific information about the problem, trying to bound the search space by intelligent choosing the nodes to be explored
  - An evaluation (heuristic) function sorts the nodes
  - Specific to the problem
- Topology
  - Global search strategies
    - Best-first search
      - Greedy best-first search
      - A* + versions of A*
  - Local search strategies
    - Tabu search
    - Hill climbing
    - Simulated annealing

# 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)*



# ISS – Best first search

- **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* = path cost
    - ISSs use heuristic functions
  - 2 possible BFS strategies
    - Expand the closest node to the objective state
    - Expand the best evaluated (best cost) node

- **Example**
  - See next slides ☺

# ISS – Best first search

- **Algorithm**

```
bool BestFS(elem, list){
     found = false;
     visited = Φ;
     toVisit = {start};   //FIFO sorted list (priority queue)
     while((toVisit != Φ) && (!found)){
          if (toVisit == Φ)
            return false
          node = pop(toVisit);
          visited = visited U {node};
          if (node == elem)
            found = true;
          else
            aux = Φ;
          for all unvisited children of node do{
            aux = aux U {child};
          }
          toVisit = toVisit U aux; //adding a node into the FIFO list based on its
                                   // evaluation (best one in the front of list)

     }  //while
     return found;
}
```

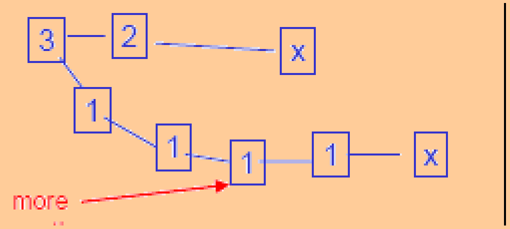
# ISS – best first search

- **Complexity analyse**
  - Time complexity
    - *b* – ramification factor
    - *d* – maximal length (depth) of solution
    - $T(n) = 1 + b + b^2 + \ldots + b^d \Rightarrow O(b^d)$
  - 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 → effort (computational, physic, etc)
  - Some paths could seem to be good

- **Applications**
  - *Web crawler (automatic indexer)*
  - Games



# ISS – 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 path's cost from the current state to the final state

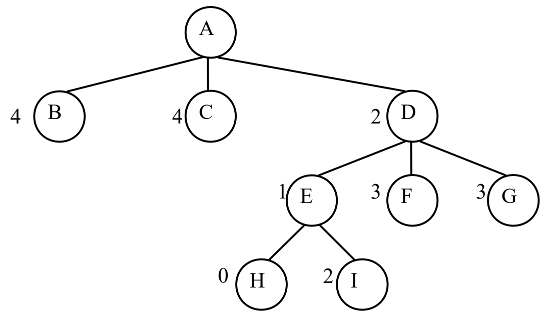
- Characteristics
  - Depends on the problem to be solved
  - New functions for new problems
  - A specific state is evaluated (instead of operators that map a state into another one)
  - Positive functions for each node *n*
    - $h(n) \geq 0$ for all states n
    - $h(n) = 0$ for final state
    - $h(n) = \infty$ for a state that starts a dead end

# ISS – heuristic functions

- Examples
  - 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

# ISS - Greedy

- **Basic elements**
  - 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
- **Example**
  - A,D,E,H




| Vizitate deja | De vizitat |
|---|---|
| Φ | A |
| A | D, B, C |
| A, D | E, F, G, B, C |
| A, D, E | H, I, F, G, B, C |
| A, D, E, H | Φ |

- **Algorithm**

```
bool BestFS(elem, list){
       found = false;
       visited = Φ;
       toVisit = {start};          //FIFO sorted list (priority queue
       while((toVisit != Φ) && (!found)){
              if (toVisit == Φ)
                  return false
              node = pop(toVisit);
              visited = visited U {node};
              if (node == elem)
                  found = true;
              else
                  aux = Φ;
              for all unvisited children of node do{
                  aux = aux U {child};
              }
              toVisit = toVisit U aux; //adding a node onto the FIFO list based on its evaluation h(n)
                                       //(best one in the front of list)

       }  //while
       return found;
}
```

# ISS - Greedy

- **Complexity analyse**
  - Time complexity → DFS
    - *b* – ramification factor
    - $d^{max}$ – maximal length (depth) of an explored tree
    - $T(n) = 1 + b + b^2 + \ldots + b^{dmax} \Rightarrow O(b^{dmax})$
  - Space complexity → BFS
    - *d* - length (depth) of solution
    - $S(n) = 1 + b + b^2 + \ldots + b^d \Rightarrow O(b^d)$
  - Completeness
    - no
  - Optimality
    - possible

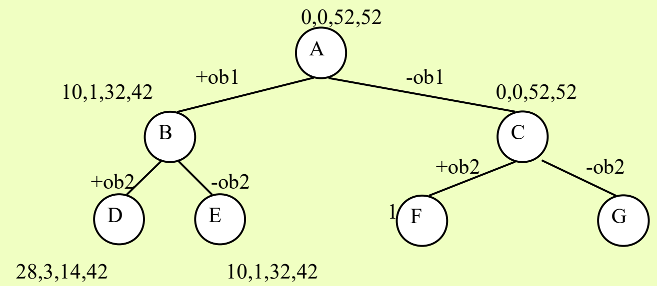
- **Advantages**
  - Quickly finds a solution (possible not-optimal), especially for small problems

- **Disadvantages**
  - Sum of optimal local decisions ≠ global optimal decision
    - Ex. TSP

- **Applications**
  - Planning problems
  - Problem of partial sums
    - Coins
    - knapsack
  - Puzzles
  - Optimal paths in graphs

# ISS – A*

- **Basic elements**
  - Combination of postivie aspects from
    - Uniform cost search
      - Optimality and completness
      - 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




- **Example**
  - Knapsack problem – capacity $W$, $n$ objects $(o_1, o_2, \ldots, o_n)$ each of then having a profit $p_i$, $i=1,2,\ldots,n$
    - Solution: for W = 5 ➔ $o_1$ and $o_3$
  - $g(n) = \Sigma p_i$, for selected objects $o_i$
  - $h(n) = \Sigma p_j$, for not-selected objects and $\Sigma w_j <= W - \Sigma w_i$
  - Fetched 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)*)

| | $o_1$ | $o_2$ | $o_3$ | $o_4$ |
|---|---|---|---|---|
| $p_i$ | 10 | 18 | 32 | 14 |
| $w_i$ | 1 | 2 | 4 | 3 |

# ISS – A*

- **Algorithm**

```
bool BestFS(elem, list){
     found = false;
     visited = Φ;
     toVisit = {start};  //FIFO sorted list (priority queue
     while((toVisit != Φ) && (!found)){
          if (toVisit == Φ)
            return false
          node = pop(toVisit);
          visited = visited U {node};
          if (node == elem)
            found = true;
          else
            aux = Φ;
          for all unvisited children of node do{
            aux = aux U {child};
          }
          toVisit = toVisit U aux; //adding a node onto the FIFO list
                                   // based on its evaluation f(n)= g(n)+ h(n)
                                   // (best one in the front of list)

     }  //while
     return found;
}
```

# ISS – A*

- **Complexity analyse**
  - Time complexity
    - *b* – ramification factor
    - $d^{max}$ – maximal length (depth) of an explored tree
    - $T(n) = 1 + b + b^2 + \dots + b^{dmax} \Rightarrow O(b^{dmax})$
  - Space complexity
    - *d* - length (depth) of solution
    - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
  - Completeness
    - Yes
  - Optimality
    - yes

- **Advantages**
  - Expands the fewest nodes of the tree
- **Disadvantages**
  - Large amount of memory

- **Applications**
  - Planning problems
  - Problems of partial sums
    - Knapsack problem
    - Coin's problem
  - Puzzles
  - Optimal paths in graphs

# ISS – 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*

# Solving problem by search

Methods

- Step-by-step construction of solution
- Identification of a possible optimal solution
  - Algorithms
    - Until now → **systematic** exploration of the search space
      - Eg. A* ➔ $10^{100}$ states ≈ 500 binary variables
    - Real-world problems can have 10 000 – 100 000 variables ➔ require new algorithms that **locally** explore the search space
    - Main idea:
      - Start with a state that does not respect some conditions and
      - Change the state for eliminating these violations
        - The search moves into a neighbourhood of the current solution
      - Such that the search will advance through the optimal state
    - Iterative algorithms
      - Only a state is retained
      - Try to improve this state
    - Intelligent version of brute force algorithm
    - Search past is not retained

```
bool LS(elem, list){
        found = false;
        crtState = initState
        while ((!found) && timeLimitIsNotExceeded){
           toVisit = neighbours(crtState)
           if (best(toVisit) is better than crtState)
          crtState = best(toVisit)
          if (crtState == elem)
                     found = true;
        }  //while
        return found;
}
```

# Solving problem by search

## Methods

- Step-by-step construction of solution
- Identification of a possible optimal solution
  - Advantages
    - Simple implementation
    - Less memory
    - can find good solution in large (continuous) search spaces where other systematic algorithms can not be applied
  - Is useful when
    - Can be generated reasonable complete solutions
    - Can be selected a good starting point
    - Exist operators for solution changing
    - Exists a progress measure (for evaluating how the search advances)
    - Exists an evaluation function for a possible solution

# Local search strategies (LSS)

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

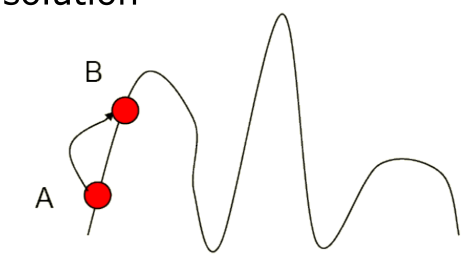
# Local search strategies

- Simple local search
  - Special elements:
    - Solution representation
    - Evaluation of a possible solution
    - Neighbourhood of a solution
      - How a neighbour solution is defined/generated
      - How neighbour solutions are identified:
        - Randomly
        - Systematically
    - How a possible solution is accepted
      - First neighbour of the current solution better than the current solution
      - Best neighbour of the current solution better than the current solution
      - Best neighbour of the current solution weaker than the current solution
      - A random neighbour of the current solution

Depends on problem

# Local search strategies – Hill climbing (HC)

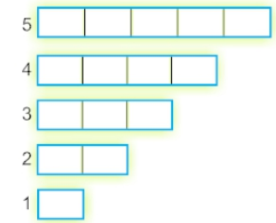
- **Basic elements**
  - Climbing a foggy mountain by an amnesiac hiker :D
  - Continuous moving to better values (larger → mountain climbing)
  - Search advances to improved states until an optimal one is identified
  - How a possible solution is accepted
    - Best neighbour of the current solution better than the current solution
  - Improvement by
    - Maximisation of state’s quality ➔ *steepest ascent HC*
    - Minimisation of state’s quality ➔ *gradient descent HC*



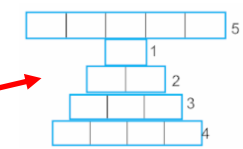

  - HC ≠ *steepest ascent/gradient descent* (SA/GD)
    - HC optimises f(x) with $x \in R^n$ by changing an element of x
    - SA/GD optimises f(x) with $x \in R^n$ by changing all the elements of x

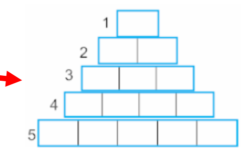
# Local search strategies – Hill climbing (HC)

- **Example**
  - Construct tours 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 tour of 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*)
      - Initial state – vector of the initial tour
      - Final state – vector of the final tour

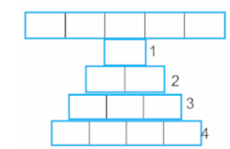


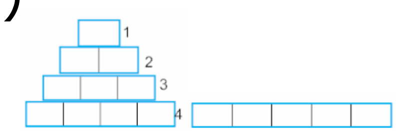

    - State evaluation
      - *f1* = # of correctly located pieces → maximisation
        - Conformably tot the final tour – *f1* = *n*
      - *f2* = # of wrongly located pieces → minimisation
        - Conformably tot the final tour– *f2* = *0*
      - *f* = *f1* – *f2* → maximization
    - Neighbourhood
      - Possible moves
        - Move a piece *i* from stack *j1* on stack *j2*
    - How a possible solution is accepted
      - Best neighbour of the current solution better than the current solution

# Local search strategies – Hill climbing (HC)

- **Example**
  - Iteration 1
    - current state = initial state:
      - $x = s_1 = ((5,1), (1,1), (2,1), (3,1), (4,1))$
      - Pieces 1, 2 and 3 are correctly located
      - Pieces 4 and 5 are wrongly located
      - $f(s_1) = 3 - 2 = 1$



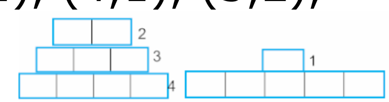

    - $x^* = x$
    - Neighbours of current state x – a single one → piece 5 moves on stack 2 →
      - $s_2 = ((1,1), (2,1), (3,1), (4,1), (5,2))$
      - $f(s_2) = 4-1=3 > f(x)$ → $x = s_2$



# Local search strategies – Hill climbing (HC)

- **Example**
  - Iteration 2
    - Current state  x = ((1,1), (2,1), (3,1), (4,1), (5,2))
      - f(x) = 3
    - Neighbours of the current state – 2 neighbours:
      - Piece 1 moves on stack 2 ➔ $s_3$ = ((2,1), (3,1), (4,1), (1,2), (5,2)) ➔ f($s_3$) = 3-2=1 < f(x)


      - Piece 1 moves on stack 3 ➔ $s_4$ = ((2,1), (3,1), (4,1), (5,2), (1,3)) ➔ f($s_4$) = 3-2=1 < f(x)



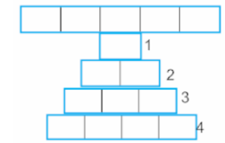

      - There is no neighbour better than x ➔ stop
  - x* = x = ((1,1), (2,1), (3,1), (4,1), (5,2))
  - But x* is a local optimum just (not a global one)

# Local search strategies – Hill climbing (HC)

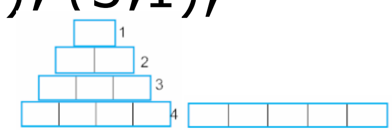
- **Example**
  - Construct tours from different geometrical shapes - other solution
    - State evaluation
      - f1 = sum of stack's height whose all pieces are correctly located (final tour f1 = 10) → maximisation
      - f2 = sum of stack's height whose pieces are wrongly located (final tour f2=0) → minimisation
      - f = f1 – f2 → maximisation
    - Neighbourhood
      - Possible moves
        - Move a piece *i* from stack *j1* on stack *j2*

# Local search strategies – Hill climbing (HC)

## Example




- Iteration 1
  - Current state x = initial state $s_1$ = ((5,1), (1,1), (2,1), (3,1), (4,1))
    - All pieces are wrongly located ➔ f1 = 0, f2 = 3+2 + 1 + 0 + 4 = 10
    - $f(s_1) = 0 – 10 = -10$
  - x* = x
  - Neighbours of current state x– a single one → piece 5 is moved on stack 2 ➔ $s_2$ = ((1,1), (2,1), (3,1), (4,1), (5,2))

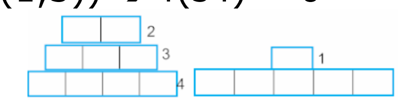



    - $f(s_2) = 0 – (3+2+1+0) = -6 > f(x)$ ➔ $x = s_2$

# Local search strategies – Hill climbing (HC)

- **Example**
  - Iteration 2
    - Current state x = ((1,1), (2,1), (3,1), (4,1), (5,2))
      - f(x) = -6
    - Neighbours of the current state – two neighbours:
      - Piece 1 is moved on stack 2 ➔ s3 = ((2,1), (3,1), (4,1), (1,2), (5,2)) ➔ f(s3) = 0 – (0+2+3+0)=-5 > f(x)


      - Piece 1 is moved 3 ➔ s4 = ((2,1), (3,1), (4,1), (5,2), (1,3)) ➔ f(s4) = 0 - (1+2+1) = -4 > f(x)




      - Best neighbour of x is s4 ➔ x = s4
  - Iteration 3
    - ...
  - Local optima are avoided

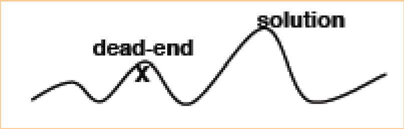
# Local search strategies – Hill climbing (HC)

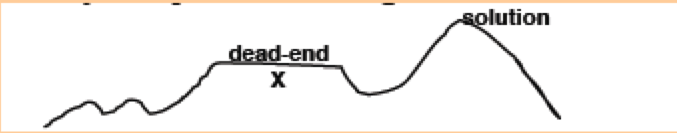
- **Algorithm**

```
Bool HC(S){
    x = s1    //initial state
    x*=x      // best solution (found until now)
    k = 0     // # of iterations
    while (not termination criteria){
    k = k + 1
    generate all neighbours of x (N)
    Choose the best solution s from N
    if f(s) is better than f(x) then x = s
    else stop
    }  //while
    x* = x
    return x*;
}
```

# Local search strategies – Hill climbing (HC)

- **Search analyse**
  - Convergence to local optima
- **Advantages**
  - Simple implementation -> solution approximation (when the real solution is difficult or impossible to find)
    - Eg. 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
  - Queen’s problem

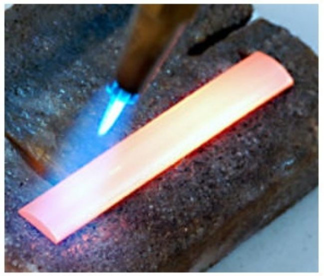


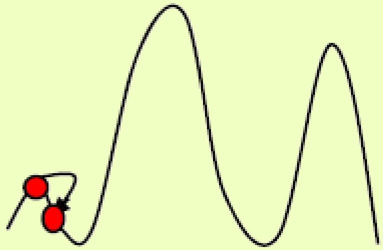



# Local search strategies – Hill climbing (HC)

- **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 → *beam local search*
    - Restart the search from a randomly initial state when the search does not advance

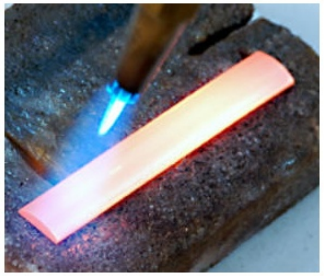
# Local search strategies – Simulated Annealing





- **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, it is retained by 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) $\Delta E$
    - Is modelled by a temperature parameter $T$
  - The frequency of weak moves and their size gradually decrease when T is decreasing
    - T = 0 → hill climbing
    - T → ∞ → weak moves are frequently performed
  - Optimal solution is identified only if the temperature slowly decreases
  - How a possible solution is accepted
    - A random neighbour of the current solution better  than the current solution or
    - Probabilistic, a random neighbour of the current solution weaker than the current solution

# Local search strategies – Simulated Annealing

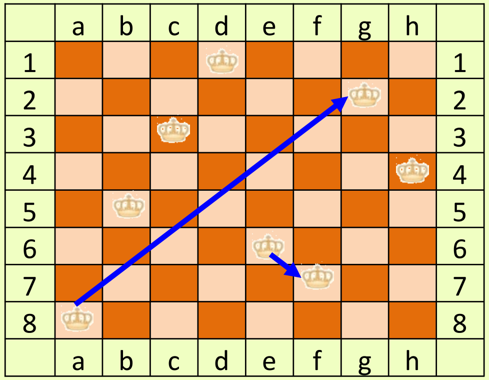


- **Example – 8-queen's problem**
  - Statement
    - Put 8 queens on a chessboard such there are no attacks between queens
  - Solution representation
    - State *x* – permutation of *n* elements x = ($x_1$,$x_2$,..,$x_n$), where $x_i$ – line where the queen of column j is located
      - There are no attacks on lines or on columns
      - It is possible to find diagonal attacks
    - Initial state – a random permutation
    - Final state – a permutation without attacks
  - Evaluation function for a state
    - F – sum of attacked queens by each queen → minimisation
  - Neighbourhood
    - Possible moves
      - Move a queen from a line to a new line (swap 2 elements from permutation)
  - How a possible solution is accepted
    - A random neighbour of the current solution
      - better than the current solution or
      - Weaker than the current solution – by a probability $P(\Delta E) = e^{-\frac{\Delta E}{T}}$ ,where
        - $\Delta$E – energy (evaluation) difference of two states
        - T – temperature, T(k) = 100/k, where k is the iteration number

# Local search strategies – Simulated Annealing

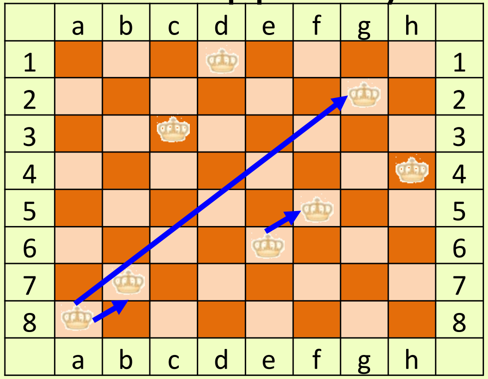
## Example – 8-queen's problem

- Iteration 1 (k = 1)
  - Current state x = initial state
    $s_1 = (8,5,3,1,6,7,2,4)$
    - $f(s_1) = 1+1 = 2$

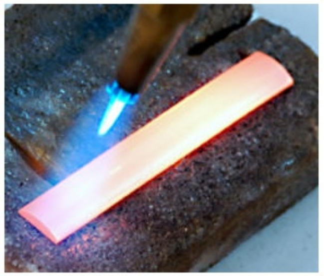



  - $x^* = x$
  - $T = 100/1 = 100$
  - A neighbour of current state x → queen of line 5 is swapped by queen of line 7

    ➔ $s_2 = (8,7,3,1,6,5,2,4)$
    - $f(s_2) = 1+1+1=3 > f(x)$
    - $\Delta E = f(s_2) - f(s_1) = 1$
    - $P(\Delta E)=e^{-1/100}$
    - $r = \mathrm{random}(0,1)$
    - if $r < P(\Delta E)$ ➔ $x = s_2$



# Local search strategies – Simulated Annealing

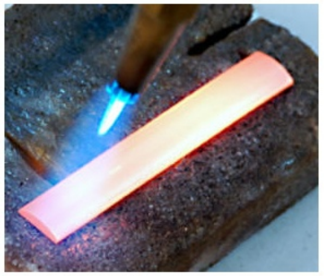


- **Algorithm**

```
bool SA(S){
      x = s1           //initial state
      x*=x  // best solution found until a given moment
      k = 0 // iteration number
      while (not termination criteria){
      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) range)
            if r < P(ΔE) then x = s
      }  //while
      x* = x
      return x*;
}
```

- Stop conditions
  - The solution is found
  - A number of iterations is reached
  - The frozen temperature (T=0) is hit
- How a small probability is chosen?
  - p = 0.1
  - P decreases along the iterations
  - P decreases along the iterations and while the "error" |f(s) – f(x)| is increasing
  - p = exp(- |f(s) – f(x)|/T)
  - Where T –temperature (that increases)
    - For a large T almost any neighbour is accepted
    - For a small T, only neighbours better that s are accepted
  - If the error is large, then the probability is small

# Local search strategies – Simulated Annealing

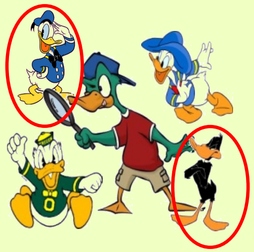


- **Search analyse**
  - Convergence (complete, optimal) through global optima is slowly

- **Advantages**
  - Statistic-based algorithm → 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 solution's quality and the time required to find it
  - Depends on some parameters (temperature)
  - The provided  optimal solution could be local or global
  - The solution's quality depends on the precision of variables involved in the algorithm

- **Applications**
  - Combinatorial optimisation problems → knapsack problem
  - Design problems → digital circuits design
  - Planning problems → production planning, tennis game planning

# Local search strategies – Tabu search

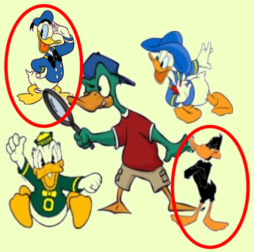


- **Basic elements**
  - “Tabu” → things that cannot be touched because they are sacred
  - Proposed in 1970 by F. Glover

  - Main idea
    - starts with a state that violates some constraints and
    - Performs changes for eliminating them (the search moves into the best neighbour solution of the current solution) in order to identify the optimal solution
    - Retains
      - Current state
      - Visited states and performed moves (limited list of states that must be avoided)
    - How a possible solution is accepted
      - Best neighbour of the current solution better than the current solution and non-visited until that moment

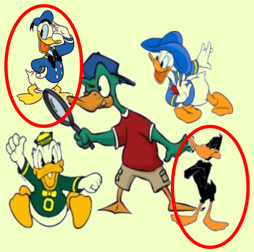
  - 2 important elements
    - Tabu moves (T) – determined by a non-Markov process that uses information obtained during last generations of search process
    - Tabu conditions – linear inequalities or logical links that depend on current solution
      - Influence the selection of tabu moves

# Local search strategies – Tabu search



- **Example**
  - Statement
    - Pay a sum $S$ by using $n$ coins of values $v_i$ as many as possible (each coin has $b_i$ copies)
  - Solution representation
    - State $x$ – vector of $n$ integers $x = (x_1, x_2, ..., x_n)$ with $x_i \in \{0, 1, 2, ..., b_i\}$
    - Initial state – randomly
  - State evaluation
    - $f_1$ = S – total value of selected coins → minimisation
      - If the total value of coins > S → penalisation (eg. 500 units)
    - $f_2$ = number of selected coins → maximisation
    - $f = f_1 - f_2$ → minimisation
  - neighbourhood
    - Possible moves
      - Including in the sum of $j$ copies of coin $i$ ($plus_{i,j}$)
      - Eliminating from the sum of $j$ copies of coin $i$ ($minus_{i,j}$)
    - Tabu list retains performed moves of an iteration
      - - move = the added/eliminated coin

# Local search strategies – Tabu search



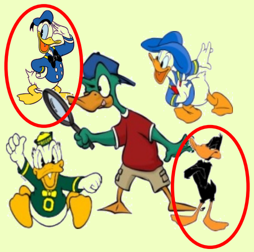
- **Example**
  - $S$ = 500, penalisation= 500, $n$ = 7

| S=500 | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ |
|---|---|---|---|---|---|---|---|
| $v_i$ | 10 | 50 | 15 | 20 | 100 | 35 | 5 |
| $b_i$ | 5 | 2 | 6 | 5 | 5 | 3 | 10 |

| Stare curentă | Val. f | Listă tabu | Stări vecine | Mutări | Val. f |
|---|---|---|---|---|---|
| 2 0 1 0 0 2 1 | 384 | ∅ | 2 0 1 3 0 2 1 | $plus_{4,3}$ | 321 |
| | | | 2 0 1 0 0 3 1 | $plus_{6,1}$ | 348 |
| | | | 0 0 1 0 0 2 1 | $minus_{1,2}$ | 406 |
| 2 0 1 3 0 2 1 | 321 | $plus_{4,3}$ | 2 0 1 3 5 2 1 | $plus_{5,5}$ | 316 |
| | | | 2 0 1 1 0 2 1 | $minus_{4,2}$ | 363 |
| | | | 2 1 1 3 0 2 1 | $plus_{2,1}$ | 270 |
| 2 1 1 3 0 2 1 | 270 | $plus_{4,3}$<br>$plus_{2,1}$ | ... | | |

  - Final solution: 4 1 5 4 1 3 10 ($f$ = -28)

# Local search strategies – Tabu search



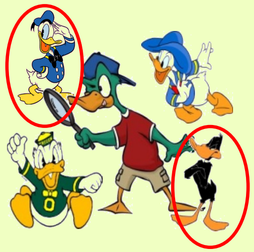
- **Example**
  - $S$ = 500, penalisation = 500, $n$ = 7

| S=50 | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ |
|---|---|---|---|---|---|---|---|
| $v_i$ | 10 | 50 | 15 | 20 | 100 | 35 | 5 |
| $b_i$ | 5 | 2 | 6 | 5 | 4 | 3 | 10 |

| Stare curentă | Val. f | Listă tabu | Stări vecine | Mutări | Val. f |
|---|---|---|---|---|---|
| 2 0 1 0 0 2 1 | 384 | ∅ | 1 0 1 4 0 2 1 | $minus_{1,1}$,$plus_{4,4}$ | 311 |
| | | | 2 0 4 0 1 2 1 | $plus_{3,3}$,$minus_{5,1}$ | 235 |
| | | | 2 0 1 0 4 2 6 | $plus_{5,4}$, $plus_{7,5}$ | 450 |
| 2 0 4 0 1 2 1 | 235 | $plus_{3,3}$, $minus_{5,1}$ | 2 0 5 0 5 2 1 | $plus_{3,1}$, $plus_{5,4}$ | 315 |
| | | | 5 0 4 0 4 2 1 | $plus_{1,3}$, $plus_{5,3}$ | 399 |
| | | | 2 2 4 0 5 2 1 | $plus_{2,2}$, $plus_{5,4}$ | 739 |
| 2 0 4 0 1 2 1 | 235 | $plus_{3,3}$, $minus_{5,1}$ | ... | | |

  - Final solution: 4 1 5 4 1 3 10 ($f$ = -28)

# Local search strategies – Tabu search

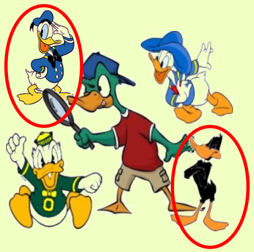


- **Algorithm**

```
bool TS(S){
    Select x∈S   //S - search space
    x*=x         //best solution until a moment
    k = 0        //iteration number
    T = ∅        //list of tabu moves
    while (not termination criteria){
    k = k + 1
    generate a subset of solutions in the neighbourhood N-T of x
    choose the best solution s from N-T and set x=s.
    if f(x)<f(x*) then x*=x
    update T with moves of generating x
    }  //while
    return x*;
}
```

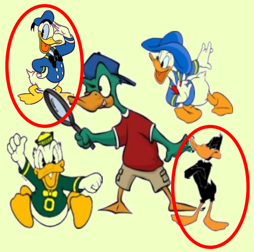
- Stop conditions
  - Fix number of iterations
  - A given number of iterations without improvements
  - Sufficient proximity to the solution (if it is known)
  - Depletion unvisited elements of a neighbourhood

# Local search strategies – Tabu search



- **Search analyse**
  - Quickly convergence to global optima
- **Advantages**
  - The algorithm is general and can be easy implemented
  - Quickly algorithm (can find in a short time the optimal solution)
- **Disadvantages**
  - Identify the neighbours in continuous search spaces
  - Large number of iterations
  - Global optima identification is not guaranteed

# Local search strategies – Tabu search



- **Applications**
  - Determination of tridimensional structure of proteins in amino acid sequences
  - Traffic optimisation in communication networks
  - Planning in production systems
  - Network design in optical telecommunication
  - Automatic routing of vehicles
  - Graph problems (partitioning)
  - Planning in audit systems

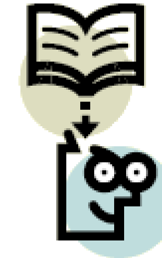
# Review

- ISS best first search
  - The best evaluated nodes are firstly expanded

  - Greedy ISS
    - Minimisation of the cost from the current state to the final state – h(n)
    - Search time < USS
    - Non-complete
    - Non-optimal

  - A* ISS
    - Minimisation of the cost from the initial state to the current state – g(n) and of the cost from the current state to the final state – h(n)
    - Avoid to re-visit a state
    - Without supra-estimation of h(n)
    - Large time and space complexity → depends on used heuristic
    - Complete
    - Optimal

# Review



- Local SS
  - Iterative algorithms
    - Work with a possible solution → optimal solution
    - Can block in local optima

| | Nex state selection | Acceptance criteria | Convergence |
|---|---|---|---|
| HC | Best neighbor | Neighbor is better than current state | Local or global optima |
| SA | Random neighbor | Neighbor is better than current state or neighbor is weaker than current state (probabilistic acceptance) | Global optima (slowly) |
| TS | Best non-visited neighbor | Neighbor is better than current state | Global optima (quickly) |