

BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics

---

# ARTIFICIAL INTELLIGENCE

**Solving search problems**

Uninformed search strategies

# Topics

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

#### A. Definition of search problems

#### B. Search strategies

A. Uninformed search strategies

B. Informed search strategies

C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)

D. Adversarial search strategies

### C. Intelligent systems

#### A. Rule-based systems in certain environments

#### B. Rule-based systems in uncertain environments (Bayes, Fuzzy)

#### C. Learning systems

A. Decision Trees

B. Artificial Neural Networks

C. Support Vector Machines

• Evolutionary algorithms

#### D. Hybrid systems

# Content

---

- Problems
- Problem solving
  - Steps of problem solving
- Solving problem by search
  - Steps of solving problem by search
  - Search strategies

# Useful information

---

- Chapters I.1, I.2 și II.3 of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
  
- Chapters 1 and 2 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
  
- Chapters 2.1 – 2.4 of

<http://www-g.eng.cam.ac.uk/mmg/teaching/artificialintelligence/>

# Problems

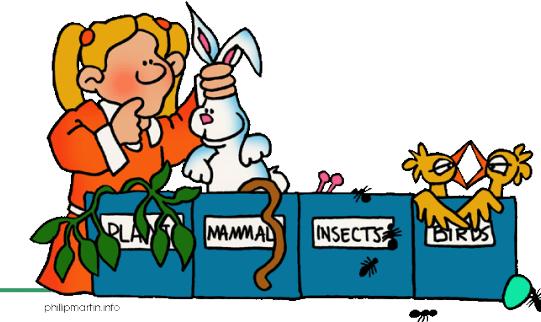
---



- Two problem types:
  - Solving in a deterministic manner
    - Computing the sinus of an angle or the square root of a number
  - Solving in a stochastic manner
    - Real-world problems → design of ABS
    - Involve the search of a solution → AI's methods

# Problems

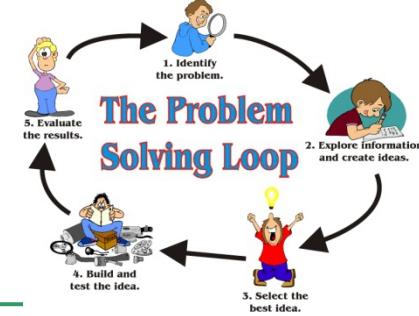
- Tipology
  - Search/optimization problems
    - Planning, satellite's design
  - Modeling problems
    - Predictions, classifications
  - Simulation problems
    - Game theory





# Problem solving

- Identification of a solution
  - In computer science (AI) → search process
  - In engineering and mathematics → optimisation process
  
- How?
  - Representation of (partial) solutions → points in the search space
  - Design of a search operators → map a potential solution into another one



# Steps in problem solving

- Problem definition
- Problem analyses
- Selection of a solving technique
  - Search
  - Knowledge representation
  - Abstract methods



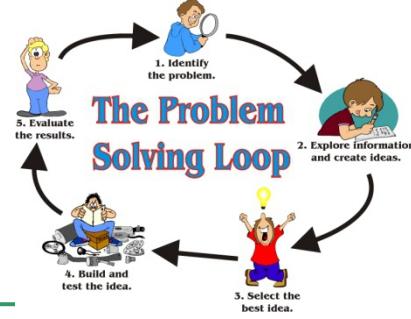
# Solving problems by search

---

- Based on some objectives
- Composed by 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

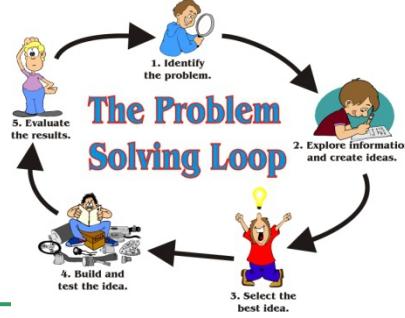
# Steps in solving problems by search

## Problem definition



### □ Problem definition involves:

- A 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



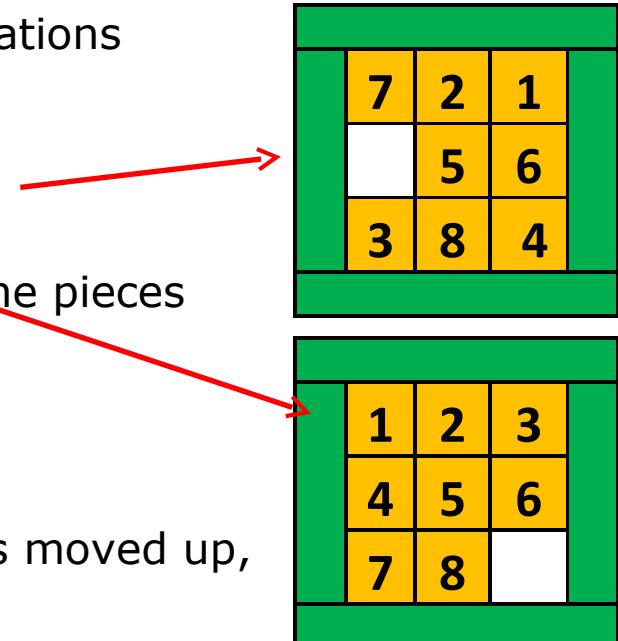
# Steps in solving problems by search

## Problem definition

### □ Examples

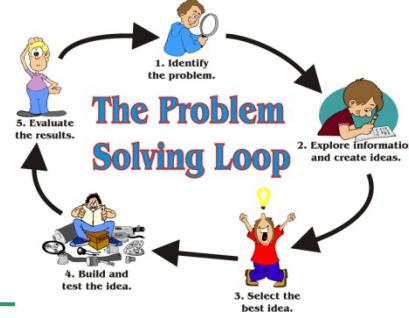
- Puzzle game with 8 pieces

- State's space – different board configurations for a game with 8 pieces
- Initial state – a random configuration
- Final state – a configuration where all the pieces are sorted in a given manner
- Rules -> white moves
  - conditions: move inside the table
  - Transformations: the white space is moved up, down, to left or to right
- Solution - optimal sequence of white moves



# Steps in solving problems by search

## Problem definition



### □ Examples

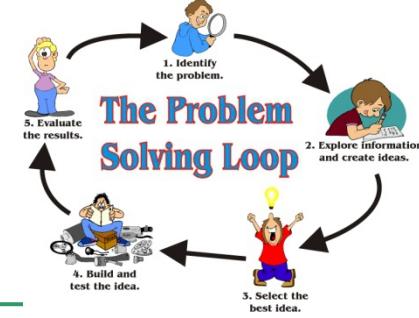
#### ■ Queen's problem

- State's space – different board configurations for a game with n queens
- Initial state – a configuration without queens
- Final state – a configuration n queens so that none of them can hit any other in one move
- Rules -> put a queen on the table
  - conditions: the queen is not hit by any other queen
  - Transformations: put a new queen in a free cell of the table
- Solution - optimal placement of queens

	a	b	c	d	e	f	g	h	
1	Q								1
2				Q					2
3			Q						3
4				Q					4
5		Q							5
6				Q					6
7			Q						7
8				Q					8
	a	b	c	d	e	f	g	h	

# Steps in solving problems by search

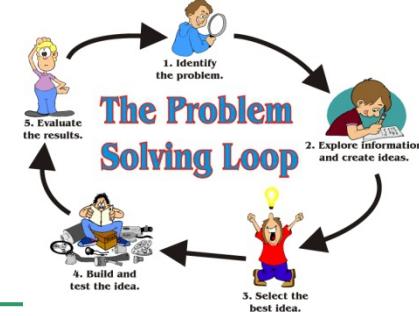
## Problem analyse



- The problem can be decomposed?
  - The sub-problems are independent or not?
- The possible state's space is predictable?
- We want a solution or an optimal solution?
- The solution is represented by a single state or by more successive states?
- We require some knowledge for limiting the search or for identifying the solution?
- The problem is conversational or solitary?
  - Human interaction is required for problem solving?

# Steps in solving problems by search

## Selection of a solving technique

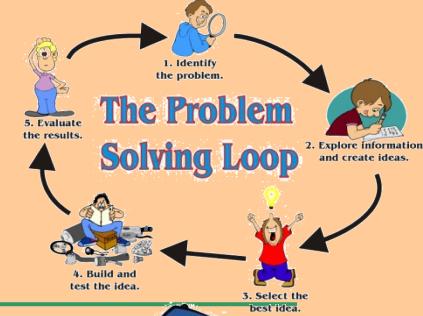


- Solving by moving rules (and control strategy) in the search space until we find a path from the initial state to the final state
- Solving by search
  - ▣ Examination of all possible states in order to identify
    - A path from the initial state to the final state
    - An optimal state
  - ▣ The search space = all possible states and the operators that maps the states



# Steps in solving problems by search

## Selection of a solving technique



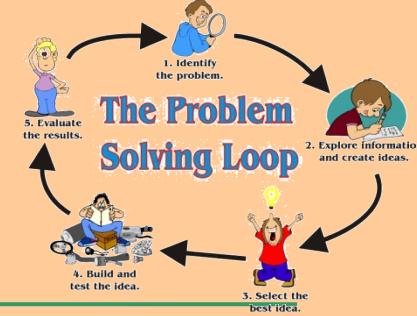
### □ Solving by search

- More searching strategies → how we select one of them?
  - Computational complexity (temporal and spatial)
  - Completeness → the algorithms always ends and finds a solution (if it exists)
  - Optimality → the algorithms finds the optimal solution (the optimal cost of the path from the initial state to the final state)



# Steps in solving problems by search

## Selection of a solving technique



### □ Solving by search

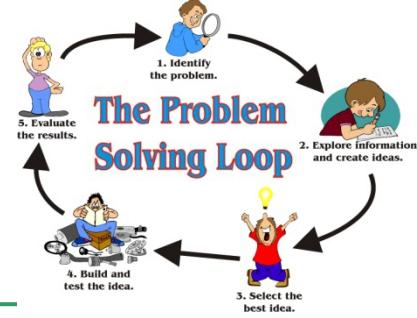
- More searching strategies → how we select one of them? → Computational complexity (temporal and spatial)

- Strategy's performance depends on
  - Time for running
  - Memory for running
  - Size of input data
  - Computer's performance
  - Compiler's quality
- Can be evaluated by complexity → computational efficiency
  - Spatial → required memory for solution identification
    - $S(n)$  – memory used by the best algorithms A that solves a decision problem  $f$  with  $n$  input data
  - Temporal → required time for solution identification
    - $T(n)$  – running time (number of steps) of the best algorithm A that solves a decision problem  $f$  with  $n$  input data



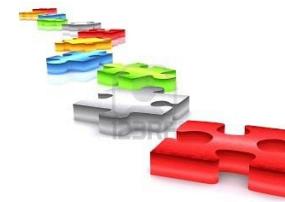
# Steps in solving problems by search

## Selection of a solving technique



- Problem solving by search can be performed by:

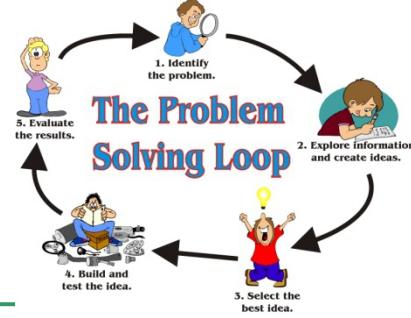
- Step by step construction of solution
- Optimal solution identification



www.shutterstock.com - 36774760

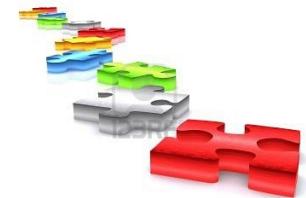
# Steps in solving problems by search

## Selection of a solving technique



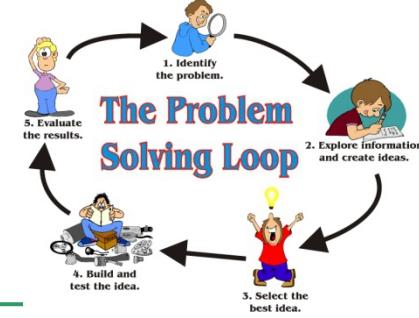
### □ Problem solving by search can be performed by:

- Step by step construction of solution
  - ▣ Problem's components
    - Initial state
    - Operators (successor functions)
    - Final state
    - Solution = a path (of optimal cost) from the initial state to the final state
  - ▣ Search space
    - All the states that can be obtained from the initial state (by using the operators)
    - A state = a component of solution
  - ▣ Example
    - Traveling Salesman Problem (TSP)
  - ▣ Algorithms
    - Main idea: start with a solution's component and adding new components until a complete solution is obtained
    - Recurrent → until a condition is satisfied
    - The search's history (path from initial state to the final state) is retained in LIFO/FIFO containers
  - ▣ Advantages
    - Do not require knowledge (intelligent information)



# Steps in solving problems by search

## Selection of a solving technique



### □ Problem solving by search can be performed by:

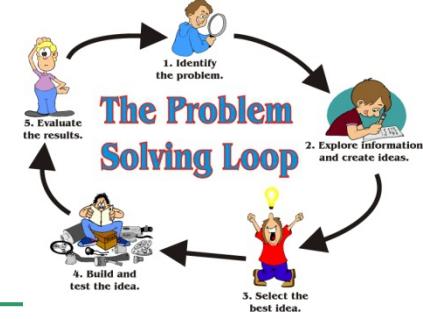
- Optimal solution identification
  - ▣ Problem's components
    - Conditions (constraints) that must be satisfied by the solution
    - Evaluation function for a potential solution → optimum identification
  - ▣ Search space
    - All possible and complete solutions
    - State = a complete solution
  - ▣ Example
    - Queen's problem
  - ▣ Algorithms
    - Main idea: start with a state that doesn't respect some conditions and change it for eliminating these violations
    - Iterative → a single state is retained and the algorithm tries to improve it
    - The searches' history is not retained
  - ▣ Advantages
    - Simple
    - Requires a small memory
    - Can find good solutions in (continuous) search spaces very large (where other algorithms can not be utilised)



www.shutterstock.com - 36774760

# Steps in solving problems by search

## Selection of a solving technique



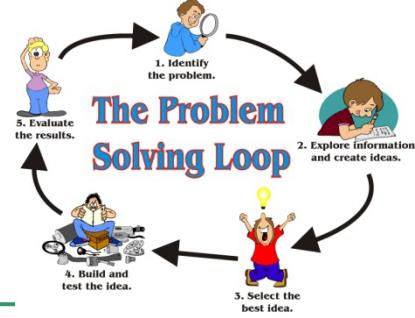
### ❑ Solving problem by search involves:

- Very complex algorithms (NP-complete problems)
- Search in an exponential space



# Steps in solving problems by search

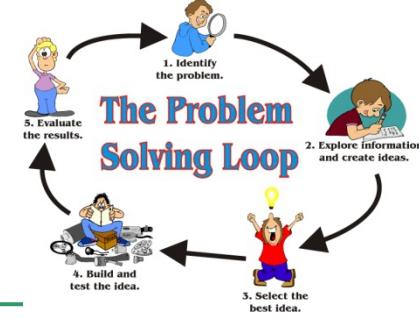
## Selection of a solving technique



- Topology of search strategies:
  - Solution **generation**
    - **Constructive** search
      - Solution is identified step by step
      - Ex. TSP
    - **Perturbative** search
      - A possible solution is modified in order to obtain another possible solution
      - Ex. SAT - Propositional Satisfiability Problem
  - Search space **navigation**
    - **Systematic** search
      - The entire search space is visited
        - Solution identification (if it exists) → complete algorithms
    - **Local** search
      - Moving from a point of the search space into a neighbor point → incomplete algorithms
      - A state can be visited more times
  - **Certain** items of the search
    - **Deterministic** search
      - Algorithms that exactly identify the solution
    - **Stochastic** search
      - Algorithms that approximate the solution
  - Search space **exploration**
    - **Sequential** search
    - **Parallel** search

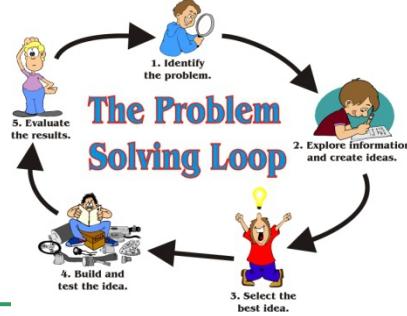
# Steps in solving problems by search

## Selection of a solving technique



Topology of search strategies:

- **Number of objectives**
  - **Single-objective** search
    - The solution must respect a single condition/constraint
  - **Multi-objective** search
    - The solution must respect more conditions/constraints
- **Number of solutions**
  - **single-modal search**
    - There is a single optimal solution
  - **multi-modal search**
    - There are more optimal solutions
- **Algorithm**
  - Search over a **finite number of steps**
  - **Iterative search**
    - The algorithms converge through the optimal solutions
  - **Heuristic search**
    - The algorithms provide an approximation of the solution
- Search **mechanism**
  - **traditional search**
  - **modern search**
- where the search takes **place**
  - **local search**
  - **global search**



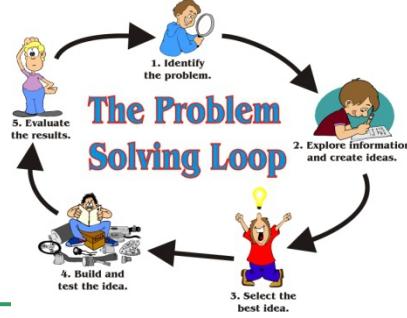
# Steps in solving problems by search

## Selection of a solving technique

---

Topology of search strategies:

- Type (**linearity**) of constraints
  - **Linear search**
  - **non-linear search**
    - Clasical (deterministic)
      - Direct – based on evaluation of the objective function
      - Indirect – based on derivates (I and/or II) of the objective function
    - Enumeration-based
      - How solution is identified
        - Uninformed – the solution is the final state
        - Informed – deals with an evaluation function for a possible solution
      - Search space type
        - Complete – the space is finite (if solution exists, then it can be found)
        - Incomplete – the space is infinite
    - Stochastic search
      - Based on random numbers
- **Agents** involves in search
  - Search by **a single agent** → without obstacle for achieving the objectives
  - **Adversarial search** → the opponent comes with some uncertainty



# Steps in solving problems by search

## Selection of a solving technique

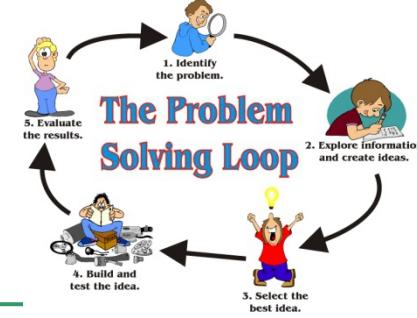
---

### Example

- Topology of search strategies
  - Solution generation
    - **Constructive search**
    - Perturbative search
  - Search space navigation
    - **Systematic search**
    - Local search
  - Certain items of the search
    - **Deterministic search**
    - Stochastic search
  - Search space exploration
    - **Sequential search**
    - Parallel search

# Steps in solving problems by search

## Selection of a solving technique

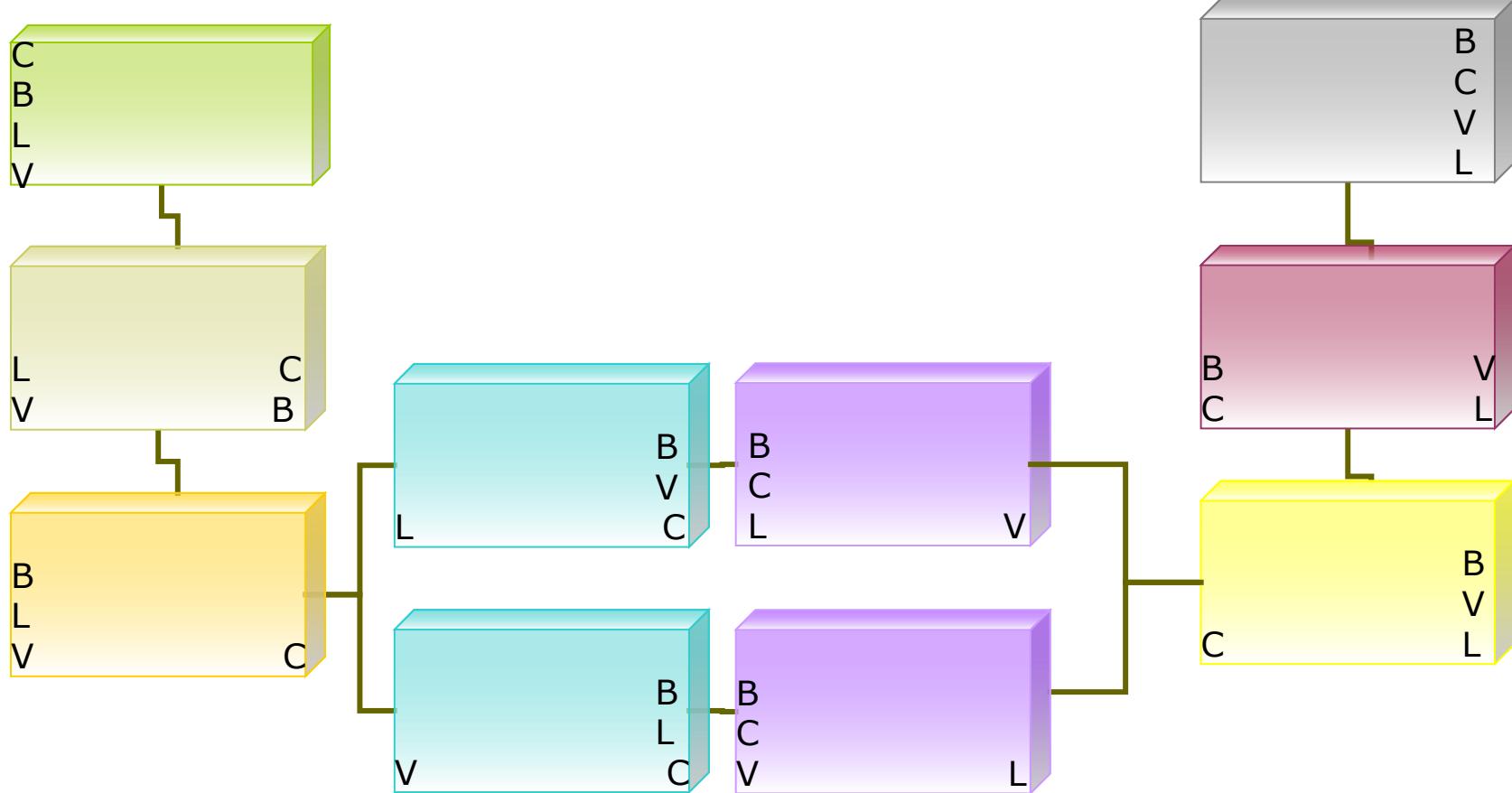
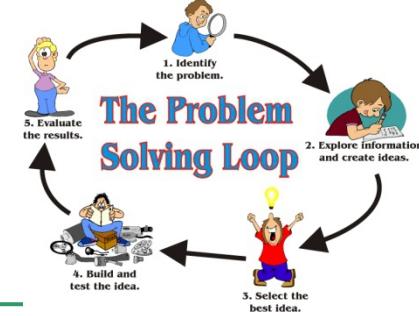


### Example

- Constructive, global, deterministic, sequential search
- Problem "capra, varza și lupul"
  - Input :
    - A goat, a cabbage and a wolf on a river-side
    - A boat with a boater
  - Output:
    - Move all the passengers on the other side of the river
    - Taking into account:
      - The boat has only 2 places
      - It is not possible to rest on the same side:
        - The goat and the cabbage
        - The wolf and the goat

# Steps in solving problems by search

## Selection of a solving technique



# Search strategies – Basic elements

---

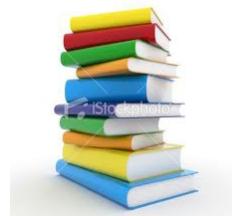


- Abstract data types (ADTs)
  - ADT list → linear structure
  - ADT tree → hierachic structure
  - ADT graph → graph-based structure
  
- ADT
  - Domain and operations
  - Representation

# Search strategies – Basic elements – ADT list



- Domain
  - $D = \{l \mid l = (el_1, el_2, \dots), \text{ where } el_i, i=1,2,3\dots, \text{ are of type } TE \text{ (type of element) and each element } el_i, i=1,2,3\dots, \text{ has a unique position in } l \text{ of type } TP \text{ (Type of position)}\}$
- Operations
  - Create(l)
  - First(l)
  - Last(l)
  - Next(l,p)
  - Prev(l,p)
  - Valid(l,p)
  - getElement(l,p)
  - getPoz (l,e)
  - Modify(l,p,e)
  - AddFirst(l,e)
  - AddToEnd(l,e)
  - AddAfter(l,p,e)
  - AddBefore(l,p,e)
  - Eliminate(l,p)
  - Search(l,e)
  - IsEmpty(l)
  - Dimension(l)
  - Destroy(l)
  - getIterator(l)
- Representation
  - Vector-based
  - Linked lists
- Special cases
  - **Stack – LIFO**
  - Queue – FIFO
  - Priority queue



# Search strategies – Basic elements – ADT list



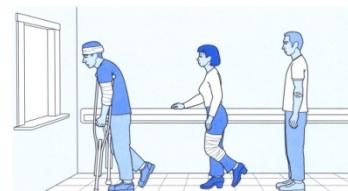
- Domain
  - $D = \{l \mid l = (el_1, el_2, \dots), \text{ where } el_i, i=1,2,3\dots, \text{ are of type } TE \text{ (type of element) and each element } el_i, i=1,2,3\dots, \text{ has a unique position in } l \text{ of type } TP \text{ (Type of position)}\}$
- Operations
  - Create(l)
  - First(l)
  - Last(l)
  - Next(l,p)
  - Prev(l,p)
  - Valid(l,p)
  - getElement(l,p)
  - getPoz(l,e)
  - Modify(l,p,e)
  - AddFirst(l,e)
  - AddToEnd(l,e)
  - AddAfter(l,p,e)
  - AddBefore(l,p,e)
  - Eliminate(l,p)
  - Search(l,e)
  - IsEmpty(l)
  - Dimension(l)
  - Destroy(l)
  - getIterator(l)
- Representation
  - Vector-based
  - Linked lists
- Special cases
  - Stack – LIFO
  - **Queue – FIFO**
  - Priority queue



# Search strategies – Basic elements – ADT list



- Domain
  - $D = \{l \mid l = (el_1, el_2, \dots), \text{ where } el_i, i=1,2,3\dots, \text{ are of type } TE \text{ (type of element) and each element } el_i, i=1,2,3\dots, \text{ has a unique position in } l \text{ of type } TP \text{ (Type of position)}\}$
- Operations
  - Create(l)
  - First(l)
  - Last(l)
  - Next(l,p)
  - Prev(l,p)
  - Valid(l,p)
  - getElement(l,p)
  - getPoz(l,e)
  - Modify(l,p,e)
  - AddFirst(l,e)
  - AddToEnd(l,e)
  - AddAfter(l,p,e)
  - AddBefore(l,p,e)
  - Eliminate(l,p)
  - Search(l,e)
  - IsEmpty(l)
  - Dimension(l)
  - Destroy(l)
  - getIterator(l)
- Representation
  - Vector-based
  - Linked lists
- Special cases
  - Stack – LIFO
  - Queue – FIFO
  - **Priority queue**



# Search strategies – Basic elements – ADT Graph

---



- Domain – container with nodes and links among nodes
  - $D = \{node_1, node_2, \dots, node_n, link_1, link_2, \dots, link_m\}$ , where  $node_i, i=1,2,\dots,n$  are nodes and  $link_i, i=1,2,\dots,m$  are edges between nodes}
- Operations
  - create
  - createNode
  - traverse
  - getIterator
  - destroy
- Representation
  - List of edges
  - List of adjacency
  - Matrix of adjacency
  - Incident matrix
- Special cases
  - Un-oriented and oriented graphs
  - Simple and multiple graphs
  - Connex and non-connex graphs
  - Complete or non-complete graph
  - With or without cycles (a-cycle → forests, trees)

# Search strategies – Basic elements – ADT Tree

---



- Domain – container with nodes and links between nodes
  - $D = \{node_1, node_2, \dots, node_n, link_1, link_2, \dots, link_m, \text{ where } node_i, i=1,2,\dots,n \text{ are nodes and } link_i, i=1,2,\dots,m \text{ are edges between nodes without cycles}\}$
- Operations
  - create
  - createLeaf
  - addSubtree
  - getInfoRoot
  - getSubtree
  - traverse
  - getIterator
  - destroy
- Representation
  - Vector-based
  - Linked lists of children
- Special cases
  - Binary trees (search trees)
  - N-ary trees

# Search strategies – Basic elements – paths in graphs

---



- *path*
  - Unique nodes
- *trail*
  - Unique edges
- *walk*
  - Without restriction
- Close path
  - Initial node = final node
- circuit
  - A closed *trail*
- cycle
  - A closed *path*



# Uninformed search strategies (USS)

- Characteristics
  - Are NOT based on problem specific information
  - Are general
  - Blind strategies
  - Brute force methods
- Topology
  - Order of 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

# USS in linear structures

## Linear search



- **Theoretical aspects**
  - Checks each element of a list until the search one is found
  - The list of elements can be sorted
- **Example**
  - List = ( 2, 3, 1, ,7, 5)
  - Elem = 7
- **Algorithm**

```
bool LS(elem, list){  
    found = false;  
    i = 1;  
    while ((!found) && (i <= list.length)) {  
        if (elem = list[i])  
            found = true;  
        else  
            i++;  
    } //while  
    return found;  
}
```



# USS in linear structures

## Linear search

### □ Search analyse

- Time complexity
  - Best case:  $elem = list[1] \Rightarrow O(1)$
  - Worst case:  $elem \notin list \Rightarrow T(n) = n + 1 \Rightarrow O(n)$
  - Average case:  $T(n) = (1 + 2 + \dots + n + (n+1))/(n+1) \Rightarrow O(n)$
- Space complexity
  - $S(n) = n$
- Completeness
  - yes
- Optimality
  - yes

### □ Advantages

- Simplicity, good time complexity for small structures
- Containers can be un-sorted

### □ Disadvantages

- Bad time complexity for large structures

### □ Applications

- Search in real data bases



# USS in linear structures

## Binary search

### □ Theoretical aspects

- Identify an element in a sorted list
- *Divide et Conquer* strategy

### □ Example

- List = ( 2, 3, 5, 6, 8, 9, 13, 16, 18), Elem = 6
  - List = ( 2, 3, 5, 6, 8, 9, 13, 16, 18)
  - List = ( 2, 3, 5, 6)
  - List = ( 5, 6)
  - List = ( 6)

### □ Algorithm

```
bool BS(elem, list){  
    found = false;  
    left = 1;  
    right = list.length;  
    while((left < right) && (!found)){  
        middle = left + (right - left)/2;  
        if (element == list[middle])  
            found = true;  
        else  
            if (element < list[middle])  
                right = middle - 1;  
            else  
                left = middle + 1;  
    } //while  
    return found;  
}
```



# USS in linear structures

## Binary search

### □ Search analyse

- Time complexity  $T(n) = 1$ , for  $n = 1$  and  $T(n) = T(n/2) + 1$ , otherwise
  - Suppose that  $n = 2k \Rightarrow k = \log_2 n$
  - Suppose that  $2k < n < 2k+1 \Rightarrow k < \log_2 n < k + 1$
  - $$\begin{aligned}T(n) &= T(n/2) + 1 \\T(n/2) &= T(n/2^2) + 1 \\&\dots \\T(n/2^{k-1}) &= T(n/2^k) + 1\end{aligned}$$

---

$$T(n) = k + 1 = \log_2 n + 1$$
- Space complexity –  $S(n) = n$
- Completeness – yes
- Optimality – yes

### □ Advantages

- Low time complexity compare to linear search

### □ Disadvantages

- Work with sorted vectors

### □ Applications

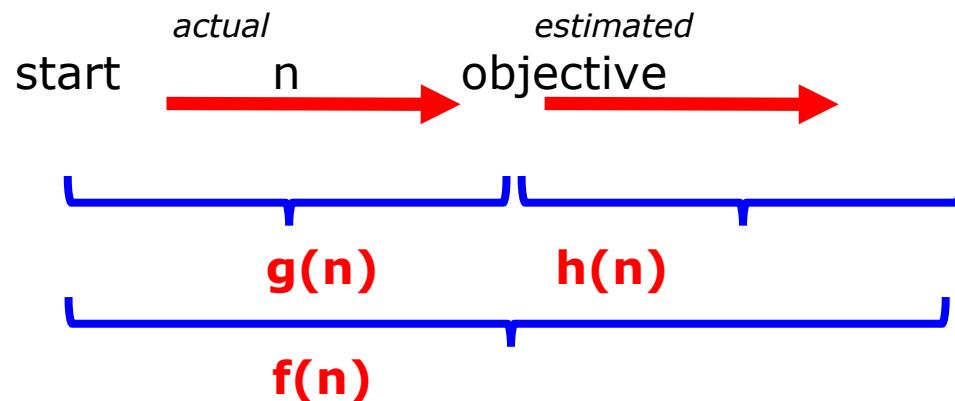
- Guess a number game
- Search in a phone book or in a dictionary



# 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)  $n$
- $f(n) = g(n) + h(n)$





# USS in tree-based structures

## Breadth-first search – BFS

### Basic elements

- All the nodes of depth  $d$  are visited before all the nodes of depth  $d+1$
- All children of current node are added into a FIFO list (queue)

### Example

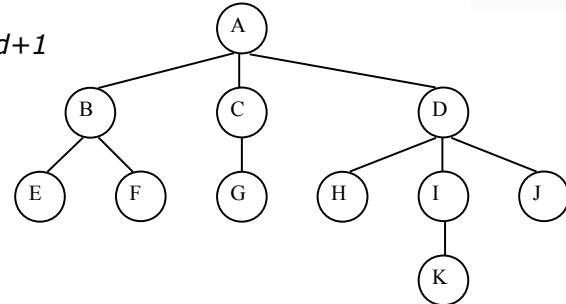
- Visiting order: A, B, C, D, E, F, G, H, I, J, K

### Algorithm

```

bool BFS(elem, list){
    found = false;
    visited = Φ;
    toVisit = {start};      //FIFO list
    while((toVisit != Φ) && (!found)){
        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;
    } //while
    return found;
}

```



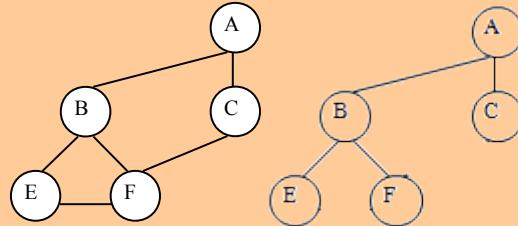
Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	C, D, E, F
A, B, C	D, E, F, G
A, B, C, D	E, F, G, H, I, J
A, B, C, D, E	F, G, H, I, J
A, B, C, D, E, F	G, H, I, J
A, B, C, D, E, F, G	H, I, J
A, B, C, D, E, F, G, H	I, J
A, B, C, D, E, F, G, H, I	J, K
A, B, C, D, E, F, G, H, I, J	K
A, B, C, D, E, F, G, H, I, J, K	Φ

# USS in tree-based structures

## Breadth-first search – BFS



- **Search analyse:**
  - Time complexity:
    - $b$  – ramification factor (number of children of a node)
    - $d$  – length (depth) of solution
    - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
  - Space complexity
    - $S(n) = T(n)$
  - Completeness
    - If solution exists, then BFS finds it
  - Optimality
    - No



- **Advantages**
  - Finds the shortest path to the objective node (the shallowest solution)

- **Disadvantages**
  - Generate and retain a tree whose size exponentially increases (with depth of objective node)
  - Exponential time and space complexity
  - [Russel&Norving experiment](#)
  - Works only for small search spaces

- **Applications**
  - Identification of connex components in a graph
  - Identification of the shortest path in a graph
  - Optimisation in transport networks → [algorithm Ford-Fulkerson](#)
  - Serialization/deserialisation of a binary tree (vs. serialization in a sorted manner) allows efficiently reconstructing of the tree
  - Collection copy (garbage collection) → [algorithm Cheney](#)

Vizitate deja	De vizitat
Φ	B
B	A, E, F
B, A	E, F, C
B, A, E	F, C
B, A, E, F	C
B, A, E, F, C	Φ



# USS in tree-based structures

## Uniform cost search – UCS

### Basic elements

- BFS +special expand procedure (based on the cost of links between nodes)
- All the nodes of depth  $d$  are visited before all the nodes of depth  $d+1$
- All children of current node are added into a **FIFO ordered** list
  - The nodes of minimum cost are firstly expanded
  - When a path to the final state is obtained, it becomes candidate to the optimal solution
- *Branch and bound* algorithm

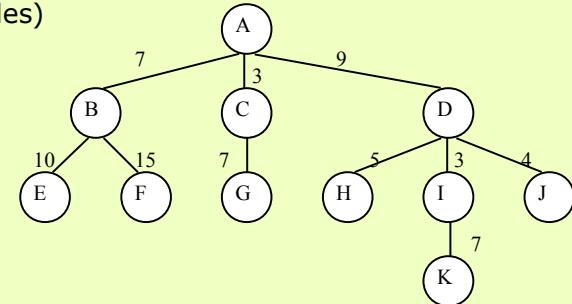
### Example

- Visiting order: A, C, B, D, G, E, F, I, H, J, K

### Algorithm

```

bool UCS(elem, list){
    found = false;
    visited = {};
    toVisit = {start}; //FIFO sorted list
    while((toVisit != {}) && (!found)) {
        node = pop(toVisit);
        visited = visited U {node};
        if (node== elem)
            found = true;
        else
            aux = {};
            for all (unvisited) children of node do{
                aux = aux U {child};
            } // for
            toVisit = toVisit U aux;
            TotalCostSort(toVisit);
    } //while
    return found;
}
  
```



visited	toVisit
$\emptyset$	A
A	C(3), B(7), D(9)
A, C	B(7), D(9), G(3+7)
A, C, B	D(9), G(10), E(7+10), F(7+15)
A, C, B, D	G(10), I(9+3), J(9+4), H(9+5), E(17), F(22)
A, C, B, D, G	I(12), J(13), H(14), E(17), F(22)
A, C, B, D, G, I	J(13), H(14), E(17), F(22), <b>K(9+3+7)</b>
A, C, B, D, G, I, J	H(14), E(17), F(22), <b>K(19)</b>
A, C, B, D, G, I, J, H	E(17), F(22), <b>K(19)</b>
A, C, B, D, G, I, J, H, E	F(22), <b>K(19)</b>
A, C, B, D, G, I, J, H, E, F	K(19)
A, C, B, D, G, I, J, H, E, F, K	$\emptyset$

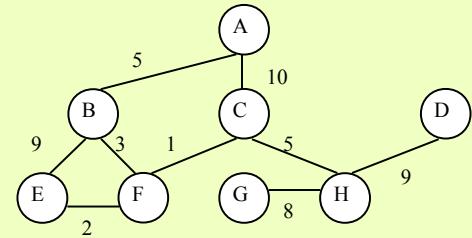


# USS in tree-based structures

## Uniform cost search – UCS

### □ Complexity analyses

- Time complexity
  - $b$  – ramification factor
  - $d$  - length (depth) of solution
  - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Space complexity
  - $S(n) = T(n)$
- Completeness
  - yes – if solutions exists, then UCS finds it
- Optimality
  - Yes



### □ Advantages

- Finding the minimum cost path to the objective node

### □ Disadvantages

- Exponential time and space complexity

### □ Applications

- Shortest path → [Dijkstra algorithm](#)

Vizitate deja	De vizitat
∅	A(0)
A(0)	B(5), C(10)
A(0), B(5)	F(8), C(10), E(14)
A(0), B(5), F(8)	C(9), E(10)
A(0), B(5), F(8), C(9)	E(10), H(14)
A(0), B(5), F(8), C(9), E(10)	H(14)



# USS in tree-based structures

## depth-first search – DFS

### □ Basic elements

- Expand a child and depth search until
  - The final node is reached or
  - The node is a leaf
- Coming back in the most recent node that must be explored
- All the children of the current node are added in a **LIFO** list (**stack**)

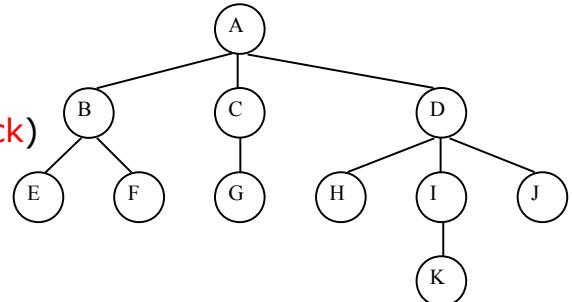
### □ Example

- Visiting order: A, B, E, F, C, G, D, H, I, K, J

### □ Algorithm

```

bool DFS(elem, list){
    found = false;
    visited = {};
    toVisit = {start}; //LIFO list
    while((toVisit != {}) && (!found)){
        node = pop(toVisit);
        visited = visited U {node};
        if (node== elem)
            found = true;
        else{
            aux = {};
            for all (unvisited) children of node do{
                aux = aux U {child};
            }
            toVisit = aux U toVisit;
        }
    } //while
    return found;
}
  
```



Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	K, J
A, B, E, F, C, G, D, H, I, K	J
A, B, E, F, C, G, D, H, I, K, J	Φ

# USS in tree-based structures

## depth-first search – DFS



### Complexity analyse

- Time complexity
  - $b$  - ramification factor
  - $d_{max}$  – maximal length (depth) of explored tree
  - $T(n) = 1 + b + b^2 + \dots + b^{d_{max}} \Rightarrow O(b^{d_{max}})$
- Space complexity
  - $S(n) = b * d_{max}$
- Completeness
  - No → the algorithm does not end for infinite paths (there is no sufficient memory for all the nodes that are visited already)
- Optimality
  - No → depth search can find a longer path than the optimal one

### Advantages

- Finding the shortest path with minimal resources (recursive version)

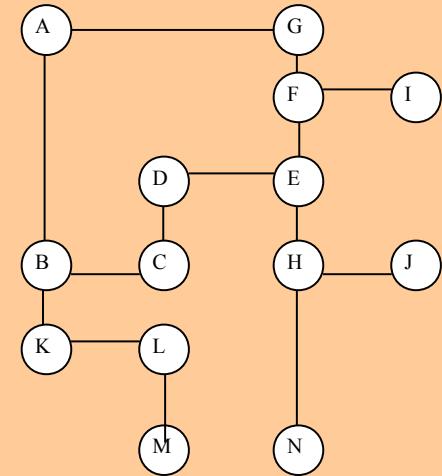
### Disadvantages

- Dead paths
  - Infinite cycles
  - Longer solution than the optimal one

### Applications

- Maze problem
- Identification of connex components
- Topological sorting
- Testing the graph planarity

A				G	
				F	I
		D		E	
B		C		H	J
K		L			
		M		N	





# USS in tree-based structures

## depth-first search – DFS

```

bool DFS_edges(elem, list){
    discovered = Φ;
    back = Φ;
    toDiscover = Φ; //LIFO
    for (all neighbours of start) do
        toDiscover = toDiscover U {(start, neighbour)}
    found = false;
    visited = {start};
    while((toDiscover != Φ) && (!found)){
        edge = pop(toDiscover);
        if (edge.out !∈ visited){
            discovered = discovered U {edge};
            visited = visited U {edge.out}
            if (edge.out == end)
                found = true;
            else{
                aux = Φ;
                for all neighbours of edge.out do{
                    aux = aux U {(edge.out, neighbour)};
                }
                toDiscover = aux U toDiscover;
            }
        }
        else
            back = back U {edge}
    } //while
    return found;
}

```

Muchia	Muchii vizitate deja	Muchii de vizitat	înapoi	Noduri vizitate
	Φ	AB, AF	Φ	A
AB	AB	BC, BK, AF	Φ	A, B
BC	AB, BC	CD, BK, AF	Φ	A, B, C
CD	AB, BC, CD	DE, BK, AF	Φ	A, B, C, D
DE	AB, BC, CD, DE	EF, EH, BK, AF	Φ	A, B, C, D, E
EF	AB, BC, CD, DE, EF	FI, FG, EH, BK, AF	Φ	A, B, C, D, E, F
FI	AB, BC, CD, DE, EF, FI	FG, EH, BK, AF	Φ	A, B, C, D, E, F, I
FG	AB, BC, CD, DE, EF, FI, FG	GA, EH, BK, AF	Φ	A, B, C, D, E, F, I, G
GA	AB, BC, CD, DE, EF, FI, FG	EH, BK, AF	GA	A, B, C, D, E, F, I, G
EH	AB, BC, CD, DE, EF, FI, FG	HJ, HN, BK, AF	GA	A, B, C, D, E, F, I, G, H
HJ	AB, BC, CD, DE, EF, FI, FG, HJ	HN, BK, AF	GA	A, B, C, D, E, F, I, G, H, J
HN	AB, BC, CD, DE, EF, FI, FG, HI, HN	BK, AF	GA	A, B, C, D, E, F, I, G, H, N



# USS in tree-based structures depth-limited search – DLS

## □ Basic elements

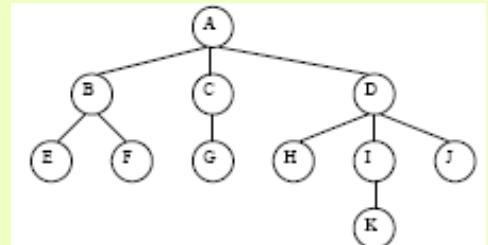
- DFS + maximal depth that limits the search ( $d_{lim}$ )
- Solved the completeness problems of DFS

## □ Example

- $d_{lim} = 2$
- Visiting order: A, B, E, F, C, G, D, H, I, J

## □ Algorithm

```
bool DLS(elem, list, dlim) {
    found = false;
    visited = Φ;
    toVisit = {start}; //LIFO list
    while((toVisit != Φ) && (!found)) {
        node = pop(toVisit);
        visited = visited ∪ {node};
        if (node.depth <= dlim) {
            if (node == elem)
                found = true;
            else{
                aux = Φ;
                for all (unvisited) children of node do{
                    aux = aux ∪ {child};
                }
                toVisit = aux ∪ toVisit;
            } //if found
        } //if dlim
    } //while
    return found;
}
```



Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	J
A, B, E, F, C, G, D, H, I, K, J	Φ



# USS in tree-based structures depth-limited search – DLS

## □ Complexity analyse

- Time complexity:
  - $b$  – ramification factor
  - $d_{lim}$  – limit of length (depth) allowed for the explored tree
  - $T(n) = 1 + b + b^2 + \dots + b^{d_{lim}} \Rightarrow O(b^{d_{lim}})$
- Space complexity
  - $S(n) = b * d_{lim}$
- Completeness
  - Yes, but  $\Leftrightarrow d_{lim} > d$ , where  $d$  = length (path) of optimal solution
- Optimality
  - No  $\rightarrow$  DLS can find a longer path than the optimal one

## □ Advantages

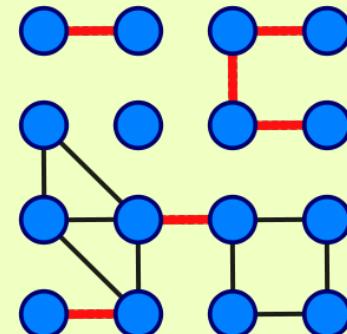
- Solves the completeness problems of DFS

## □ Disadvantages

- How to choose a good limit  $d_{lim}$ ?

## □ Applications

- Identification of bridges in a graph





# USS in tree-based structures

## iterative deepening depth search – IDDS

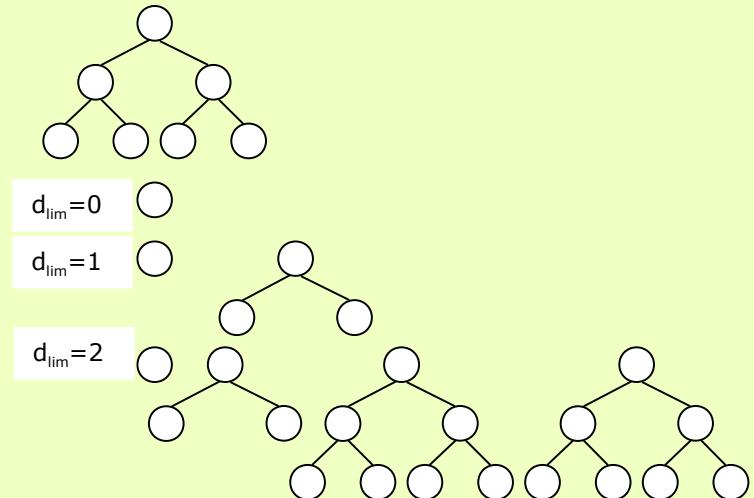
### □ Basic elements

- U DLS( $d_{lim}$ ), where  $d_{lim} = 1, 2, 3, \dots, d_{max}$
- Solves the identification of the optimal limit  $d_{lim}$  from DLS
- Usually, it works when:
  - The search space is large
  - The length (depth) of solution is known

### □ Example

### □ Algorithm

```
bool IDS(elem, list){  
    found = false;  
    dlim = 0;  
    while ((!found) && (dlim < dmax)) {  
        found = DLS(elem, list, dlim);  
        dlim++;  
    }  
    return found;  
}
```





# USS in tree-based structures

## iterative deepening depth search – IDDS

### □ Complexity analysis

- Time complexity:
  - $b^{d_{max}}$  nodes at depth  $d_{max}$  are expanded once =>  $1 * b^{d_{max}}$
  - $b^{d_{max}-1}$  nodes at depth  $d_{max}-1$  are expanded twice =>  $2 * (b^{d_{max}-1})$
  - ...
  - $b$  nodes at depth 1 are expanded  $d_{max}$  times =>  $d_{max} * b^1$
  - 1 node (the root) at depth 0 is expanded  $d_{max}+1$  times =>  $(d_{max}+1)*b^0$
- $$T(n) = \sum_{i=0}^{d_{max}} (i+1)b^{d_{max}-i} \Rightarrow O(b^{d_{max}})$$

- Space complexity
  - $S(n) = b * d_{max}$
- Completeness
  - yes
- Optimality
  - yes

### □ Advantages

- Requires linear memory
- The goal state is obtained by a minimal path
- Faster than BFS and DFS

### □ Disadvantages

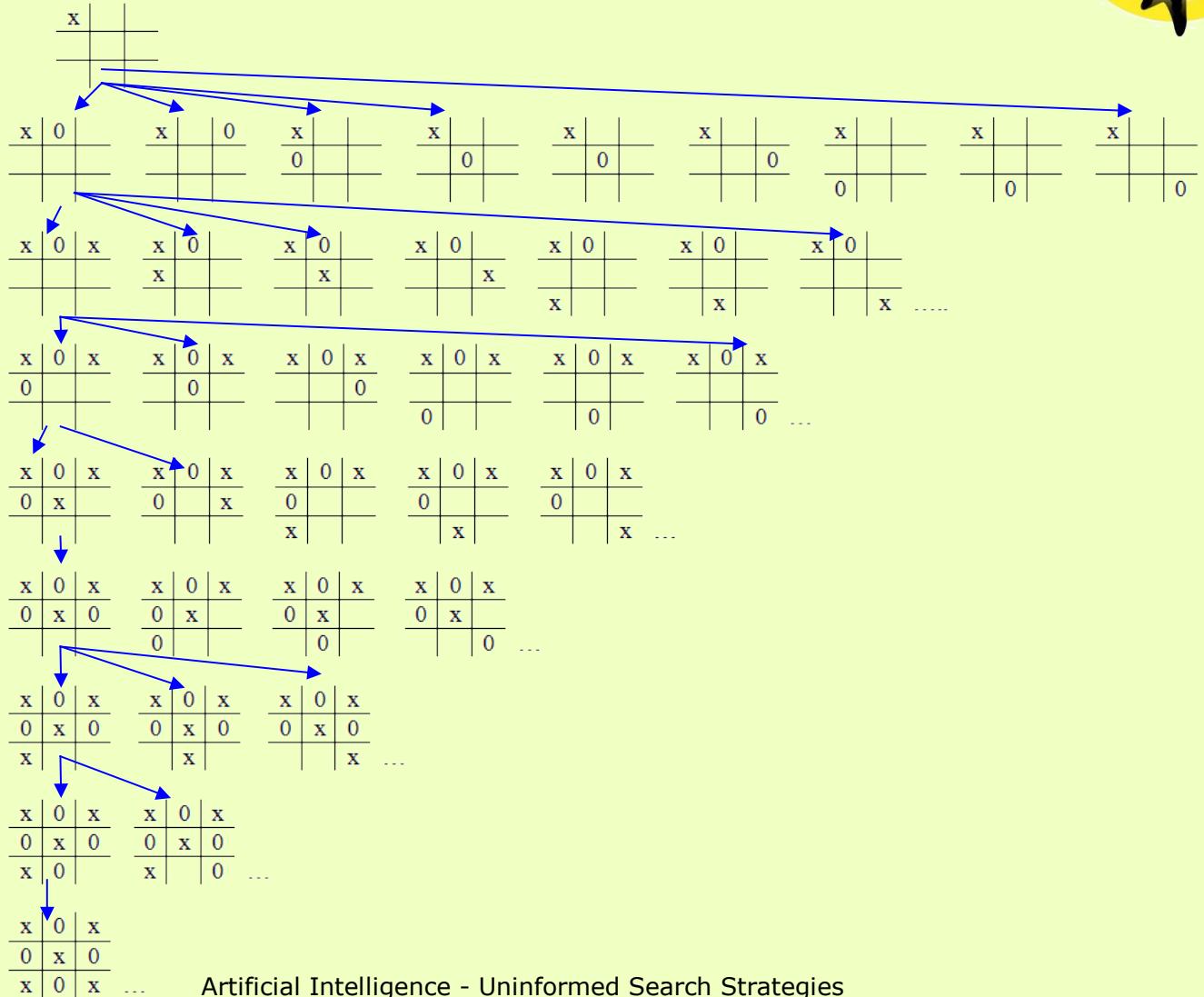
- Requires to know the solution depth

### □ Applications

- Tic tac toe game



# USS in tree-based structures iterative deepening depth search – IDDS



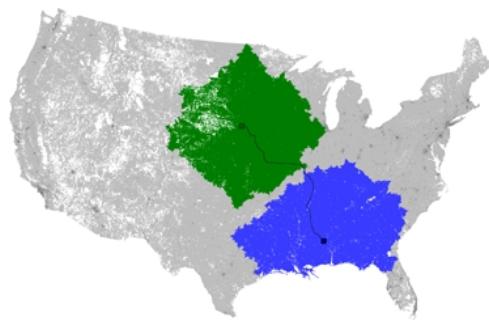


# USS in tree-based structures bi-directional search – BDS

## □ Basic elements

- 2 parallel search strategies
  - *forward*: from root to leaves
  - *backward*: from leaves to root
- that end when they meet
- any SS can be used in a direction
- Requires establishing:
  - the parents and the children of each node
  - the meeting point

## □ Example



## □ Algorithm

- Depend on the SS used



# USS in tree-based structures bi-directional search – BDS

## □ Complexity analyse

- Time complexity
  - $b$  – ramification factor
  - $d$  – solution length (depth)
  - $O(b^{d/2}) + O(b^{d/2}) \Rightarrow O(b^{d/2})$
- Space complexity
  - $S(n) = T(n)$
- Completeness
  - yes
- Optimality
  - yes

## □ Advantages

- Good time and space complexity

## □ Disadvantages

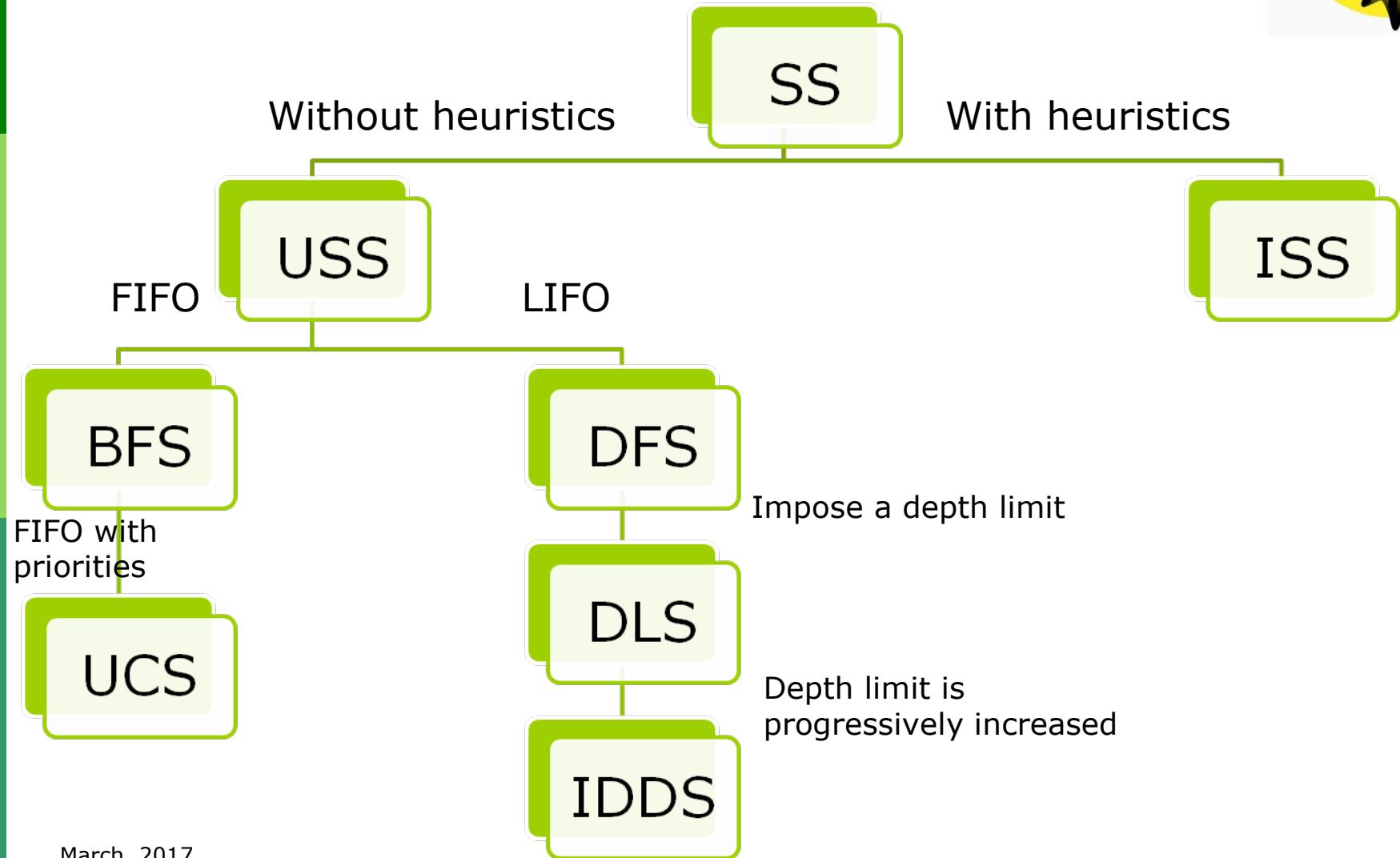
- Each state must be reversed
  - From had to tail
  - From tail to head
- Difficult to implement
- Identification of parents and children for all the nodes
- The final state must be known

## □ Applications

- Partitioning problem
- Shortest path



# USS in tree-based structures





# USS in tree-based structures

Comparison of performances

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	$O(b^{d_{\max}})$	$O(b * d_{\max})$	No	No
DLS	$O(b^{d_{\text{lim}}})$	$O(b * d_{\text{lim}})$	Yes, if $d_{\text{lim}} > d$	No
IDS	$O(b^d)$	$O(b * d)$	Da	Yes
BDS	$O(b^{d/2})$	$O(b^{d/2})$	Yes	Yes

# Next lecture

---

A. Short introduction in Artificial Intelligence (AI)

## A. Solving search problems

A. Definition of search problems

B. Search strategies

A. Uninformed search strategies

B. Informed search strategies

C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)

D. Adversarial search strategies

## C. Intelligent systems

A. Rule-based systems in certain environments

B. Rule-based systems in uncertain environments (Bayes, Fuzzy)

C. Learning systems

A. Decision Trees

B. Artificial Neural Networks

C. Support Vector Machines

• Evolutionary algorithms

D. Hybrid systems

## Next lecture – Useful information

---

- Chapter II.4 of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapters 3 and 4 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter 2.5 of  
<http://www-g.eng.cam.ac.uk/mmg/teaching/artificialintelligence/>

# References

---

- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
  - PhD. Prof. Laura Diosan – [www.cs.ubbcluj.ro/~lauras](http://www.cs.ubbcluj.ro/~lauras)
  - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)

BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics

# ARTIFICIAL INTELLIGENCE



**Solving search problems**

Informed search strategies

Global and local

# Topics

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

- A. Definition of search problems
- B. Search strategies

- A. Uninformed search strategies
- B. Informed search strategies
- C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
- D. Adversarial search strategies

### C. Intelligent systems

- A. Rule-based systems in certain environments
- B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
- C. Learning systems
  - A. Decision Trees
  - B. Artificial Neural Networks
  - C. Support Vector Machines
    - Evolutionary algorithms
- D. Hybrid systems

# Content

---

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

# Useful information

---

- Chapter II.4 of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapters 3 and 4 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter 2.5 of  
<http://www-g.eng.cam.ac.uk/mmg/teaching/artificialintelligence>



# Solving problems by search

---

- Search strategies
  - Topology
  - Available information
    - Uninformed search (blind search)
    - Informed search (heuristic 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

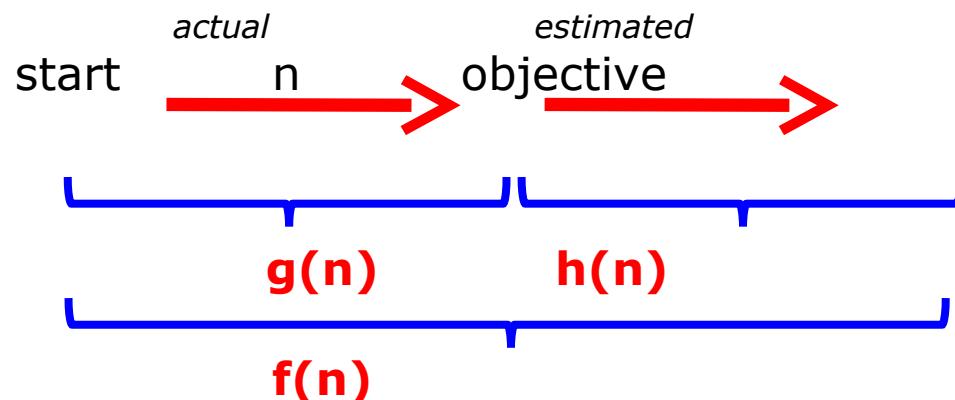
- Global search strategies
  - Best-first search
    - Greedy best-first search
    - A\* + versions of A\*
- Local search strategies
  - Tabu search
  - Hill climbing
  - Simulated annealing



# 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)
    - $f = \text{path cost}$
  - 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 =  $\emptyset$ ;  
    toVisit = {start}; //FIFO sorted list (priority queue)  
    while((toVisit !=  $\emptyset$ ) && (!found)){  
        if (toVisit ==  $\emptyset$ )  
            return false  
        node = pop(toVisit);  
        visited = visited U {node};  
        if (node == elem)  
            found = true;  
        else  
            aux =  $\emptyset$ ;  
        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
  - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- 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

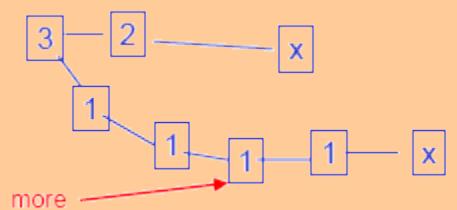
- Specific information helps the search
- Good speed to find the final state

## □ 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

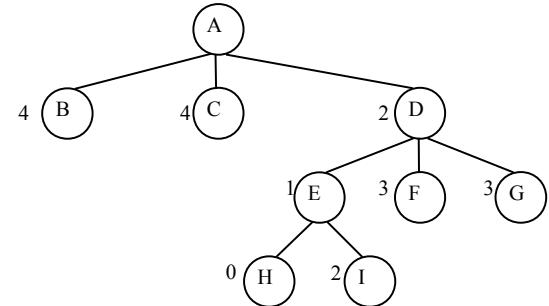
- Evaluation function  $f(n) = h(n)$ 
  - Cost path estimation from the current state to the final one –  $h(n)$
  - cost minimization for the path that must be followed

## □ Example

- A,D,E,H

## □ Algorithm

```
bool BestFS(elem, list){  
    found = false;  
    visited = ∅;  
    toVisit = {start};           //FIFO sorted list (priority queue  
    while((toVisit != ∅) && (!found)){  
        if (toVisit == ∅)  
            return false  
        node = pop(toVisit);  
        visited = visited U {node};  
        if (node == elem)  
            found = true;  
        else  
            aux = ∅;  
        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;  
}
```



Vizitate deja	De vizitat
∅	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	∅



# ISS - Greedy

## □ Complexity analyse

- Time complexity → DFS
  - $b$  – ramification factor
  - $d^{\max}$  – maximal length (depth) of an explored tree
  - $T(n) = 1 + b + b^2 + \dots + b^{d^{\max}} \Rightarrow O(b^{d^{\max}})$
- Space complexity → BFS
  - $d$  - length (depth) of solution
  - $S(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Completeness
  - nu
- Optimality
  - possible

## □ Advantages

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

## □ Disadvantages

- Sum of optimal local decisions  $\neq$  global optimal decision
  - Ex. TSP

## □ Applications

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

# ISS – A\*

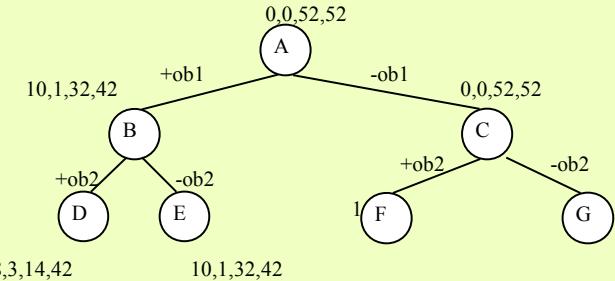


## □ Basic elements

- Combination of 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 through 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  $(o_1, o_2, \dots, o_n)$  each of them having a profit  $p_i$ ,  $i=1,2,\dots,n$ 
  - 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 taking 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
  - $T(n) = 1 + b + b^2 + \dots + b^{d^{\max}} \Rightarrow O(b^{d^{\max}})$
- 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\*

## ❑ Bibliography

- 02/A\_IDA.pdf
- 02/A\_IDA\_2.pdf
- 02/SMA\_RTA.pdf
- 02/Recursive Best-First Search.ppt
- 02/IDS.pdf
- 02/IDA\_MA.pdf
- [http://en.wikipedia.org/wiki/IDA<sup>A</sup>](http://en.wikipedia.org/wiki/IDA%2A)
- [http://en.wikipedia.org/wiki/SMA<sup>A</sup>](http://en.wikipedia.org/wiki/SMA%2A)



# Solving problem by search

- Topology of search strategies:
  - Solution **generation**
    - **Constructive** search
      - Solution is identified step by step
      - Ex. TSP
    - **Perturbative** search
      - A possible solution is modified in order to obtain another possible solution
      - Ex. SAT - Propositional Satisfiability Problem
  - Search space **navigation**
    - **Systematic** search
      - The entire search space is visited
        - Solution identification (if it exists) → complete algorithms
    - **Local** search
      - Moving from a point of the search space into a neighbour point → incomplete algorithms
      - A state can be visited more times
  - **Certain** items of the search
    - **Deterministic** search
      - Algorithms that exactly identify the solution
    - **Stochastic** search
      - Algorithms that approximate the solution
  - Search space **exploration**
    - **Sequential** search
    - **Parallel** search

# Solving problem by search



## Methods

- Step-by-step construction of solution
- Identification of a possible optimal solution
  - Problem's components
    - Conditions (constraints) that solution must satisfy (partially or totally)
    - Evaluation function of a possible solution → optimal solution identification
  - Search space
    - Set of all possible complete solutions
    - State = a possible complete solution
    - Final state = optimal solution
  - Example
    - Queen's problem
      - Possible states: boards with 8 queens
      - Operators: change the column of a queen
      - Search objective: the board without attacks
      - Evaluation function: number of attacks
    - Planning problems
    - Design of digital circuits



# 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



www.shutterstock.com - 36774760

## 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

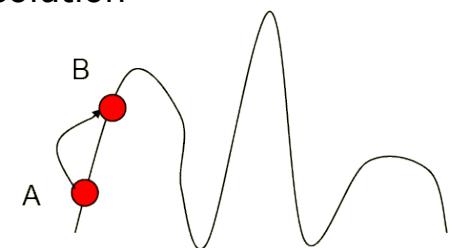
Depends on problem

# 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  $x \in \mathbb{R}^n$  by changing an element of  $x$
  - SA/GD optimises  $f(x)$  with  $x \in \mathbb{R}^n$  by changing all the elements of  $x$

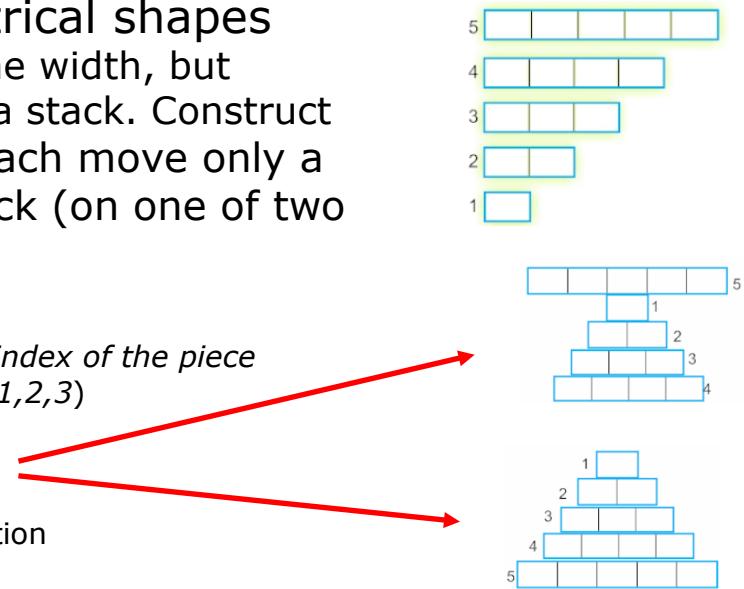


# Local search strategies – Hill climbing (HC)



## 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 ( $i=1,2,\dots,n$ ) 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
    - $f_1 = \#$  of correctly located pieces  $\rightarrow$  maximisation
      - Conform tot the final tour –  $f_1 = n$
    - $f_2 = \#$  of wrongly located pieces  $\rightarrow$  minimisation
      - Conform tot the final tour –  $f_2 = 0$
    - $f = f_1 - f_2 \rightarrow$  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



# Local search strategies – Hill climbing (HC)

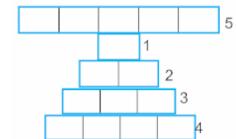


## □ Example

### ■ Iteration 1

#### □ current state = initial state:

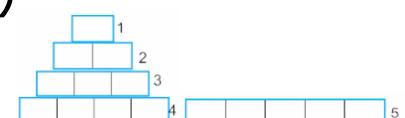
- $x = s_1 = ((5,1), (1,1), (2,1), (3,1), (4,1))$
- Pieces 1, 2 and 3 are correctly located
- Pieces 4 and 5 are wrongly located
- $f(s_1) = 3 - 2 = 1$



#### □ $x^* = x$

#### □ Neighbours of current state $x$ – a single one → piece 5 moves on stack 2 →

- $s_2 = ((1,1), (2,1), (3,1), (4,1), (5,2))$
- $f(s_2) = 4 - 1 = 3 > f(x) \rightarrow x = s_2$



# 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  $\rightarrow s_3 = ((2,1), (3,1), (4,1), (1,2), (5,2)) \rightarrow f(s_3) = 3-2=1 < f(x)$



- Piece 1 moves on stack 3  $\rightarrow s_4 = ((2,1), (3,1), (4,1), (5,2), (1,3)) \rightarrow f(s_4) = 3-2=1 < f(x)$
- There is no neighbour better than  $x \rightarrow$  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
    - $f_1$  = sum of stack's height whose all pieces are correctly located (final tour  $f_1 = 10$ ) → maximisation
    - $f_2$  = sum of stack's height whose pieces are wrongly located (final tour  $f_2=0$ ) → minimisation
    - $f = f_1 - f_2 \rightarrow$  maximisation
  - Neighbourhood
    - Possible moves
      - Move a piece  $i$  from stack  $j_1$  on stack  $j_2$

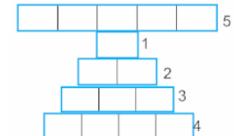
# Local search strategies – Hill climbing (HC)



## □ Example

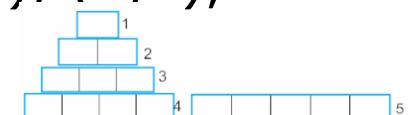
### ■ Iteration 1

- Current state  $x$  = initial state  $s_1 = ((5,1), (1,1), (2,1), (3,1), (4,1))$ 
  - All pieces are wrongly located  $\rightarrow f_1 = 0, f_2 = 3+2 + 1 + 0 + 4 = 10$
  - $f(s_1) = 0 - 10 = -10$



- $x^* = x$

- Neighbours of current state  $x$  – a single one  $\rightarrow$  piece 5 is moved on stack 2  $\rightarrow s_2 = ((1,1), (2,1), (3,1), (4,1), (5,2))$



- $f(s_2) = 0 - (3+2+1+0) = -6 > f(x) \rightarrow x = s_2$

# 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  $\rightarrow s_3 = ((2,1), (3,1), (4,1), (1,2), (5,2)) \rightarrow f(s_3) = 0 - (0+2+3+0) = -5 > f(x)$ A diagram of a 5x5 grid of boxes. The heights of the stacks are labeled: stack 1 is 5, stack 2 is 3, stack 3 is 4, stack 4 is 2, and stack 5 is 1. The boxes are represented by small squares in the grid.  
$$f(s_3) = 0 - (0+2+3+0) = -5 > f(x)$$
  - Piece 1 is moved 3  $\rightarrow s_4 = ((2,1), (3,1), (4,1), (5,2), (1,3)) \rightarrow f(s_4) = 0 - (1+2+1) = -4 > f(x)$ A diagram of a 5x5 grid of boxes. The heights of the stacks are labeled: stack 1 is 5, stack 2 is 3, stack 3 is 4, stack 4 is 2, and stack 5 is 1. The boxes are represented by small squares in the grid.  
$$f(s_4) = 0 - (1+2+1) = -4 > f(x)$$
- Best neighbour of  $x$  is  $s_4 \rightarrow x = s_4$

### Iteration 3

- ...

### Local optima are avoided

# Strategii de căutare locală – Hill climbing (HC)



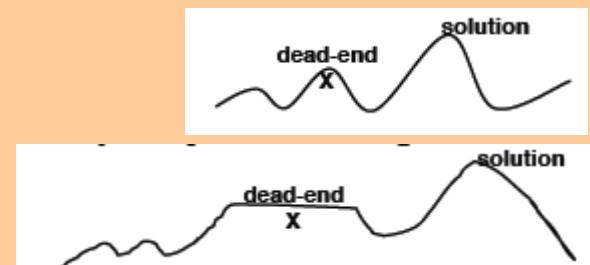
## □ Algorithm

```
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



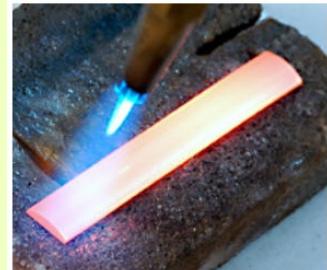
# Local search strategies – Hill climbing (HC)



## ❑ Versions

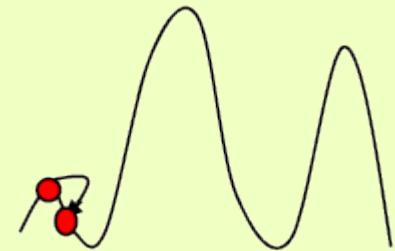
- Stochastic HC
  - ❑ The next state is randomly selected
- First-choice HC
  - ❑ Randomly generation of successors until a new one is identified
- Random-restart HC → *beam local search*
  - ❑ Restart the search from a randomly initial state when the search does not advance

# 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
  - $T = 0 \rightarrow$  hill climbing
  - $T \rightarrow \infty \rightarrow$  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

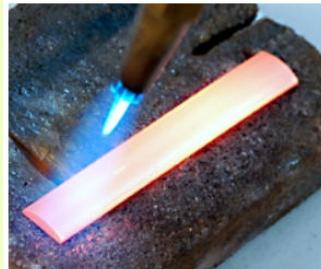


# Local search strategies – Simulated Annealing



## □ 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  $x = (x_1, x_2, \dots, x_n)$ , 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  $\rightarrow$  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  $x \rightarrow$  queen of line 5 is swapped by queen of line 7

$\rightarrow s_2 = (8, 7, 3, 1, 6, 5, 2, 4)$

- $f(s_2) = 1+1+1=3 > f(x)$

- $\Delta E = f(s_2) - f(s_1) = 1$

- $P(\Delta E) = e^{-1/100}$

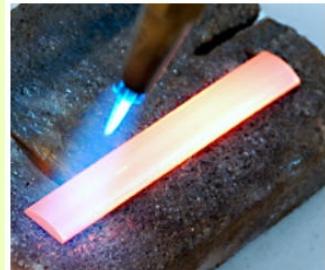
- $r = \text{random}(0,1)$

- $\text{if } r < P(\Delta E) \rightarrow x = s_2$

	a	b	c	d	e	f	g	h	
1	Q				Q				1
2					Q				2
3	Q								3
4			Q						4
5		Q							5
6				Q					6
7					Q				7
8	Q								8
	a	b	c	d	e	f	g	h	

	a	b	c	d	e	f	g	h	
1	Q				Q				1
2					Q				2
3			Q						3
4				Q					4
5					Q				5
6					Q				6
7		Q							7
8					Q				8
	a	b	c	d	e	f	g	h	

# 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( $\Delta E$ ) 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 than  $s$  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

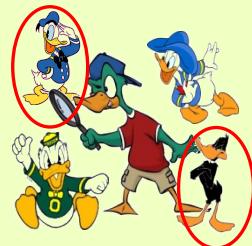
- Statistic-based algorithm → it is able to identified the optimal solution, but it requires many iterations
- Easy to implement
- Generally, if find a good (global) solution
- Can solve complex problems (with noise and many constraints)

## □ 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

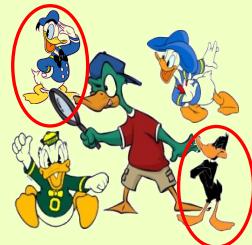
- Combinatorial optimisation problems → knapsack problem
- Design problems → digital circuits design
- Planning problems → production planning, tenis game planning



# 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 unvisited 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

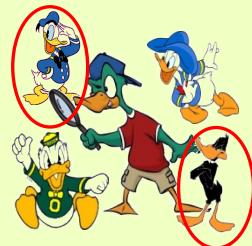
- State  $x$  – vector of  $n$  integers  $x = (x_1, x_2, \dots, x_n)$  with  $x_i \in \{0, 1, 2, \dots, b_i\}$
- Initial state – randomly

### ■ State evaluation

- $f_1 = S$  – total value of selected coins → minimisation
  - If the total value of coins  $> S$  → penalisation (eg. 500 units)
- $f_2 = \text{number of selected coins}$  → maximisation
- $f = f_1 - f_2 \rightarrow \text{minimisation}$

### ■ neighbourhood

- Possible moves
  - Including in the sum of  $j$  copies of coin  $i$  (plus,)
  - Eliminating from the sum of  $j$  copies of coin  $i$  (minus,)
- Tabu list retains performed moves of an iteration
  - - move = the added/eliminated coin



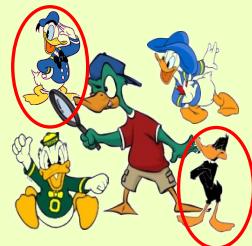
# Local search strategies – Tabu search

## Example

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

<b><math>S=500</math></b>	<b><math>m_1</math></b>	<b><math>m_2</math></b>	<b><math>m_3</math></b>	<b><math>m_4</math></b>	<b><math>m_5</math></b>	<b><math>m_6</math></b>	<b><math>m_7</math></b>
$v_i$	10	50	15	20	100	35	5
$b_i$	5	2	6	5	5	3	10

<b>Stare curentă</b>	<b>Val. f</b>	<b>Listă tabu</b>	<b>Stări vecine</b>	<b>Mutări</b>	<b>Val. f</b>
2 0 1 0 0 2 1	384	$\emptyset$	2 0 1 3 0 2 1	plus <sub>4,3</sub>	321
			2 0 1 0 0 3 1	plus <sub>6,1</sub>	348
			0 0 1 0 0 2 1	minus <sub>1,2</sub>	406
2 0 1 3 0 2 1	321	plus <sub>4,3</sub>	2 0 1 3 5 2 1	plus <sub>5,5</sub>	316
			2 0 1 1 0 2 1	minus <sub>4,2</sub>	363
			2 1 1 3 0 2 1	plus <sub>2,1</sub>	270
2 1 1 3 0 2 1	270	plus <sub>4,3</sub> plus <sub>2,1</sub>	...		



# 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	$\emptyset$	1 0 1 4 0 2 1	minus <sub>1,1</sub> , plus <sub>4,4</sub>	311
			2 0 4 0 1 2 1	plus <sub>3,3</sub> , minus <sub>5,1</sub>	235
			2 0 1 0 4 2 6	plus <sub>5,4</sub> , plus <sub>7,5</sub>	450
2 0 4 0 1 2 1	235	plus <sub>3,3</sub> , minus <sub>5,1</sub>	2 0 5 0 5 2 1	plus <sub>3,1</sub> , plus <sub>5,4</sub>	315
			5 0 4 0 4 2 1	plus <sub>1,3</sub> , plus <sub>5,3</sub>	399
			2 2 4 0 5 2 1	plus <sub>2,2</sub> , plus <sub>5,4</sub>	739
2 0 4 0 1 2 1	235	plus <sub>3,3</sub> , minus <sub>5,1</sub>	...		

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

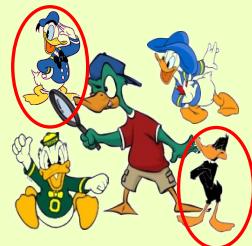


# Local search strategies – Tabu search

## □ 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 easily 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 three-dimensional 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
    - incomplete
    - not 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

	<b>Nex state selection</b>	<b>Acceptance criteria</b>	<b>Convergence</b>
HC	Best neighbour	Neighbour is better than current state	Local or global optima
SA	Random neighbour	Neighbour is better than current state or neighbour is weaker than current state (probabilistic acceptance)	Global optima (slowly)
TS	Best un-visited neighbour	Neighbour is better than current state	Global optima (quickly)

# Next lecture

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

- A. Definition of search problems
- B. Search strategies

- A. Uninformed search strategies
- B. Informed search strategies
- C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
- D. Adversarial search strategies

### C. Intelligent systems

- A. Rule-based systems in certain environments
- B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
- C. Learning systems
  - A. Decision Trees
  - B. Artificial Neural Networks
  - C. Support Vector Machines
  - Evolutionary algorithms
- D. Hybrid systems

## Next lecture – Useful information

---

- Chapter 14 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- Chapter 7.6 of *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapter 9 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

- 
- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean - [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)

BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics

# ARTIFICIAL INTELLIGENCE



**Solving search problems**

Informed local search strategies

Evolutionary Algorithms

# Topics

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

- A. Definition of search problems
- B. Search strategies

- A. Uninformed search strategies
- B. Informed search strategies
- C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
- D. Adversarial search strategies

### C. Intelligent systems

- A. Rule-based systems in certain environments
- B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
- C. Learning systems
  - A. Decision Trees
  - B. Artificial Neural Networks
  - C. Support Vector Machines
  - Evolutionary algorithms
- D. Hybrid systems

# Content

---

- ❑ Solving problems by search
  - Informed search strategies
    - ❑ Local strategies
      - Evolutionary algorithms

# Useful information

---

- Chapter 14 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- Chapter 7.6 of *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapter 9 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

# Local search

---

## □ Typology

- Simple local search – a single neighbour state is retained
  - Hill climbing → selects the best neighbour
  - Simulated annealing → selects probabilistic the best neighbour
  - Tabu search → retains the list of visited solutions
- Beam local search – more states are retained (a population of states)
  - Evolutionary algorithms
  - Particle swarm optimisation
  - Ant colony optmisation

# Nature-inspired search

---

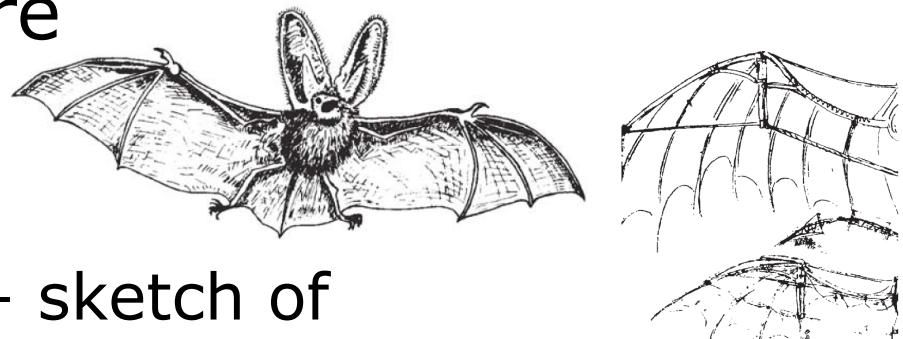
- Best method for solving a problem
  - Human brain
    - Has created the wheel, car, town, etc.
  - Mechanism of evolution
    - Has created the human brain
- Simulation of nature
  - By machines' help → the artificial neural networks simulate the brain
    - Flying vehicles, DNA computers, membrane-based computers
  - By algorithms' help
    - Evolutionary algorithms simulate the evolution of nature
    - Particle Swarm Optimisation simulates the collective and social behaviour
    - Ant Colony Optimisation

# Evolutionary Algorithms (EAs) – Basic elements

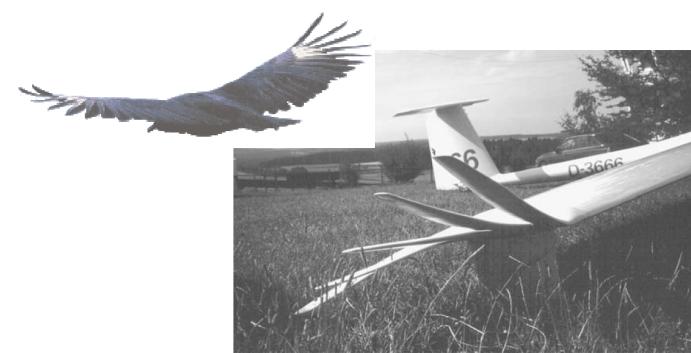
---

## □ Simulation of nature

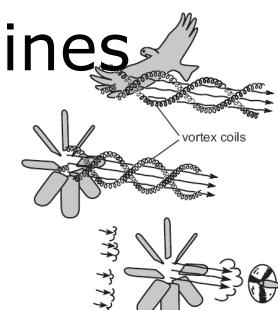
- Fly of bats
- Leonardo da Vinci – sketch of a flying machine



- Flies of birds and planes



- Flies of birds and wind-turbines



# EAs – basic elements

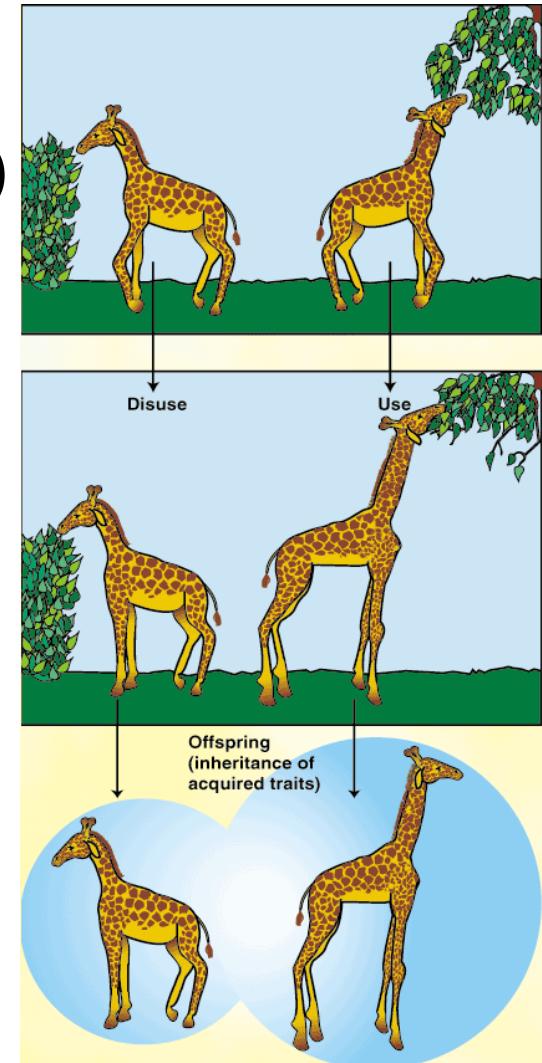
---

- Main characteristics of EAs
  - Iterative and parallel processes
  - Based on random search
  - Bio-inspired – involve mechanisms as:
    - Natural selection
    - Reproduction
    - Recombination
    - Mutation

# EAs – basic elements

## Historical points

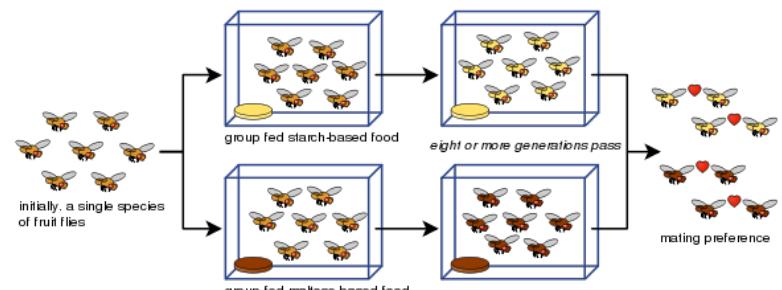
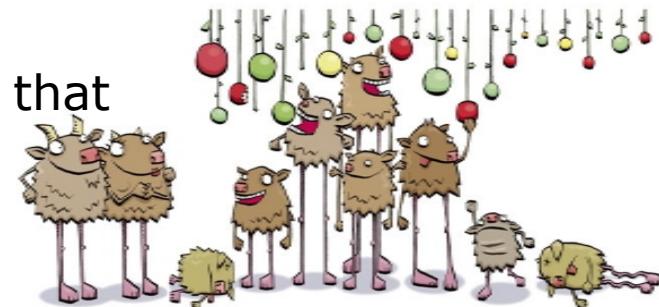
- ❑ Jean Baptise de Lamark (1744-1829)
  - Has proposed in 1809 an explanation for origin of species in the book *Zoological Philosophy*:
    - ❑ Needs of an organism determine the evolving characteristics
    - ❑ Useful characteristics could be transferred to offspring
  - *use and disuse law*



# EAs – basic elements

## Historical points

- ▣ Charles Darwin (1807-1882)
  - In the book *Origin of Species* he proved that all the organisms have evolved based on:
    - ▣ Variation
      - Overproduction of offspring
    - ▣ Natural selection
      - Competition (generation of constant size)
      - Fitness survival
    - Reproduction
    - Occurrence of new species



# EAs – basic elements

---

## Historical points

- Modern theory of evolution
  - Darwin's theory is improved by mechanism of genetic inheritance
  - Genetic variance is produced by
    - Mutation and
    - Sexual recombination
- L. Fogel 1962 (San Diego, CA)– *Evolutionary Programming (EP)*
- J. Holland 1962 (Ann Arbor, MI) → *Genetic Algorithms (GAs)*
- I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany) → *Evolution Strategies (ESs)*
- J. Koza 1989 (Palo Alto, CA) → *Genetic Programming (GP)*

# EAs – basic elements

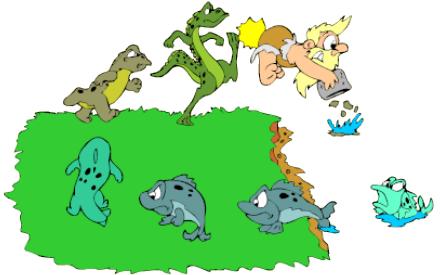
---

## ❑ Evolutionary metaphor

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

# EAs - algorithm

---



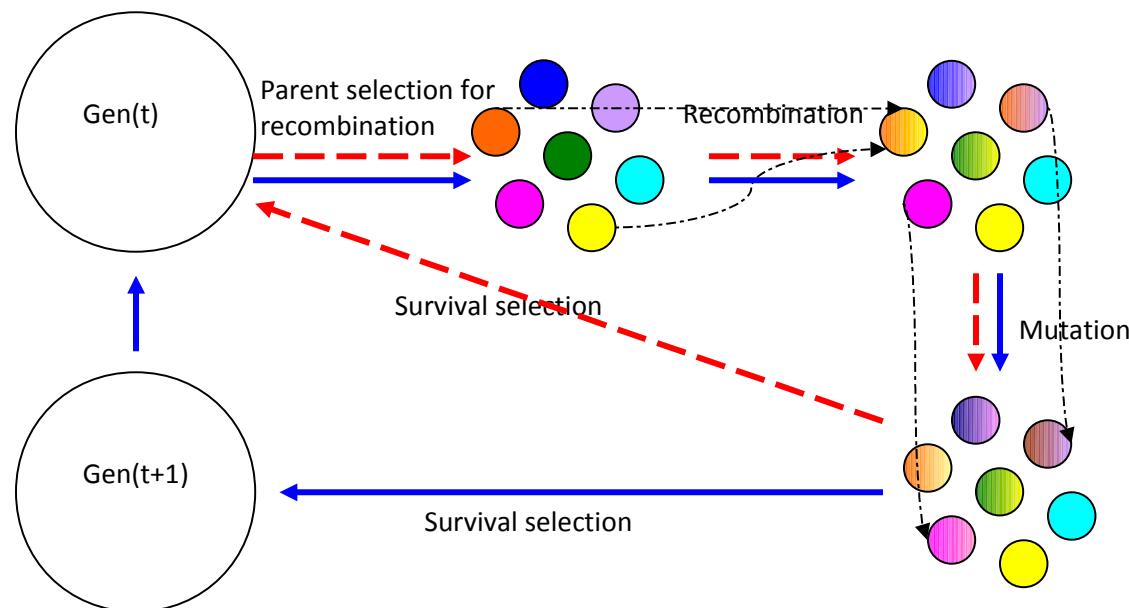
- General sketch
- Design

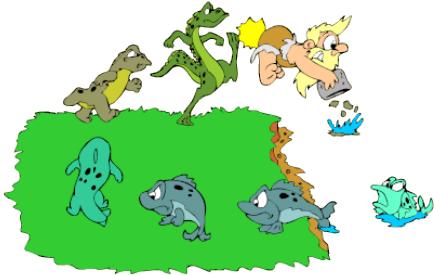


# EAs - algorithm

## □ General sketch of an EA

- Generational →
- Steady-state →





# EAs - algorithm

- Design
  - Chromosome representation
  - Population model
  - Fitness function
  - Genetic operators
    - Selection
    - Mutation
    - Crossover
  - Stop condition

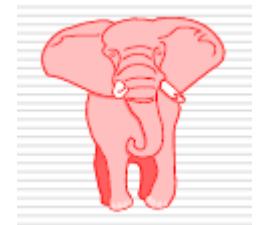
# EAs - algorithm

## Design – representation

---

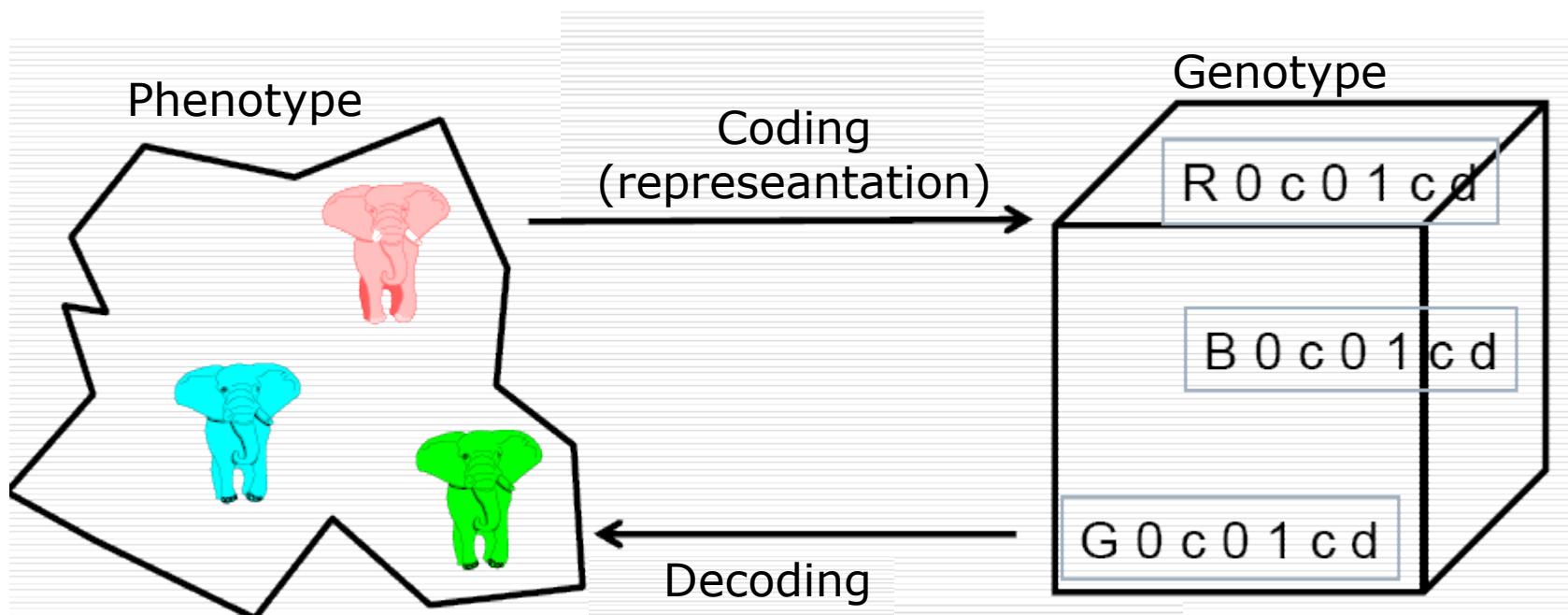
- 2 levels of each possible solution

- External level → phenotype
  - Individual – original object in the context of the problem
  - The possible solutions are evaluated here
  - Ant, knapsack, elephant, towns, ...
- Internal level → genotype
  - Chromosome – code associated to an object
    - Composed by genes, located in loci (fix positions) and having some values (alleles)
    - The possible solutions are searched here
    - One-dimensional vector (with numbers, bits, characters), matrix, ...



# EAs - algorithm design – representation

- Representation must be representative for:
  - Problem
  - Fitness function and
  - Genetic operators



# EAs - algorithm

## Design - representation

---

### Typology of chromosome's representation

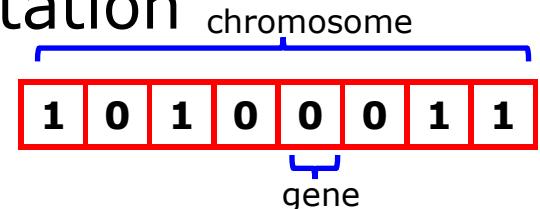
- Linear
  - Discrete
    - Binary → knapsack problem
    - Not-binary
      - Integers
        - Random → image processing
        - Permutation → travelling salesman problem (TSP)
      - Class-based → map colouring problem
    - Continuous (real) → function optimization
  - Tree-based → regression problems

# EAs - algorithm

## Design - representation

---

- Linear discrete and binary representation
  - Genotype
    - Bit-strings



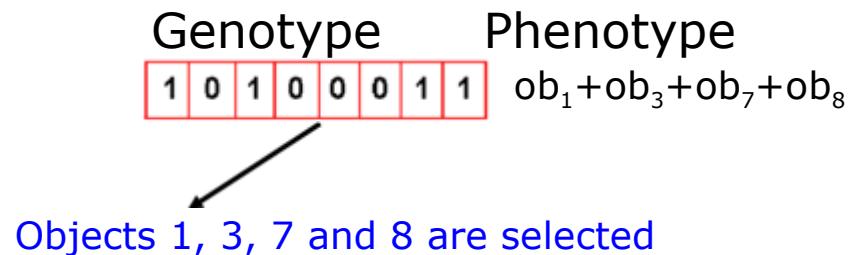
# EAs - algorithm

## Design - representation

---

### □ Linear discrete and binary representation

- Genotype
  - Bit-strings
- Phenotype
  - Boolean elements
    - Eg. Knapsack problem – selected objects for the bag



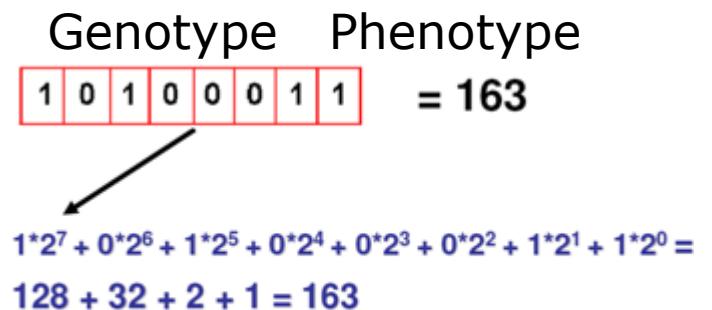
# EAs - algorithm

## Design - representation

---

- Linear discrete and binary representation
  - Genotype
    - Bit-strings
  - Phenotype
    - Boolean elements
      - Eg. Knapsack problem – selected objects for the bag

- Integers



# EAs - algorithm

## Design - representation

---

- Linear discrete and binary representation
  - Genotype
    - Bit-strings
  - Phenotype
    - Boolean elements
      - Example: Knapsack problem – selected objects for the bag
    - Integers
    - Real numbers from a range  
(ex. [2.5, 20.5])

Genotype	Phenotype																
<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td style="width: 10px;"></td><td style="width: 10px;"></td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>									1	0	1	0	0	0	1	1	$= 13.9609$
1	0	1	0	0	0	1	1										

$x = 2.5 + \frac{163}{256} (20.5 - 2.5) = 13.9609$

# EAs - algorithm Design - representation

---

Transformation of real values from binary representation

- Let be  $z \in [x,y] \subseteq \mathcal{R}$  represented as  $\{a_1, \dots, a_L\} \in \{0,1\}^L$
- Function  $[x,y] \rightarrow \{0,1\}^L$  must be inverse (a phenotype corresponds to a genotype)
- Function  $\Gamma: \{0,1\}^L \rightarrow [x,y]$  defines the representation
$$\Gamma(a_1, \dots, a_L) = x + \frac{y-x}{2^L-1} \cdot \left( \sum_{j=0}^{L-1} a_{L-j} \cdot 2^j \right) \in [x,y]$$
- Remarks
  - $2^L$  values can be represented
  - L – maximum precision of solution
  - For a better precision → long chromosomes → slowly evolution

# EAs - algorithm

## Design - representation

---

- Linear discrete non-binary integer random representation
  - Genotype
    - Vector of integers from a given range
  - Phenotype
    - Utility of numbers in the problem
  - Example: Pay a sum  $S$  by using different  $n$  coins
    - Genotype → vector of  $n$  integers from range  $[0, S/\text{value of current coin}]$
    - Phenotype → how many coins of each type must be considered

# EAs - algorithm

## Design - representation

---

- ❑ Linear discrete non-binary integer permutation representation
  - Genotype
    - Permutation of  $n$  elements ( $n$  – number of genes)
  - Phenotype
    - Utility of permutation in problem
- Example Traveling Salesman Problem
  - Genotype → permutation of  $n$  elements
  - Phenotype → visiting order of towns (each town has associated a number from  $\{1,2,\dots,n\}$ )

# EAs - algorithm

## Design - representation

---

- ❑ Linear discrete non-binary integer class-based representation
  - Similarly to integer one, but labels are used instead numbers
  - Genotype
    - ❑ Vector of labels from a given set
  - Phenotype
    - ❑ Labels' meaning
  - Example Map colouring problem
    - ❑ Genotype → vector of  $n$  colours ( $n$  – number of countries)
    - ❑ Phenotype → what colour has to be used for each country

# EAs - algorithm

## Design - representation

---

- Linear continuous (real) representation
  - Genotype
    - Vector of real numbers
  - Phenotype
    - Number meaning
  - Example Function optimisation  $f:R^n \rightarrow R$ 
    - Genotype  $\rightarrow$  more real numbers  $X=[x_1, x_2, \dots, x_n], x_i \in R$
    - Phenotype  $\rightarrow$  values of function f arguments

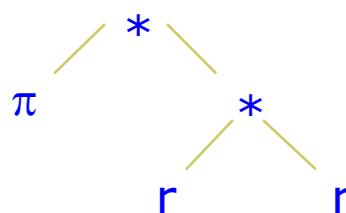
# EAs - algorithm

## Design - representation

---

- Tree-based representation
  - Genotype
    - Trees than encode S-expressions
    - Internal nodes → functions (F)
      - Mathematical
        - Arithmetic operators
        - Boolean operators
      - Statements
        - Of a given programming language
        - Of other language type
    - Leaf → terminals (T)
      - Real or Boolean values, constants or variables
      - Sub-programs
  - Phenotype
    - Meaning of S-expressions
  - Example Computing the circle area

$$\pi * r^2$$



# EAs - algorithm

## Design – creation of population

---

### □ Population – concept

- Aim

- To keep a collection of possible solutions (candidate solutions)
    - Repetitions are allowed

- Properties

- (usually) fixed dimension  $\mu$
  - Diversity
    - Number of different fitnesses/phenotypes/genotypes

- Remarks

- Represents the basic unit that evolves
    - The entire population evolves, not only the individuals!!!

# EAs - algorithm

## Design – creation of population

---

- Population - initialisation
  - Uniformly distributed in the search space (if it is possible)
    - Binary strings
      - Randomly generation of 0 and 1 with a 0.5 probability (fifty-fifth)
    - Arrays of real numbers uniformly generated (in a given range)
    - Permutations
      - Generation of identical permutation and making some changes

# EAs - algorithm

## Design – creation of population

---

- Population - initialisation
  - Uniformly distributed in the search space (if it is possible)
    - Trees
      - *Full* method – complete trees
        - Nodes of depth  $d < D_{\max}$  are randomly initialised by a function from function set  $F$
        - nodes of depth  $d = D_{\max}$  are randomly initialised by a terminal from the terminal set  $T$
      - *Grow* method – incomplete trees
        - Nodes of depth  $d < D_{\max}$  are randomly initialised by an element from  $F \cup T$
        - nodes of depth  $d = D_{\max}$  are randomly initialised by a terminal from the terminal set  $T$
      - *Ramped half and half* method
        - $\frac{1}{2}$  of population is initialised by *Full* methods
        - $\frac{1}{2}$  of population is initialised by *Grow* methods
        - By using different depths

# EAs - algorithm

## Design – creation of population

---

- Population model:
  - Generational EA
    - Each generation creates  $\mu$  offspring
    - Each individual survives a generation only
    - Set of parents is totally replaced by set of offspring
  - Steady-state EA
    - Each generation creates a single offspring
    - A single parent (the worst one) is replaced by the offspring
- Generation Gap
  - Proportion of replaced population
  - $1 = \mu/\mu$ , for generational model
  - $1/\mu$ , for steady-state model

# EAs - algorithm

## Design – fitness function

---

### □ Aim

- Reflects the adaptation to environment
- Quality function or objective function
- Associates a value to each candidate solution
  - Consequences over selection → the more different values, the better

### □ Properties

- Costly stage
  - Un-changed individuals could not be re-evaluated

### □ Typology:

- Number of objectives
  - One-objective
  - Multi-objective → Pareto fronts
- Optimisation direction
  - Maximisation
  - Minimisation
- Degree of precision
  - Deterministic
  - Heuristic

# EAs - algorithm

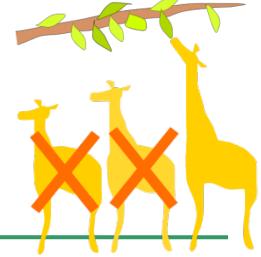
## Design – fitness function

---

- Examples
  - Knapsack problem
    - Representation → linear, discrete and binary
    - Fitness →  $\text{abs}(\text{knapsack's capacity} - \text{weight of selected objects}) \rightarrow \min$
  - Problem of paying sum  $s$  by using different coins
    - Representation → linear, discrete and integer
    - Fitness →  $\text{abs}(\text{sum to be paid} - \text{sum of selected coins}) \rightarrow \min$
  - TSP
    - Representation → linear, discrete, integer, permutation
    - Fitness → cost of path → min
  - Numerical function optimization
    - Representation → linear, continuous, real
    - Fitness → value of function → min/max
  - Computing the circle's area
    - Representation → tree-based
    - Fitness → sum of square errors (difference between the real value and the computed value for a given set of examples) → min

# EAs - algorithm Design – selection

---



- Aim:
  - Gives more reproduction/survival chances to better individuals
    - Weaker individuals have chances also because they could contain useful genetic material
  - Orients the population to improve its quality
  
- Properties
  - Works at population-level
  - Is based on fitness only (is independent to representation)
  - Helps to escape from local optima (because its stochastic nature)



# EAs - algorithm

## Design – selection

---

### □ Typology

#### ■ Aim

- Parent selection (from current generation) for reproduction
- Survival selection (from parents and offspring) for next generation

#### ■ Winner strategy

- Deterministic – the best wins
- Stochastic – the best has more chances to win

#### ■ Mechanism

##### □ Selection for recombination

- Proportional selection (based on fitness) } Based on entire population
- Rank-based selection
- Tournament selection -----> Based on a part of population

##### □ Survival selection

- Age-based selection
- Fitness-based selection

# EAs - algorithm

## Design – recombination selection



### □ Proportional selection (fitness-based selection) - PS

#### □ Main idea

- Roulette algorithm for entire population
- Estimation of the copies # of an individual (selection pressure)

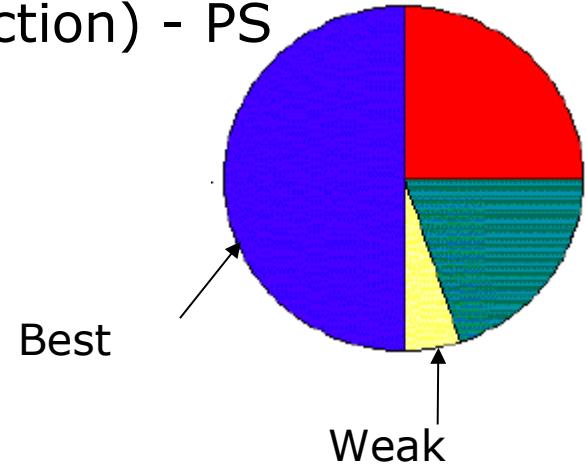
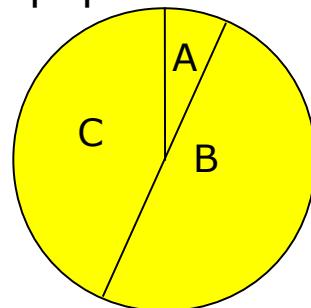
$$E(n_i) = \mu \frac{f(i)}{\langle f \rangle}, \text{ where:}$$

- $\mu$  = size of population,
- $f(i)$  = fitness of individual  $i$ ,
- $\langle f \rangle$  = mean fitness of population

#### □ Better individuals

- Have more space on roulette
- Have more chances to be selected

#### □ Ex. A population of $\mu = 3$ individuals



	$f(i)$	$P_{selPS}(i)$
A	1	1/10=0.1
B	5	5/10=0.5
C	4	4/10=0.4
Suma	10	1

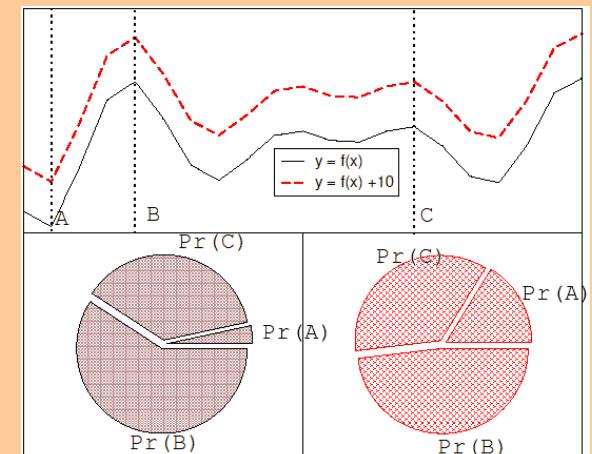


# EAs - algorithm

## Design – recombination selection

### Proportional selection (fitness-based selection) – PS

- Advantages
- Simple algorithm
- Disadvantages
- Premature convergence
  - Best chromosomes predispose to dominate the population
- Low selection pressure when fitness functions are very similar (at the end of a run)
- Real results are different to theoretical probabilistic distribution
- Works at the entire population level
- Solutions
- Fitness scaling
  - Windowing
    - $f'(i) = f(i) - \beta^t$ , where  $\beta$  is a parameter that depends on evolution history
      - e.g.  $\beta$  is the fitness of the weakest individual of current population (the  $t^{\text{th}}$  generation)
  - Sigma scaling (Goldberg type)
    - $f'(i) = \max\{f(i) - (\langle f \rangle - c * \sigma_f), 0.0\}$ , where:
      - $C$  – a constant (usually, 2)
      - $\langle f \rangle$  – average fitness of population
      - $\sigma_f$  – standard deviation of population fitness
  - Normalisation
    - Starts by absolute (initial) fitnesses
    - Standardize these fitnesses such as the fitnesses:
      - Belong to  $[0,1]$
      - Best fitness is the smallest one (equal to 0)
      - Sum of them is 1
- Another selection mechanism





# EAs - algorithm

## Design – selection for recombination

### ❑ Ranking selection – RS

#### ■ Main idea

- ❑ Sort the entire population based on fitness
  - Increases the algorithm complexity, but it is negligible related to the fitness evaluation
- ❑ Each individual receives a rank
- ❑ Computes the selection probabilities based on these ranks
  - Best individual has rank  $\mu$
  - Worst individual has rank 1
- ❑ Tries to solve the problems of proportional selection by using relative fitness (instead of absolute fitness)



# EAs - algorithm

## Design – selection for recombination

### □ Ranking selection - RS

#### ■ Ranking procedures

□ Linear (LR)       $P_{lin\_rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$

- $s$  – selection pressure
  - Measures the advantages of the best individual
  - $1.0 < s \leq 2.0$
  - In the generational algorithm  $s$  represents the copies number of an individual
- Eg. For a population of  $\mu = 3$  individuals

	$f(i)$	$P_{selPS}(i)$	Rank	$P_{selLR}(i)$ for $s=2$	$P_{selRL}(i)$ for $s=1.5$
A	1	$1/10=0.1$	1	0.33	0.33
B	5	$5/10=0.5$	3	1.00	0.33
C	4	$4/10=0.4$	2	0.67	0.33
Sum	10	1			

□ Exponential (ER)       $P_{exp\_rank}(i) = \frac{1-e^{-i}}{c}$

- Best individual can have more than 2 copies
- C – normalisation factor
  - Depends on the population size ( $\mu$ )
  - Must be chosen such that the sum of selection probabilities is 1



# EAs - algorithm

## Design – selection for recombination

---

### ❑ Ranking selection - RS

- Advantages

- ❑ Keep the selection pressure constant

- Disadvantages

- ❑ Works with the entire population

- Solutions

- ❑ Another selection procedure



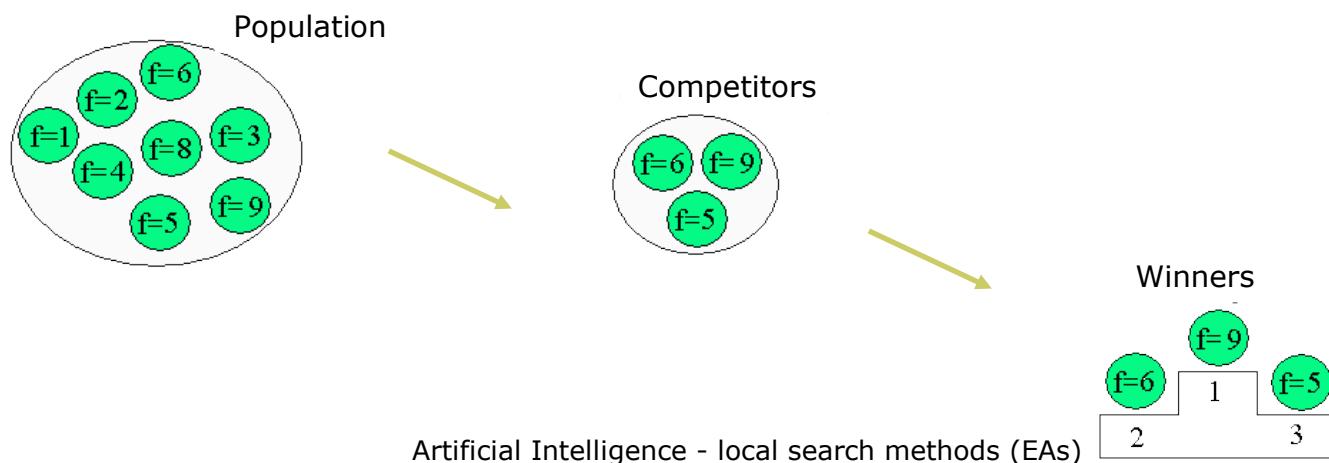
# EAs - algorithm

## Design – selection for recombination

### □ Tournament selection

#### ■ Main idea

- Chooses  $k$  individuals → sample of  $k$  individuals ( $k$  – tournament size)
- Selects the best individual of the sample
- Probability of sample selection depends on
  - Rank of individual
  - Sample size ( $k$ )
    - The larger  $k$  is, the greater selection pressure is
  - Choosing manner – with replacement (steady-state model) or without replacement
    - Selection without replacement increases the selection pressure
  - For  $k = 2$  the time required by the best individual to dominate the population is the same to that from linear ranking selection with  $s = 2 * p$ ,  $p$  – selection probability of the best individual from population





# EAs - algorithm

## Design – selection for recombination

---

### ❑ Tournament selection

#### ■ Advantages

- ❑ Does not work with the entire population
- ❑ Easy to implement
- ❑ Easy to control the selection pressure by using parameter k

#### ■ Disadvantages

- ❑ The real results of this selection are different to theoretical distribution (similarly to roulette selection)

# EAs - algorithm

## Design – survival selection



- Survival selection (selection for replacement)
  - Based on age
    - ▣ Eliminates the oldest individuals
  - Based on fitness
    - ▣ Proportional selection
    - ▣ Ranking selection
    - ▣ Tournament selection
    - ▣ Elitism
      - Keep the best individuals from a generation to the next one (if the offspring are weaker than parents, then keep the parents)
    - ▣ GENITOR (replaces the worst individual)
      - Elimination of the worst  $\lambda$  individuals



# EAs - algorithm

## Design – variation operators

---

- Aim :
  - Generation of new possible solutions
- Properties
  - Works at individual level
  - Is based on individual representation (fitness independent)
  - Helps the exploration and exploitation of the search space
  - Must produce valid individuals
- Typology
  - Arity criterion
    - Arity 1 → mutation operators
    - Arity > 1 → recombination/crossover operators



# EAs - algorithm

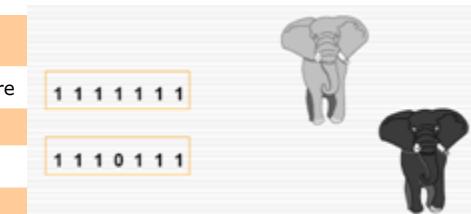
## Design – mutation

### □ Aim

- Reintroduces in population the lost genetic material
- Unary search operator (continuous space)
- Introduces the diversity in population (discrete space)

### □ Properties

- Works at genotype level
- Based on random elements
- Responsible to the exploration of promising regions of the search space
- Responsible to escape from local optima
- Must introduce small and stochastic changes for an individual
- Size of mutation must be controllable
- Can probabilistic take place (by a given probability  $p_m$ ) at the gene level



# EAs - algorithm

## Design – mutation



### □ Typology

- Binary representation
  - Strong mutation – bit-flipping
  - Weak mutation
- Integer representation
  - Random resetting
  - Creep mutation
- Permutation representation
  - Insertion mutation
  - Swap mutation
  - Inverse mutation
  - scramble mutation
  - K-opt mutation
- Real representation
  - Uniform mutation
  - Non-uniform mutation
    - Gaussian mutation
    - Cauchy mutation
    - Laplace mutation
- Tree-based representation → future lecture
  - Grow mutation
  - Shrink mutation
  - Switch mutation
  - Cycle mutation
  - Koza mutation
  - Mutation for numerical terminals



# EAs - algorithm

## Design – mutation (binary representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in \{0,1\}$ , for  $i=1,2,\dots,L$
- Strong mutation – *bit flipping*
  - Main idea
    - Changes by probability  $p_m$  (mutation rate) all the genes in their complement
      - $1 \rightarrow 0$
      - $0 \rightarrow 1$
    - Eg. A chromosome of  $L = 8$  genes,  $p_m = 0.1$





# EAs - algorithm

## Design – mutation (binary representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in \{0,1\}$ , for  $i=1,2,\dots,L$

### □ Weak mutation

- Main idea

- Changes by probability  $p_m$  (mutation rate) some of the genes in 0 or 1
  - $1 \rightarrow 0/1$
  - $0 \rightarrow 1/0$

- Eg. A chromosome of  $L = 8$  genes,  $p_m = 0.1$





# EAs - algorithm

## Design – mutation (integer representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in \{val_1, val_2, \dots, val_k\}$  for  $i=1,2,\dots,L$
- *Random resetting* mutation
  - Main idea
    - The value of a gene is changed (by probability  $p_m$ ) into another value (from the definition domain)





# EAs - algorithm

## Design – mutation (integer representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in \{val_1, val_2, \dots, val_k\}$ , for  $i=1,2,\dots,L$
- *Creep* mutation
  - Main idea
    - The value of a gene is changed (by probability  $p_m$ ) by adding a positive/negative value
      - New value follows a 0 symmetric distribution
      - The performed change is very small





# EAs - algorithm

## Design – mutation (permutation representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  with  $g_i \neq g_j$  for all  $i \neq j$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in \{val_1, val_2, \dots, val_L\}$ , for  $i = 1, 2, \dots, L$  s.a.  $g'_i \neq g'_j$  for all  $i \neq j$ .
- *Swap mutation*
  - Main idea
    - Randomly choose 2 genes and swap their values





# EAs - algorithm

## Design – mutation (permutation representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  with  $g_i \neq g_j$  for all  $i \neq j$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$ , for  $i=1,2,\dots,L$  s.a.  $g'_i \neq g'_j$  for all  $i \neq j$ .

### □ Insertion mutation

#### ■ Main idea

- Randomly choose 2 genes  $g_i$  and  $g_j$  with  $j > i$
- Insert gene  $g_j$  after gene  $g_i$  s.a.  $g'_i = g_i, g'_{i+1} = g_j, g'_{k+2} = g_{k+1}$ , for  $k=i, i+1, i+2, \dots$





# EAs - algorithm

## Design – mutation (permutation representation)

- A chromosome  $c=(g_1, g_2, \dots, g_L)$  with  $g_i \neq g_j$  for all  $i \neq j$  becomes  $c'=(g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in \{val_1, val_2, \dots, val_L\}$ , for  $i=1, 2, \dots, L$  s.a.  $g'_i \neq g'_j$  for all  $i \neq j$ .

### □ Inversion mutation

#### ■ Main idea

- Randomly choose 2 genes and inverse the order of genes between them (sub-string of genes)





# EAs - algorithm

## Design – mutation (permutation representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  with  $g_i \neq g_j$  for all  $i \neq j$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in \{val_1, val_2, \dots, val_L\}$ , for  $i = 1, 2, \dots, L$  s.a.  $g'_i \neq g'_j$  for all  $i \neq j$ .

- *scramble mutation*

- Main idea

- Randomly choose a (continuous or discontinuous) sub-array of genes and re-organise that genes





# EAs - algorithm

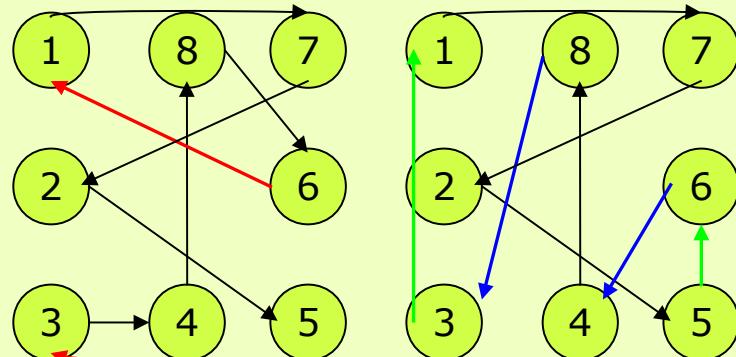
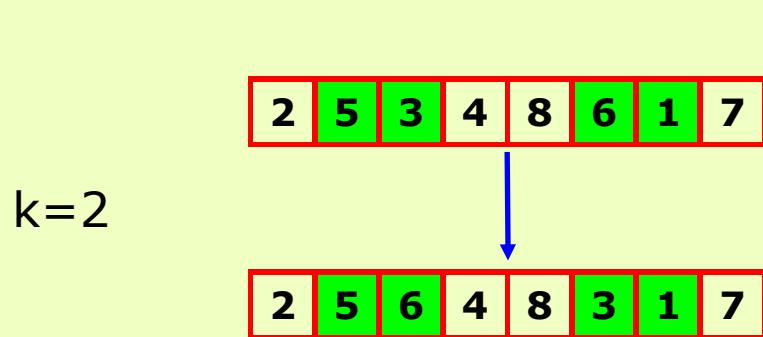
## Design – mutation (permutation representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  with  $g_i \neq g_i$  for all  $i \neq i$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$ , for  $i=1,2,\dots,L$  s.a.  $g'_i \neq g'_i$  for all  $i \neq i$ .

### □ K-opt mutation

#### ■ Main idea

- Choose 2 disjoint sub-strings of length k
- Interchange 2 elements of these sub-strings





# EAs - algorithm

## Design – mutation (real representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- Uniform mutation
  - Main idea
    - $g'_i$  is changed by probability  $p_m$  into a new value that is randomly uniform generated in  $[LI_i, LS_i]$  range



# EAs - algorithm

## Design – mutation (real representation)

- A chromosome  $c = (g_1, g_2, \dots, g_L)$  becomes  $c' = (g'_1, g'_2, \dots, g'_L)$ , where  $g_i, g'_i \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$

### □ Non-uniform mutation

#### ■ Main idea

The value of a gene is changed by adding a positive/negative value with a given probability ( $p_m$ )

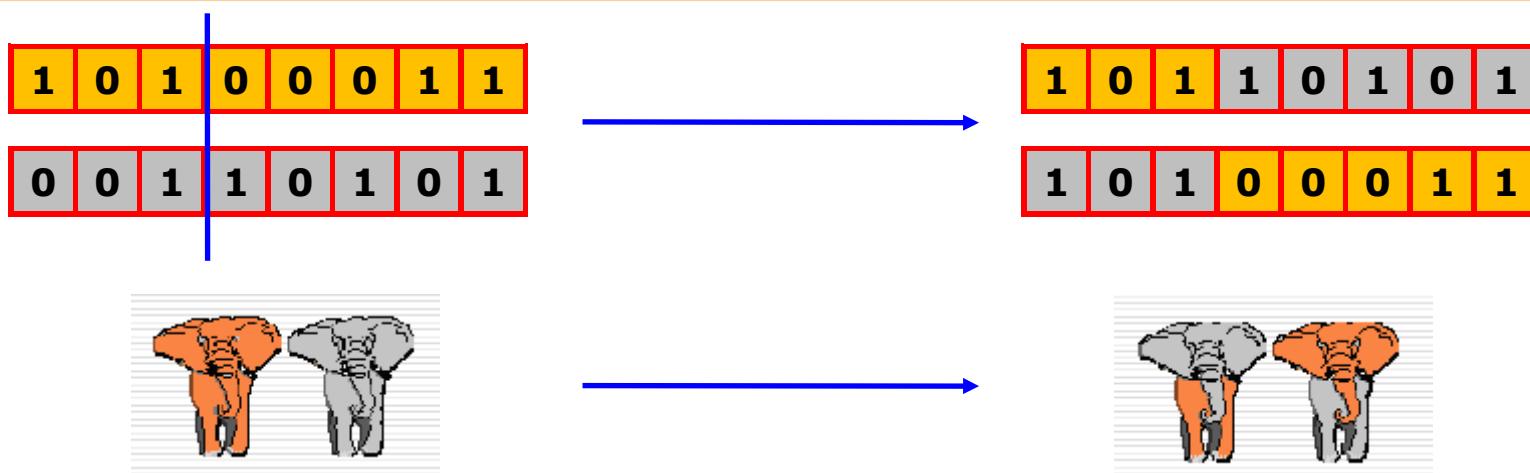
- The added value belongs to a distribution of type
  - $N(\mu, \sigma)$  (Gaussian) with  $\mu = 0$
  - Cauchy ( $x_0, \gamma$ )
  - Laplace ( $\mu, b$ )
- And it is re-introduced in  $[LI_i, LS_i]$  range (if it is necessary) – *clamping*

# EAs - algorithm Design – recombination



- Aim
  - Mix the parents' information

- Properties
  - The offspring has to inherit something from both parents
    - Selection of mixed information is randomly performed
  - Operator for exploitation of already discovered possible solutions
  - The offspring can be better, the same or weaker than their parents
  - Its effects are reducing while the search converges



# EAs - algorithm

## Design – recombination



### □ Typology

- Binary and integer representation
  - With cutting points
  - Uniforme
- Permutation representation
  - Order crossover (version 1 and version 2)
  - Partially Mapped Crossover
  - Cycle crossover
  - Edge-based crossover
- Real representation
  - Discrete
  - Arithmetic
    - Singular
    - Simple
    - Complete
  - Geometric
  - Shuffle crossover
  - Simulated binary crossover
- Tree-based representation
  - Sub-tree based crossover → future lecture



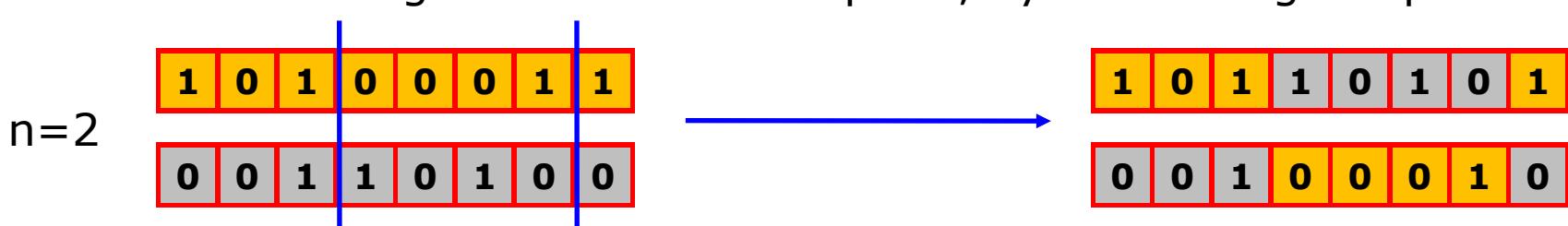
## EAs - algorithm

### Design – recombination (binary and integer representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - where  $g_i^1, g_i^2, g_i', g_i'' \in \{0,1\} / \{\text{val}_1, \text{val}_2, \dots, \text{val}_k\}$ , for  $i=1,2,\dots,L$

### □ N-cutting point crossover

- Main idea
  - Choose n cutting-points ( $n < L$ )
  - Cut the parents through these points
  - Put together the resulted parts, by alternating the parents





# EAs - algorithm

## Design – recombination (binary and integer representation)

### □ N cutting point crossover

#### ■ Properties

- Average of values encoded by parents = average of values encoded by offspring
  - Eg binary representation on 4 bits of integer numbers – XO with n=1 after second bit
    - $p_1 = (1,0,1,0)$ ,  $p_2 = (1,1,0,1)$
    - $c_1 = (1,0, 0,1)$ ,  $c_2 = (1,1,1,0)$
    - $\text{val}(p_1) = 10$ ,  $\text{val}(p_2) = 13 \rightarrow (\text{val}(p_1) + \text{val}(p_2))/2 = 23/2=11.5$
    - $\text{val}(c_1) = 9$ ,  $\text{val}(c_2) = 14 \rightarrow (\text{val}(c_1) + \text{val}(c_2))/2 = 23/2=11.5$
  - Eg. Binary representation on 4 bits for knapsack problem (K=10, 4 items of weight and value: (2,7), (1,8), (3,1), (2,3))
    - $p_1 = (1,0,1,0)$ ,  $p_2 = (1,1,0,1)$
    - $c_1 = (1,0, 0,1)$ ,  $c_2 = (1,1,1,0)$
    - $\text{val}(p_1) = 8$ ,  $\text{val}(p_2) = 18 \rightarrow (\text{val}(p_1) + \text{val}(p_2))/2 = 26/2=13$
    - $\text{val}(c_1) = 10$ ,  $\text{val}(c_2) = 16 \rightarrow (\text{val}(c_1) + \text{val}(c_2))/2 = 26/2=13$
  - Probability of  $\beta \approx 1$  is the largest one  $\beta = \left| \frac{\text{val}(d_1) - \text{val}(d_2)}{\text{val}(p_1) - \text{val}(p_2)} \right|$ 
    - Contracting crossover  $\beta < 1$ 
      - Offspring values are between parent values
    - Expanding crossover  $\beta > 1$ 
      - Parent values are between offspring values
    - Stationary crossover  $\beta = 1$ 
      - Offspring values are equal to parent values



## EAs - algorithm

### Design – recombination (binary and integer representation)

- From 2 parent chromosomes

- $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$

- 2 offspring are obtained

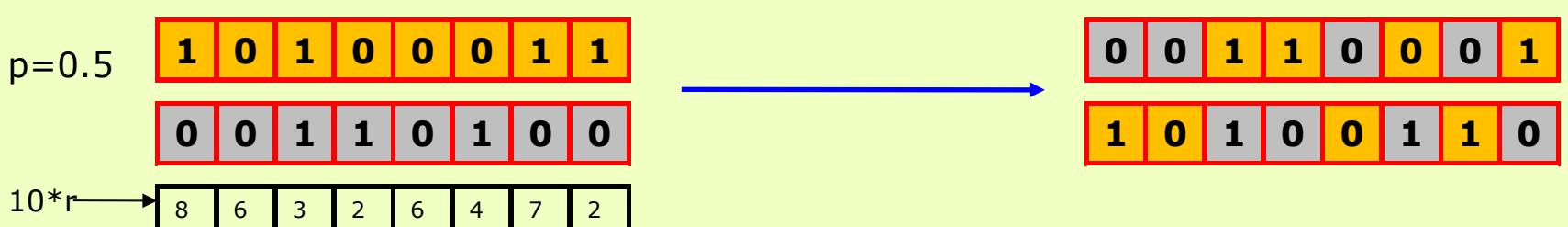
- $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - where  $g_i^1, g_i^2, g_i', g_i'' \in \{0,1\} / \{\text{val}_1, \text{val}_2, \dots, \text{val}_k\}$ , for  $i=1,2,\dots,L$

- Uniform crossover

- Main idea

- Each gene of an offspring comes from a randomly and uniform selected parent:

- For each gene a uniform random number  $r$  is generated
      - If  $r < \text{probability } p$  (usually,  $p=0.5$ ),  $c_1$  will inherit that gene from  $p_1$  and  $c_2$  from  $p_2$ ,
      - otherwise,  $c_1$  will inherit  $p_2$  and  $c_2$  will inherit  $p_1$





## EAs - algorithm

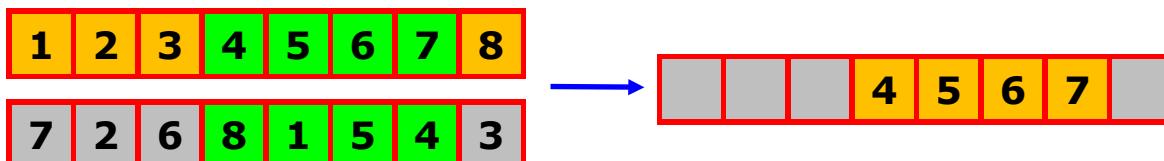
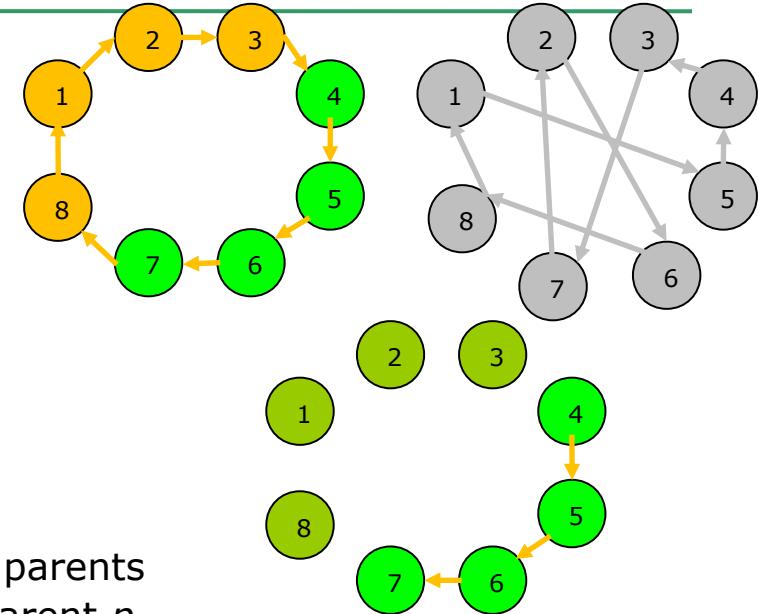
### Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$

#### □ Order crossover

##### ■ Main idea

- Offspring keep the order of genes from parents
- Choose a substring of genes from the parent  $p_1$
- Copy the substring from  $p_1$  into offspring  $d_1$  (on corresponding positions)

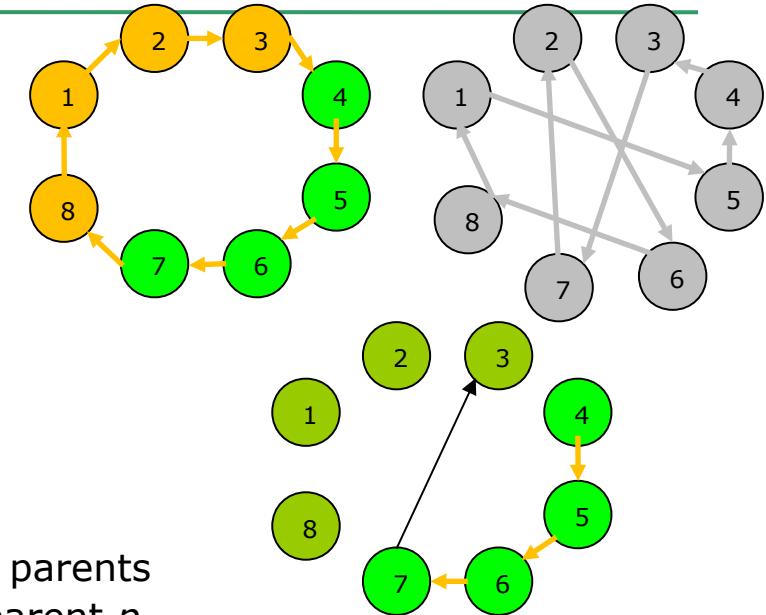




## EAs - algorithm

### Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$



#### □ Order crossover

- Main idea
  - Offspring keep the order of genes from parents
  - Choose a substring of genes from the parent  $p_1$
  - Copy the substring from  $p_1$  into offspring  $d_1$  (on corresponding positions)
  - Copy the genes of  $p_2$  in offspring  $d_1$ :
    - Starting with the first position after sub-string
    - Respecting gene's order from  $p_2$  and
    - Re-loading the genes from start (if the end of chromosome is reached)

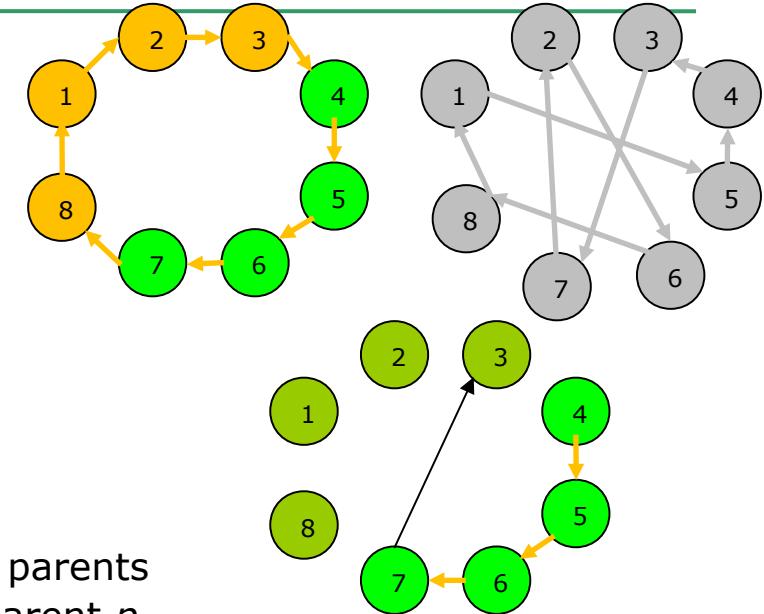




## EAs - algorithm

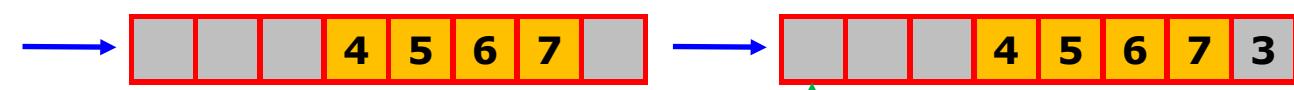
### Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$



#### □ Order crossover

- Main idea
  - Offspring keep the order of genes from parents
  - Choose a substring of genes from the parent  $p_1$
  - Copy the substring from  $p_1$  into offspring  $d_1$  (on corresponding positions)
  - Copy the genes of  $p_2$  in offspring  $d_1$ :
    - Starting with the first position after sub-string
    - Respecting gene's order from  $p_2$  and
    - Re-loading the genes from start (if the end of chromosome is reached)

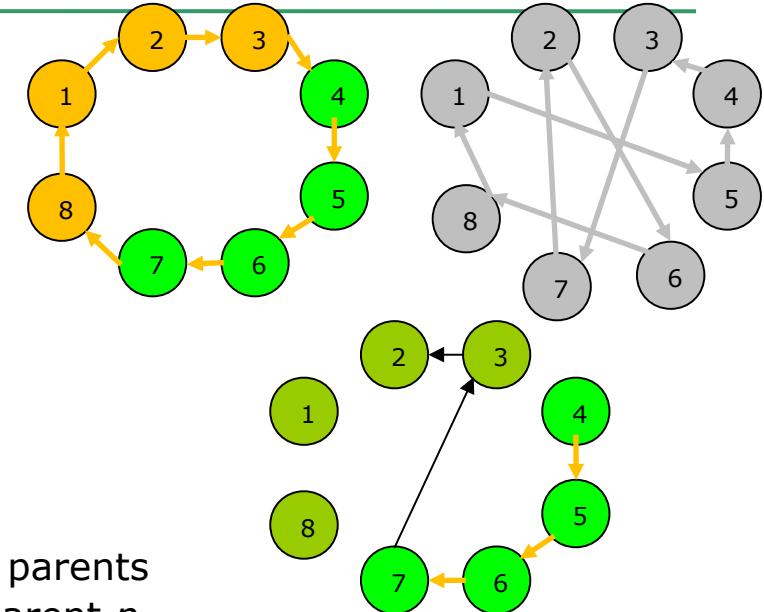




# EAs - algorithm

## Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$



### □ Order crossover

- Main idea
  - Offspring keep the order of genes from parents
  - Choose a substring of genes from the parent  $p_1$
  - Copy the substring from  $p_1$  into offspring  $d_1$  (on corresponding positions)
  - Copy the genes of  $p_2$  in offspring  $d_1$ :
    - Starting with the first position after sub-string
    - Respecting gene's order from  $p_2$  and
    - Re-loading the genes from start (if the end of chromosome is reached)

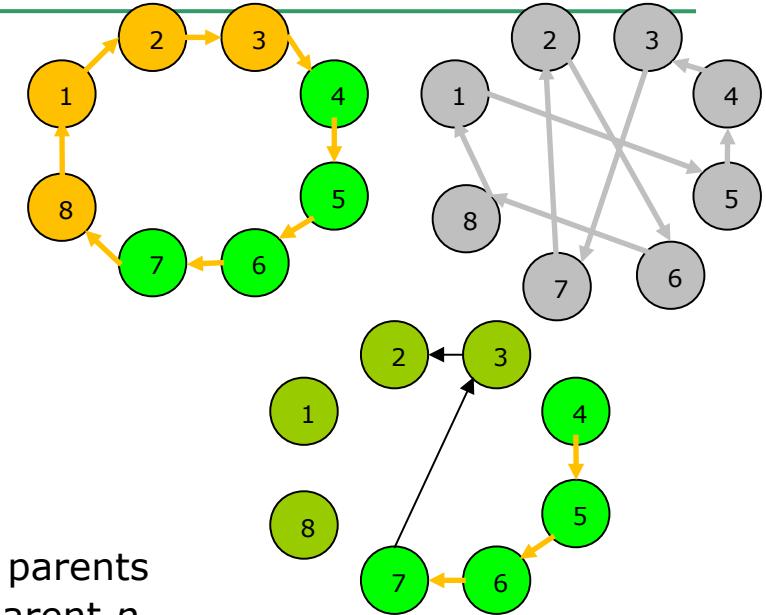




# EAs - algorithm

## Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$



### □ Order crossover

- Main idea
  - Offspring keep the order of genes from parents
  - Choose a substring of genes from the parent  $p_1$
  - Copy the substring from  $p_1$  into offspring  $d_1$  (on corresponding positions)
  - Copy the genes of  $p_2$  in offspring  $d_1$ :
    - Starting with the first position after sub-string
    - Respecting gene's order from  $p_2$  and
    - Re-loading the genes from start (if the end of chromosome is reached)

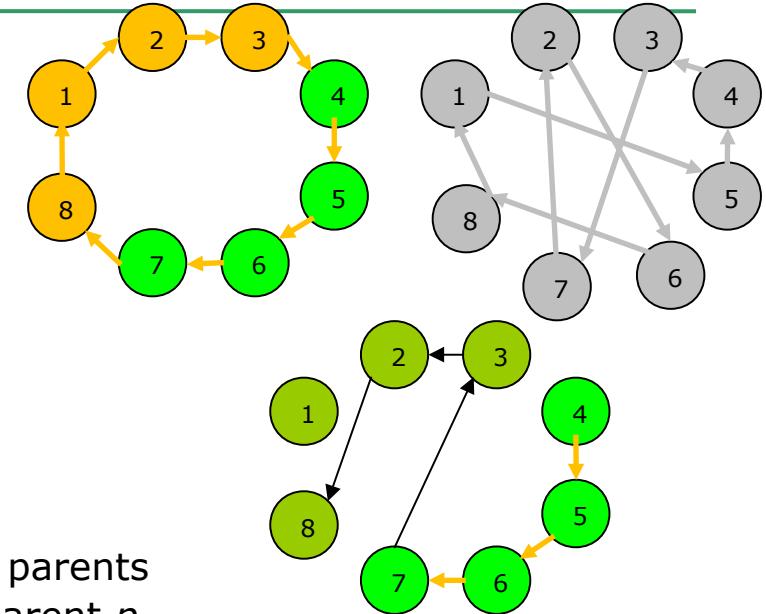




# EAs - algorithm

## Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$



### □ Order crossover

- Main idea
  - Offspring keep the order of genes from parents
  - Choose a substring of genes from the parent  $p_1$
  - Copy the substring from  $p_1$  into offspring  $d_1$  (on corresponding positions)
  - Copy the genes of  $p_2$  in offspring  $d_1$ :
    - Starting with the first position after sub-string
    - Respecting gene's order from  $p_2$  and
    - Re-loading the genes from start (if the end of chromosome is reached)

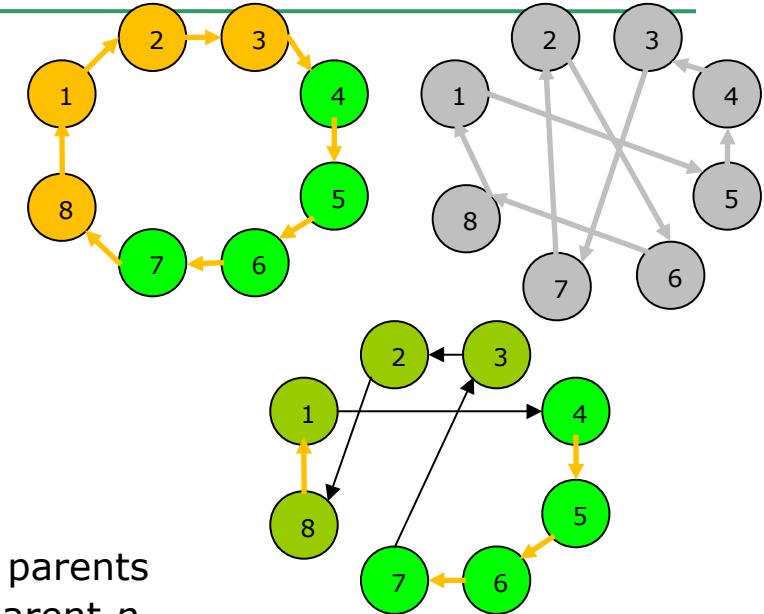




# EAs - algorithm

## Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$



### □ Order crossover

- Main idea
  - Offspring keep the order of genes from parents
  - Choose a substring of genes from the parent  $p_1$
  - Copy the substring from  $p_1$  into offspring  $d_1$  (on corresponding positions)
  - Copy the genes of  $p_2$  in offspring  $d_1$ :
    - Starting with the first position after sub-string
    - Respecting gene's order from  $p_2$  and
    - Re-loading the genes from start (if the end of chromosome is reached)

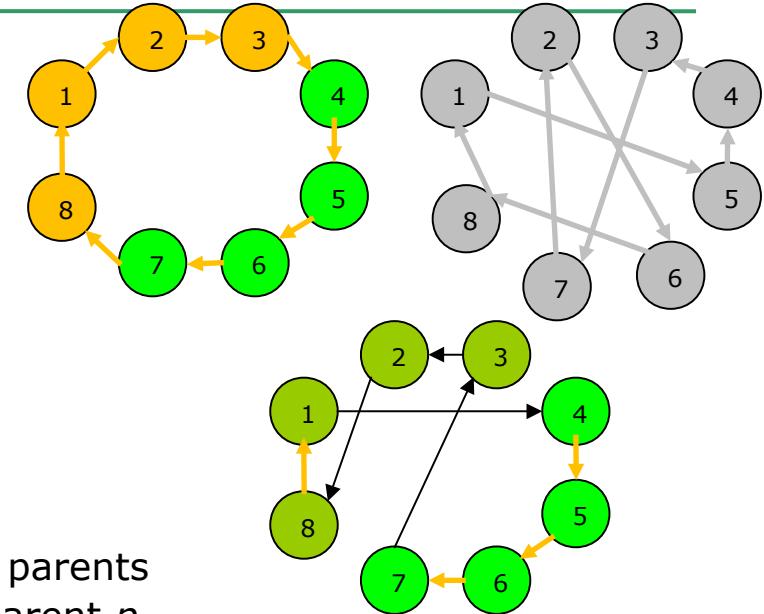




# EAs - algorithm

## Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$



- Order crossover

- Main idea
  - Offspring keep the order of genes from parents
  - Choose a substring of genes from the parent  $p_1$
  - Copy the substring from  $p_1$  into offspring  $d_1$  (on corresponding positions)
  - Copy the genes of  $p_2$  in offspring  $d_1$ :
    - Starting with the first position after sub-string
    - Respecting gene's order from  $p_2$  and
    - Re-loading the genes from start (if the end of chromosome is reached)
  - Repeat all the previous steps for the second offspring  $d_2$ .





# EAs - algorithm

## Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- *Partially mapped XO*
  - Main idea
    - Choose a substring of genes from parent  $p_1$
    - Copy the sub-string into offspring  $d_1$  (on corresponding positions)
    - Take elements  $i$  from substring of  $p_2$  that do not belong to substring from  $p_1$  and determine the element  $j$  that was copied instead of it from  $p_1$
    - Put  $i$  in  $d_1$  on position of  $j$  in  $p_2$  (if that place is empty)
    - If the place of  $j$  in  $p_2$  is already filled by element  $k$  in  $d_1$ , then  $i$  will be put in the position of  $k$  in  $p_2$
    - All the other elements are copied from  $p_2$  into  $d_1$
    - Repeat the procedure for  $d_2$  by swapping the parents





## EAs - algorithm

### Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- *Partially mapped XO*
  - Main idea
    - Choose a substring of genes from parent  $p_1$
    - Copy the sub-string into offspring  $d_1$  (on corresponding positions)
    - Take elements  $i$  from substring of  $p_2$  that do not belong to substring from  $p_1$  and determine the element  $j$  that was copied instead of it from  $p_1$
    - Put  $i$  in  $d_1$  on position of  $j$  in  $p_2$  (if that place is empty)
    - If the place of  $j$  in  $p_2$  is already filled by element  $k$  in  $d_1$ , then  $i$  will be put in the position of  $k$  in  $p_2$
    - All the other elements are copied from  $p_2$  into  $d_1$
    - Repeat the procedure for  $d_2$  by swapping the parents





# EAs - algorithm

## Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$

### □ Partially mapped XO

- Main idea
  - Choose a substring of genes from parent  $p_1$
  - Copy the sub-string into offspring  $d_1$  (on corresponding position(s))
  - Take elements  $i$  from substring of  $p_2$  that do not belong to substring from  $p_1$  and determine the element  $j$  that was copied instead of it from  $p_1$
  - Put  $i$  in  $d_1$  on position of  $j$  in  $p_2$  (if that place is empty)
  - If the place of  $j$  in  $p_2$  is already filled by element  $k$  in  $d_1$ , then  $i$  will be put in the position of  $k$  in  $p_2$
  - All the other elements are copied from  $p_2$  into  $d_1$
  - Repeat the procedure for  $d_2$  by swapping the parents

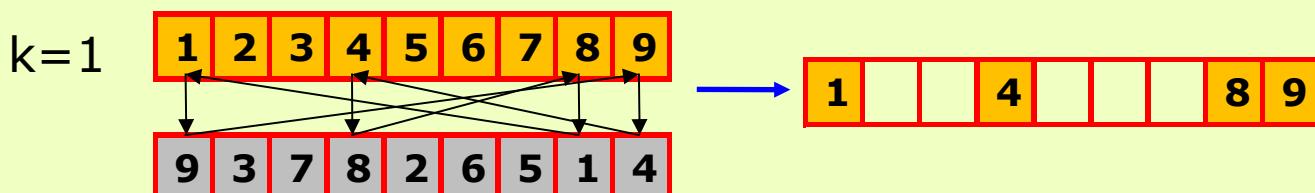




# EAs - algorithm

## Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- Cycle crossover
  - Main idea
    1. Initially,  $k = 1$
    2. Create a cycle:
      - Add into the cycle the gene from position  $k$  from  $p_1$  ( $g_k^1$ )
      - Take the gene of position  $k$  from  $p_2$  ( $g_k^2$ )
      - Select the gene of  $p_1$  whose value is equal to  $g_k^2$  ( $g_r^1$ ) and include it in the cycle
      - Take the gene of position  $r$  from  $p_2$  ( $g_r^2$ )
      - Repeat the previous steps until the gene of position  $k$  from  $p_1$  is considered
    3. Copy the genes of cycle into  $d_1$  (by respecting the appearance positions in  $p_1$ )
    4. Increase  $k$  and compose a new cycle with the genes from  $p_2$ 
      - Copy the genes of cycle into  $d_1$  (by respecting the appearance positions in  $p_2$ )
      - Repeat steps 2-5 until  $k = L$

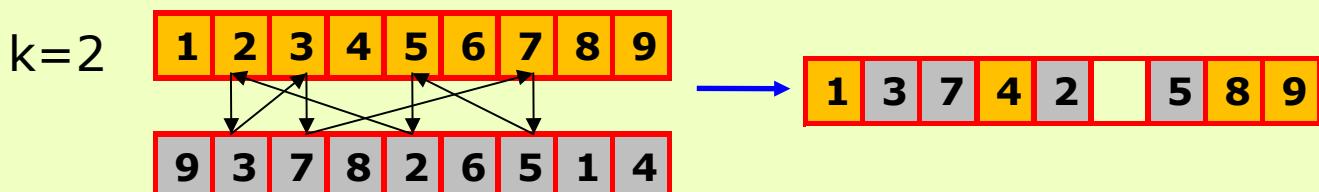




# EAs - algorithm

## Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- Cycle crossover
  - Main idea
    1. Initially,  $k = 1$
    2. Create a cycle:
      - Add into the cycle the gene from position  $k$  from  $p_1$  ( $g_k^1$ )
      - Take the gene of position  $k$  from  $p_2$  ( $g_k^2$ )
      - Select the gene of  $p_1$  whose value is equal to  $g_k^2$  ( $g_r^1$ ) and include it in the cycle
      - Take the gene of position  $r$  from  $p_2$  ( $g_r^2$ )
      - Repeat the previous steps until the gene of position  $k$  from  $p_1$  is considered
    3. Copy the genes of cycle into  $d_1$  (by respecting the appearance positions in  $p_1$ )
    4. Increase  $k$  and compose a new cycle with the genes from  $p_2$ 
      - Copy the genes of cycle into  $d_1$  (by respecting the appearance positions in  $p_2$ )
      - Repeat steps 2-5 until  $k = L$

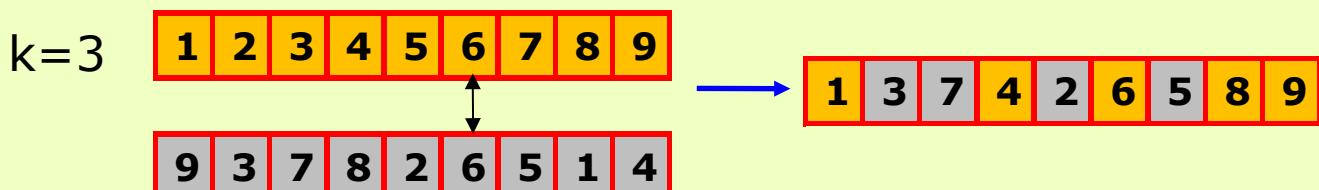




## EAs - algorithm

### Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [L_{I_i}, L_{S_i}]$ , for  $i=1,2,\dots,L$
- Cycle crossover
  - Main idea
    1. Initially,  $k = 1$
    2. Create a cycle:
      - Add into the cycle the gene from position  $k$  from  $p_1$  ( $g_k^1$ )
      - Take the gene of position  $k$  from  $p_2$  ( $g_k^2$ )
      - Select the gene of  $p_1$  whose value is equal to  $g_k^2$  ( $g_r^1$ ) and include it in the cycle
      - Take the gene of position  $r$  from  $p_2$  ( $g_r^2$ )
      - Repeat the previous steps until the gene of position  $k$  from  $p_1$  is considered
    3. Copy the genes of cycle into  $d_1$  (by respecting the appearance positions in  $p_1$ )
    4. Increase  $k$  and compose a new cycle with the genes from  $p_2$ 
      - Copy the genes of cycle into  $d_1$  (by respecting the appearance positions in  $p_2$ )
      - Repeat steps 2-5 until  $k = L$





## EAs - algorithm

### Design – recombination (permutation representation)

---

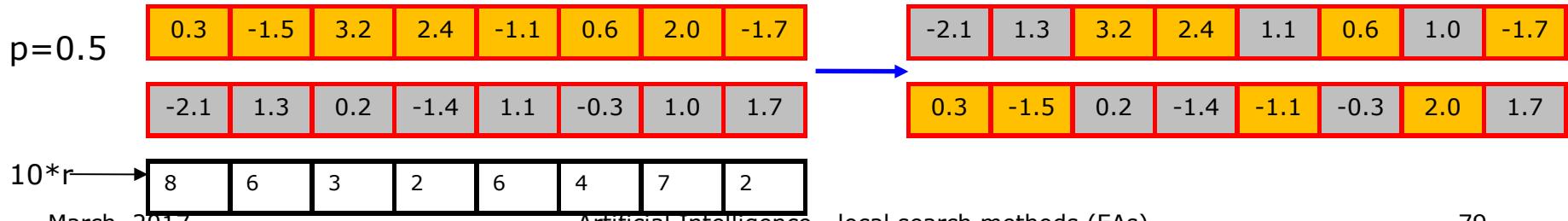
- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
  
- Edge-based crossover
  - See *Whitley, Darrell, Timothy Starkweather, D'Ann Fuquay (1989). "Scheduling problems and traveling salesman: The genetic edge recombination operator". International Conference on Genetic Algorithms. pp. 133–140* [link](#)



# EAs - algorithm

## Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- Discrete crossover
  - Main idea
    - Each gene offspring is taken (by the same probability,  $p = 0.5$ ) from one of the parents
    - Similarly to uniform crossover for binary/integer representation
    - The absolute values of genes are not changed (no new information is created)





# EAs – algorithm

## Design – recombination (real representation)

- From 2 parent chromosomes
    - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
  - 2 offspring are obtained
    - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
    - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- 
- Arithmetic crossover
    - Main idea
      - Create offspring between parents  $\rightarrow$  arithmetic crossover
        - $z_i = \alpha x_i + (1 - \alpha) y_i$  where  $\alpha : 0 \leq \alpha \leq 1$ .
      - Parameter  $\alpha$  can be:
        - Constant  $\rightarrow$  uniform arithmetic crossover
        - Variable  $\rightarrow$  eg. Depends on the age of population
        - Random  $\rightarrow$  generated for each new XO that is performed
      - New values of a gene can appear
    - Typology
      - Singular arithmetic crossover
      - Simple arithmetic crossover
      - Complete arithmetic crossover



# EAs – algorithm Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- Singular arithmetic crossover
  - Choose one gene from two parents (of the same position  $k$ ) and combine them
    - $g_k' = \alpha g_k^1 + (1-\alpha)g_k^2$
    - $g_k'' = (1-\alpha)g_k^1 + \alpha g_k^2$
  - The rest of genes are unchanged
    - $g_i' = g_i^1$
    - $g_i'' = g_i^2$ , for  $i = 1,2,\dots,L$  and  $i \neq k$

$$[LI, LS] = [-2.5, +3]$$

$$k=3$$

$$\alpha = 0.6$$

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7



$0.6 * 3.2 + (1-0.6) * 0.2 = 2.0$	$(1-0.6) * 3.2 + 0.6 * 0.2 = 1.4$	<table border="1"><tbody><tr><td>0.3</td><td>-1.5</td><td>2.0</td><td>2.4</td><td>-1.1</td><td>0.6</td><td>2.0</td><td>-1.7</td></tr><tr><td>-2.1</td><td>1.3</td><td>1.4</td><td>-1.4</td><td>1.1</td><td>-0.3</td><td>1.0</td><td>1.7</td></tr></tbody></table>	0.3	-1.5	2.0	2.4	-1.1	0.6	2.0	-1.7	-2.1	1.3	1.4	-1.4	1.1	-0.3	1.0	1.7
0.3	-1.5	2.0	2.4	-1.1	0.6	2.0	-1.7											
-2.1	1.3	1.4	-1.4	1.1	-0.3	1.0	1.7											



# EAs – algorithm Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- Simple arithmetic crossover
  - Select a position  $k$  and combine all the genes after that position
    - $g_i' = \alpha g_i^1 + (1-\alpha)g_i^2$
    - $g_i'' = (1-\alpha)g_i^1 + \alpha g_i^2$ , for  $i=k, k+1, \dots, L$
  - Genes from positions  $< k$  rest unchanged
    - $g_i' = g_i^1$
    - $g_i'' = g_i^2$ , for  $i = 1, 2, \dots, k-1$

$$[LI, LS] = [-2.5, +3]$$

$$k=6$$

$$\alpha = 0.6$$

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

0.3	-1.5	3.2	2.4	-1.1	0.24	1.6	-0.34
-2.1	1.3	0.2	-1.4	1.1	0.06	1.4	0.34

$$0.6*0.6+(1-0.6)*(-0.3)=0.24$$

$$(1-0.6)*0.6+0.6*(-0.3)=0.06$$



# EAs – algorithm

## Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$

### □ Complete arithmetic crossover

- All of the genes are combined
  - $g_i' = \alpha g_i^1 + (1-\alpha)g_i^2$
  - $g_i'' = (1-\alpha)g_i^1 + \alpha g_i^2$  , for  $i=1,2,\dots,L$

$$0.6*0.3+(1-0.6)*(-2.1)=-0.66$$

$$[LI, LS] = [-2.5, +3]$$

$$\alpha = 0.6$$

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

$$(1-0.6)*0.3+0.6*(-2.1)=-1.14$$

-0.66	4.3	2.0	0.48	-0.22	0.24	1.6	-0.34
-1.14	0.18	1.4	0.12	0.22	0.06	1.4	0.34



# EAs – algorithm

## Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- Geometric crossover
  - Main idea
    - Each gene of an offspring represents the product between parent's genes, each of them by a given exponent  $\omega$  and  $1-\omega$ , respectively (where  $\omega$  is a real positive number  $\leq 1$ )
    - $g_i' = (g_i^1)^\omega (g_i^2)^{1-\omega}$
    - $g_i'' = (g_i^1)^{1-\omega} (g_i^2)^\omega$

$$[LI, LS] = [-2.5, +3]$$

$$\omega = 0.7$$

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7

$$0.3^{0.7} + 2.1^{1-0.7} = 1.68$$

$$0.3^{1-0.7} + 2.1^{0.7} = 2.38$$

1.68	2.41	2.87	2.95	2.10	1.40	2.62	2.62
2.38	2.33	1.74	2.57	2.10	1.29	2.23	2.62



# EAs – algorithm

## Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- 1 offspring is obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$
  - Where  $g_i' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- Blend crossover – BLX
  - Main idea
    - A single offspring is created
    - Offspring's genes are randomly generated from  $[Min_i - I*a, Max_i + I*a]$  range, where:
      - $Min_i = \min\{g_i^1, g_i^2\}$ ,  $Max_i = \max\{g_i^1, g_i^2\}$
      - $I = Max - Min$ ,  $a$  – parameter from  $[0,1]$

$$[LI, LS] = [-2.5, +3]$$

$$a = 0.7$$



Min	0.3	1.3	0.2	1.4	1.1	0.3	1.0	1.7
Max	2.1	1.5	3.2	2.4	1.1	0.6	2.0	1.7
I	0.8	0.2	3.0	1.0	0	0.3	1.0	0.0

Min-Ia	-0.26	1.16	-1.90	0.70	1.10	0.09	0.30	1.70
Max+Ia	2.66	1.50	3.20	2.40	1.10	0.60	2.00	1.70



# EAs – algorithm

## Design – recombination (real representation)

- ▣ From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$  and  $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ▣ 2 offspring are obtained
  - $c_1 = (g_1', g_2', \dots, g_L')$  and  $c_2 = (g_1'', g_2'', \dots, g_L'')$ ,
  - Where  $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$ , for  $i=1,2,\dots,L$
- ▣ Simulated binary crossover
  - Main idea
    - Each gene of an offspring is a combination of parent's genes

$$d_1 = \frac{p_1 + p_2}{2} - \beta \frac{p_2 - p_1}{2}, \quad d_2 = \frac{p_1 + p_2}{2} + \beta \frac{p_2 - p_1}{2}$$

- ▣ Such as the two properties of n-cutting point XO to be respected (for binary representation)
  - Average of parent values = average of offspring values
  - Probability of a spread factor  $\beta \approx 1$  is greater to any other factor



# EAs – algorithm Design – recombination

---

## ❑ Multiple recombination

- Based on the value's frequencies from parents (general uniform XO)
- Based on segmentation and crossover (general XO with diagonal cutting points)
- Based on numeric operations that are specific to real values (XO based on gravity center, general arithmetic XO)

# EAs – algorithm

## Design – recombination or mutation?

---

- Intense debates
  - Questions:
    - Which is the best operator?
    - Which is the most necessary operator?
    - Which is the most important operator?
  - Answers:
    - Depend on problem, but,
    - In general, is better to use both operators
    - Each of them having another role (purpose).
    - EAs with mutation only are possible, but EAs with crossover only are not possible
- Search aspects:
  - Exploration → discovering promising regions in the search space (accumulating useful information about the problem)
  - Exploitation → optimising in a promising region of the search space (by using the existent information)
  - Cooperation and competition must exist between these 2 aspects
- Recombination
  - Exploitation operator → performs a large jump into a region somewhere between the regions associated to parents
    - Effects of exploitation decrease while AE is converging
  - Binary/n-ary operator that can combine information from 2/more parents
  - Operator that does not change the frequency of values from chromosome at the population level
- Mutation
  - Exploration operator → performs small random diversions, remaining in a neighbourhood of parent
    - Local optima escape
  - Operator that can introduce new genetic information
  - Operator that changes the frequency of values from chromosome at the population level



# EAs – algorithm

## Design – stop condition

---

- Choosing a stop condition
  - An optimal solution was found
  - The physical resources were ended
    - A given number of fitness evaluation has been performed
  - The user resources (time, patience) were ended
    - Several generation without improvements have been born

# EAs – algorithm Evaluation

---



- Performance evaluation of an EA
  - After more runs
    - Statistical measures are computed
      - Average of solutions
      - Median of solutions
      - Best solution
      - Worst solution
      - Standard deviation of solutions – for comparisons
  - The number of independent runs must be large enough

# EAs

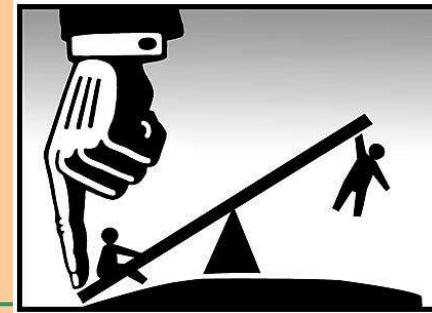
---



- Analyse of complexity
  - The most costly part → fitness evaluation

# EAs

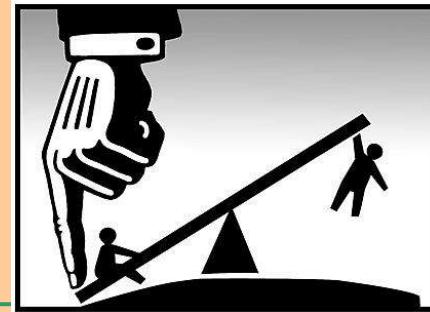
---



- Advantages
  - AEs have a general sketch for all the problems
    - Only
      - representation
      - fitness function
    - are changed
  - AEs are able to give better results than classical optimisation methods because
    - They do not require linearization
    - They are not based on some presumptions
    - They do not ignore some possible solutions
  - AEs are able to explore more possible solutions than human can

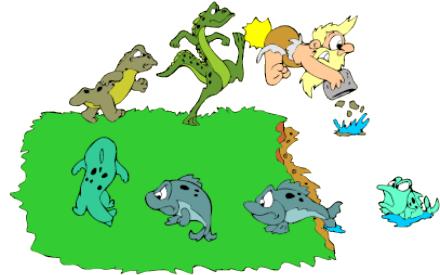
# AEs

---



- Disadvantages
  - Large running time

# AEs

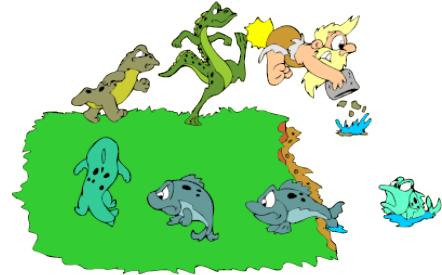


## ❑ Applications

- Vehicle design
  - Material composition
  - Vehicle shape
- Engineering design
  - Structural and organisational optimisation of constructions (buildings, robots, satellites, turbines)
- Robotics
  - Design and components optimisation
- Hardware evolution
  - Digital circuits optimisation
- Telecommunication optimisation
- Cross-word game generation
- Biometric inventions (inspired by natural architectures)
- Traffic and transportation routing
- PC games
- Cryptography
- Genetics
- Chemical analyse of kinematics
- Financial and marketing strategies

# AEs

---



## ❑ Typology

- Evolutionary strategies
- Evolutionary programming
- Genetic algorithms
- Genetic programming

# Next lecture

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

- A. Definition of search problems
- B. Search strategies

- A. Uninformed search strategies
- B. Informed search strategies
- C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
- D. Adversarial search strategies

### C. Intelligent systems

- A. Rule-based systems in certain environments
- B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
- C. Learning systems
  - A. Decision Trees
  - B. Artificial Neural Networks
  - C. Support Vector Machines
    - Evolutionary algorithms
- D. Hybrid systems

# Next lecture – Useful information

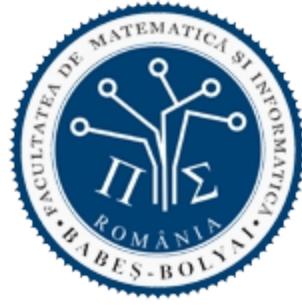
---

- Chapter 16 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *James Kennedy, Russel Eberhart, Particle Swarm Optimisation, Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948, 1995* (04\_ACO\_PSO/PSO\_00.pdf)
- *Marco Dorigo, Christian Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243 – 27* (04\_ACO\_PSO/Dorigo05\_ACO.pdf)

- 
- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean - [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)



BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics



# ARTIFICIAL INTELLIGENCE

**Solving search problems**

Informed local search strategies

Nature-inspired algorithms

# Topics

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

- A. Definition of search problems
- B. Search strategies

- A. Uninformed search strategies
- B. Informed search strategies
- C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
- D. Adversarial search strategies

### C. Intelligent systems

- A. Rule-based systems in certain environments
- B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
- C. Learning systems
  - A. Decision Trees
  - B. Artificial Neural Networks
  - C. Support Vector Machines
  - D. Evolutionary algorithms
- D. Hybrid systems

# Useful information

---

- Chapter 16 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *James Kennedy, Russel Eberhart, Particle Swarm Optimisation, Proceedings of IEEE International Conference on Neural Networks. IV.* pp. 1942–1948, 1995  
(04\_ACO\_PSO/PSO\_00.pdf)
- *Marco Dorigo, Christian Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243 – 27* (04\_ACO\_PSO/Dorigo05\_ACO.pdf)

# Local search

---

## □ Typology

- Simple local search – a single neighbor state is retained
  - Hill climbing → selects the best neighbor
  - Simulated annealing → selects probabilistic the best neighbor
  - Tabu search → retains the list of visited solutions
- Beam local search – more states are retained (a population of states)
  - Evolutionary algorithms
  - Particle swarm optimisation
  - Ant colony optimisation

# Nature-inspired algorithms

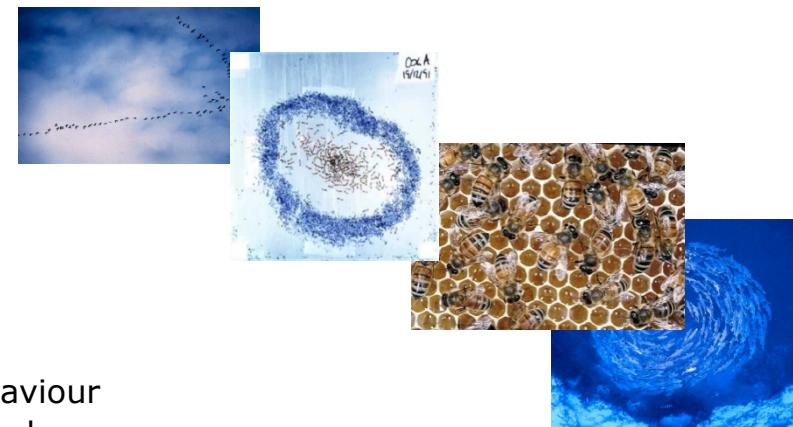
---

- Best method for solving a problem
  - Human brain
    - Has created the wheel, car, town, etc.
  - Mechanism of evolution
    - Has created the human brain
- Simulation of nature
  - By machines' help → the artificial neural networks simulate the brain
    - Flying vehicles, DNA computers, membrane-based computers
  - By algorithms' help
    - Evolutionary algorithms simulate the evolution of nature
    - Particle Swarm Optimisation simulates the collective and social behaviour
      - <http://www.youtube.com/watch?feature=endscreen&v=JhZKc1Mgub8&NR=1>
      - <http://www.youtube.com/watch?v=ulucJnxT7B4&feature=related>
    - Ant Colony Optimisation (ACO)
      - [http://www.youtube.com/watch?v=jrW\\_TTxP1ow](http://www.youtube.com/watch?v=jrW_TTxP1ow)

# Nature-inspired algorithms

---

- ▣ Swarm intelligence (collective intelligence)
  - A group of individuals that interact in order to achieve some objectives by collective adaptation to a global or local environment
  - A computational metaphor inspired by
    - Birds'' flying (V shape)
    - Ants that are searching food
    - Bees'' swarms that are constructing their nest
    - Schools of fish
  - Because:
    - Control is distributed among more individuals
    - Individuals local communicate
    - system behaviour transcends the individual behaviour
    - System is robust and can adapt to environment changes
- Social insects (2% of total)
  - Ants
    - 50% of social insects
    - 1 ant has  $\sim$  1mg  $\rightarrow$  total weight of ants  $\sim$  total weight of humans
    - Live for over 100 millions of years (humans live for over 50 000 years)
  - Termites
  - Bees



# Nature-inspired algorithms

---

- Swarm (Group)
  - More individuals, apparently non-organized, that are moving in order to form a group, but each individual seems to move in a particular direction
  - Inside the group can appear some social processes
  - The collection is able to do complex tasks
    - Without a guide or an external control
    - Without a central coordination
  - The collection can have performances better than the independent individuals
- Collective adaptation → self-organisation
  - Set of dynamic mechanisms that generates a global behaviour as a result of interaction among individual components
  - Rules that specify this interaction are executed based on local information only, without global references
  - Global behaviour is an emergent property of the system (and not one external imposed)

# PSO

---

- Theoretical aspects
- Algorithm
- Example
- Properties
- Applications

# PSO – theoretical aspects

---

- Proposed by
  - Kennedy and Eberhart in 1995 <http://www.particleswarm.info/>
  - Inspired by social behavior of bird swarms and school of fish
- Search is
  - **Cooperative**, guided by the relative quality of individuals
- Search operators
  - A kind of mutation

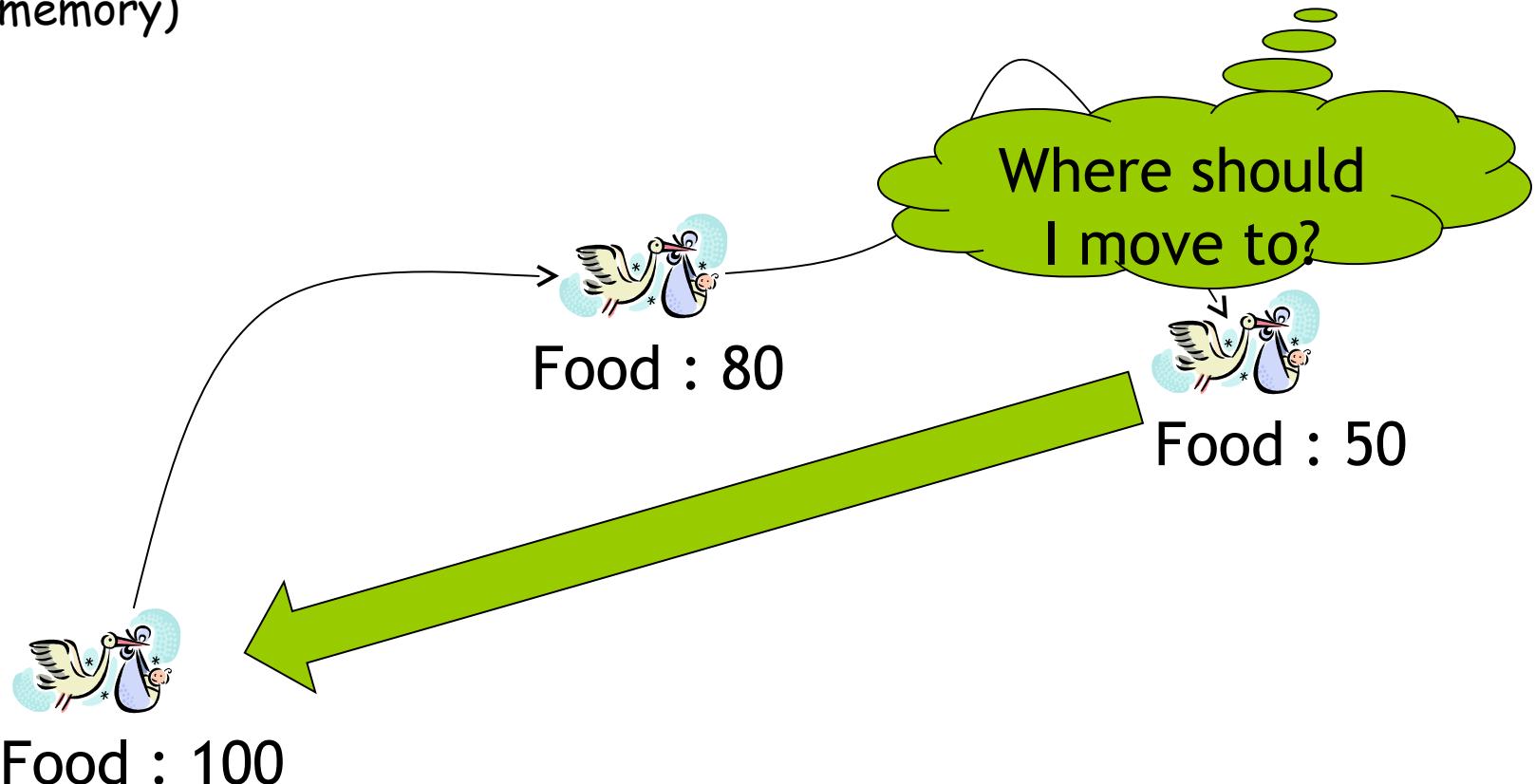
# PSO – theoretical aspects

---

- Special elements
  - Optimisation method based on:
    - Populations ( $\approx$  EAs) of particles ( $\approx$  chromosomes) that search the optimal solution
    - Cooperation (instead of concurrence, like in EAs case)
  - Each particle
    - moves (in the search space) and has a velocity (velocity  $\approx$  movement, because the time is discrete)
    - Retains the place where it has obtained the best results
    - Has associated a neighbourhood of particles
  - Particles cooperate
    - Exchange information among them (regarding the discovering performed in the places already visited)
    - Each particle knows the fitness of neighbours such as it can use the position of the best neighbour for adjusting its velocity

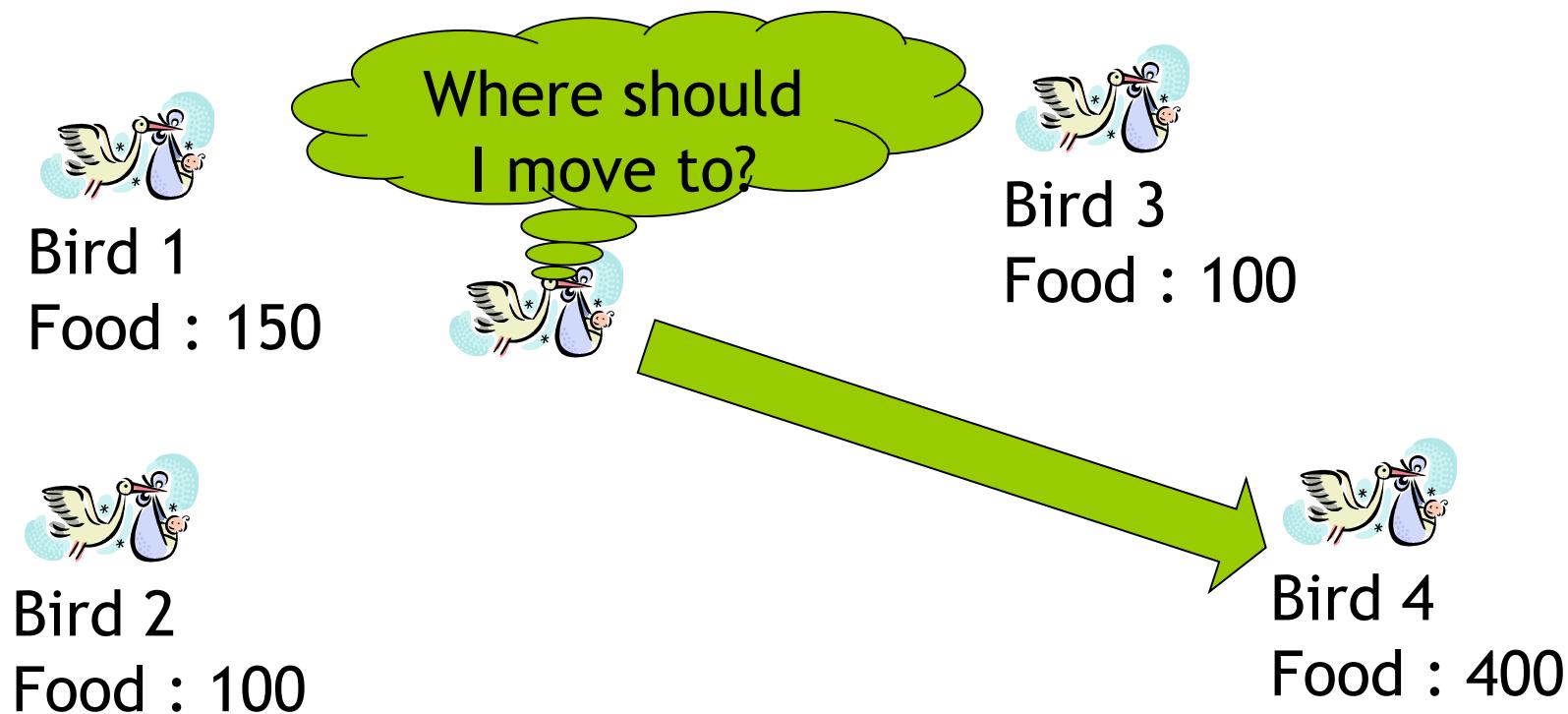
# PSO – theoretical aspects

Main idea: cognitive behaviour → an individual remembers past knowledge (has memory)



# PSO – theoretical aspects

Main idea: social behaviour → an individual relies on the knowledge of other members of the group



# PSO – algorithm

---

## □ General sketch

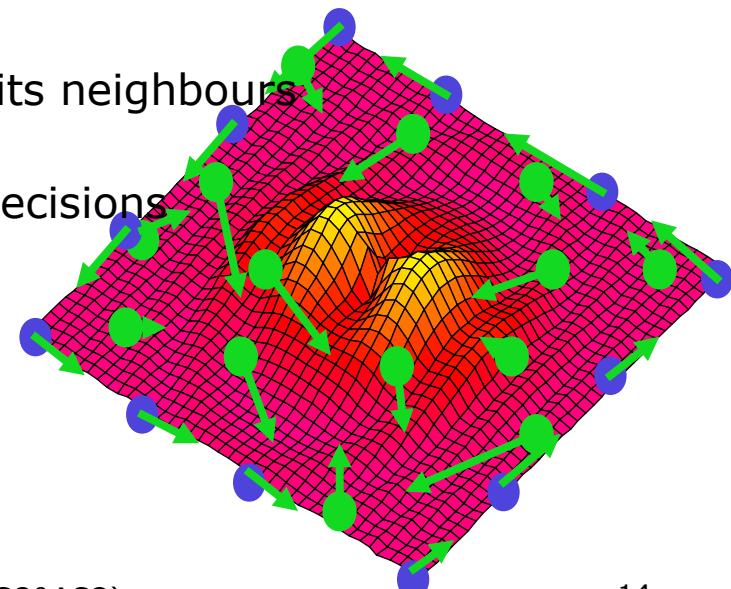
1. Creation of the initial population of particles
  1. Random positions
  2. Null/random velocities
2. Evaluation of particles
3. For each particle
  - Update the memory
    - Identify the best particle of the swarm ( $g_{Best}$ ) / of the current neighborhood ( $l_{Best}$ )
    - Identify the best position (with the best fitness) reached until now –  $p_{Best}$
  - Update the velocity
  - Update the position
4. If the stop conditions are not satisfied, go back to step 2; otherwise STOP.

# PSO – algorithm

---

## 1. Creation of the initial population of particles

- Each particle has associated
  - A position – possible solution of the problem
  - A velocity – changes a position into another position
  - A quality function (fitness)
- Each particle has to:
  - Interact (exchange information) with its neighbours
  - Memorise a previous position
  - Use the information in order to take decisions
- Initialisation of particles
  - Random positions
  - Null/random velocities



# PSO – algorithm

---

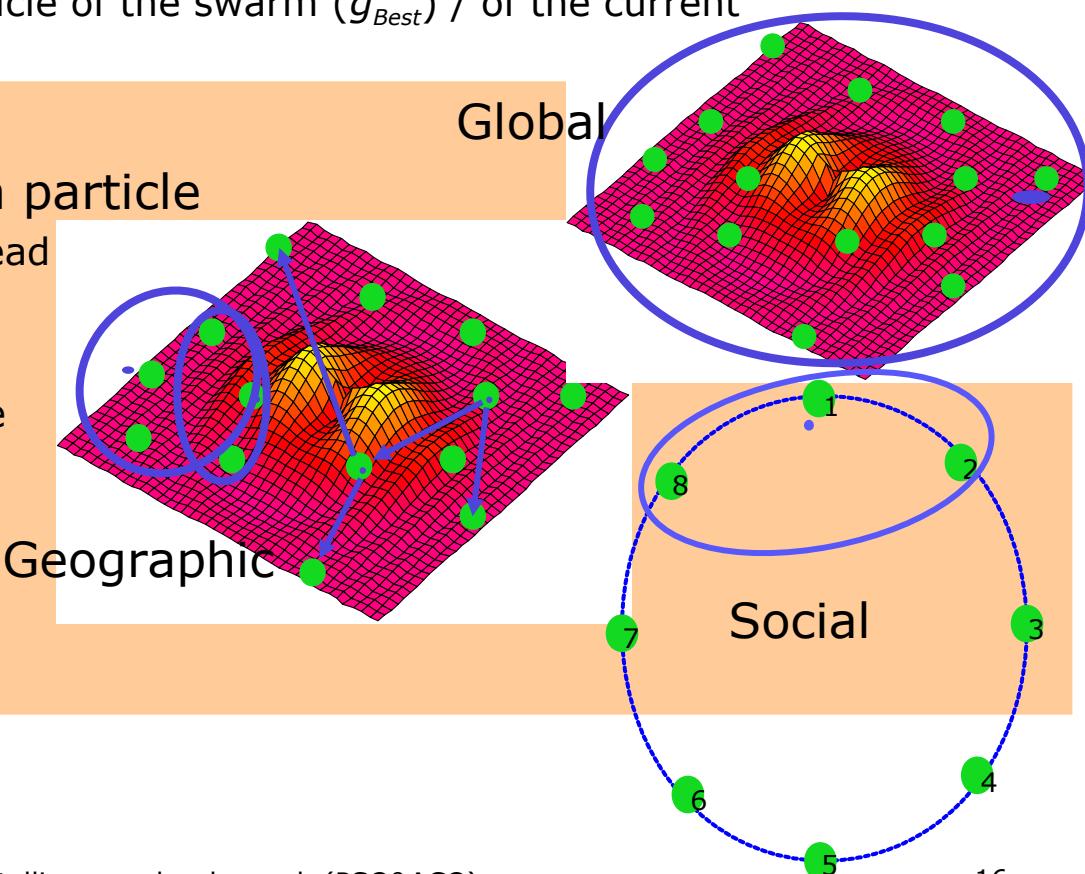
## 2. Evaluation of particles

- Depends on problem

# PSO – algorithm

## 3. For each particle $p_i$

- Update the memory
  - Identify the best particle of the swarm ( $g_{Best}$ ) / of the current neighbourhood ( $I_{Best}$ )
- Neighbourhood for a particle
  - Neighbourhood's spread
    - Global
    - Local
  - Neighbourhood's type
    - Geographic
    - Social
    - Circular



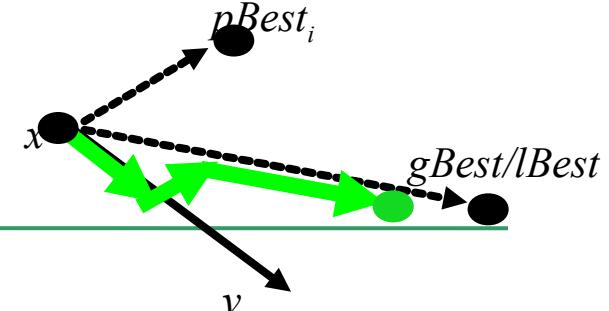
# PSO – algorithm

---

## 3. For each particle $p_i$

- Update the memory
  - Identify the best particle of the swarm ( $g_{Best}$ ) / of the current neighbourhood ( $I_{Best}$ )
  - Identify the best position (with the best fitness) reached until now –  $p_{Best}$

# PSO – algorithm



3. For each particle  $p_i$ 
  - Update the velocity  $\mathbf{v}_i$  and position  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iD})$  (on each dimension)
    - $v_{id} = w * v_{id} + c_1 * rand() * (p_{Best\_d} - x_{id}) + c_2 * rand() * (g_{Best\_d} - x_{id})$
    - $x_{id} = x_{id} + v_{id}$
    - where:
      - $i=1, N$  ( $N$  – total number of particles/swarm size);
      - $d = 1, D$
      - $w$  – inertia coefficient (Shi, Eberhart)
        - $w * v_{id}$  inertial factor → forces the particle to move in the same direction until now (*audacious*)
        - Balance the search between global exploration (large  $w$ ) and local exploration (small  $w$ )
        - Can be constant or descending (while the swarm is getting old)
      - $c_1$  - cognitive learning coefficient
        - $c_1 * rand() * (p_{Best\_d} - x_{id})$  – cognitive factor → forces the particle to move towards its best position (*conservation*)
      - $c_2$  - social learning coefficient
        - $c_2 * rand() * (g_{Best\_d} - x_{id})$  – social factor → forces the particle to move towards the best neighbour (*follower*)
      - $c_1$  and  $c_2$  can be equal or different ( $c_1 > c_2$  și  $c_1 + c_2 < 4$  – Carlisle, 2001)
  - 3. Each component of velocity vector must belong to a given range  $[-v_{max}, v_{max}]$  in order to keep the particles inside the search space

# PSO – properties

---

- PSO principles:
  - Proximity – the group has to perform computing in space and time
  - Quality – the group has to be able of answering to the quality of environment
  - Stability – the group has not to change its behaviour at each environment change
  - Adaptability – the group has to be able of changing its behaviour when the cost on change is not prohibit
- Differences from EAs:
  - There is no recombination operator – information exchange takes place based on particle's experience and based on the best neighbour (not based on the parents selected based on quality only)
  - Position update ~ mutation
  - Selection is not utilised – survival is not based on quality (fitness)
- PSO versions
  - PSO binary and discrete
  - PSO with more social learning coefficients
  - PSO with heterogeneous particles
  - Hierarchical PSO

# PSO – properties

---

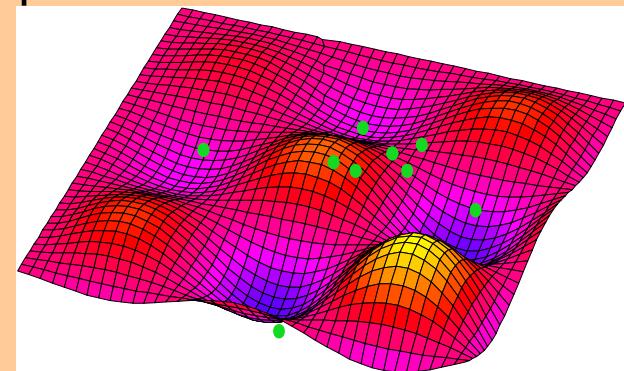
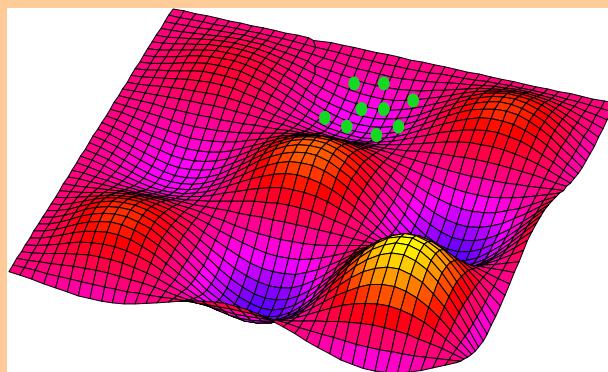
- PSO discrete (binary)
  - PSO version for a discrete search space
  - Position of a particle
    - Possible solution of the problem → binary string
    - Changes based on the velocity of particle
  - Velocity of a particle
    - Element from a continuous space
    - Changes based on standard PSO principles
    - Can be viewed as changing probability of the corresponding bit from the particle's position

$$x_{ij} = \begin{cases} 1, & \text{if } \tau < s(v_{ij}) \\ 0, & \text{otherwise} \end{cases}, \text{ where } s(v_{ij}) = \frac{1}{1 + e^{-v_{ij}}}$$

# PSO – properties

## ❑ Risks

- Particles has the trend to group in the same place
  - ❑ To rapid convergence and the impossibility to escape from local optima
  - ❑ Solution:
    - Re-initialization of some particles



- Particles move through unfeasible regions

# PSO – properties

---

## □ Analyses of PSO algorithm

- Dynamic behavior of the swarm can be analyzed by 2 index:

- Dispersion index

- Measures the spreading degree of particle around the best particle of the swarm
    - Average of absolute distances (on each dimension) between each particle and the best particle of the swarm
    - Explains the cover degree (small or spread) of the search space

- Velocity index

- Measures the moving velocity of the swarm into a iteration
    - Average of absolute velocities
    - Explain how the swarm moves (aggressive or slow)

# PSO – applications

---

- ❑ Control and design of antenna
- ❑ Biological, medical and pharmaceutics applications
  - Analysis of tremor in Parkinson's disease
  - Cancer Classification
  - Prediction of protein structure
- ❑ Network communication
- ❑ Combinatorial optimisation
- ❑ Financial optimisation
- ❑ Image&video analyse
- ❑ Robotics
- ❑ Planning
- ❑ Network security, intrusion detection, cryptography
- ❑ Signal processing

# ACO

---

- Theoretical aspects
- Algorithm
- Example
- Properties
- Applications

# ACO – theoretical aspects

---

- Proposed
  - By Colomi and Dorigo in 1991 for solving discrete optimisation problems – TSP – as a comparison for EAs –  
<http://iridia.ulb.ac.be/~mdorigo/ACO/about.html>
  - Inspired by social behaviour of ants that search a path from their nest and food
  - Why ants?
    - Colony system (from several ants to millions of ants)
    - Labor division
    - Social behaviour is very complex
- Search
  - **Cooperative**, guided by the **relative** quality of individuals
- Search operators
  - Constructive ones, adding elements in solution

# ACO – theoretical aspects

---

## □ Special elements

- The optimisation problem must be transformed into a problem of identifying the optimal path in an oriented graph
- Ants construct the solution by walking through the graph and put pheromones on some edges
- Optimisation method based on
  - Ant colonies ( $\approx$ EAs) that search the optimal solution
  - Cooperation (instead of concurrence like in EAs)
- Each ant:
  - Moves (in the search space) and put some pheromones on its path
  - Memorises the path
  - Selects the path based on
    - The existing pheromones on that path
    - Heuristic information associated to that path
  - Cooperates with other ants through the pheromone trail (that corresponds to a path) that
    - Depends on the solution quality
    - Evaporates while the time is passing

# ACO – theoretical aspects

---

- Natural ants
  - An ant colony start to search some food

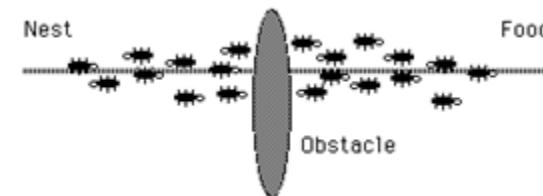


# ACO – theoretical aspects

---

## □ Natural ants

- An ant colony start to search some food
- At a moment, an obstacle appears

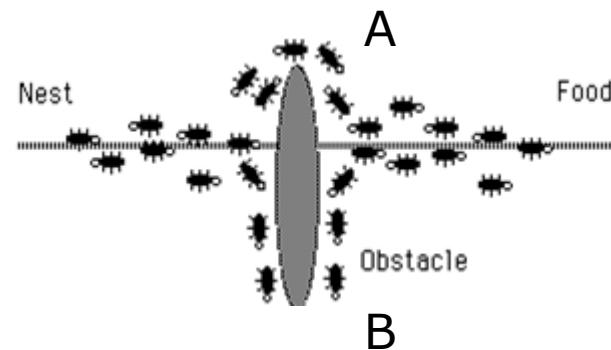


# ACO – theoretical aspects

---

## □ Natural ants

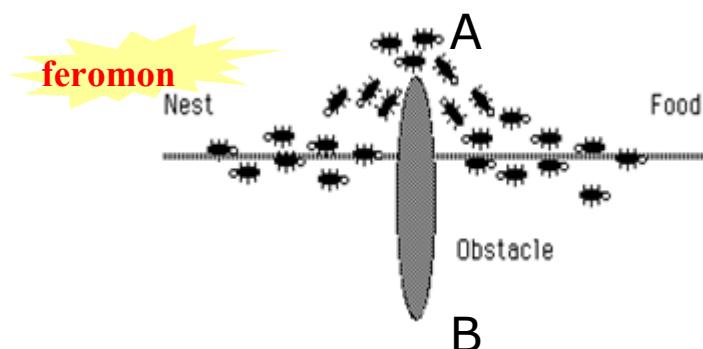
- An ant colony start to search some food
- At a moment, an obstacle appears
- The ants will surround the obstacle either on path A or path B



# ACO – theoretical aspects

## □ Natural ants

- An ant colony start to search some food
- At a moment, an obstacle appears
- The ants will surround the obstacle either on path A or path B
- Because the path A is shorter, the ants of this path will performed more rounds and, therefore, will put more pheromones
- Pheromone concentration will quickly increase on path A (relative to path B) such as the ants from path B will re-oriented to path A
- Because the ants do not follow path B and because the pheromone trail evaporates, the trail of ants from path B will disappear
- Therefore, the ants will take the shortest path (path A)



# ACO – theoretical aspects

---

- Artificial ants look like natural ants
  - Walk from their nest towards food
  - Discover the shortest path based on pheromone trail
    - Each ant performed random moves
    - Each ant put some pheromone on its path
    - Each ant detects the path of “boss ant” and tends to follow it
    - Increasing the pheromone of a path will determine to increase the probability to follow that path by more ants
- But they have some improvements:
  - Has memory
    - Retains performed moves → has a proper state (retaining the history of decisions)
    - Can come back to their nest (based on pheromone trail)
  - Are not completely blind – can appreciate the quality of their neighbour space
  - Perform move in a discrete space
  - Put pheromone based on the identified solution, also

# ACO – theoretical aspects

---

- Pheromone trail plays the role of
  - A collective, dynamic and distributed memory
  - A repository of the most recent ants' experiences of searching food
  
- Ants can indirectly communicate and can influence each-other
  - By changing the chemical repository
  - In order to identify the shortest path from nest to food

# ACO – algorithm

---

- While iteration < maximum # of iterations
  - 1. Initialisation
  - 2. While # of steps required to identify the optimal solution is not performed
    - For each ant of the colony
      - Increase the partial solution by an element (ant moves one step)
      - Change locally the pheromone trail based on the last element added in solution
  - 3. Change the pheromone trail on the paths traversed by
    - all ants/the best ant
  - 4. Return the solution identified by the best ant

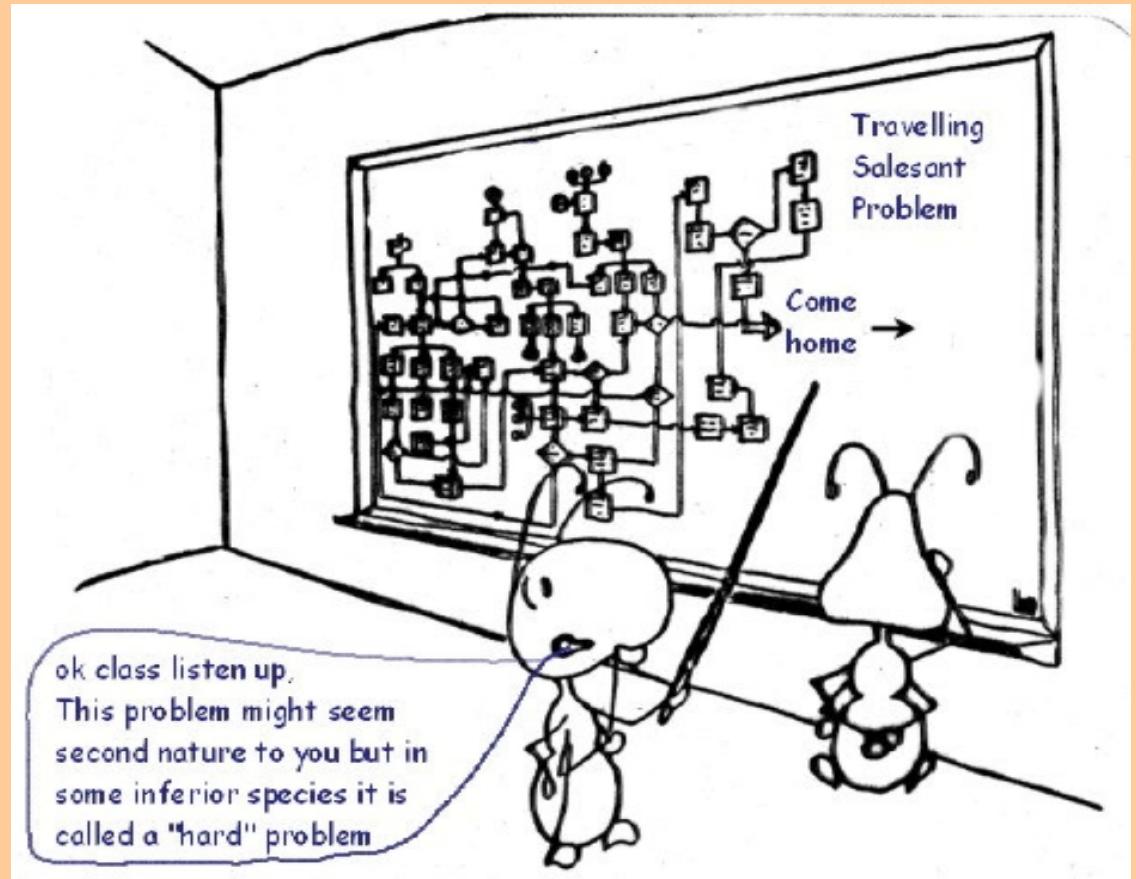
# ACO – algorithm

---

- 3 versions – differences:
  - Rules for transforming a state into another state (moving rules for ants)
  - Moment when the ants deposit pheromones
    - While the solution is constructed
    - At the end of solution's construction
  - Which ant deposits pheromones
    - All the ants
    - The best ant only
- Versions :
  - Ant system (AS)
    - **All** the ants deposit pheromones **after** a solution is **complete** constructed (global collective update)
  - MaxMin Ant System (MMAS)  $\approx$  AS, but
    - The **best** ant only deposits pheromones **after** a solution is **complete** constructed (global update of the leader)
    - Deposited pheromones is **limited** to a given range
  - Ant Colony System (ACO)  $\approx$  AS, but
    - **All** the ants deposit pheromones at **each step** of solution construction (collective local update)
    - The **best** ant only deposits pheromone after the solution is complete constructed (global update of the leader)

# ACO – example

- Travelling salesman problem - TSP
  - Finds the shortest path that visits only once all the n given cities.



# ACO – example

---

## 1. Initialisation:

- $t := 0$  (time)
- For each edge  $(i,j)$  2 elements are initialised:
  - $\tau_{ij}^{(t)} = c$  (intensity of pheromone trail on edge  $(i,j)$  at time  $t$ )
  - $\Delta\tau_{ij} = 0$  (quantity of pheromone deposited on edge  $(i,j)$  by all the ants)
- $m$  ants are randomly places in  $n$  city-nodes ( $m \leq n$ )
- Each ant updates its memory (list of visited cities)
  - Adds in the list the starting city

# ACO – example for TSP

1. While # of steps required to identify the optimal solution is not performed (# of steps = n)
  - For each ant of the colony
    - Increase the partial solution by an element (ant moves one step)
      - Each ant k (from city i) selects the next city j:

$$j = \begin{cases} \arg \max_{l \in \text{permis}_k} \{\tau_{il}\}^\alpha [\eta_{il}]^\beta \}, & \text{if } q \leq q_0 \\ J, & \text{otherwise} \end{cases}$$

Random proportional rule

- where:

- $q$  – random uniform number from  $[0,1]$
- $q_0$  – parameter,  $0 \leq q_0 \leq 1$  ( $q_0 = 0 \rightarrow$  AS/MMAS, otherwise ACO)
- $J$  is a city selected by probability

Pseudo-random proportional rule

$$p_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}^{(t)}]^\alpha [\eta_{ij}]^\beta}{\sum_{s=allowed_k(t)} [\tau_{is}^{(t)}]^\alpha [\eta_{is}]^\beta}, & j - allowed \\ 0, & otherwise \end{cases}$$

where:

- $p_{ij}^k$  – probability of transition of ant k from city i to city j
- $\eta_{ij} = \frac{1}{d_{ij}}$  – visibility from city I towards city j (attractive choice of edge  $(i,j)$ )
- $allowed_k$  – cities that can be visited by ant k at time t
- $\alpha$  – controls the trail importance (how many ants have visited that edge)
- $\beta$  – controls the visibility importance (how close is the next city)

# ACO – example for TSP

---

1. While # of steps required to identify the optimal solution is not performed
  - For each ant of the colony
    - Increase the partial solution by an element (ant moves one step)
    - Change locally the pheromone trail based on the last element added in solution

$$\tau_{ij}^{(t+1)} = (1 - \varphi) \tau_{ij}^{(t)} + \varphi * \tau_0$$

- where:
  - $\varphi$  – pheromone degradation coefficient;  $\varphi \in [0,1]$ ; for  $\varphi = 0 \rightarrow$  AS/MMAS, otherwise ACO
  - $\tau_0$  – initial value of pheromone
  - $(i,j)$  – last edge visited by ant

# ACO – example for TSP

---

3. Change the pheromone trail from
  - **Paths covered by all ants (AS)**
    - For each edge
      - Compute the unit quantity of pheromones put by the  $k^{\text{th}}$  ant on edge  $(i,j)$ 
        - $\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{- if the } k^{\text{th}} \text{ ant used the edge } (i,j) \\ 0 & \text{otherwise} \end{cases}$
        - $Q$  – quantity of pheromone deposited by an ant.
        - $L_k$  – length (cost) of tour performed by the  $k^{\text{th}}$  ant
      - Compute the total quantity of pheromone from edge  $(ij)$  
$$\Delta\tau_{ij} = \sum_{k=1}^m \Delta\tau_{ij}^k$$
      - Compute the intensity of pheromone trail as a sum of old pheromone evaporation and the new deposited pheromone
$$\tau_{ij}^{(t+n)} = (1 - \rho) * \tau_{ij}^{(t)} + \Delta\tau_{ij}$$
    - 3. Where  $\rho$  ( $0 < \rho < 1$ ) – evaporation coefficient of pheromone trail from a complete tour to another complete tour

# ACO – example for TSP

---

3. Change the pheromone trail from
  - **The best path (ACO)**
  - **The best path of the best ant (MMAS)**
  - For each edge of the best path
    - Compute the unit quantity of pheromone deposited by the best ant on edge  $(ij)$ 
      - $\Delta\tau_{ij} = \frac{1}{L_{best}}$
    - $L_{best}$  – length (cost) of the best path
      - Of current iteration
      - Over all executed iteration (until that time)
    - Compute the intensity of pheromone trail as sum of old pheromone evaporation and the new deposited pheromone

$$\tau_{ij}^{(t+n)} = [(1 - \rho) * \tau_{ij}^{(t)} + \rho * \Delta\tau_{ij}^{best}]_{\tau_{min}}^{\tau_{max}}$$

3. Where  $\rho$  ( $0 < \rho < 1$ ) – – evaporation coefficient of pheromone trail from a complete tour to another complete tour

- $\tau_{min}$  și  $\tau_{max}$  – limits (inferior and superior) of pheromone;
  - For  $\tau_{min} = -\infty$  and  $\tau_{max} = +\infty \rightarrow$  ACO, otherwise MMAS

# ACO – properties

---

- Properties
  - Iterative algorithm
  - Algorithm that progressively constructs the solution based on
    - Heuristic information
    - Pheromone trail
  - Stochastic algorithm
- Advantages
  - Run continuous and real-time adaptive change input
    - Ex. for TSP the graph can be dynamically changed
  - Positive feedback helps to quickly discovering of solution
  - Distribute computing avoids premature convergence
  - Greedy heuristic helps to identify an acceptable solution from the first stages of search
  - Collective interaction of individuals
- Disadvantages
  - Slowly convergence vs other heuristic search
  - For TSP instances with more than 75 cities it finds weak solutions
  - In AS there is no central process to guide the search towards good solutions

# ACO – applications

---

- Optimal paths in graphs
  - Ex. Traveling Salesman Problem
- Problems of quadratic assignments
- Problems of network optimisation
- Transport problems



# Review

---

## □ PSO

- Beam local search
- Possible solutions → particles that have:
  - A position in the search space
  - A velocity
- Cooperative and perturbative search based on:
  - Position of the best particle of the swarm
  - Best position of particle (particle has memory)

## □ ACO

- Beam local search
- Possible solutions → ants that have:
  - Memory – retain steps of solution construction
  - Smell – take decisions based on pheromones deposited by other ants (social, collective, collaborative behaviour)
- Cooperative and constructive search

# Next lecture

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

- A. Definition of search problems
- B. Search strategies

- A. Uninformed search strategies
- B. Informed search strategies
- C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
- D. Adversarial search strategies

### C. Intelligent systems

- A. Rule-based systems in certain environments
- B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
- C. Learning systems
  - A. Decision Trees
  - B. Artificial Neural Networks
  - C. Support Vector Machines
  - D. Evolutionary algorithms
- D. Hybrid systems

# Next lecture – useful information

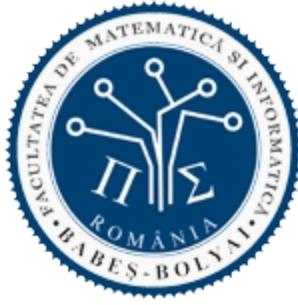
---

- Chapter II.5 of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 6 of *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- Documents from folder *05\_adversial\_minimax*

- 
- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean - [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)



BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics



# ARTIFICIAL INTELLIGENCE

**Solving search problems**

Adversarial search

# Topics

---

A. Short introduction in Artificial Intelligence (AI)

**B. Solving search problems**

**A. Definition of search problems**

**B. Search strategies**

- A. Uninformed search strategies
- B. Informed search strategies
- C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
- D. **Adversarial search strategies**

**C. Intelligent systems**

A. Rule-based systems in certain environments

B. Rule-based systems in uncertain environments (Bayes, Fuzzy)

C. Learning systems

- A. Decision Trees
- B. Artificial Neural Networks
- C. Support Vector Machines
- D. Evolutionary algorithms

D. Hybrid systems

# Useful information

---

- Chapter II.5 of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 6 of *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- Documents from folder *05\_adversial\_minimax*

# Content

---

## Games

- ❑ A bit of history
- ❑ Some theoretical aspects
- ❑ Games and search
  - Definitions, components, topology
  - Strategies and search algorithms



# Games – a bit of history

---

- Challenges of intelligence
  - Checkers (Mesopotamia, cc. 3000 ïHr.)
  - Go (China, sec. VI ï.Hr.)
  - Chess (India, sec. VI d.Hr.)
- Games as search algorithms
  - Classical search – a single agent that tries, without obstacles, to meet its target
  - Games – search in the presence of an adversary (opponent agent) that comes with some uncertainty
- Games and AI
  - Algorithm design and testing
  - One the oldest domain of AI

# Games – a bit of history

---

- Game playing
  - By humans
    - Intelligent activity
    - Comparing the abilities
  - By computers
    - Favorable environment for developing AI techniques
      - Game = a problem that is
        - Structured (success or fail)
        - Solvable by search (simple search or heuristic search)
      - Limits

# Games – some theoretical aspects

---

- Difficult problems with minimal initial knowledge
- States and actions easier to be represented
- Few knowledge about environment is required
- Games are a particular case of search problems
- Frequently, they have huge search spaces
  - Game's complexity → uncertainty
    - Due to insufficient time for computing the consequences of all possible moves
    - Not due to lack of information
- Game are interesting ☺

# Games and search

---

- Formalism
- Game's representation
- Typology
- Search space
  - Representation
  - Exploration methods

# Games and search – formalism

---

- ▣ Solving a search problem
  - Stage x: definition of the search space
    - ▣ Linear spaces
    - ▣ Tree-based spaces
      - Trees
      - Graphs
  - Stage x + 1: selection of a search strategy
    - ▣ Which move/strategy is the best?
      - That wins the game
      - That can be quickly identified (time complexity is reduced)
      - That can be identified by minimal resources (space complexity is reduced)
- ▣ Problem:
  - How the best next move can be identified as soon as possible?
- ▣ One solution:
  - Search among possible moves and their consequences

# Games and search – formalism

---

- Adversarial search is used in games with players that try to maximise their score, but they have one or more opponents
- Game's representation as search problems
  - **States**
    - Board configurations + the player that has to move
    - All the possible states → search tree
  - **Operators (actions, successor functions)**
    - Allowed moves
  - **Initial state**
    - Initial game configuration
  - **Final state**
    - (winning) configuration that ends the game
  - **Utility function (score function)**
    - Associates a numerical value to each state
- Difficulties
  - Games can be difficult problems
    - Simple to be formalised
  - Optimal solution identification can be infeasible
    - A winning solution is acceptable
    - An unacceptable solution has large costs and could cause the defeat

# Games and search – formalism

---

## □ Definitions

### ■ Conflict

- Situation when more parts act and have contrary purposes
- The consequence of a part' action depends on the actions of other parts

### ■ Game

- Simplified modelling of a conflict
- Set of decisions (actions, moves) of the parts

### ■ Move

- A function  $H : \{\text{game' positions}\} \rightarrow \{\text{game' positions}\}$
- If  $p$  – a game position,  $H(p)$  – a new position

### ■ Rules

- System of conditions that regard the possible moves

### ■ Strategy

- Rules that define the free moves of the game for a given game' state

# Games and search – representation

---

- Components
  - Players
  - Moves (strategies)
  - Winning criteria
  
- Game representations
  - Strategic form → matrix
    - Players
    - Strategies
    - Profits for each player and each strategy
  - Expanded form → tree
    - Level → player
    - Node → move

# Games and search – representation



## □ Strategic form → matrix

- Eg. Prisoner's Dilemma: two prisoners are questioned by the police. Police knows something about what they did, but it has not all the information. To find out, the two prisoners are detained in two cells and are questioned. Prisoners have two options:
  - can tell the whole story (can betray one another)
  - can not say anything (can cooperate)
- No prisoner knows what the other will respond. Depending on their answers, the penalties are:
  - If both silent (cooperates), the sentence is easy (one year each)
  - If one talks (betray) and one silent (cooperates), the betrayer is released, and the defendant gets 10 years
  - If both talk (betray each other), the sentence is 5 years each
- What will the two prisoners do?

P1 \ P2	Cooperation	Betrayal
Cooperation	(1,1)	(10,0)
Betrayal	(0,10)	(5,5)

# Games and search – representation



## □ Strategic form → matrix

- Eg. Hunting deer: two guys go hunting. Each can choose to hunt:
  - a deer or
  - a rabbit
- without knowing what the other person chose.
  - Hunting deer is more profitable (gain = 4) and can only be done in two.
  - Hunting rabbits is less valuable (gain = 1) can be done individually.
- What they choose to hunt?

P1 \ P2	Deer	Rabbit
Deer	(4,4)	(1,3)
Rabbit	(3,1)	(3,3)

# Games and search – representation

## □ Strategic form → matrix

### ■ Eg. economic

- Publicity of two companies in the same market
- Agreements level cartel pricing

### ■ Eg. Sport



- Two cyclists are in the front row wearing platoon train (cooperate) to not be overtaken. Often, only one train goes (co) and the finish line is betrayed by his opponent.

### ■ Ex. sociological



When you meet a new person, we tend to be very careful to have a safe position (competition). Both may indicate a desire to move from defensive positions to

- interaction and
- recognition of a common intention

# Games and search – typology

---

- Number of players

- 1
- 2
- More

- Communication between players

- Cooperative
- Non-cooperative

- Game's strategies

- Symmetric
- Asymmetric

# Games and search – typology

---

## □ Player's profit

- Zero sum

- Profit of a player = loss of other players → a player must win or the game ends in a draw

- Non-zero sum

- Profit of a player ≠ loss of other players

## □ Nature of performed moves

- Free moves

- A move is selected from the set of possible moves → deterministic games

- Random moves

- Random factor (dice, playing cards, coins) → non-deterministic games

# Games and search – typology

---

## □ Game's information

### ■ Perfect information

- A player, before to move, knows the consequences of all the other previous moves (its moves and the moves of the other players)
- Usually, the game involves sequential moves

### ■ Imperfect information

- A player does not know all the previous moves

# Games and search – typology

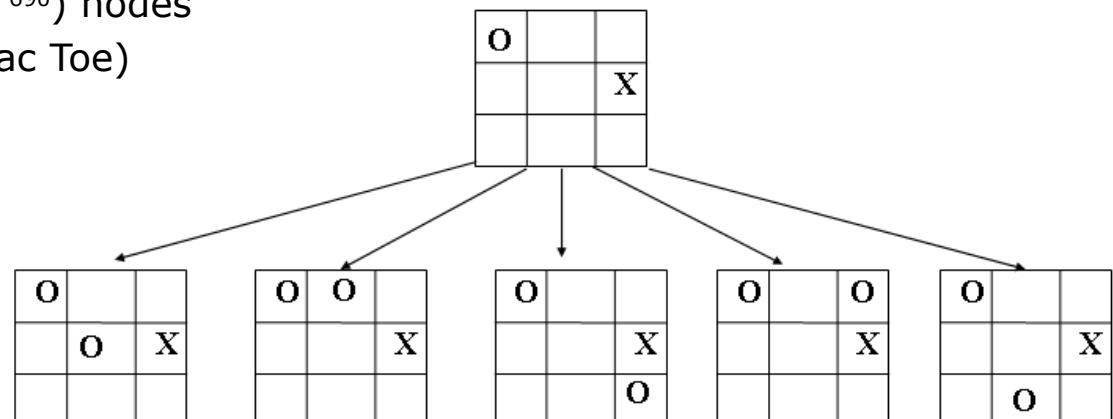
	Deterministic	Non-deterministic (random)
Perfect information		 
Imperfect information	Vaporășe/Avioane/ Submarine	 

# Games and search – search space



## □ Representation

- Linear space
- Tree-based space → game's tree
  - Identify the winning strategy → explore the entire tree
  - Very large for some games
    - Chess (AI drosophila)
      - Ramification factor ≈ 35
      - ≈ 50 moves/player
      - ≈  $35^{100}$  ( $10^{154}$ ) nodes
      - $10^{40}$  different nodes
    - Go
      - ≈ 200 moves/state, 300 levels
      - $200^{300}$  ( $10^{690}$ ) nodes
  - Example – XO (Tic Tac Toe)



# Games and search – search space



## □ Strategy (for a player)

### ■ Definition

- All the moves performed by a player
- Based on
  - Game's rules
  - Current state of game
- ≠ move

### ■ Typology

- Aim
  - Step-by-step strategies
    - At each step, the optimal move is identified
  - Complete strategies
    - A sequence of moves is identified

# Games and search – search space



## □ Game's strategy

### ■ Step by step

- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - Linear structures
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - Tree-based structures
    - AndOr trees
    - MiniMax (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - An optimal path between 2 locations
  - A sequence of actions for moving the player from a location into another location

# Games and search – search space



## □ Game's strategy

### ■ Step by step

- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - **Linear structures**
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - Tree-based structures
    - AndOr trees
    - MiniMax (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - An optimal path between 2 locations
  - A sequence of actions for moving the player from a location into another location

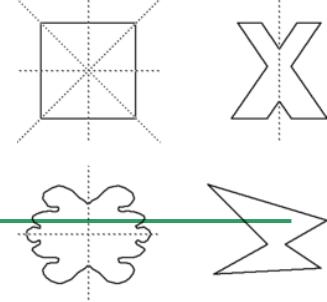
# Games and search – search space

Game's strategy → strategy of symmetry

## □ Main idea

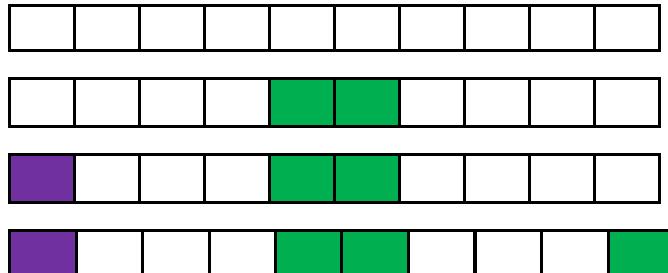
- Player B imitates player A based on a symmetry axis (center)
- If A can move, then B can move also
  - Game ends when A can not move
- It is possible that the first move to not have symmetric

# Games and search – search space

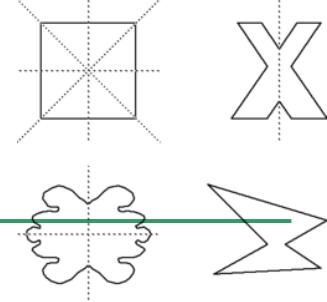


## Game's strategy → strategy of symmetry

- Eg. Game fill the cells
  - We have:
    - A strip of paper is divided into  $N$  cells. Alternatively two players A and B each fill maximum  $k$  adjacent empty cells ( $n$  and  $k$  have the same parity). One who can not move loses. Originally moves the player A.
  - It requires:
    - To determine if the player A has a winning strategy. If so, to program moves of A, the B's being read from the keyboard.
  - Study case:
    - $n = 10, k = 4$

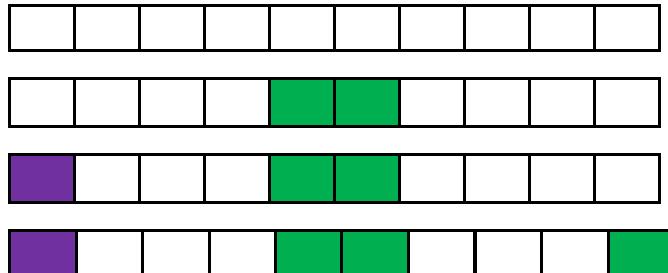


# Games and search – search space



## Game's strategy → strategy of symmetry

- Eg. Game fill the cells
  - We have:
    - A strip of paper is divided into  $N$  cells. Alternatively two players A and B each fill maximum  $k$  adjacent empty cells ( $n$  and  $k$  have the same parity). One who can not move loses. Originally moves the player A.
  - It requires:
    - To determine if the player A has a winning strategy. If so, to program moves of A, the B's being read from the keyboard.
  - Study case:
    - $n = 10, k = 4$
- Solution
  - if  $n$  – even
    - First move → player A fill an even # of cells in the middle of the strip
    - For next moves, player A imitates the moves of player B (taking into account the middle of the strip)
  - If  $n$  - odd
    - First move → player A fill an odd # of cells in the middle of the strip
    - For next moves, player A imitates the moves of player B (taking into account the middle of the strip)



# Games and search – search space



## Game's strategy → strategy of pairs

### □ Main idea

- Generalisation for strategy of symmetry
- Groups the moves in pairs
  - (move of player A, move of player B)
- If A can move, then B can move also
  - Game ends when A can not move
- It is possible that the first move does not have a pair

# Games and search – search space



## Game's strategy → strategy of pairs

### □ Eg. *Sliding squares*

#### ■ It gives:

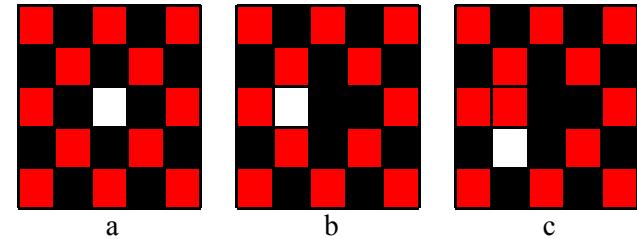
- Each square of a table  $n * n$  ( $n$  - odd) is red or black so the board resembles a checker board. Box in the middle of the sheet is empty (does not contain a square). Alternatively, two players A (red squares) and B (black squares) move one of their squares on the board free square. Square must be initially moved into a free neighbour square (horizontal or vertical). The player that can not move any of his squares loses the game. Player B moves first.

#### ■ It requires:

- Determine if the player A has a winning strategy. If so, program moves to A, the B's being read from the keyboard.

#### ■ Study case:

- $n = 5$  (initial game proposed by G.W.Lewthwaite)





# Games and search – search space

## Game's strategy → strategy of pairs

### □ Eg. *Sliding squares*

#### ■ It gives:

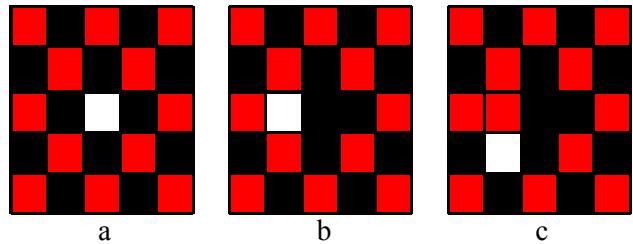
- Each square of a table  $n * n$  ( $n - \text{odd}$ ) is red or black so the board resembles a checker board Box in the middle of the sheet is empty (does not contain a square). Alternatively, two players A (red squares) and B (black squares) move one of their squares on the board free square. Square must be initially moved into a free neighbour square (horizontal or vertical). The player that can not move any of his squares loses the game. Player B moves first.

#### ■ It requires:

- Determine if the player A has a winning strategy. If so, program moves to A, the B's being read from the keyboard.

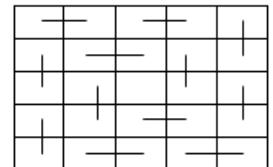
#### ■ Study case:

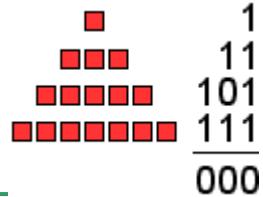
- $n = 5$  (initial game proposed by G.W.Lewthwaite)



#### ■ Solution

- Split the table in Dominoes, the square of the middle (initially free) does not belong to a domino
- After each move of player B, the free square belongs to a domino that covers a red square also. This red square will be moved by player A for its next move (player A has always a square to move)
- Covering the board by dominoes starts from upper-left corner and follows a spiral way





# Games and search – search space

---

## Game's strategy – parity strategy

### □ Main idea

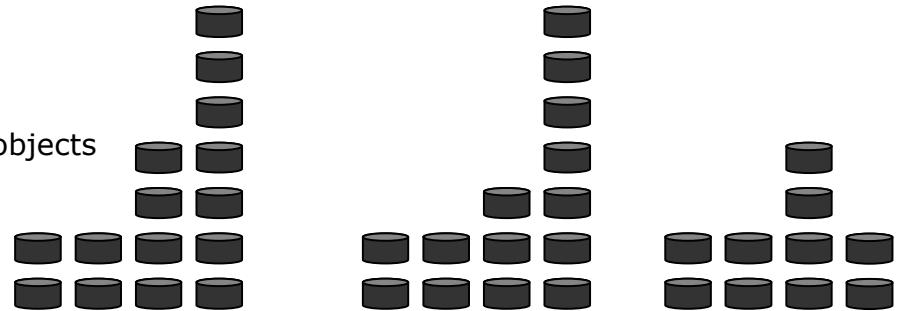
- 2 positions for a game
  - Singular position (even position)
  - Non-singular position (odd position)
- Theorems
  - T1: an odd position is transformed, by a suitable move, into an even position (player A)
  - T2: an even position is transformed, by any move, into an odd position (player B)
- Winner
  - → end position = even position

# Games and search – search space

## Game's strategy – parity strategy

### □ Eg. – NIM game

- It gives:
  - N stacks, each with  $\pi_i$  objects and 2 players A and B that, alternatively, extracts from a single stack a number of objects. The player that performed the last move wins the game. Player A starts the game.
- It requires:
  - Determine if player A has a winning strategy. If it has, program its moves, while B's are read from the keyboard.
- study case:
  - 4 stacks with 2, 2, 4 and 7, respectively, objects



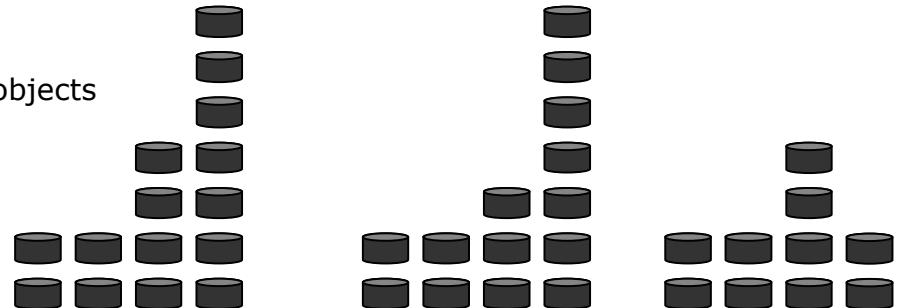
# Games and search – search space

## Game's strategy – parity strategy

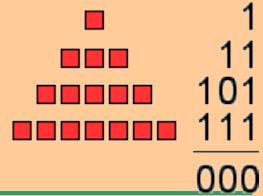
### ❑ Eg. – NIM game

- It gives:
  - N stacks, each with  $p_i$  objects and 2 players A and B that, alternatively, extracts from a single stack a number of objects. The player that performed the last move wins the game. Player A starts the game.
- It requires:
  - Determine if player A has a winning strategy. If it has, program its moves, while B's are read from the keyboard.

- study case:
  - 4 stacks with 2, 2, 4 and 7, respectively, objects



- Solution
  - # of objects of each stack can be binary represented (as sum of 2's powers)
    - Each natural number admits a unique decomposition as a sum of 2's powers
  - Even state – all 2's powers from game representation appear for an even number of times (on all stacks)
  - Off state – any state that is not even
    - T1 → an odd position is transformed, by a suitable move, into an even position (player A)
    - T2 → an even position is transformed, by any move, into an odd position (player B)
  - Select the stack that contains the most significant bit on position k
    - k – position of the most representative bit of the NIM sum (for all the stacks)



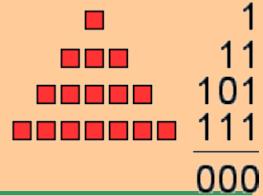
# Games and search – search space

Game's strategy – parity strategy

## □ Eg. – NIM game

### ■ Theorem proving

- $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
- $2^0 - 2, 2^1 - 2, 2^2 - 2 \rightarrow$  even state
- Extract from a stack with an even number of objects
  - An even # of objects (eg. 2 objects from  $S_3$ ) = >
    - $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4-2 = 2^1, S_4: 5 = 2^2 + 2^0$
    - $2^0 - 2, 2^1 - 3, 2^2 - 1 \rightarrow$  odd state
  - An odd # of objects (eg. 1 object from  $S_1$ )
    - $S_1: 2-1 = 2^0, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
    - $2^0 - 3, 2^1 - 1, 2^2 - 2 \rightarrow$  odd state
- Extract from a stack with an odd number of objects
  - An even # of objects (eg. 2 objects from  $S_4$ ) = >
    - $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5-2 = 2^1 + 2^0$
    - $2^0 - 2, 2^1 - 3, 2^2 - 1 \rightarrow$  odd state
  - An odd # of objects (eg. 1 object from  $S_2$ )
    - $S_1: 2 = 2^1, S_2: 3-1 = 2^1, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
    - $2^0 - 1, 2^1 - 2, 2^2 - 2 \rightarrow$  odd state



# Games and search – search space

Game's strategy – parity strategy

- Eg. – NIM game

- Algorithm

- Suppose the game:

- $S_1: 3 = 2^1 + 2^0 \rightarrow 011, S_2: 4 = 2^2 \rightarrow 100, S_3: 5 = 2^2 + 2^0 \rightarrow 101$

- Steps:

1. Compute the NIM sum ( $S_{\text{Nim}}$ ) for all the stacks

- $S_{\text{Nim}} = S_1 \text{ XOR } S_2 \text{ XOR } S_3 = 3 \text{ XOR } 4 \text{ XOR } 5 = 011 \text{ XOR } 100 \text{ XOR } 101 = 010 = 2$

2. Identify the position  $k$  of the most significant 1 in the NIM sum

- $S_{\text{Nim}} = 2 = 010_{(2)} \rightarrow$  second position ( $k = 2$ )

3. Search the stack that contains the most significant bit on position  $k$  (the stack that decreases if XOR is applied between it and the NIM sum)

- $S_1: 3 = 010, S_2: 4 = 100, S_3: 5 = 101 \Rightarrow S_1$  or

- $S_1 \text{ XOR } S_{\text{Nim}} = 3 \text{ XOR } 2 = 011 \text{ XOR } 010 = 001 = 1$

- $S_2 \text{ XOR } S_{\text{Nim}} = 4 \text{ XOR } 2 = 100 \text{ XOR } 010 = 110 = 6$

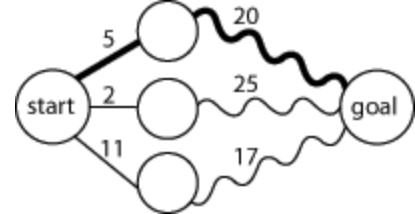
- $S_3 \text{ XOR } S_{\text{Nim}} = 5 \text{ XOR } 2 = 101 \text{ XOR } 010 = 111 = 7 \rightarrow S_1$

4. Extract from this stack some objects such as the other stacks form an even state; # of object that remains on that stack = that stack XOR NIM sum

- $S_1 \text{ XOR } S_{\text{Nim}} = 3 \text{ XOR } 2 = 011 \text{ XOR } 010 = 001 = 1 \rightarrow 2 (=3 - 1)$  objects are taken from stack 1

5. Goto step 1

# Games and search – search space

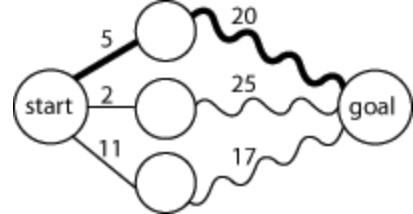


## Game's strategy – dynamic programming (DP)

### □ Main idea

- Steps for problem solving by DP:
  - Decompose the problem in more sub-problems
  - Solve the sub-problems
  - Combine the sub-solutions in order to obtain the final solution
  
- Steps for game solving by DP:
  - Decomposed the game in more sub-games
  - Identify a perfect winning strategy for each sub-game
  - Combine the sub-strategies in order to obtain the final strategy

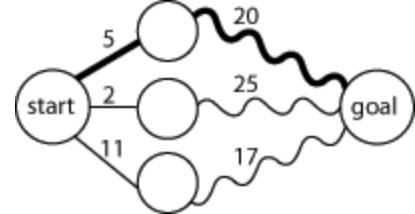
# Games and search – search space



## Game's strategy – dynamic programming (DP)

- Main idea
  - How a game is solved by DP?
    - Decompose the game in sub-games  $G_h$  (lowest level sub-games) and
      - Label each game  $G_h$  by T (true) or F (false) based on the probability of the player (that has to move) to have a winning strategy
    - A sub-game  $G_k$  from an interior level ( $k < h$ ) will be labelled by:
      - T, if there is at least a sub-game  $G_i$ ,  $k < i$ , labelled by F, that can be reached by a move from game  $G_k$
      - F, if all sub-games  $G_i$ ,  $k < i$ , that can be reached by a move from the game  $G_k$  are tagged with T

# Games and search – search space



## Game's strategy – dynamic programming (DP)

### ❑ Eg. – array of cells

- It gives:

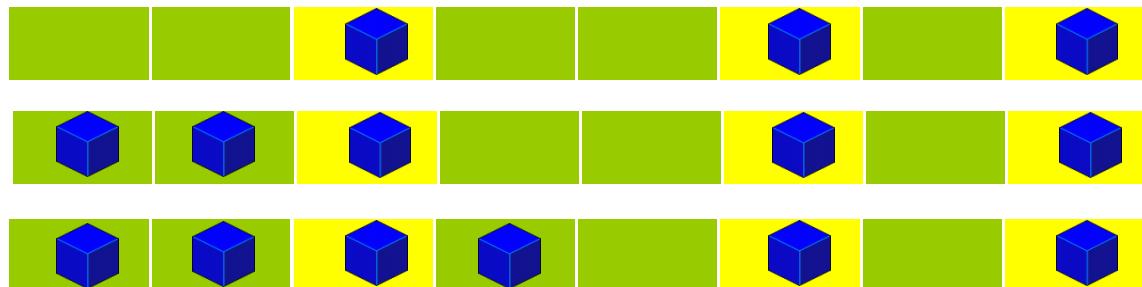
- An array of  $n$  cells that can contain cubes (maximum one cube in a cell). The goal of the game is to fill all the cells by cubes. 2 players, alternatively, fill by cubes (complete or partially) the most left sub-array of empty cells. The player that can not move losses the game. First player is A.

- It requires:

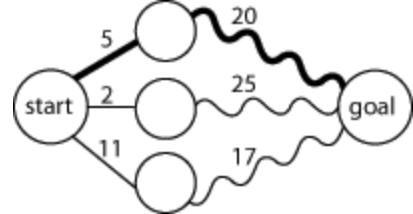
- Determine if the player A has a winning strategy, and if it has, that program the A's moves, that of B's being read from keyboard.

- Example

- $n = 8$



# Games and search – search space



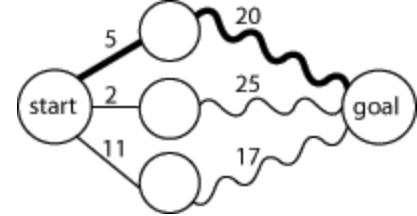
Game's strategy – dynamic programming (DP)

- Eg. – array of cells

- Solution:



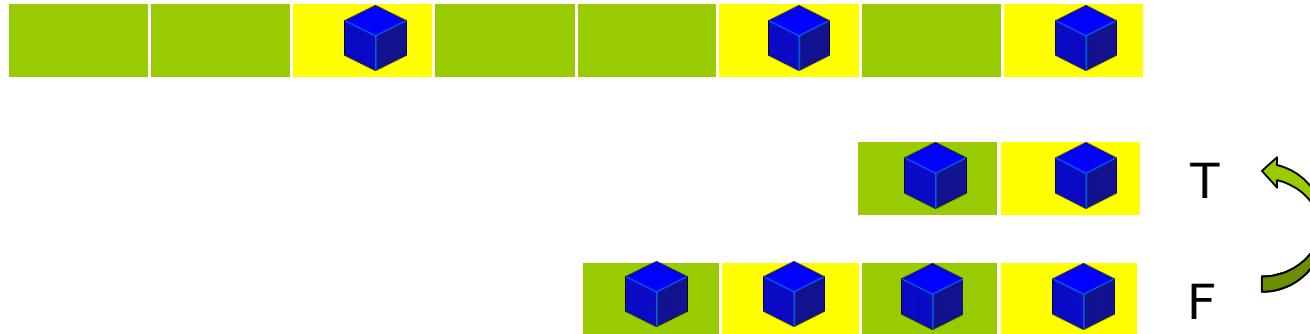
# Games and search – search space



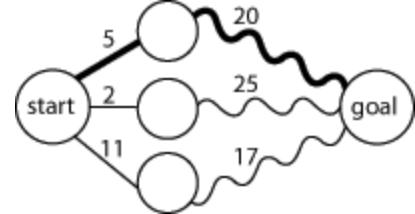
Game's strategy – dynamic programming (DP)

- Eg. – array of cells

- Solution:



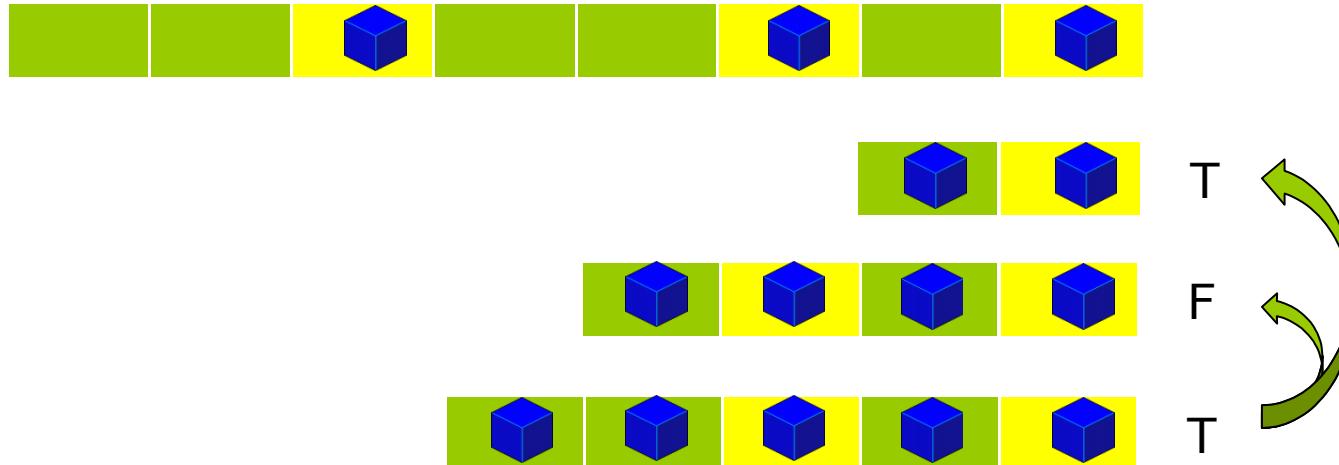
# Games and search – search space



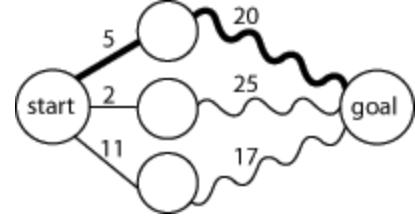
Game's strategy – dynamic programming (DP)

- Eg. – array of cells

- Solution:



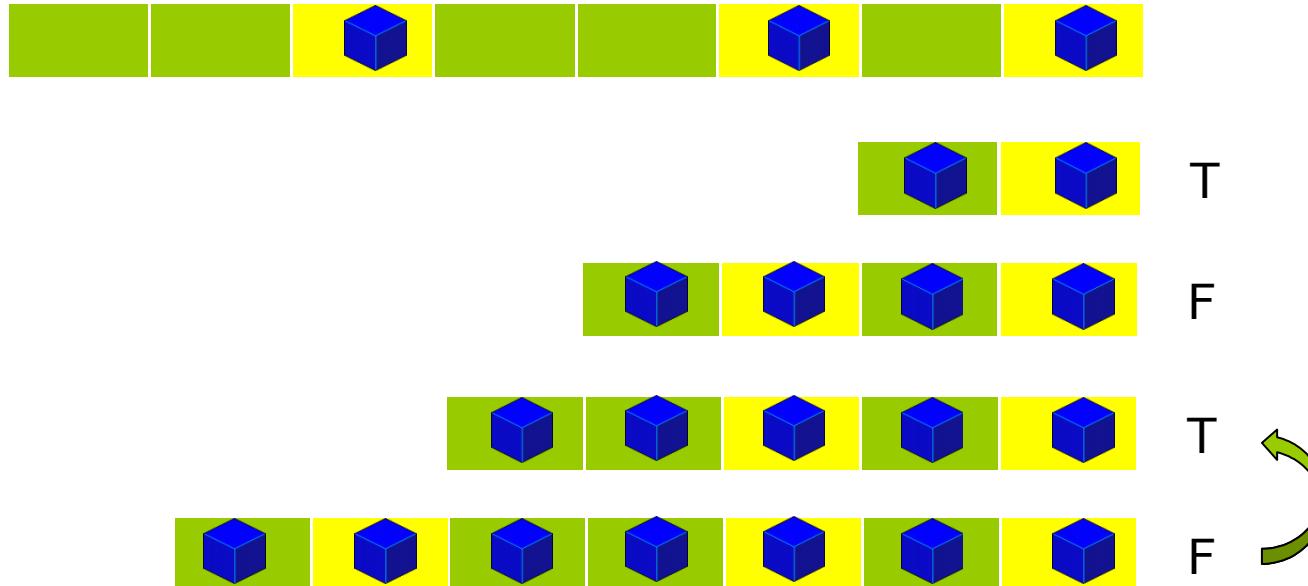
# Games and search – search space



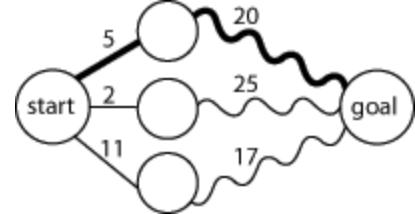
Game's strategy – dynamic programming (DP)

- Eg. – array of cells

- Solution:



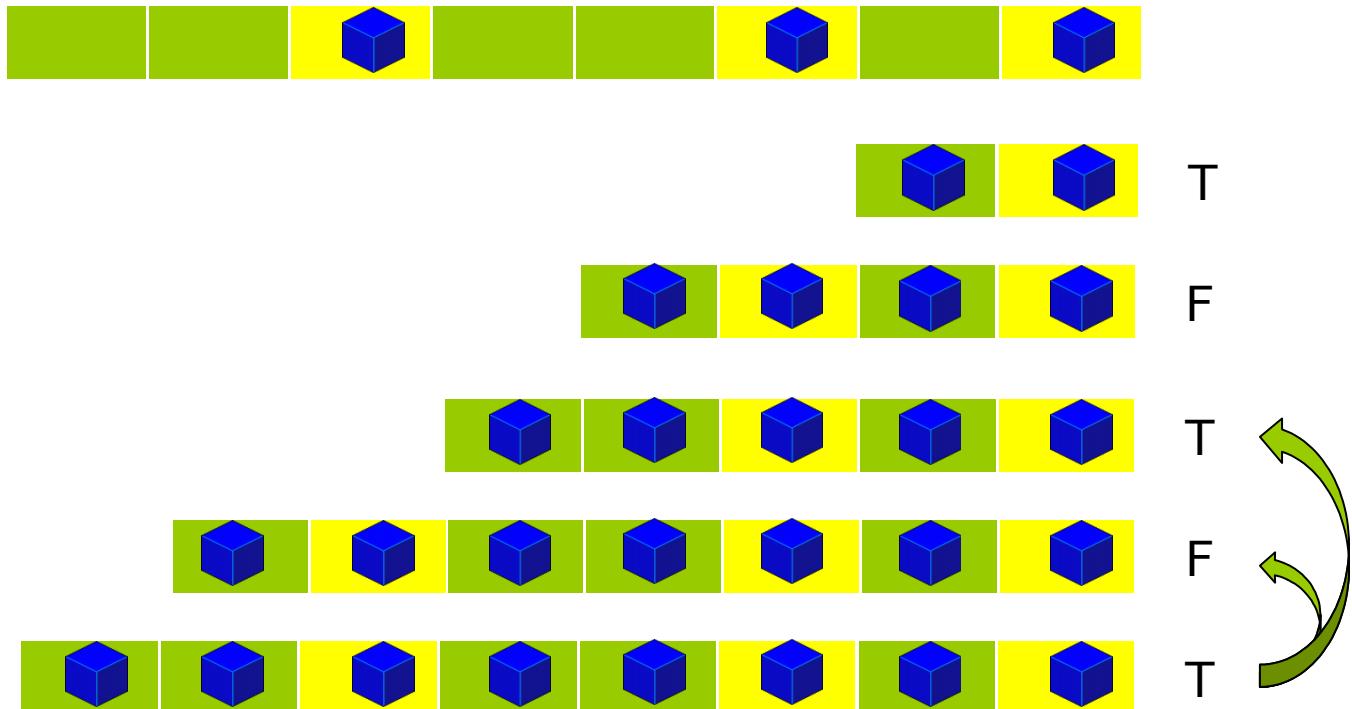
# Games and search – search space



Game's strategy – dynamic programming (DP)

- Eg. – array of cells

- Solution:



# Games and search – search space



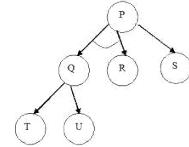
## □ Game's strategy

### ■ Step by step

- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - Linear structures
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - **Tree-based structures**
    - **AndOr trees**
    - MiniMax (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - An optimal path between 2 locations
  - A sequence of actions for moving the player from a location into another location



# Games and search – search space

---

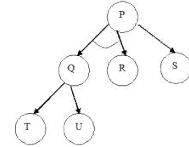
## Game's strategies – AndOr trees (AOT)

### □ Main idea

- **Search space representation** by help of AOTs for a 2 players game
  - OR nodes
    - → select a move for the current player A
    - → there is a possibility (a move) for player A to move in an AND node
  - AND nodes
    - → consider all the possible moves of the opponent
    - → by any move, player B moves in an OR node
  - Other meaning:
    - Root → problem of the winning player (that starts the game)
    - Possible moves for player A are combined by dis-junction (OR)
    - Possible moves of player B are combine by conjunction (AND)
  - The game can be won if it starts by an OR node

### ■ Difficulties

- To large trees

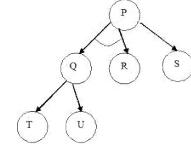


# Games and search – search space

## Game's strategies – AndOr trees (AOT)

### □ Main idea

- Steps for solving a game by AOTs
  - Decompose the game in more sub-games
    - Nodes on more levels
    - Nodes of a level correspond to the possible moves of a player
  - Identify a perfect winning strategy for each sub-game (node)
    - Label the nodes by T or F, depending on the possibility of player A to have a winning strategy for that sub-game
    - Labeling rules
      - Leaves are labeled by T or F depending on game configuration
      - Internal nodes are labeled by:
        - T (for player A) if there is at least one child node labeled by T – OR rule
        - T (for player B) if all the children are labeled by T – AND rule
  - Combine the sub-strategies in order to obtain the final strategy



# Games and search – search space

## Game's strategies – AndOr trees (AOT)

### □ Example

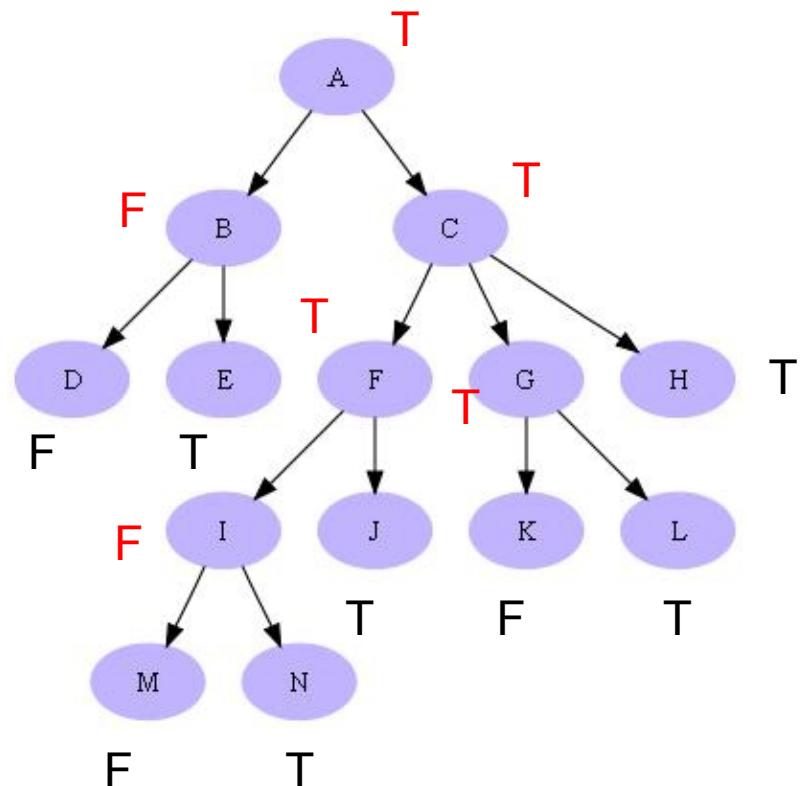
A (OR)

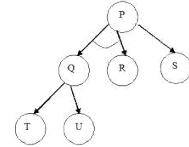
B (AND)

A (OR)

B (AND)

A (OR)





# Games and search – search space

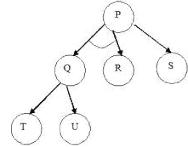
## Game's strategies – AndOr trees (AOT)

### Algorithm

```

bool backAndOr(Node N, int level){
    //level = 0 for the node in the top of the tree.
    if (N is a terminal) {
        if (the first player won)
            return true;
        else
            return false;
    }
    else{
        if (level % 2){           //B is about to move; AND
            result = true;
            for each child Ni of N
                result = result && backAndOr(Ni, level+1);
        }
        else{                   // A is about to move; OR
            result = false;
            for each child Ni of N
                result = result || backAndOr(Ni, level+1);
        }
        return result;
    }
}

```



# Games and search – search space

## Game's strategies – AndOr trees (AOT)

### □ Advantages

- Can be applied for solving any game

### □ Disadvantages

- Require a large memory
- Require a large computing time

### □ → mini-max algorithm

# Games and search – search space



## □ Game's strategy

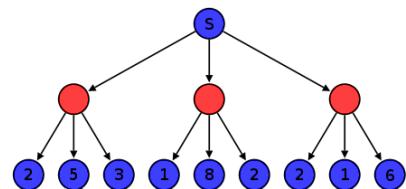
### ■ Step by step

- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - Linear structures
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - **Tree-based structures**
    - AndOr trees
    - **Minimax** (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - An optimal path between 2 locations
  - A sequence of actions for moving the player from a location into another location

# Games and search – search space

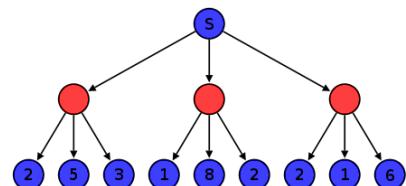


## Game's strategies → MiniMax

### □ Theoretical aspects

- Proposed by John von Neuman in 1944
- Main idea: maximize the position of a player, while the position of the opponent is minimizing
- The search tree = levels for profit maximization of one player's profit alternate with levels for profit minimization of the opponent
  - First player tries to maximize its profit (MAX player -> first move)
  - Second player tries to minimize the gain of the first player (MIN player → the opponent)
- Identify the best moves
  - Construct the tree for all the possible moves (a move for each state of the game)
  - Evaluate the leaf (where a player wins)
  - Propagate down-to-top the profits
  - Tree exploration respects DFS method
- Generate all the moves
  - Possible only for simple games (ex. Tic-Tac-Toe)
  - Impossible for complex games

# Games and search – search space



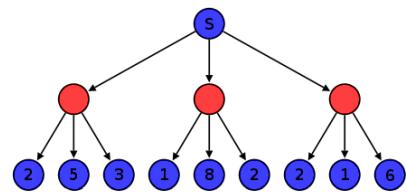
## Game's strategies → MiniMax

### Algorithm

- Construct the tree of all possible moves
- Evaluate the leaf
- While it is possible to select a node for whose children were evaluated
  - Choose a node
  - If the node is on a Min level, then it will be evaluated to the value of the minimum children
  - If the node is on a Max level, then it will be evaluated to the value of the maximum children
- Return the root's value (root is on a Max level)

```
int backMiniMax(Node N, int level){  
    //level = 0 for the node in the top of the tree.  
    if (level == MaxLevel)  
        return the quality of N computed with a heuristic;  
    else if (level < MaxLevel){  
        if (level % 2){  
            //B is about to move; minimize  
            result = MaxInt;  
            for each child Ni of N  
                result = minim(result, backMiniMax(Ni, level+1));  
        }  
        else{  
            // A is about to move; Maximize  
            result = -MaxInt;  
            for each child Ni of N  
                result = maxim(result, backMiniMax(Ni, level+1));  
        }  
        return result;  
    }  
}
```

# Games and search – search space

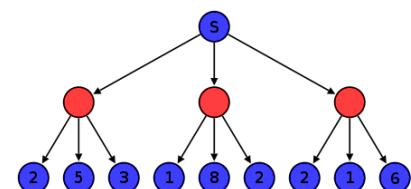


Game's strategies → MiniMax

- Example - *Tic-Tac-Toe* game

- Evaluation of a node:

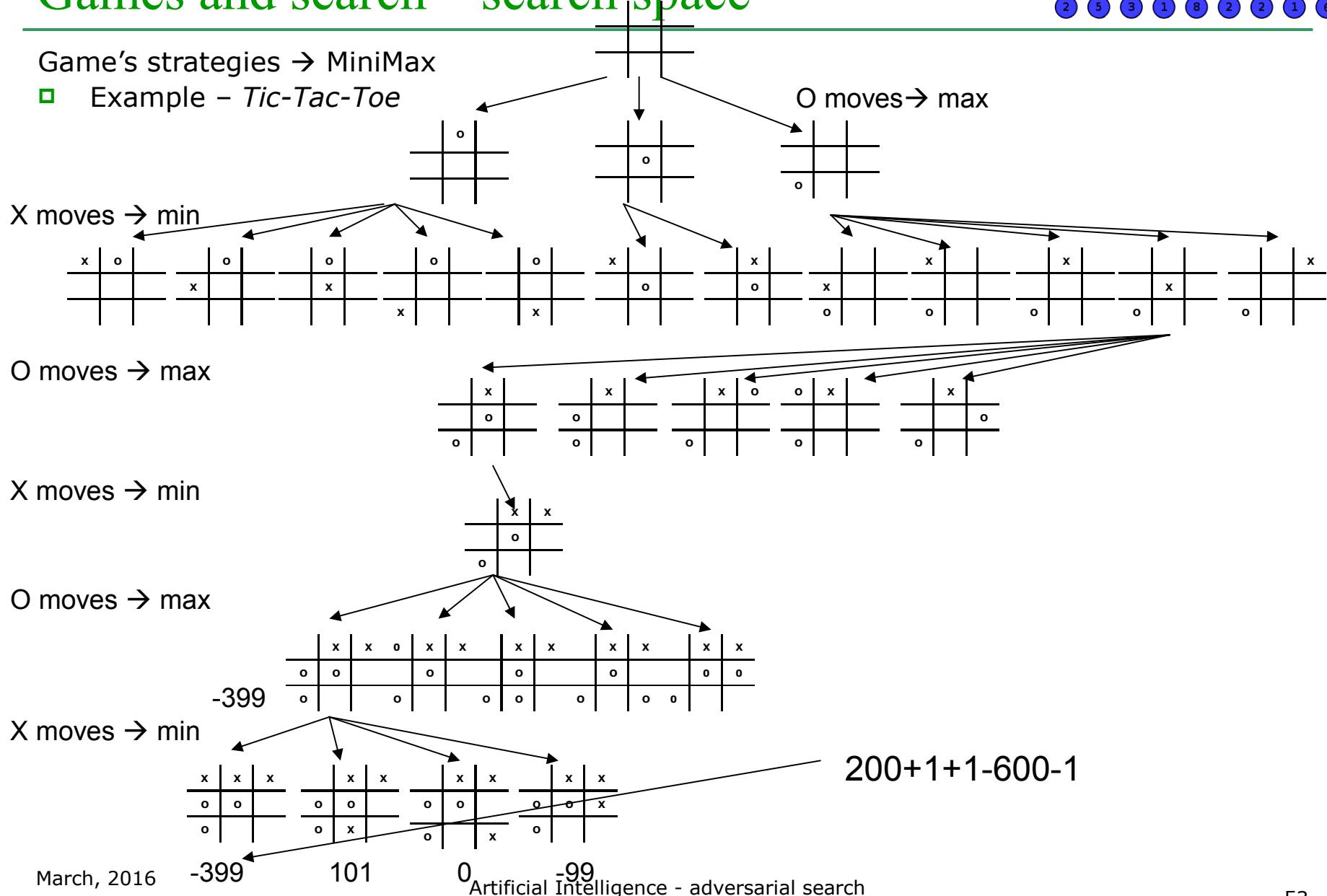
- If a line is almost complete (2 equal signs) → 200 points
- If there are 2 almost complete lines (2 equal signs) → 300 points
- A complete line → 600 points
- A potential line (a sign) → 1 point
- Points of the player above to move are added
- Points of the other player are subtracted



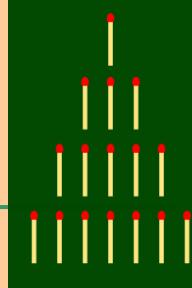
# Games and search – search space

Game's strategies → MiniMax

- Example – Tic-Tac-Toe



# Games and search – search space



Game's strategies → MiniMax

## □ Example – NIM game

### ■ Input :

- N stacks, each of them having  $\pi_i$  objects. 2 players A and B extract, alternatively, from a single stack any number of objects. The player that performs the last move wins the game.  
Player A moves first.

### ■ Output:

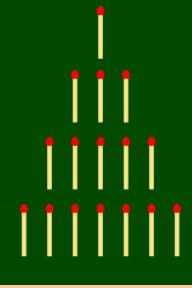
- Determine if player A has a winning strategy and, if it has, program the A's moves, while those of B's are read from keyboard.

### ■ Study case

- 3 stacks with 1,1 and 2, respectively, objects



# Games and search – search space



Game's strategies → MiniMax

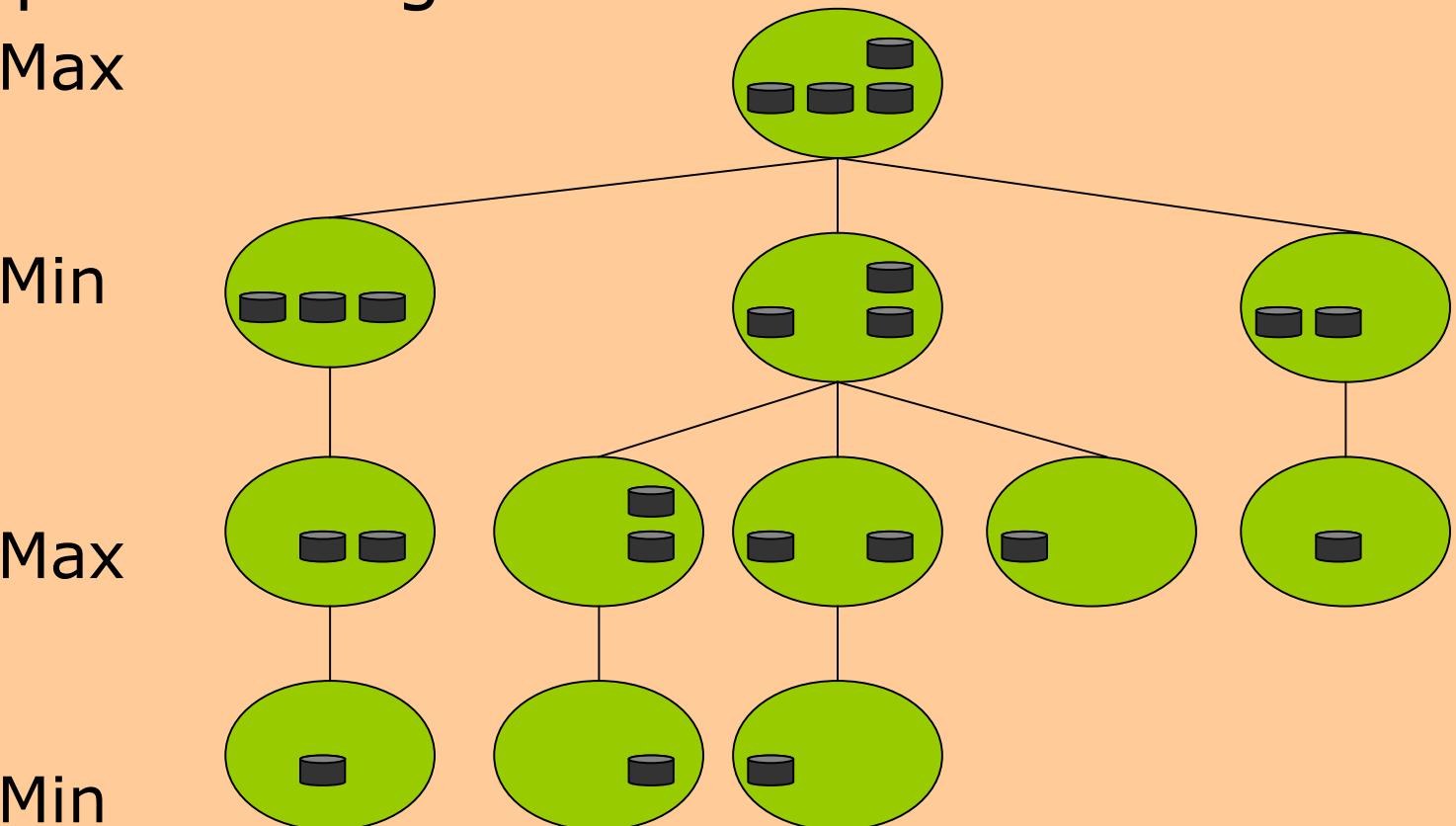
□ Example – NIM game

Max

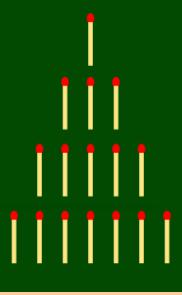
Min

Max

Min



# Games and search – search space



Game's strategies → MiniMax

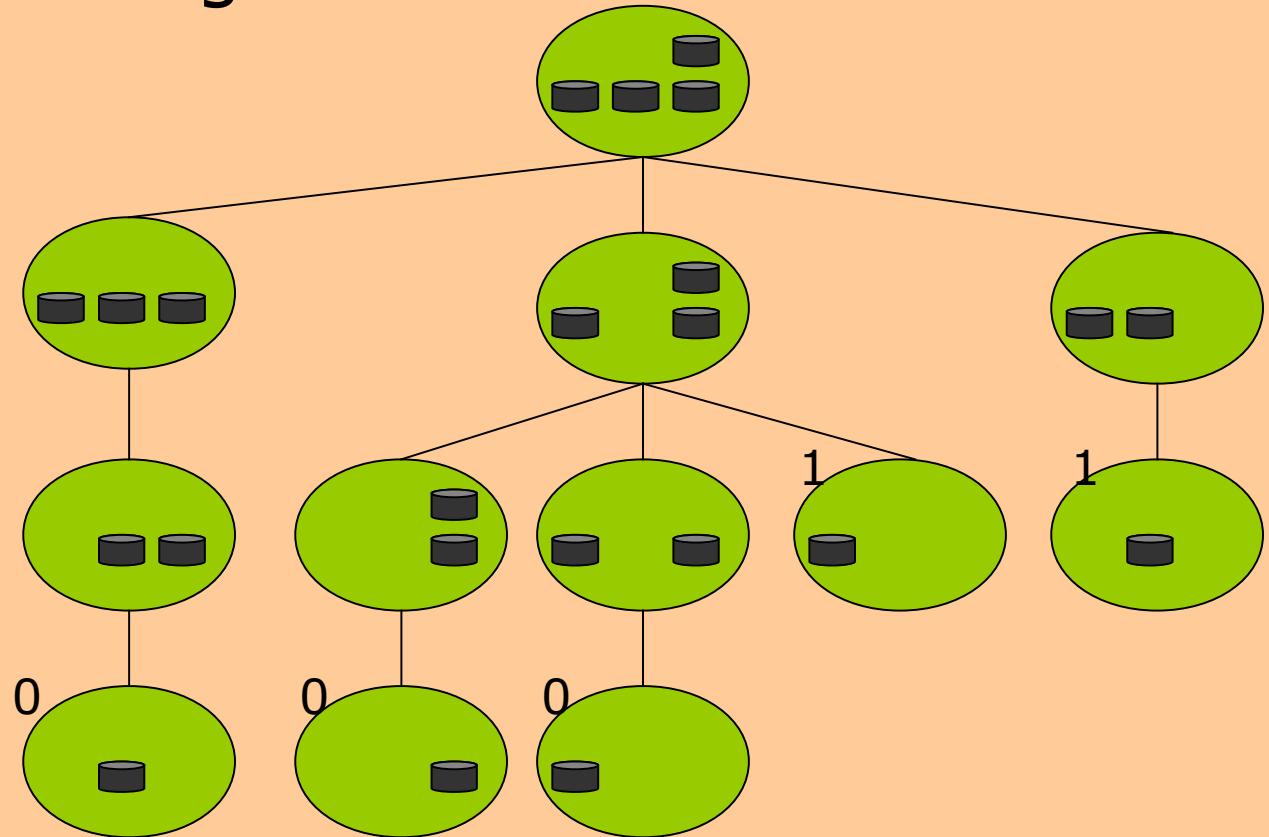
- Example – NIM game

Max

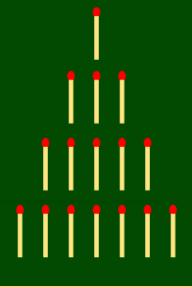
Min

Max

Min



# Games and search – search space



Game's strategies → MiniMax

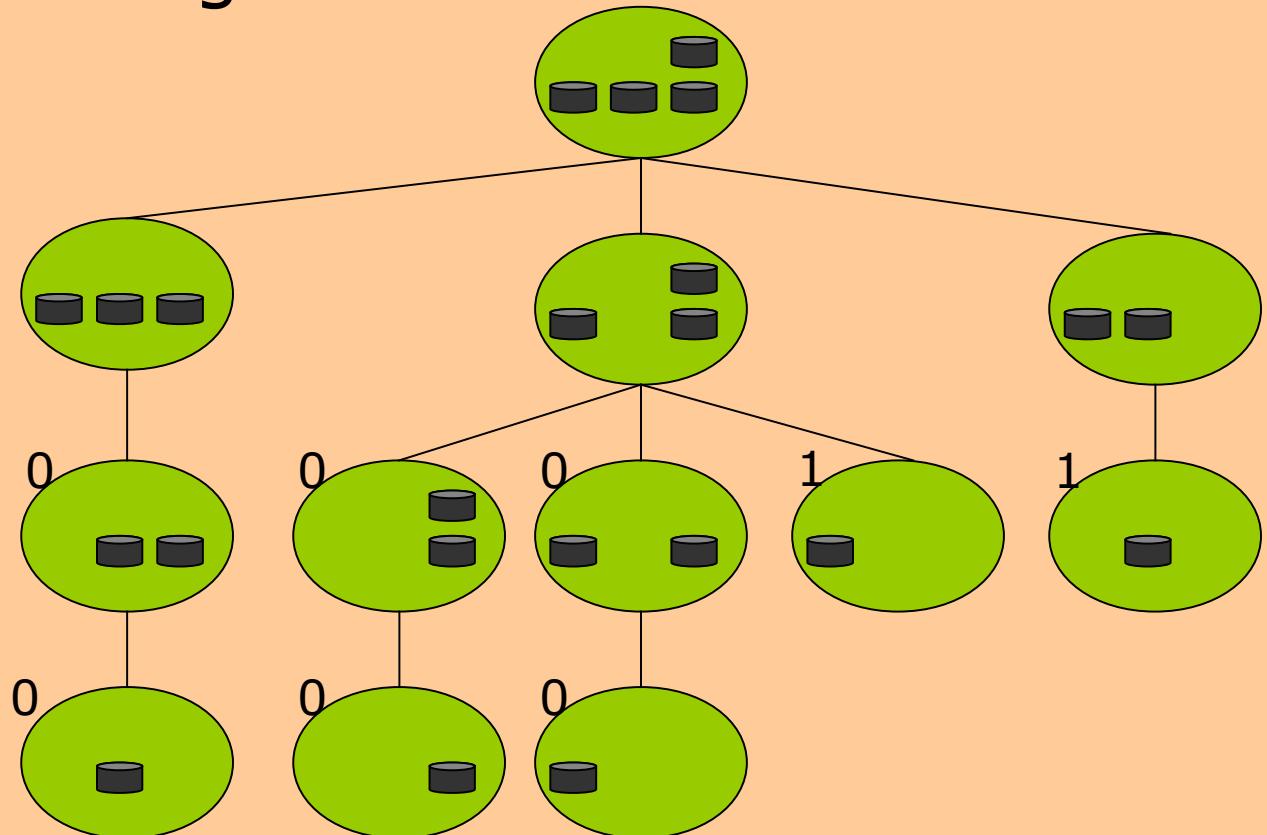
- Example – NIM game

Max

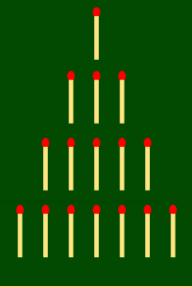
Min

Max

Min



# Games and search – search space



Game's strategies → MiniMax

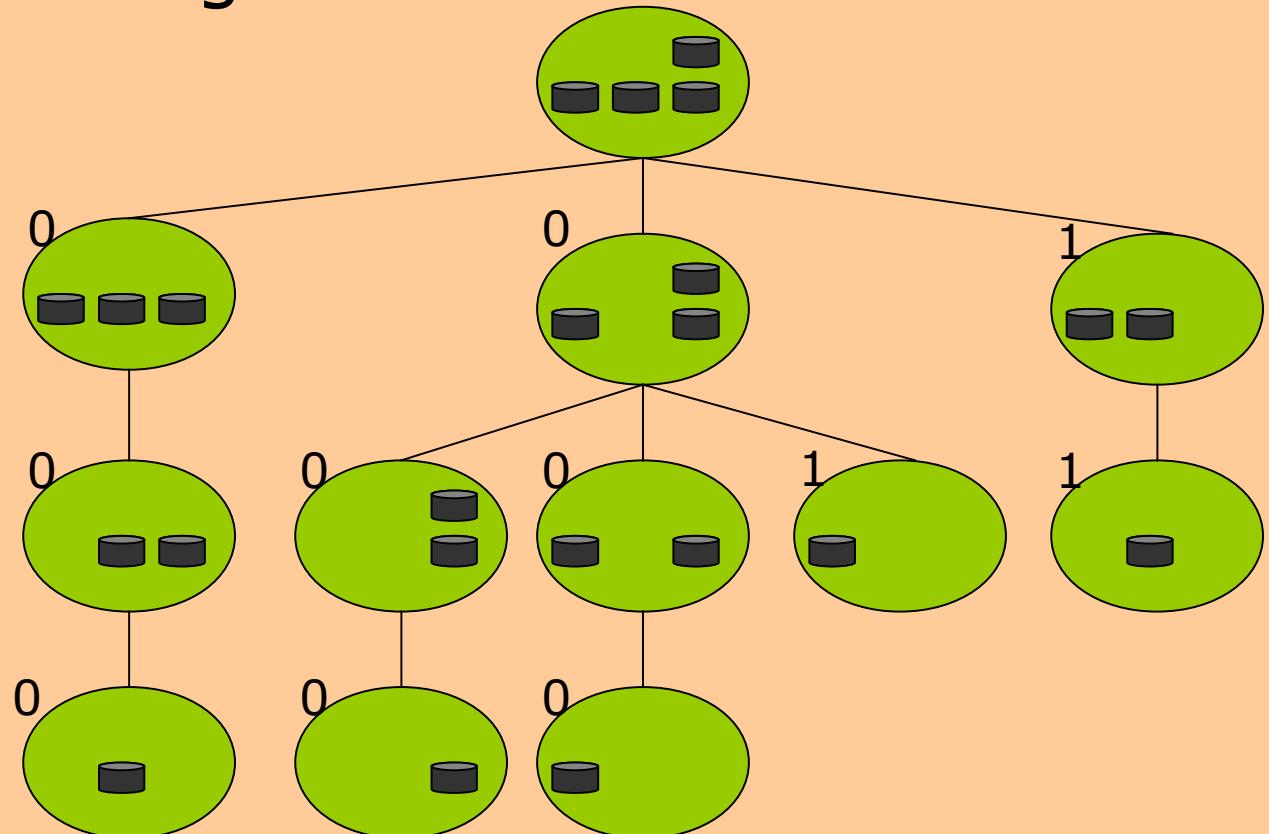
- Example – NIM game

Max

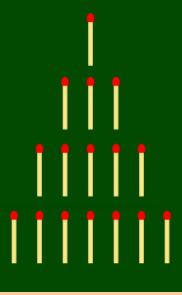
Min

Max

Min



# Games and search – search space



Game's strategies → MiniMax

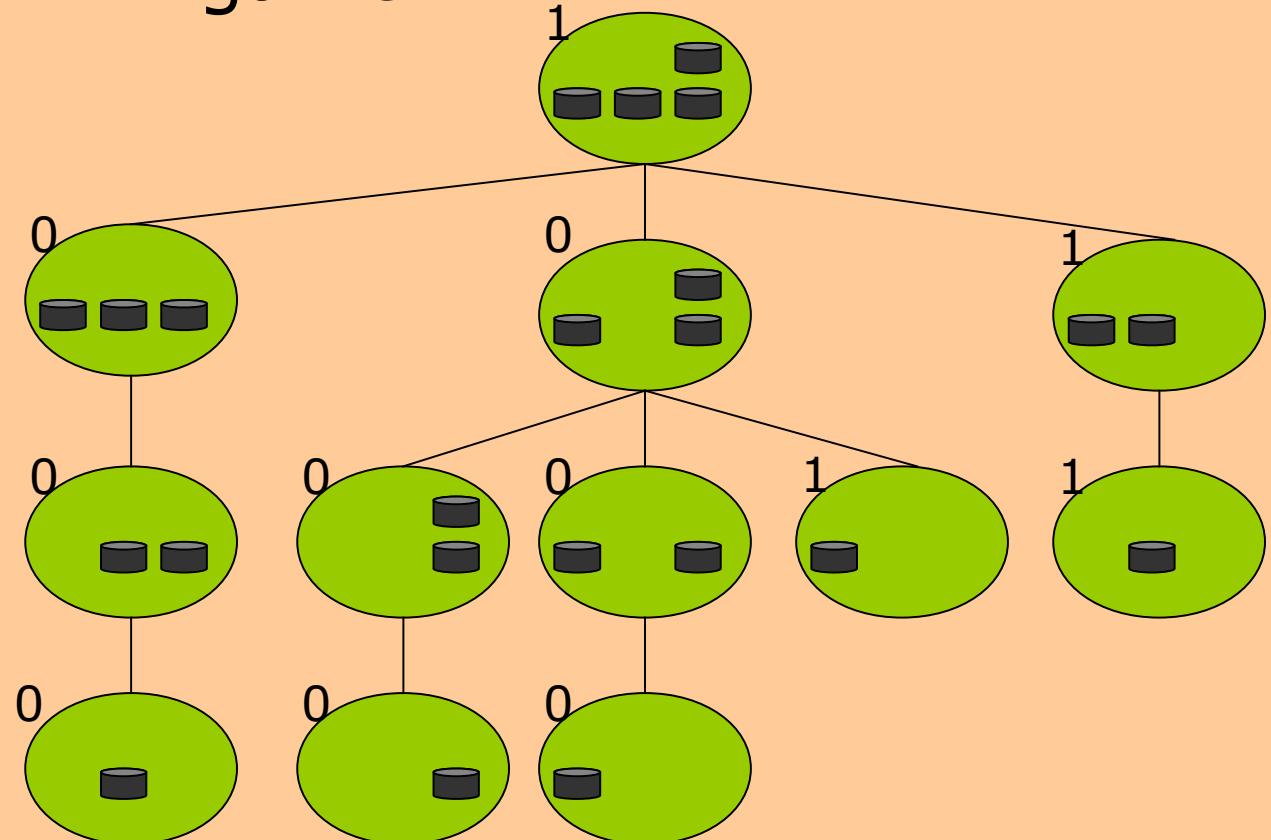
- Example – NIM game

Max

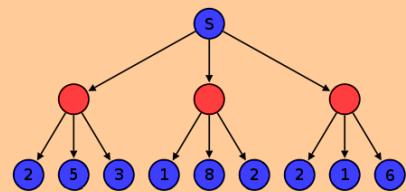
Min

Max

Min



# Games and search – search space

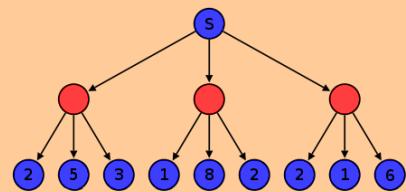


Game's strategies → MiniMax

## □ Difficulties

- Not explore the entire tree
  - Depth-first search
- When the game starts, we have to fix
  - A maximal depth
    - Which is the optimal depth?
      - Large depth → long time to search
      - Small depth → some paths can be missed (early sacrifices for later gains)
  - An evaluation function
    - Each node (state) must be evaluated
    - How? → by using heuristic functions

# Games and search – search space



Game's strategies → MiniMax

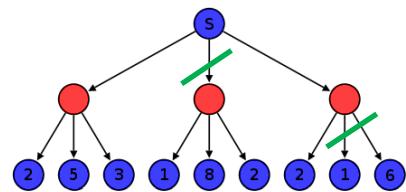
## □ Disadvantages

- Only 100 nodes are explored in a second
- Chess – move time = 150s → 150 000 positions  
→ 3-4 move in advance, only

## □ Solution

- Avoid some nodes by pruning → alpha-beta reducing → MiniMax with  $\alpha$ - $\beta$  pruning

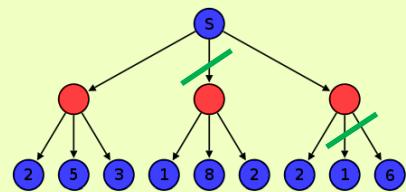
# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

- MiniMax
  - Creates the entire tree
  - Bottom-up propagation of values
- MiniMax with  $\alpha$ - $\beta$  pruning
  - Simultaneously create and propagate
  - If a path is bad, no energy is consumed in order to establish how bad is that path
    - Some useless evaluations can be eliminated
    - Some node's expansions can be avoided

# Games and search – search space

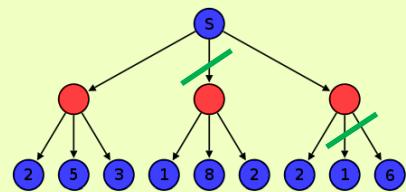


Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

## □ Theoretical aspects

- Discovered by McCarthy in 1956, but published for first time in a technical report of MIT
- Main idea
  - MiniMax value of the root can be determined by evaluating all the nodes on the searching frontier
  - Similarly to branch and bound algorithm
- $\alpha$ - $\beta$  calling
  - $\alpha$  - value of the best selection (largest selection) done over the MAX path
    - If a node has a value weaker than  $\alpha$ , then MAX will avoid it and will cut the sub-tree rooted in that node
  - $\beta$  - similarly of  $\alpha$ , but for MIN player

# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

## □ Theoretical aspects

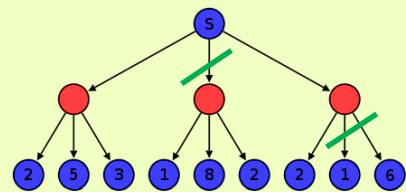
### ■ $\alpha$

- Value of the best selection (largest selection) performed until now over its path by the MAX player
- Minimum score that can be obtained by MAX player
- If a node  $v$  is weaker (less) than  $\alpha$ , MAX will avoid it by eliminating that branch  
→
  - If the search is in a MIN node whose value  $\leq \alpha$ , then all children of that node that are not worth exploring because they will be ignored anyway by MAX player

### ■ $\beta$

- The smallest value found until now over its path by MIN player
- Minimum score that can MAX player hopes to obtain
- If a node  $v$  is better (greater) than  $\beta$ , MIN will avoid it by eliminating that branch  
→
  - If the search is in a MAX node whose value  $\geq \beta$ , then all children of that node that are not worth exploring because they will be ignored anyway by MIN player

# Games and search – search space

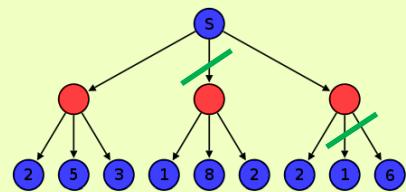


Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

## □ Theoretical aspects

- $\alpha$  value of a node
  - Initially, the score of that node (if the node is a leaf) or  $-\infty$
  - Than,
    - MAX node = highest score of children
    - MIN node =  $\alpha$  value of the predecessor
- $\beta$  value of a node
  - Initially, the score of that node (if the node is a leaf) or  $+\infty$
  - Than,
    - MIN node = smallest score of children
    - MAX node =  $\beta$  value of the predecessor
- Score of a node
  - MAX node  $\rightarrow$  final  $\alpha$  value
  - MIN node  $\rightarrow$  final  $\beta$  value

# Games and search – search space

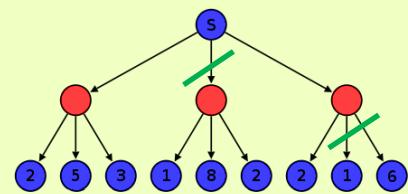


Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

## Algorithm

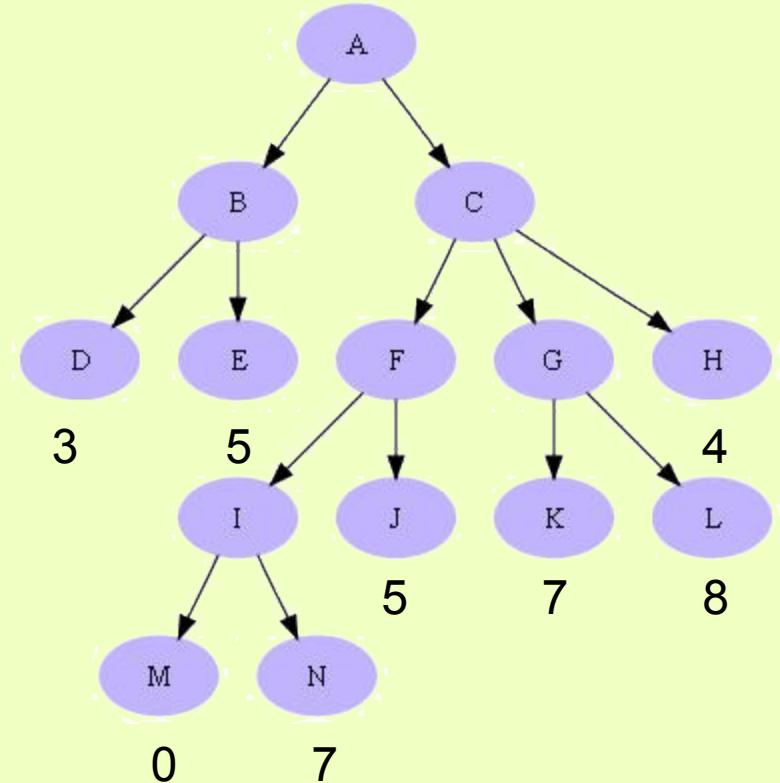
```
int backMiniMaxAB(Node N, int A, int B){  
    Set Alpha value of N to A and Beta value of N to B;  
    if N is a leaf  
        return the estimated score of this leaf  
    else{  
        if N is a Min node  
            for each child Ni of N {  
                Val = backMiniMaxAB(Ni, Alpha of N, Beta of N);  
                Beta value of N = minim(Beta value of  
N, Val);  
                if Beta value of N ≤ Alpha value of N then exit loop;  
            }  
            return Beta value of N;  
        else // N is a Max node  
            for each child Ni of N do {  
                Val = MINIMAX-AB(Ni, Alpha of N, Beta of N);  
                Alpha value of N = Max{Alpha value of N, Val};  
                if Alpha value of N ≥ Beta value of N then exit loop;  
            }  
            Return Alpha value of N;  
    }  
}
```

# Games and search – search space

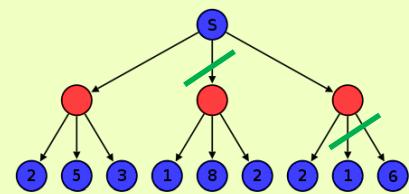


Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

- Example



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

## □ Example

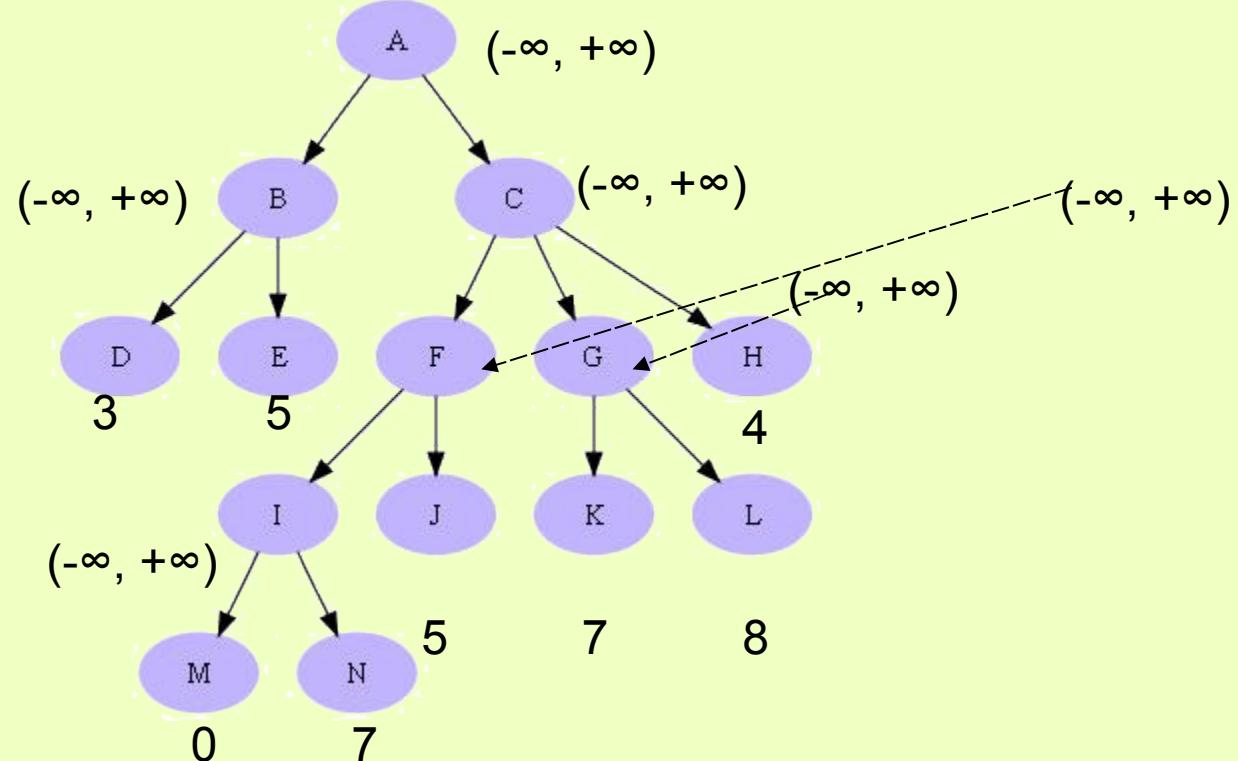
MAX

MIN

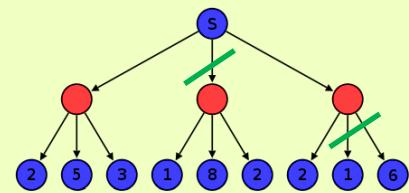
MAX

MIN

MAX



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

## □ Example

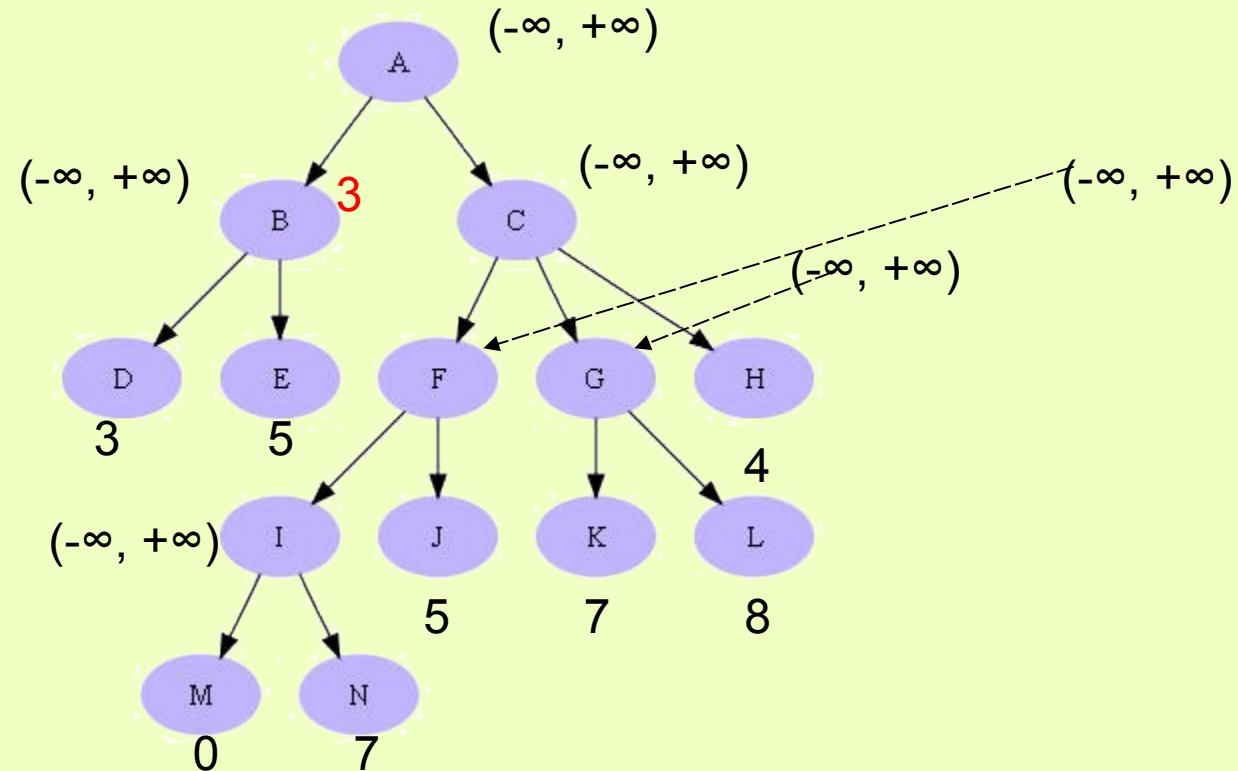
MAX( $\alpha$ )

MIN( $\beta$ )

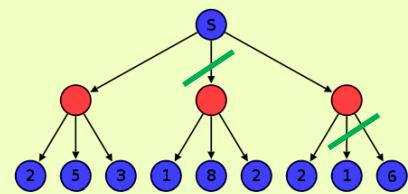
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

- Example

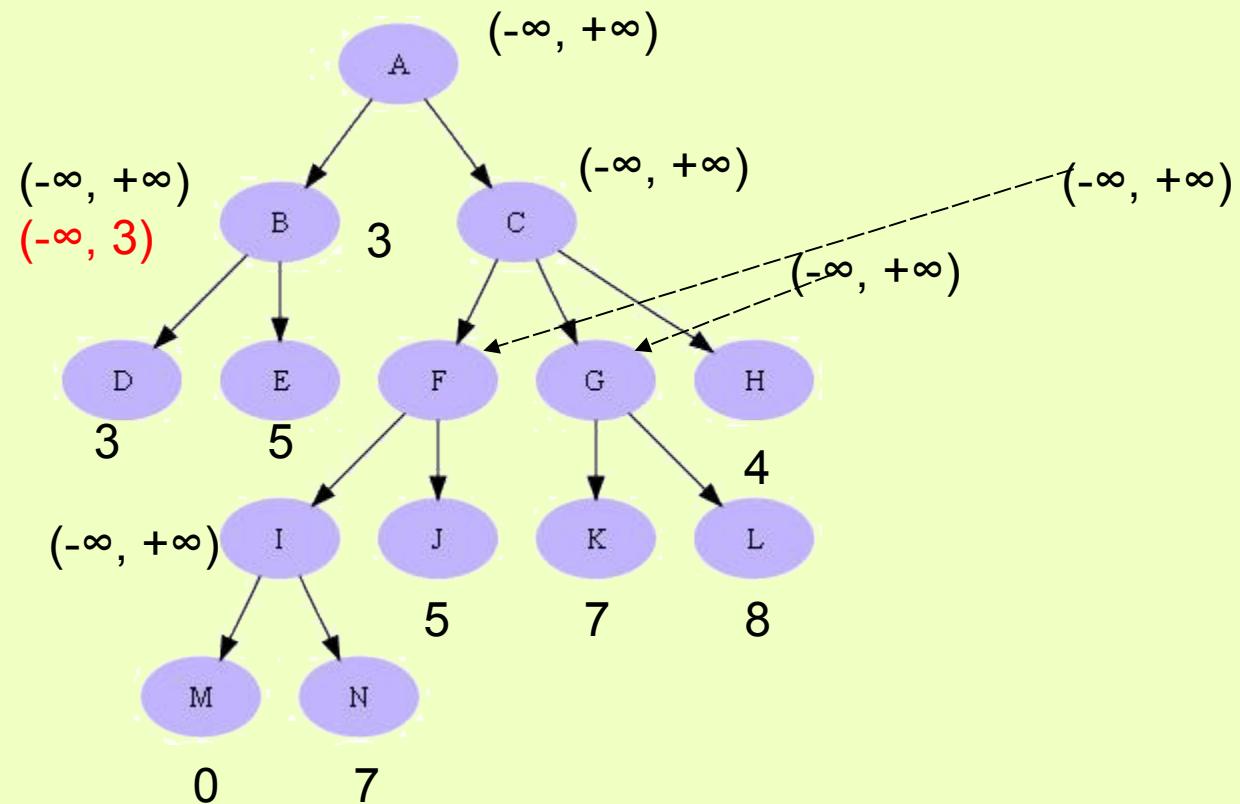
$\text{MAX}(\alpha)$

$\text{MIN}(\beta)$

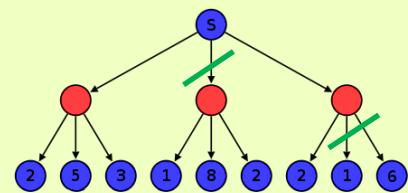
$\text{MAX}(\alpha)$

$\text{MIN}(\beta)$

$\text{MAX}(\alpha)$



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

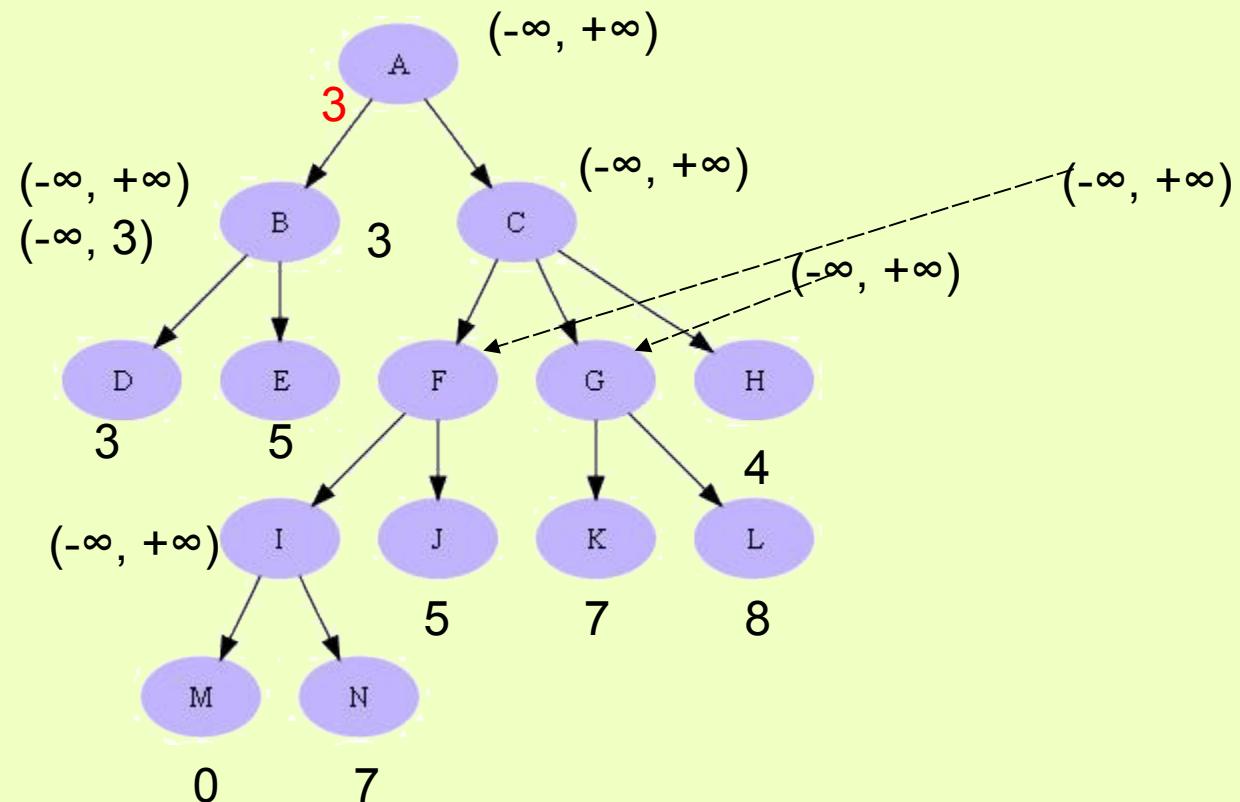
MAX( $\alpha$ )

MIN( $\beta$ )

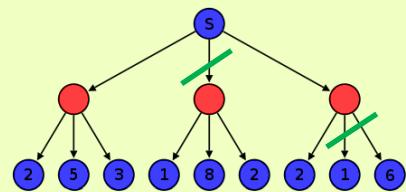
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

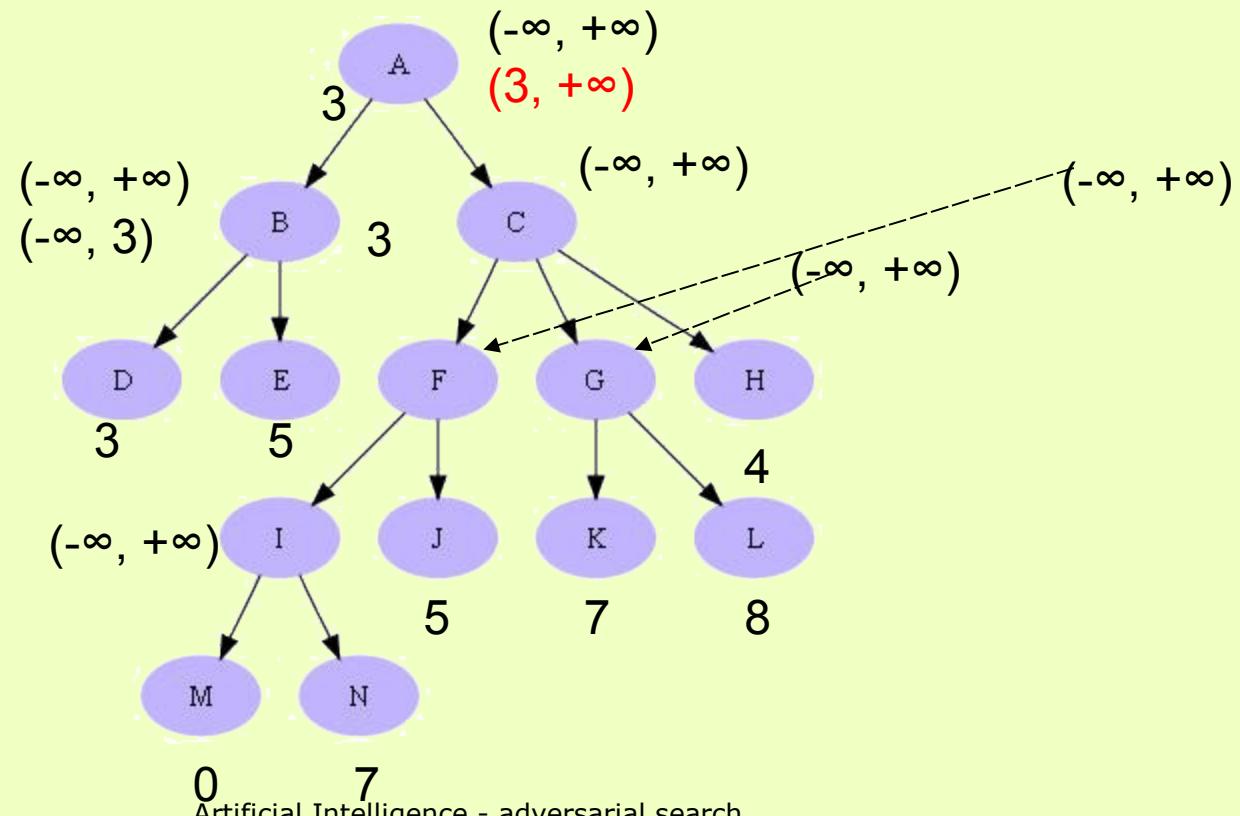
MAX( $\alpha$ )

MIN( $\beta$ )

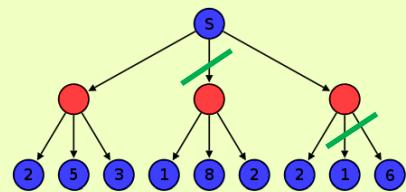
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

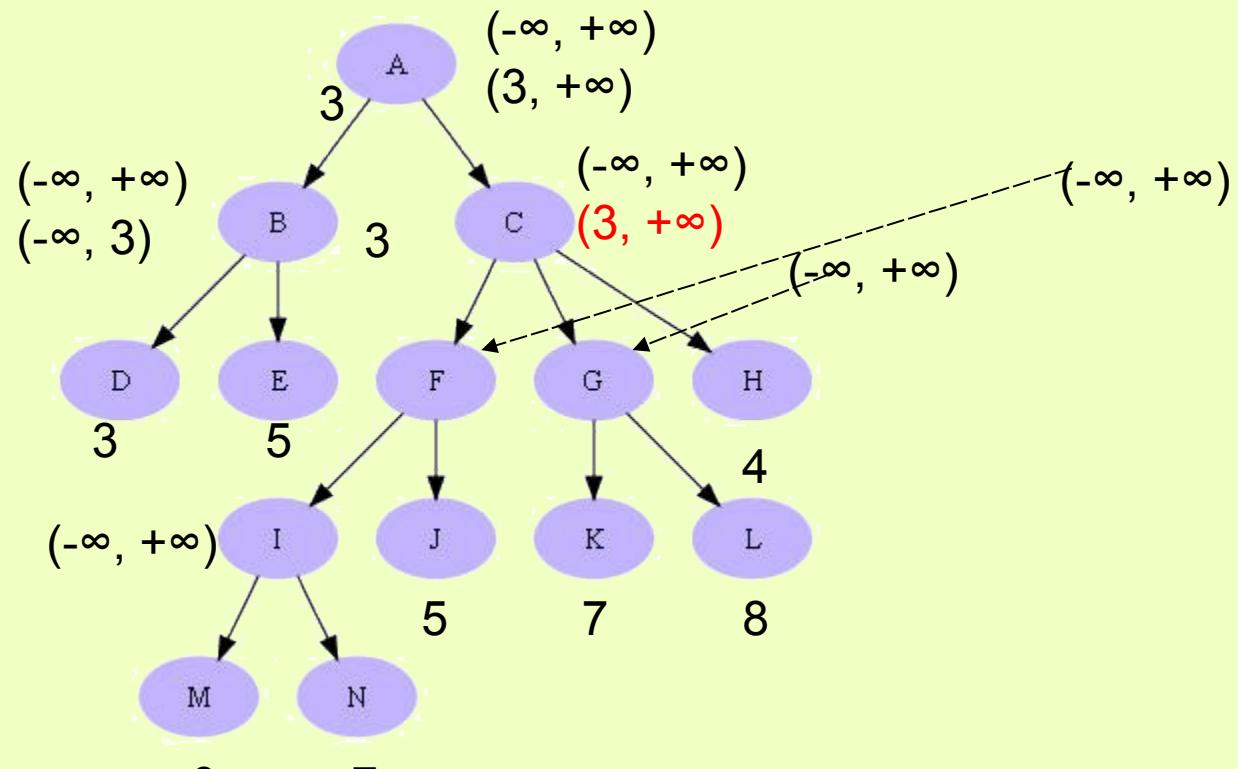
MAX( $\alpha$ )

MIN( $\beta$ )

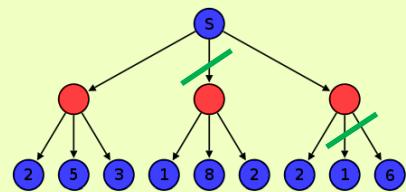
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

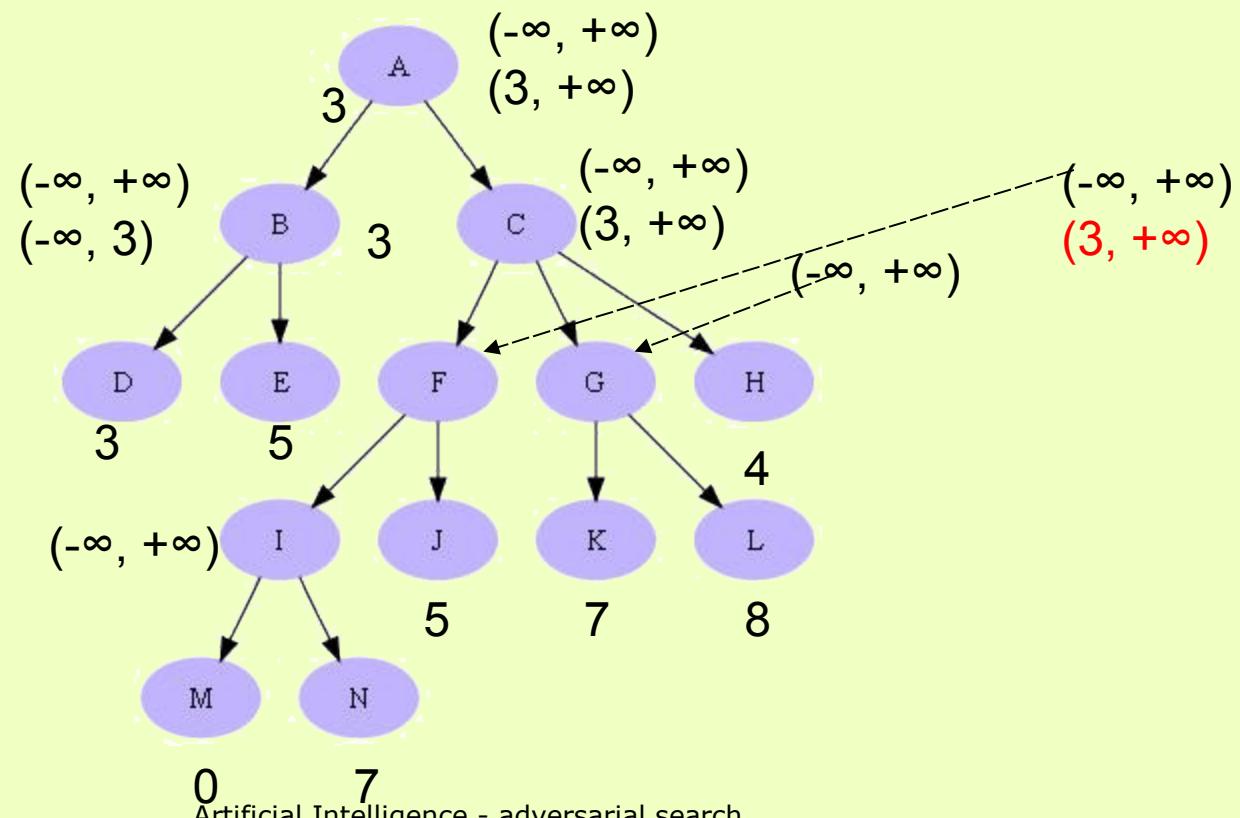
MAX( $\alpha$ )

MIN( $\beta$ )

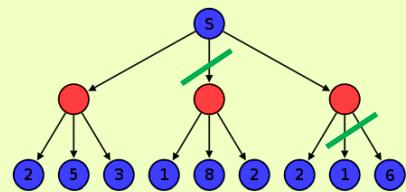
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

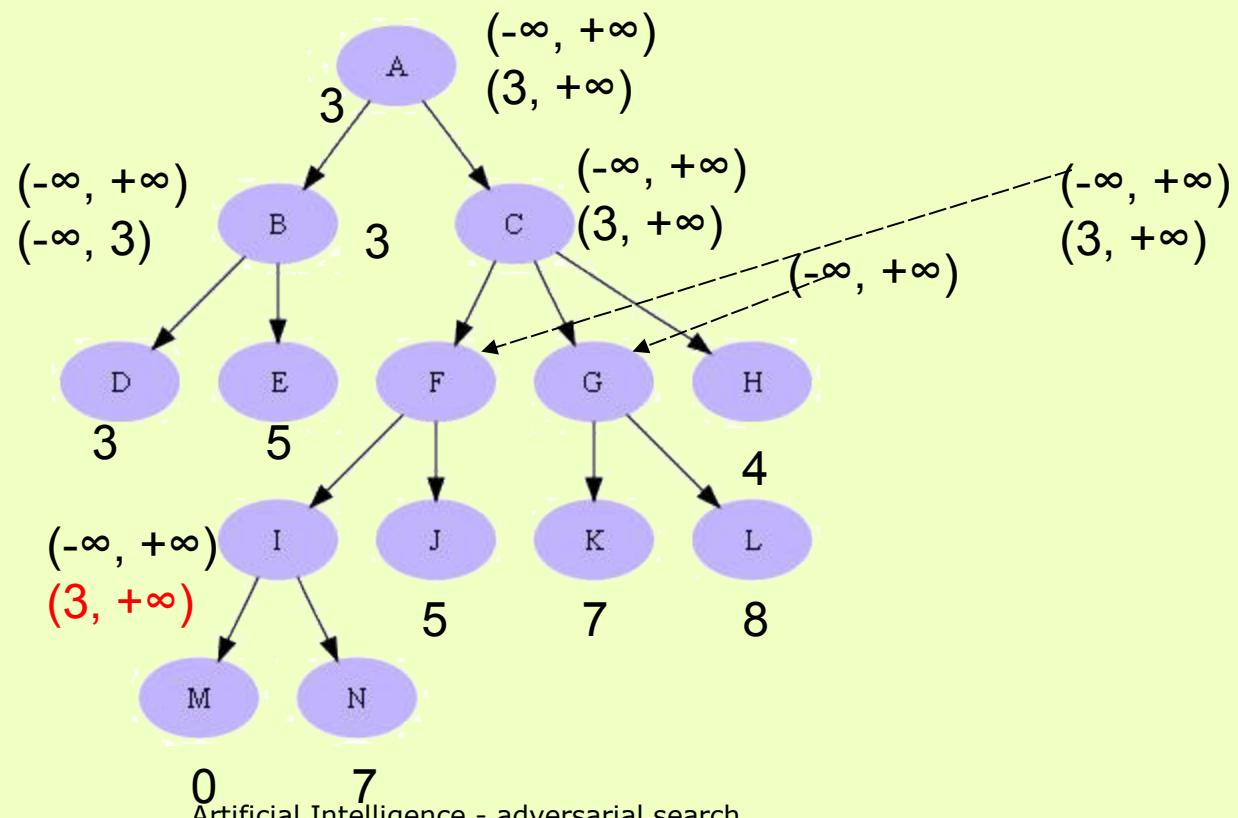
MAX( $\alpha$ )

MIN( $\beta$ )

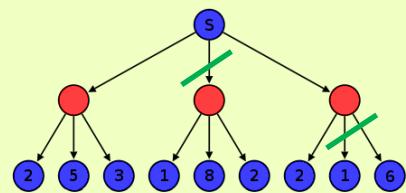
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

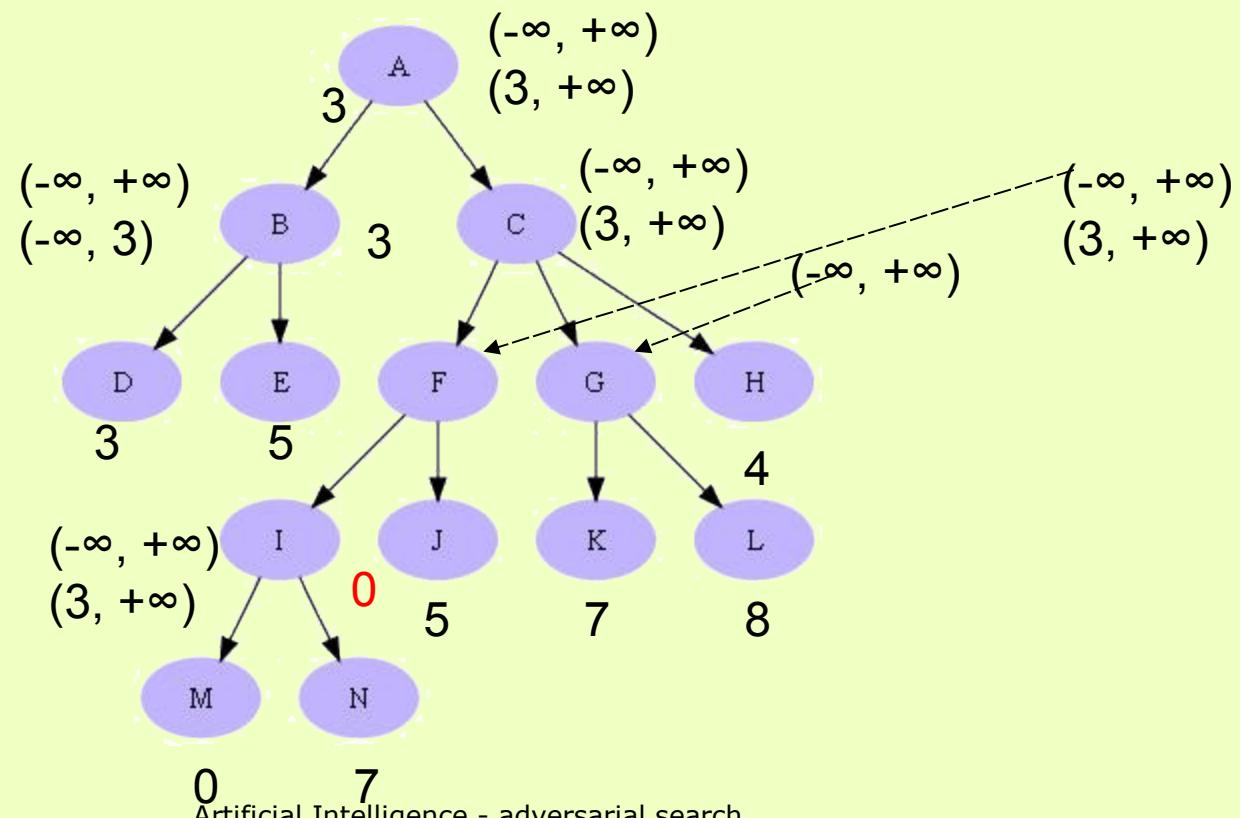
MAX( $\alpha$ )

MIN( $\beta$ )

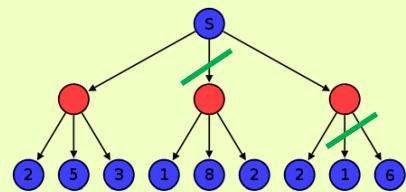
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

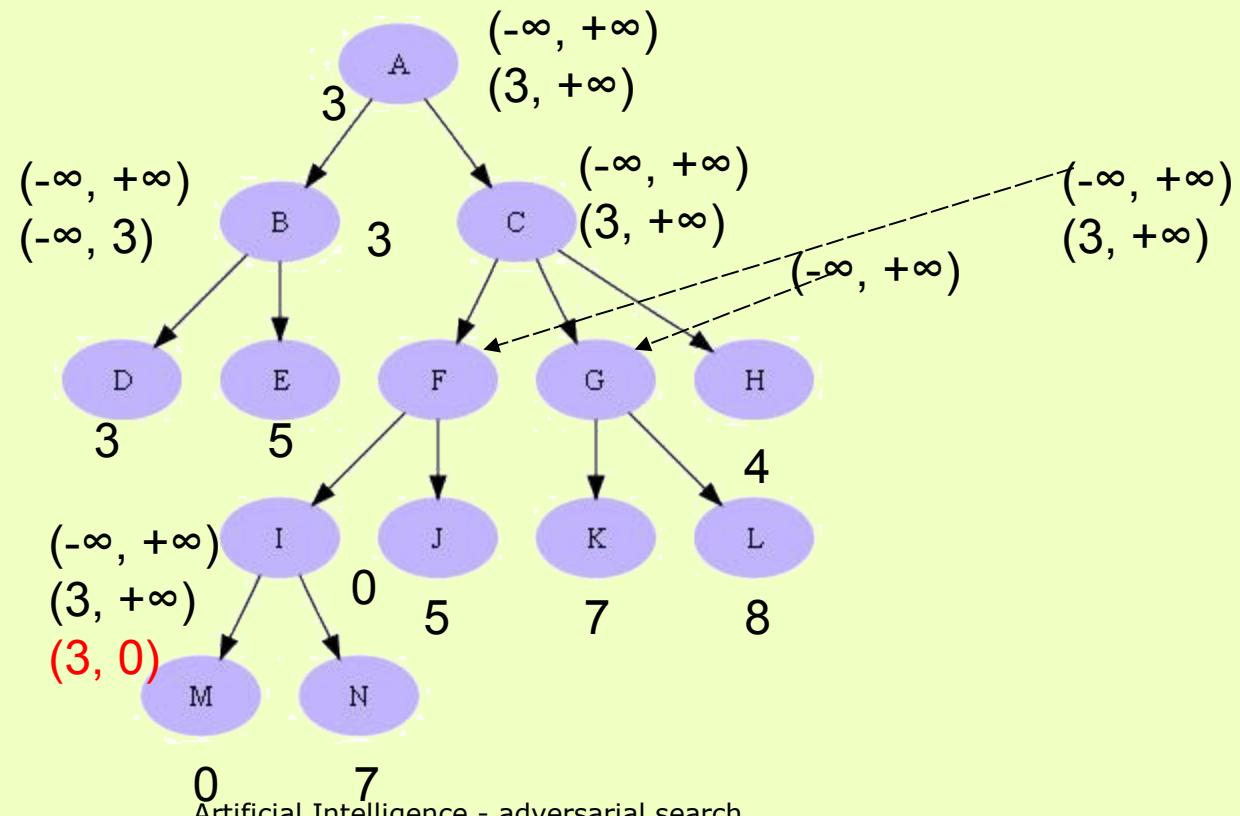
MAX( $\alpha$ )

MIN( $\beta$ )

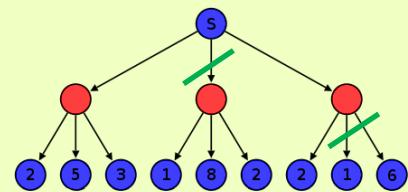
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

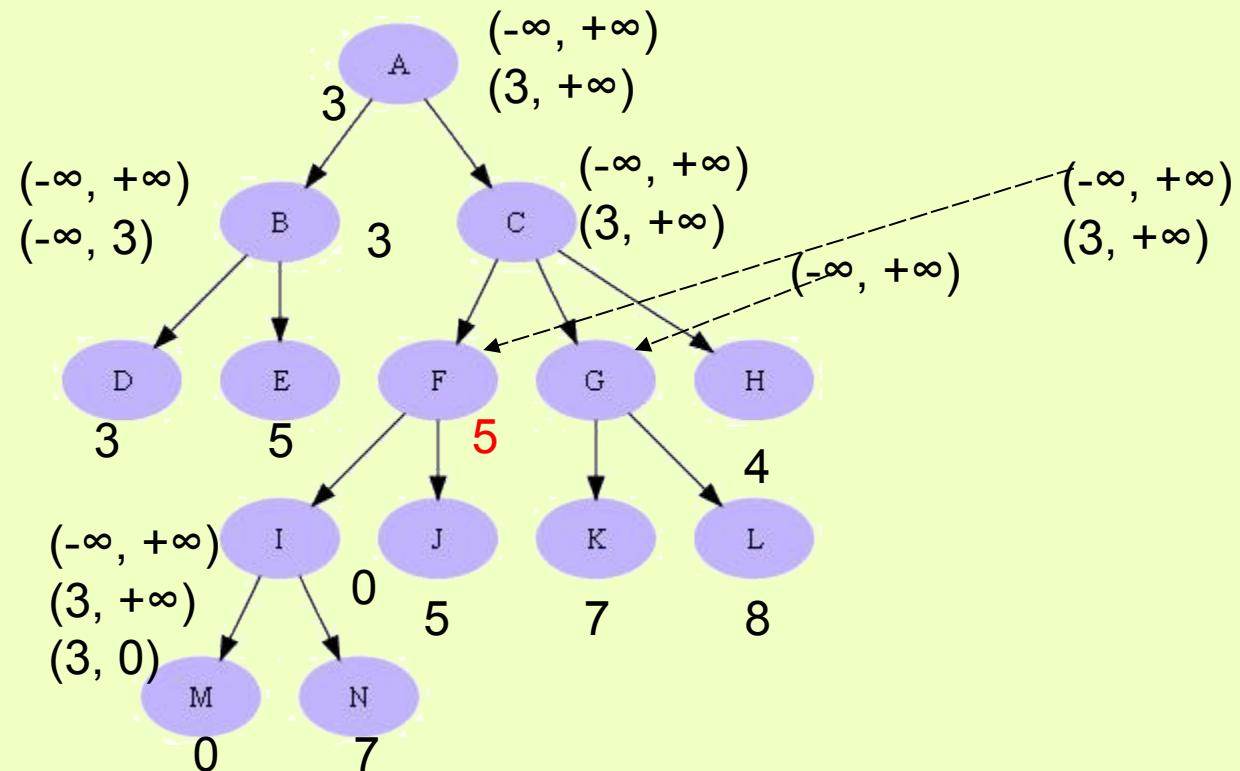
MAX( $\alpha$ )

MIN( $\beta$ )

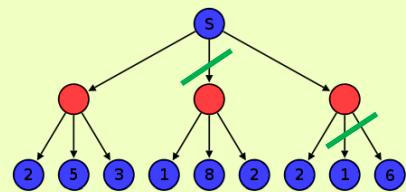
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

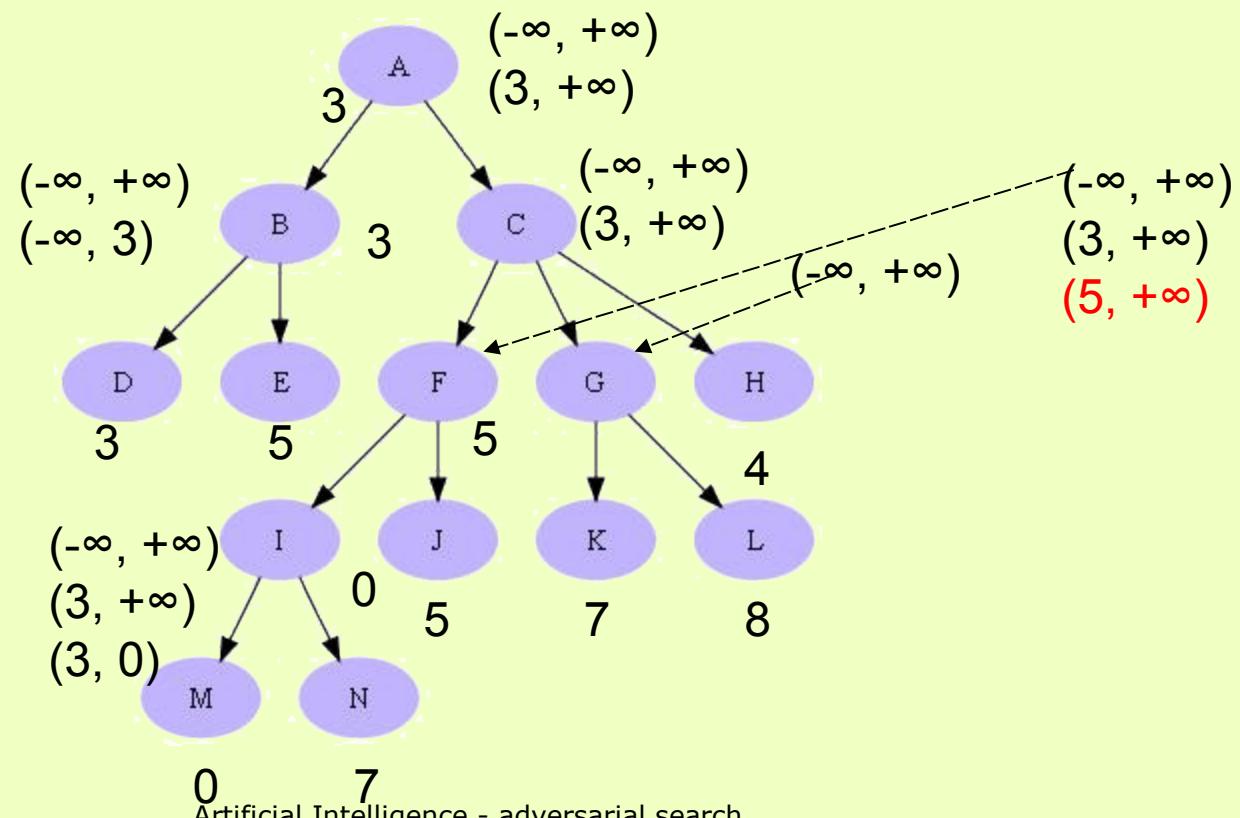
MAX( $\alpha$ )

MIN( $\beta$ )

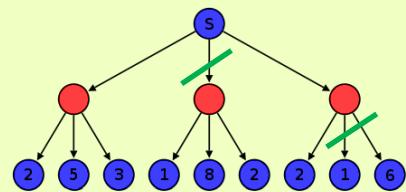
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

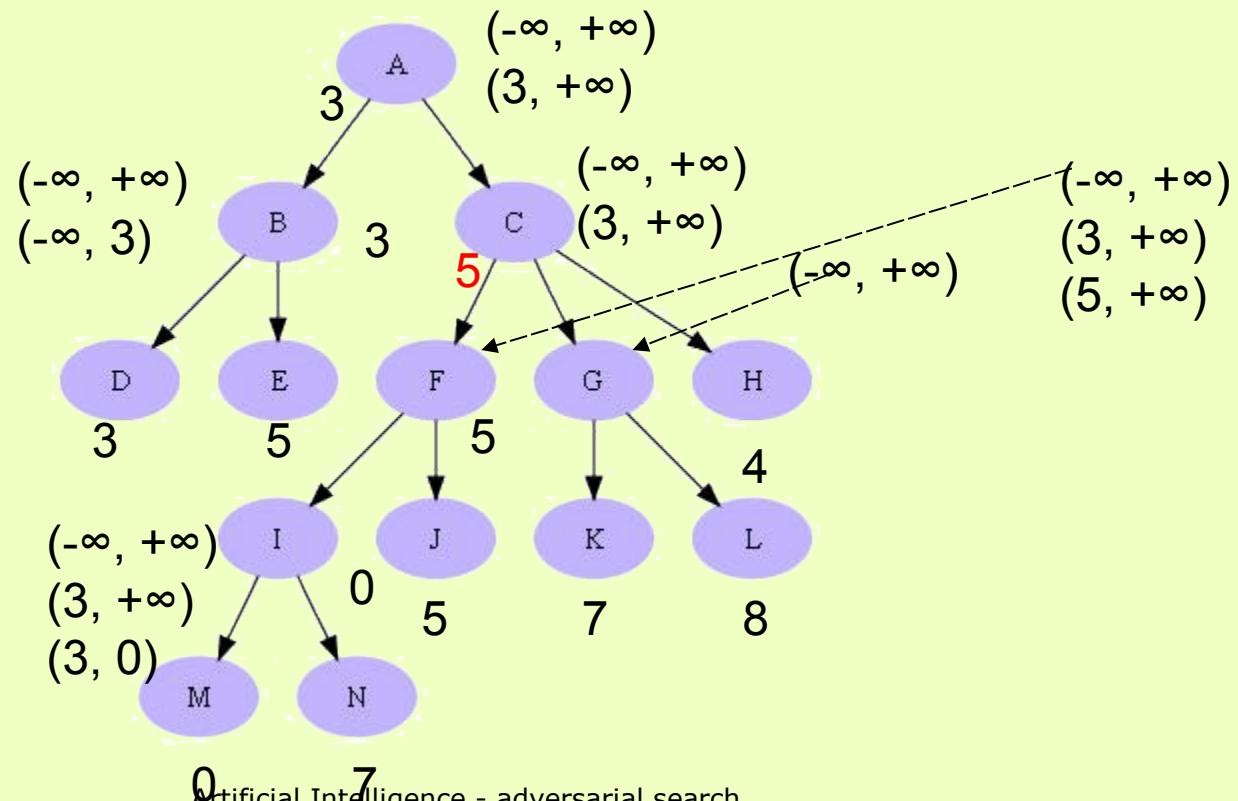
MAX( $\alpha$ )

MIN( $\beta$ )

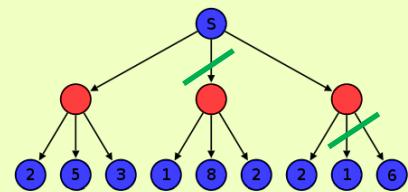
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

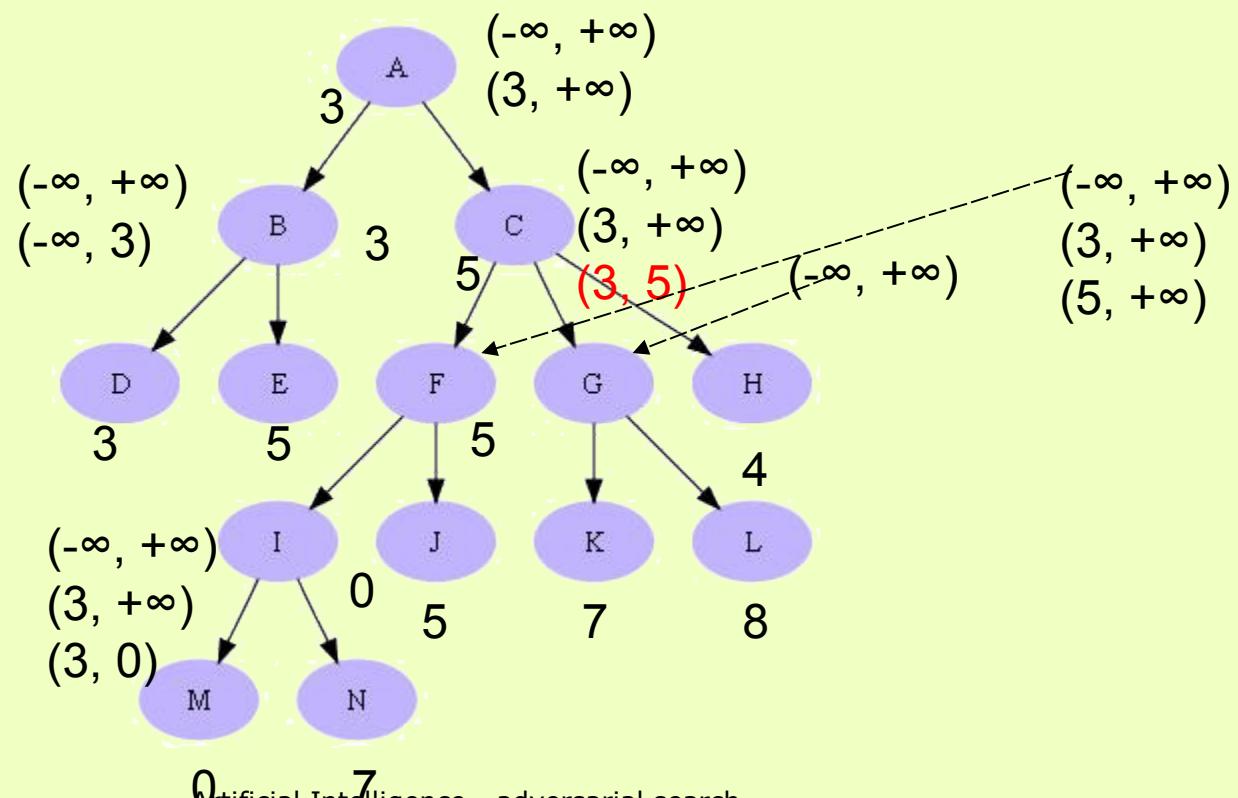
MAX( $\alpha$ )

MIN( $\beta$ )

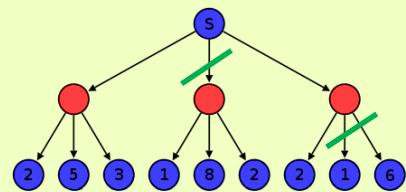
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

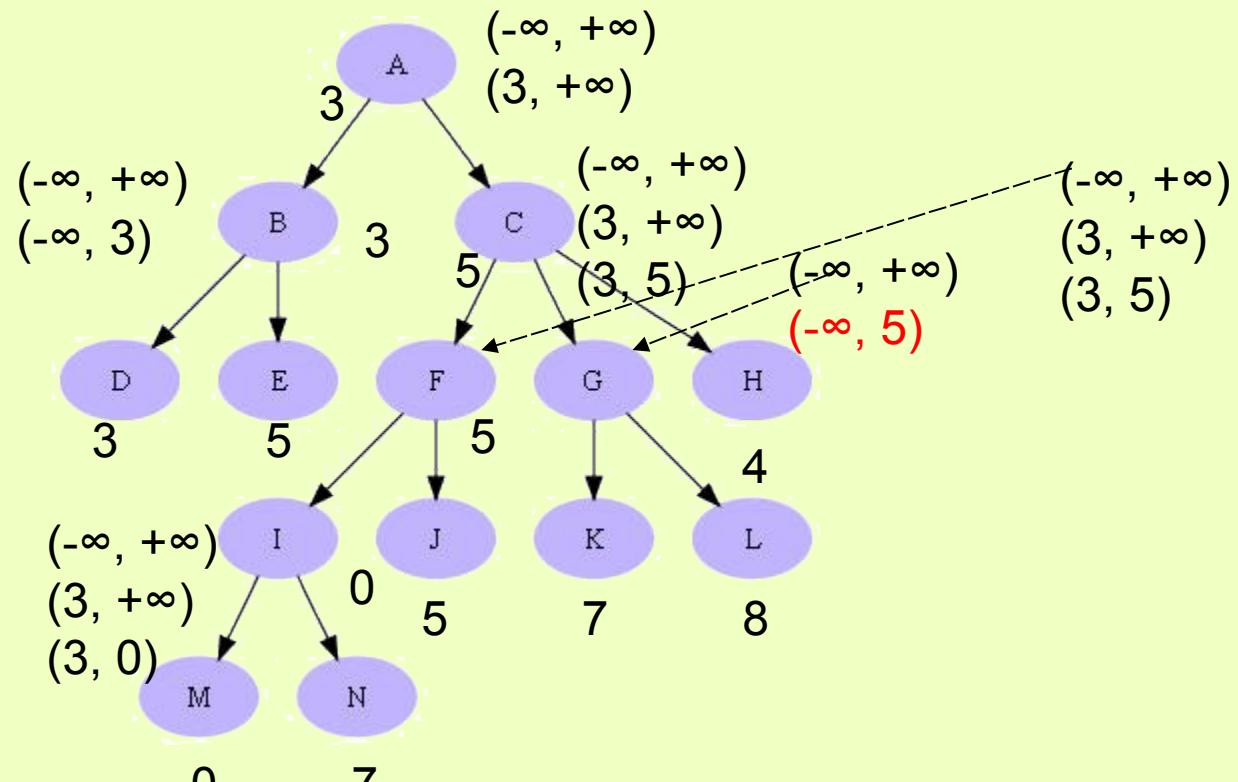
MAX( $\alpha$ )

MIN( $\beta$ )

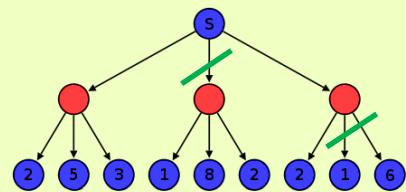
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

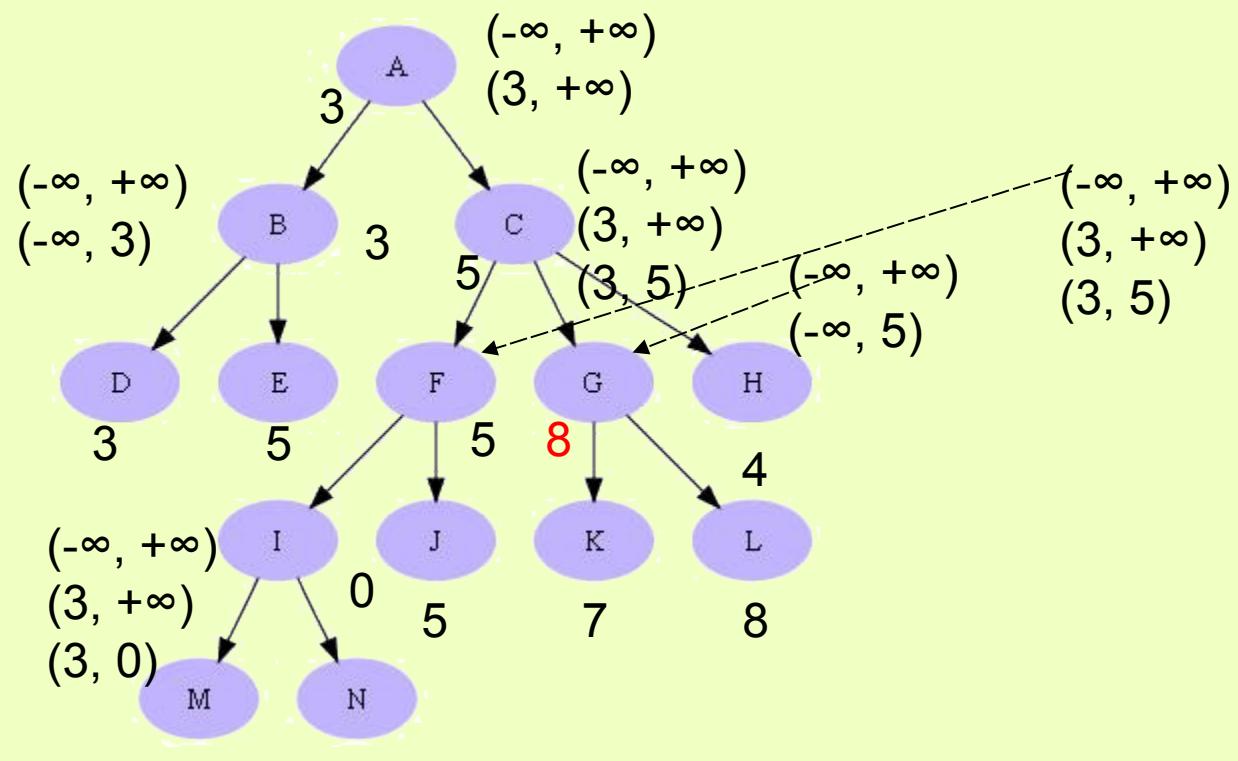
MAX( $\alpha$ )

MIN( $\beta$ )

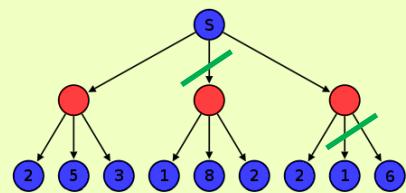
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

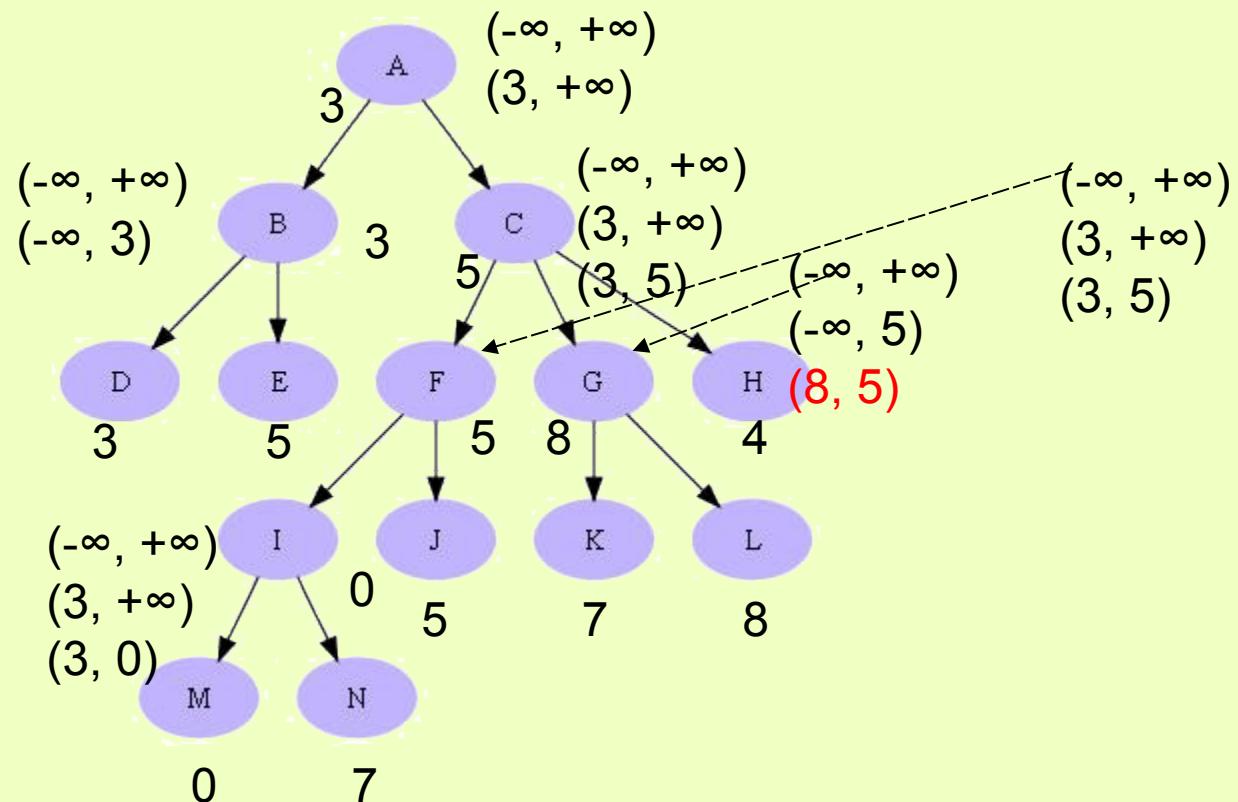
MAX( $\alpha$ )

MIN( $\beta$ )

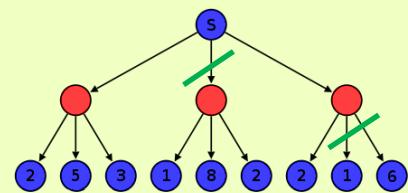
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

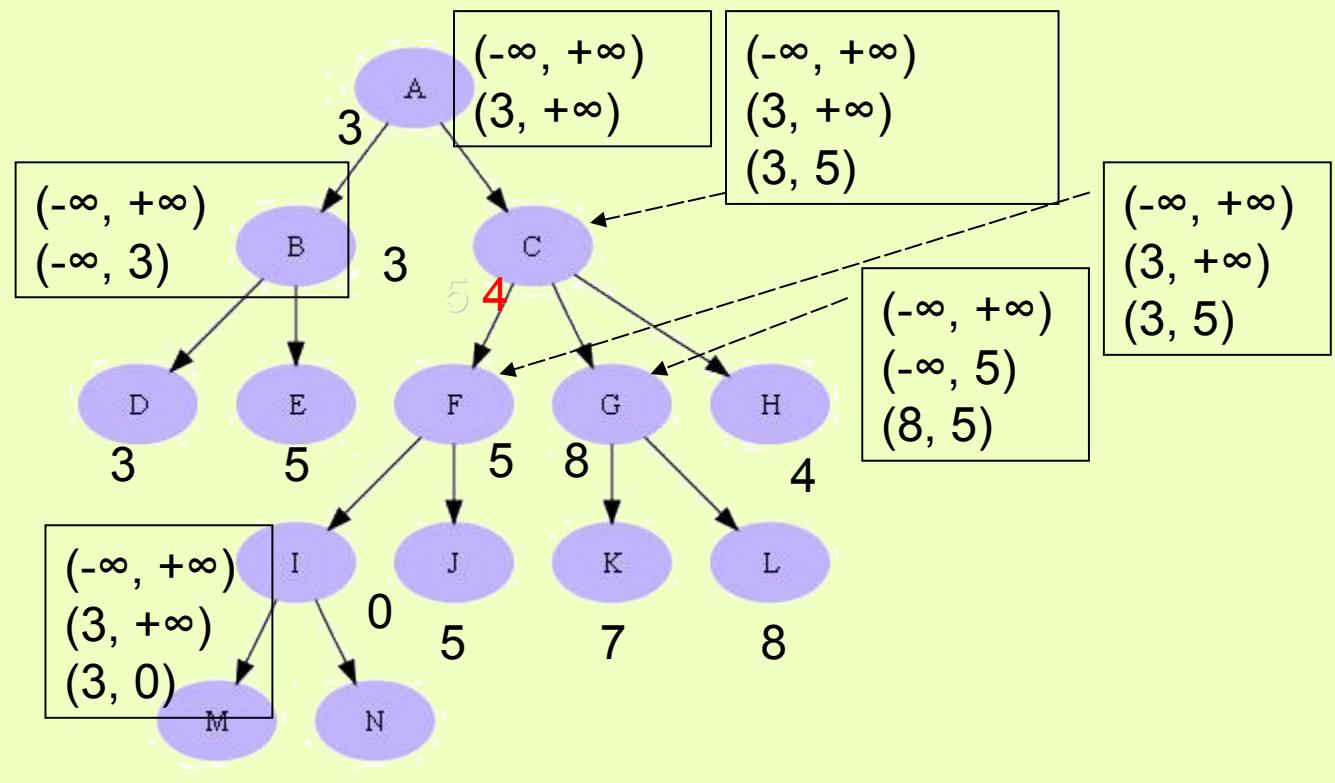
MAX( $\alpha$ )

MIN( $\beta$ )

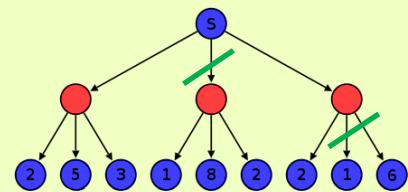
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

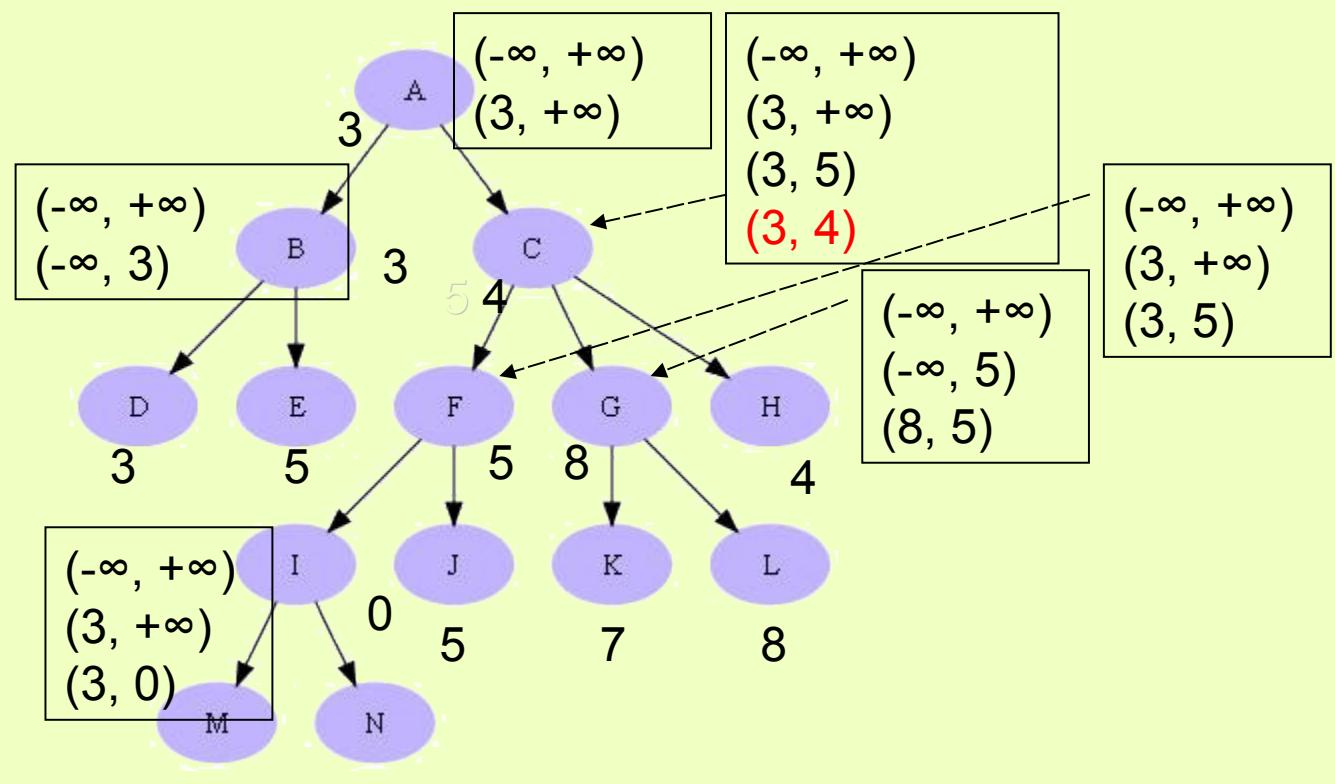
MAX( $\alpha$ )

MIN( $\beta$ )

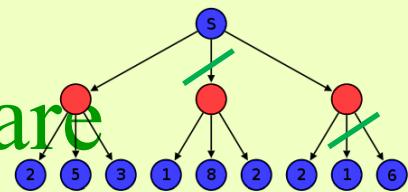
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

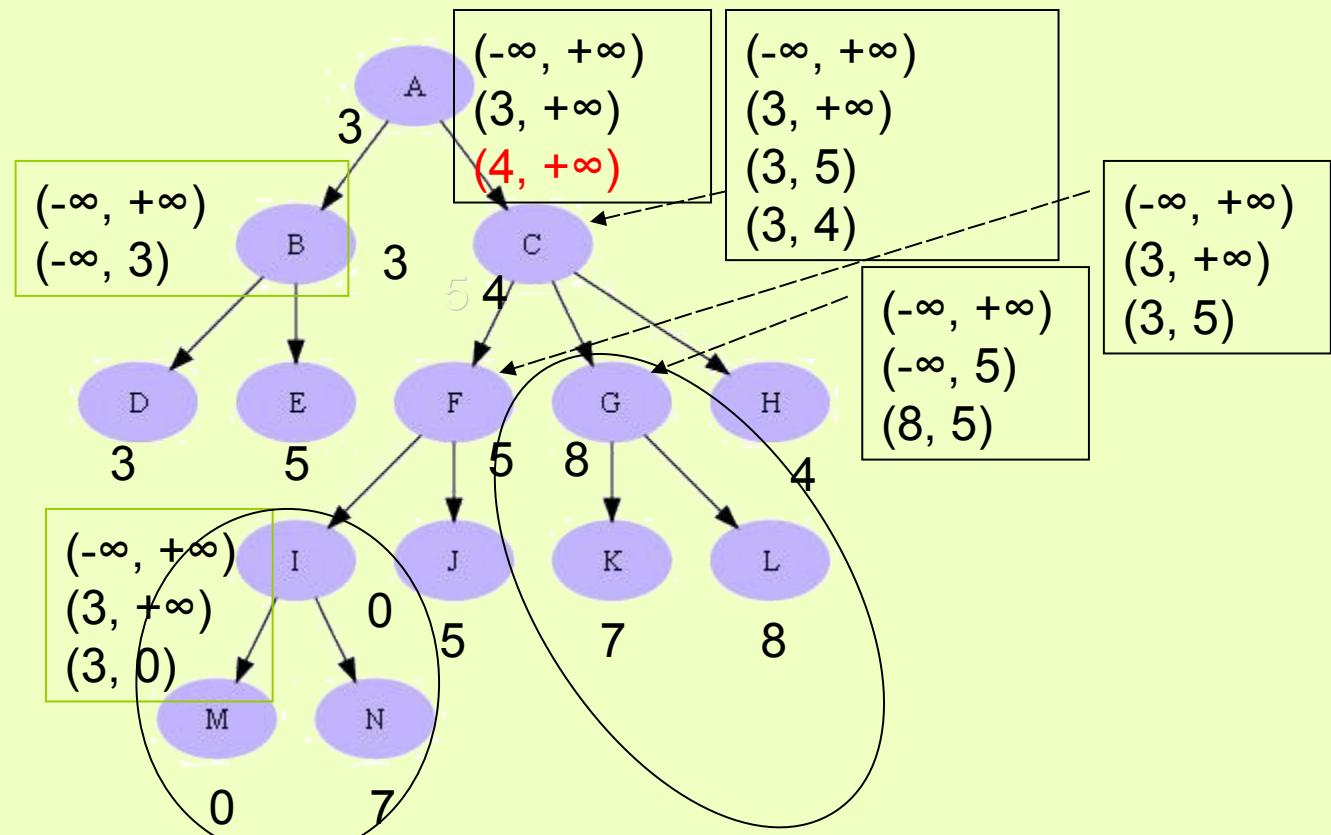
MAX( $\alpha$ )

MIN( $\beta$ )

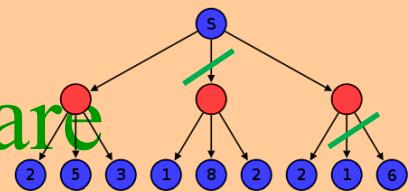
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare

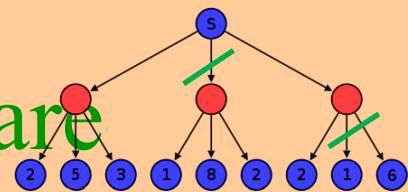


Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

## ❑ Properties

- Pruning does not influence the final result
- A good sorting of the moves improves the pruning algorithm
- If the successors are perfectly sorted (the best are the first), then the time complexity is  $O(b^{d/2})$ , instead of  $O(b^d)$  (classic MiniMax)
- A beginner-level program can be transformed in an expert-level program

# Jocurile și căutarea – spațiul de căutare

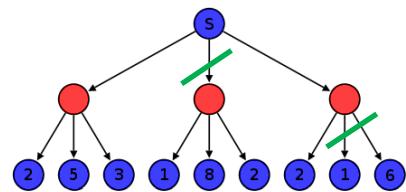


Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

- MinMax vs. MiniMax with  $\alpha$ - $\beta$  pruning

	MinMax	MinMax $\alpha$ - $\beta$
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

# Games and search – search space



- AndOr trees, MiniMax, MiniMax with  $\alpha$ - $\beta$  pruning
  - Large complexity → efficient search methods are required (for solving games)
    - Chess
      - $b \sim 35$
      - $d \sim 100$
      - $b^d \sim 35^{100} \sim 10^{154}$  nodes
    - Tic-Tac-Toe
      - $\sim 5$  legal moves (from 9)
      - $5^9 = 1\ 953\ 125$
      - $9! = 362\ 880$  (if the computer moves first)
      - $8! = 40\ 320$  (if the computer is the second player)
    - Go game
      - $b > 361$  (for a  $19 \times 19$  board)

# Games and search – search space



## ❑ Reality

- Checkers → Chinok
  - Chinok beats Marion Tinsley in 1994 (after 40 years of challenges)
  - A data base with final states computed for all positions of a perfect game of 8 pieces is utilized (444 billions of possible positions)
- Chess → Deep Blue
  - Deep Blue defeated Garry Kasparov in 1997
  - 200 millions of positions are checked in a second
  - Ramification factor is reduced to 6 by using  $\alpha$ - $\beta$  pruning (classic, 35-40)
- Othello game
  - Human players refuse to play against computers (because they are too good) – Moor (1980), Logistello (1997)
- Go game
  - Human players refuse to play against computers (because they are too weak)
  - Ramification factor > 300 → pattern-based algorithms are required
- backgammon
  - BKG (fuzzy logic), TD-Gammon (artificial neural networks)
  - Computer defeated the world champion (because it was lucky)

# Games and search – search space



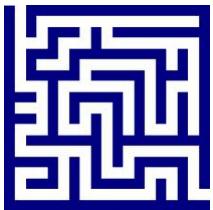
## □ Game's strategy

### ■ Step by step

- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - **Linear structures**
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - Tree-based structures
    - AndOr trees
    - MiniMax (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - **An optimal path between 2 locations**
  - A sequence of actions for moving the player from a location into another location



# Games and search – search space

Game's strategy → Path-finding

## ❑ Theoretical aspects

### ■ Problem

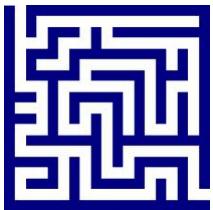
- ❑ Identify on a map (possible, with obstacles) an optimal path between 2 given locations.
- ❑ Where?
  - On a map
    - Map as a matrix
    - Map as a graph (simple, mesh, quad-tree)
  - In an environment (known/unknown, static/dynamic)

### ❑ By who?

- An agent
- 2 agents
- More agents

### ■ Solution

- ❑ Using a search (optimization) method



# Games and search – search space

Game's strategy → Path-finding

## ❑ Theoretical aspects

### ■ Solution

#### ❑ How?

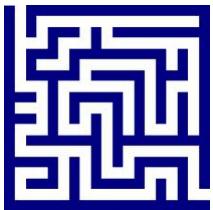
- Using a search (optimisation) method

#### ❑ Which is the best solution?

- Computing speed
- Length of the path
- Quality of the path
- Available information

#### ❑ Difficulties

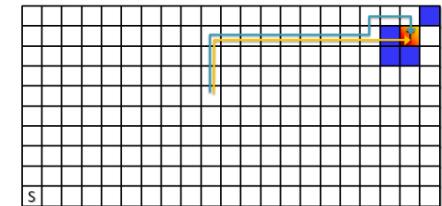
- Real-time solution
- Limited resources
- Very large search space
- Modelling the real-world involves uncertainty and heterogeneous elements (terrain, units, agents)

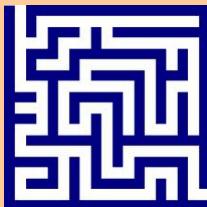


# Games and search – search space

Game's strategy → Path-finding

- Search algorithms for maps represented as a matrix of tiles (tile mapped game) → labyrinths
  - Why?
    - Simplicity
  - Characteristics
    - Discrete search space
    - Tiles are square
    - Tiles can be traversed or blocked
    - Linear moves from a tile to another one (horizontal, vertical or diagonal moves)
  - Methods
    - A\*
      - *Best-first search* algorithms
      - Represents a standard in game's industry
    - Heuristics → Manhattan (gen. of Minkowski norm)
      - Estimation of the distance between current point and destination
      - Ignore obstacles

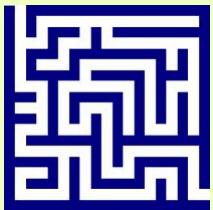




# Games and search – search space

Game's strategy → Path-finding

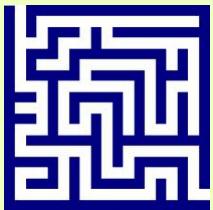
- ❑ Search algorithms for maps represented as a graph
  - Why?
    - ❑ Matrix representation → large regions of the map have the same cost
  - Characteristics
    - ❑ Discrete search space
    - ❑ Tiles can have any shape
    - ❑ Tiles can be traversed or blocked
    - ❑ Random moves from a tile to another one
  - Methods
    - ❑ Paths of edges
    - ❑ Paths of nodes
      - Points
      - Edges of a polygon
      - Middle of the polygon
    - ❑ Eg. Dijkstra, Floyd-Warshall, Kruskal, etc



# Games and search – search space

Game's strategy → Path-finding

- Improvements of the search algorithms
  - Better heuristics
  - Hierarchical abstracting
  - Decentralised approaches



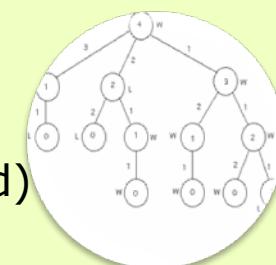
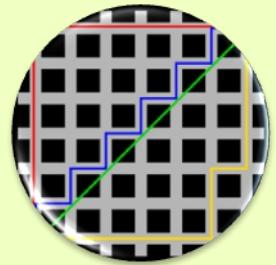
# Games and search – search space

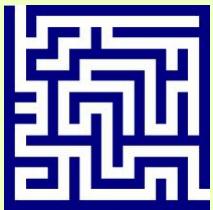
Game's strategy → Path-finding

- Improvements of the search algorithms
  - Better heuristics

- Heuristics without memory
    - Eg. Manhattan
    - Very fast to compute
    - Do not require supplementary memory
    - Good quality (sometimes)
  - Heuristics based on memory
    - Eg. Heuristics based on landmarks
    - Fast enough to compute
    - Require memory
    - Good quality (identification of dead ends)

- Imperfect information
    - Very expansive to compute and store
    - Very fast to utilize (after they have been computed)
    - Impractical





# Jocurile și căutarea – spațiul de căutare

Game's strategy → Path-finding

## ❑ Improvements of the search algorithms

### ■ Hierarchical abstracting

#### ❑ Graphs where

- Flows of motion are annotated
- Domination relations are associated to edges
- Levels are identified (eg. Room, house, town, city)

### ■ Decentralised approaches

#### ❑ Main idea:

- Decompose the problem in sub-problems that
  - Can be completely solved
  - Can be sub-optimal
  - Can be incomplete

#### ❑ Aim

- All the elements (collaborative identification) have to arrive at destination

#### ❑ Difficulties

- Increased search space:  $O(n) \rightarrow O(n^m)$
- Ramification factor explodes (from 4 or 8 to  $5^m$  or  $9^m$ )

# Jocurile și căutarea – spațiul de căutare



## Game's strategy → Planning

- Theoretical aspects
  - Problem
    - Planning (sorting) some actions
      - Moves, selections, rotations, etc.
    - Input:
      - An initial state



# Jocurile și căutarea – spațiul de căutare



## Game's strategy → Planning

- Theoretical aspects
  - Problem
    - Planning (sorting) some actions
      - Moves, selections, rotations, etc.
    - Input:
      - An initial state
      - A final state

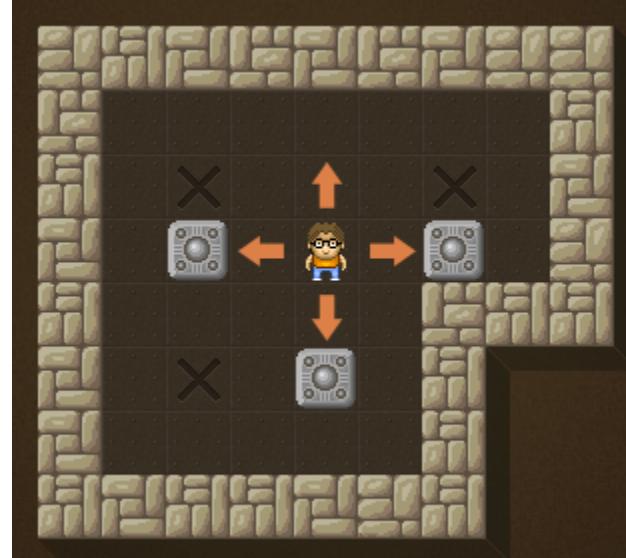


# Games and search – search space



## Game's strategy → Planning

- ▣ Theoretical aspects
  - Problem
    - ▣ Planning (sorting) some actions
      - Moves, selections, rotations, etc.
    - ▣ Input:
      - An initial state
      - A final state
      - A set of possible actions



# Games and search – search space



## Game's strategy → Planning

- ▣ Theoretical aspects
  - Problem
    - ▣ Planning (sorting) some actions
      - Moves, selections, rotations, etc.
    - ▣ Input:
      - An initial state
      - A final state
      - A set of possible actions
    - ▣ Output:
      - Identify a sequence of actions that transforms the initial state into the final state



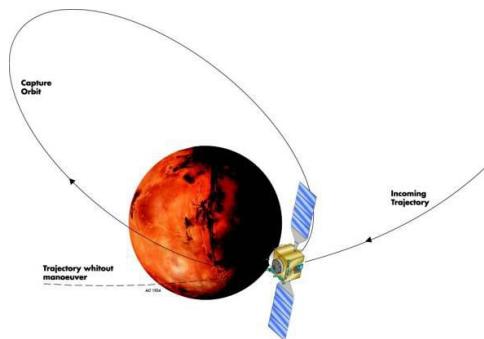
# Games and search – search space



## Game's strategy → Planning

### □ Why?

- NPC games (*First-person shooter, Role-playing, Real-time strategy*)
- Real-world planning problems
  - Space telescope
  - Robots
  - UAVs



# Games and search – search space



## Game's strategy → Planning

### □ How?

- Algorithms
  - For selecting an action (shooting)
  - For evaluating the environment
- Approaches
  - STRIPS →  
<http://www.dis.uniroma1.it/~degiacom/didattica/dottorato-stavros-vassos/>
  - HTN
- Behaviour simulation
  - Finite state machines
  - Behaviour trees
  - GOAP (FEAR)



# Review

## □ Game's definition

- Initial state (how the elements are initially located)
- Possible actions (allowed moves)
- Final test (when a game ends)
- Utility functions (who wins and what is the profit)

## □ Solving strategy for games

- Based on (almost) complete exploration of the search tree
  - AndOr → who wins
  - MiniMax → who wins and what is its profit
  - MiniMax with  $\alpha$ - $\beta$  pruning → who wins and what is its profit and what moves do not deserve to be performed
- Based on intelligent strategies
  - Symmetry
  - Pairs
  - Parity
  - Dynamic programming
  - A\* (Path-finding, Planning)

# Next lecture

---

- A. Short introduction in Artificial Intelligence (AI)
- B. Solving search problems
  - A. Definition of search problems
  - B. Search strategies
    - A. Uninformed search strategies
    - B. Informed search strategies
    - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
    - D. Adversarial search strategies
- C. Intelligent systems
  - A. Rule-based systems in certain environments
  - B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
  - C. Learning systems
    - A. Decision Trees
    - B. Artificial Neural Networks
    - C. Support Vector Machines
    - D. Evolutionary algorithms
  - D. Hybrid systems

# Next lecture - useful information

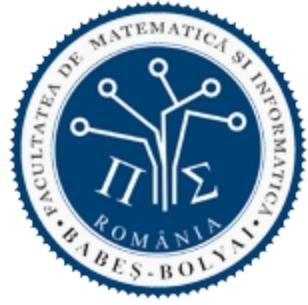
---

- Chapter III of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 4 and 5 of *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- Chapter 2 of *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapter 6 and 7 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

- 
- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean – [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)



BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics



# ARTIFICIAL INTELLIGENCE

**Intelligent systems**

Rule-based systems

# Topics

---

- A. Short introduction in Artificial Intelligence (AI)
- B. Solving search problems
  - A. Definition of search problems
  - B. Search strategies
    - A. Uninformed search strategies
    - B. Informed search strategies
    - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
    - D. Adversarial search strategies
- C. Intelligent systems
  - A. Rule-based systems in certain environments
  - B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
  - C. Learning systems
    - A. Decision Trees
    - B. Artificial Neural Networks
    - C. Support Vector Machines
    - D. Evolutionary algorithms
  - D. Hybrid systems

# Useful information

---

- Chapter III of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 4 and 5 of *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- Chapter 2 of *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapter 6 and 7 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

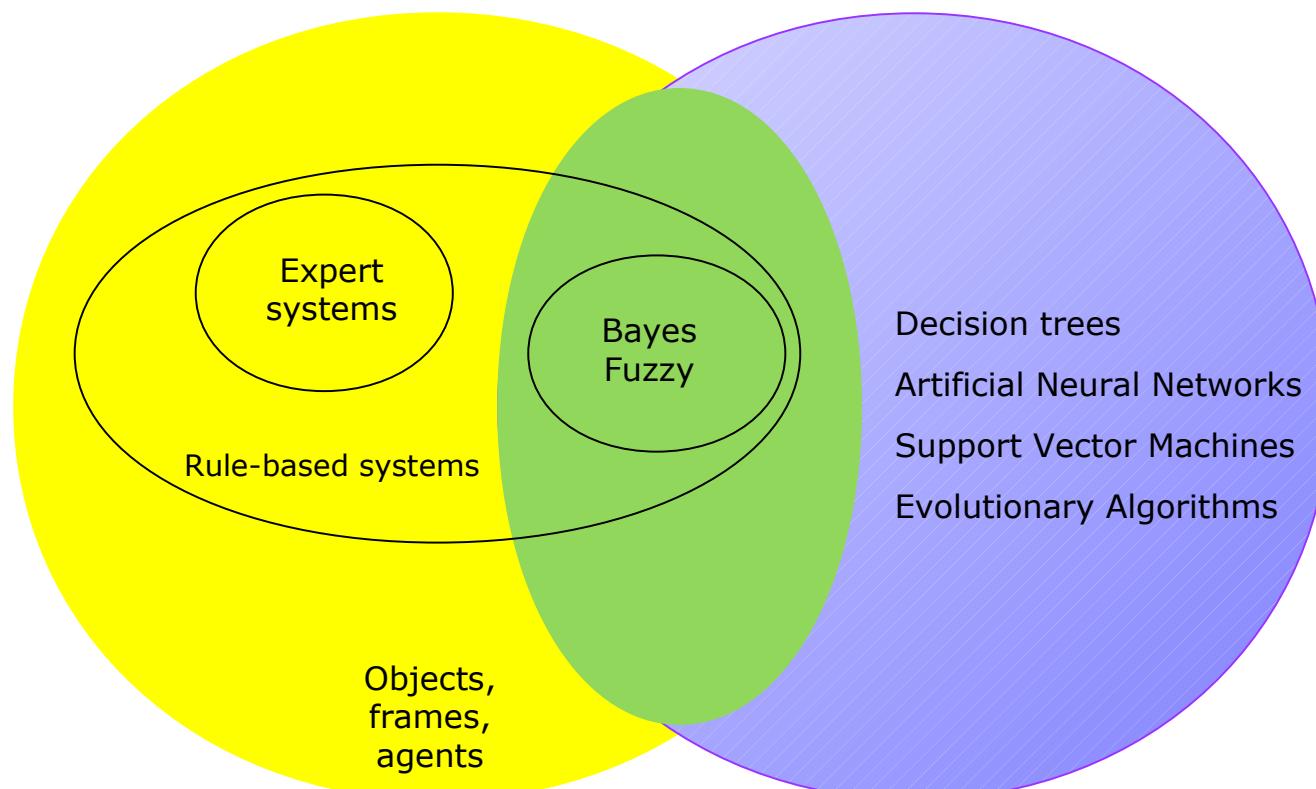
# Content

---

- ❑ Intelligent systems
  - Knowledge-based systems
    - ❑ Logic-based systems
    - ❑ Rule-based systems in certainty

# Intelligent systems

---



Sisteme bazate pe cunoștințe (SBC)

Inteligență computațională

# Intelligent systems – knowledge-based systems (KBS)

## ❑ Computational systems - 2 main modules

- *knowledge base (KB)*

- ❑ Specific information

- *inference engine (IE)*

- ❑ Rules that can determine new information
  - ❑ Domain independent algorithms

# Intelligent systems – KBS

---

## Knowledge base

- ❑ Content
- ❑ Typology
- ❑ Representation
- ❑ Storing

# Intelligent systems – KBS

---

## Knowledge base (KB)

- Content
  - Information (in a given representation – eg. Sentences) about domain
  - Required information for problem solving
  - Set of sentences (represented in a formal way) that describe an environment
    - Interpretability of representation → knowledge representation language
    - Mechanism for obtaining new sentences from the old ones → inference
- Typology
  - Perfect knowledge
  - Imperfect knowledge
    - Not-exactly
    - Incomplete
    - Non-measurable
- Knowledge representation
  - Formal logic (formal languages)
  - Rules
  - Semantic nets

# Intelligent systems – KBS

---

## Knowledge base (KB)

### ❑ Knowledge representation

- Formal logic (formal languages)
  - Definition
    - Science of formal principles for rationing
  - Components
    - Syntax – atomic symbols used by language and the constructing rules of the language
    - Semantic – associates a meaning to each symbol and a truth value (true or false) to each rule
    - Syntactic inference – rules for identifying a subset of logic expressions → theorems (for generating new expressions)
  - Typology
    - True value
      - Dual logic
      - Polyvalent logic
    - Basic elements
      - Classic → primitives = sentences (predicates)
      - Probabilistic → primitives = random variables
    - Working manner
      - Propositional logic → declarative propositions and fix or unique objects (Ionica is student)
      - First-order logic → declarative propositions, predicates and quantified variables, unique objects or variables associated to a unique object
- Rules
- Semantic nets

# Intelligent systems – KBS

---

## Knowledge base (KB)

### ❑ Knowledge representation

- Formal logic
- Rules
  - Special heuristics that generate information
  - A possible representation of information
  - Eg. If Ionica works at Facebook, then he earns a lot, but he has little free time.
  - Interdependences among rules → inference network
  - Link the cause and effect → if clause then effect
- Semantic nets
  - Oriented graphs with nodes that contain concepts and edges that represent semantic relations among concepts
    - *Meronymy* (A is a meronym of B if A is a part of B)
      - Eg. Finger is a meronym of hand, wheel is a meronym of car
    - *Holonymy* (A is holonym of B dacă B is a part of A)
      - Ex. Tree is a holonym of branch
    - *Hyponymy* (A is hyponym of B if A is a kind of B)
      - Ex. Tractor is a hyponym of vehicle
    - *Hypernymy* (A is hypernym of B if A is a generalisation of B)
      - Ex. Fruit is a hypernym of orange
    - *Synonymy* (A is a synonym of B if A and B have the same meaning)
      - Ex. run is synonym to jog
    - *Antonymy* (A is an antonym of B if A and B reflect opposed things)
      - Ex. Dry is an antonym to wet

# Intelligent systems – KBS

---

## Knowledge base (KB)

- ❑ Storage
  - Relations
    - ❑ Simple → data bases
    - ❑ Hierarchical → semantic nets
  - Formal logic
    - ❑ Rules
  - Procedural logic
    - ❑ Algorithms

# Intelligent systems – KBS

---

## Inference engine (IE)

- Content
  - Determine a conclusion by taking into account some premises and by applying some inference rules
  - IE depends on the complexity and type of knowledge
- Typology
  - Inference direction
    - IE with *forward chaining*
      - Start from available information and determine a conclusion
      - Based on data (*data driven*)
    - IE with *backward chaining*
      - Start from a possible conclusion and identify some explanations for it
      - Based on aim (*goal driven*)
- Techniques for inference
  - Certainty environment
    - Logic-base
    - Rule-based
  - Uncertainty environment
    - Based on theory of probability
    - Based on theory of possibility

# Intelligent systems – KBS

---

## Typology

- ❑ **Logic based systems (LBS)**
- ❑ **Rule based systems (RBS)**
- ❑ *Case-based reasoning*
- ❑ *Hypertext manipulating systems*
- ❑ *Data bases and intelligent UI*
- ❑ *Intelligent tutoring systems*

# Intelligent systems – KBS

---

## Logic based systems (LBS)

- ❑ Content and aim
- ❑ Architecture
- ❑ Typology
- ❑ Tools
- ❑ Advantages and limits

# Intelligent systems – KBS – LBS

---

## Content and aim

### □ Content

- Explore a lot of information for obtaining conclusions about difficult activities by using methods of formal logic
- A logic system is composed of
  - Language (syntax and semantic)
  - Deduction method (inference)

### □ Aim of a LBS

- Problem solving by help of declarative programming
- Eg. Automatic theorems proving

### □ Why LBS are studied?

- Formal logic is precise and well-defined

# Intelligent systems – KBS – LBS

---

## Architecture

- Knowledge base (KB)
  - Syntax
    - Atomic symbols used by a language and construction rules of expressions
  - Semantic
    - Meaning of symbols and the truth value of rules
- Inference engine (IE)
  - Inference, as method, - rules for determining a subset of logic expressions → theorems

# Intelligent systems – KBS – LBS

---

## Typology

- ▣ **Systems based on propositional logic**
  - Declarative propositions only
  - Described objects are unique and fixed (*Ionica is student*)
- ▣ Systems based on first-order logic
  - Declarative propositions, predicates (*Student(a)*) and quantified variables (*for any a, Student(a) → WifiAccess(a)*)
  - Described objects can be unique or dynamic (*all the students are present*)
  - Predicates have simple arguments (*Student(a)*)
- ▣ Systems based on higher-order logic ( $\geq 2$ )
  - Declarative propositions, predicates and quantified variables
  - Variables can represent more relations among objects
  - Predicates can have simple arguments or predicates arguments (*StudentSenator(Student(a))*) or function arguments (*Bursier(a has average over 9.50)*)
- ▣ Temporal systems
  - Represent the truth value of fact along time (*Ionica is sometimes nervous*)
- ▣ Modal systems
  - Represents doubtful facts also (*Ionica could pass the exam*)

# Intelligent systems – KBS – LBS

---

## Systems based on propositional logic

- ❑ Knowledge base
- Composed by
  - ❑ symbols (A, B, P, Q, ...)
  - ❑ Propositions (formula)
    - Defined as
      - A symbol
      - If P is a proposition, then  $\neg P$  is a proposition too
      - If P and Q are propositions, then  $P \wedge Q$ ,  $P \vee Q$ ,  $P \Rightarrow Q$ ,  $P \Leftrightarrow Q$  are propositions too
      - A finite number of applying the rules (1)-(3)
    - Meaning of a proposition → truth value
    - Model → interpretation of a set of propositions such as all the propositions are true
- ❑ Inference engine
- Performs inference
  - ❑ Establish the truth value of a proposition taking into account the KB
- Typology
  - ❑ Model checking
    - All the possible combinations of truth value for all the involved symbols and propositions
  - ❑ Model deduction

# Intelligent systems – KBS – LBS

## Systems based on propositional logic – inference engine

- Problem
- For a KB = {P<sub>1</sub>, P<sub>2</sub>, ..., P<sub>m</sub>} composed by symbols {X<sub>1</sub>, X<sub>2</sub>, ..., X<sub>n</sub>} and an objective proposition O
- Is it possible to deduce O from KB?

## ■ Model checking

- Steps
  - Construct the table for all possible combinations of truth values for all the symbols
  - Determine if all the models of KB are models for O too
    - KB models – that models where all propositions are true

## ■ Example

- P = Afară este foarte cald
- Q = Afară este umezeală
- R = Afară plouă
  
- BC = {P  $\wedge$  Q  $\Rightarrow$  R, Q  $\Rightarrow$  P, Q}
- R = Va plouă?

P	Q	R	P $\wedge$ Q $\Rightarrow$ R	Q $\Rightarrow$ P	Q	BC	R	BC $\Rightarrow$ R
T	T	T	T	T	T	T	T	T
T	T	F	F	T	T	F	F	T
T	F	T	T	T	F	F	T	T
T	F	F	T	T	F	F	F	T
F	T	T	T	F	T	F	T	T
F	T	F	T	F	T	F	F	T
F	F	T	T	T	F	F	T	T
F	F	F	T	T	F	F	F	T

## ■ Difficulties

- Exponential number for all combinations  $\rightarrow$  large computing time
- Solution: deduction by using inference rules

# Intelligent systems – KBS – LBS

## Intelligent systems – KBS – LBS

- ❑ Problem
- For a KB = {P<sub>1</sub>, P<sub>2</sub>, ..., P<sub>m</sub>} composed by symbols {X<sub>1</sub>, X<sub>2</sub>, ..., X<sub>n</sub>} and an objective proposition O
- Is it possible to deduce O from KB?
  
- ❑ Model deduction by using inference rules
- Steps
  - ❑ Construct a demonstration (proof) of truth value for the objective proposition based on:
    - Propositions
      - Original (that from KB)
      - Derivative
    - Inference rules
- Example
  - ❑ Problem

- P = Afară este foarte cald
- Q = Afară este umedeală
- R = Afară plouă
  
- BC = {P  $\wedge$  Q  $\Rightarrow$  R, Q  $\Rightarrow$  P, Q}
- R = Va plouă?

Regulă de inferență	Premisă	Propoziția derivată
Modus ponens	A, A $\Rightarrow$ B	B
Și introductiv	A, B	A $\wedge$ B
Și eliminativ	A $\wedge$ B	A
Negație dublă	$\neg\neg$ A	A
Rezoluție unitară	A $\vee$ B, $\neg$ B	A
Rezoluție	A $\vee$ B, $\neg$ B $\vee$ C	A $\vee$ C

### Solution

1. Q Premise
2. Q  $\Rightarrow$  P Premise
3. P Modus Ponens (1,2)
4. (P  $\wedge$  Q)  $\Rightarrow$  R Premise
5. P  $\wedge$  Q introductiv AND(1,3)
6. R Modus Ponens (4,5)

# Intelligent systems – KBS

---

## Typology

- ❑ Logic based systems (LBS)
- ❑ **Rule based systems (RBS)**
- ❑ *Case-based reasoning*
- ❑ *Hypertext manipulating systems*
- ❑ *Data bases and intelligent UI*
- ❑ *Intelligent tutoring systems*

# Intelligent systems – KBS

---

## □ Rule-based systems (RBS)

- Content and objectives
- Design
- Architecture
- Tools and example
- Advantages and limits

# Intelligent systems – KBS – RBS

---

## Content and objectives

### □ Content

- Explore a lot of knowledge for new conclusions about activities that are difficult to be analyzed by using methods as humans do
- Can solve problems for which there is no deterministic algorithm
- try to simulate a human expert (in a given domain).
- RBSs do not replace human experience, but they facilitate the work of non-experts → Expert Systems

### □ RBS's aim

- Solving problems that require human experts
  - To transfer expertise from an expert to a computer system and
  - Than on to other humans (non-experts)
- Examples of problems solved by RBSs → consulting problems
  - Detector problems in operating systems - Microsoft Windows troubleshooting
  - Financial consultant
  - Medical diagnosis - program takes place of a doctor; given a set of symptoms the system suggests a diagnosis and treatment
  - Car fault diagnosis - given car's symptoms, suggest what is wrong with it

### □ Why RBS are investigated?

- In order to understand the human reason
- Human experts need holidays, can go to other companies, can be sick, can ask more money
- Have a big commercial success

# Intelligent systems – KBS – RBS

---

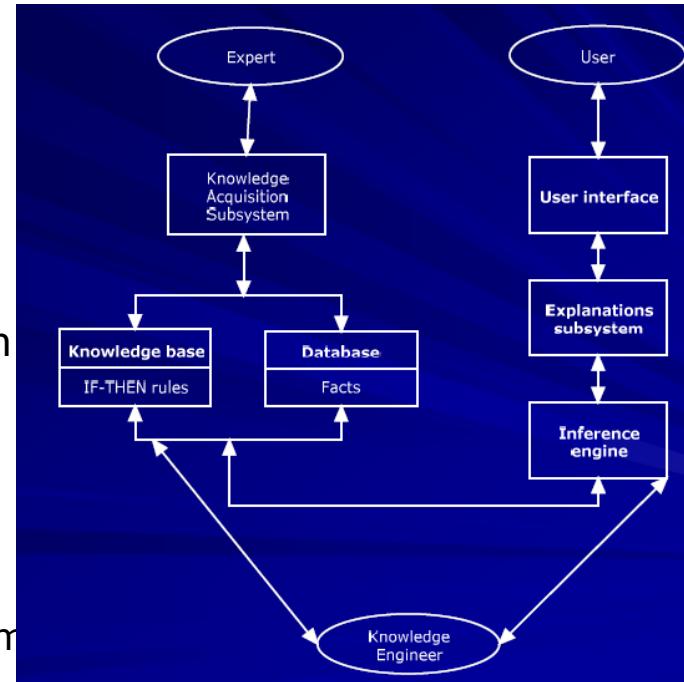
## Design

- ❑ Knowledge acquisition
- ❑ Knowledge representation
- ❑ Knowledge inference
- ❑ Knowledge transfer to the user

# Intelligent systems – KBS – RBS

## Architecture

- Knowledge base (KB)
  - Specific information
- Inference engine (IE)
  - Rules utilized for generating new information
- User interface
  - Allows dialog with users and accessing facts and knowledge from KB (to be added or updated)
  - A facility for the user to interact with the system
- Knowledge Acquisition Facility
  - An automatic way to acquire knowledge
- Explanation Facility
  - Explains reasoning of the system to the user



# Intelligent systems – KBS – RBS

---

Architecture → knowledge base

## □ Content

- knowledge necessary for understanding, formulating, and solving problems
  - ▣ Facts – correct sentences
  - ▣ Rules - special heuristics, or rules that directly generates knowledge
  - ▣ Knowledge is the primary raw material of ES
  - ▣ Incorporates knowledge representation

## □ Aim

- Storing all knowledge (facts, rules, solving methods, heuristics) about a specific domain taken from human experts or other sources

# Intelligent systems – KBS – RBS

---

## Architecture → knowledge base

### ❑ Facts

- Definition
  - Correct affirmations (sentences, propositions)
  - Memorised as some data structures
- Example
  - *Ionica works at Facebook*
- Typology
  - Persistence (changing rate)
    - Static facts - ~ permanently (*Ionica works at Facebook*)
    - Dynamic facts – specific for an instance/run (*Ionica takes a lunch brake*)
  - Generation
    - Given facts (*Ionica assists to a meeting*)
    - Derived facts – resulted by applying some rules (*If Ionica is PM, then he must lead the meeting*)

# Intelligent systems – KBS – RBS

---

## Architecture → knowledge base

### □ Rules

#### ■ Definition

- Special heuristics that generate knowledge (information)
- A possible knowledge representation
- Inter-dependencies among rules → inference network
- Link a cause to an effect → clauses *IF cond THEN effect1 ELSE effect2*
  - IF this condition (or premise or antecedent) occurs,
  - THEN some action (or result, or conclusion, or consequence) will (or should) occur
  - Example
    - IF temperature < -20oC AND you are in Romania, THEN is winter.
    - Deduction – cause + rule → effect
    - Abduction – effect + rule → cause
    - Induction – cause + effect → rule
- Rules can be viewed as a simulation of the cognitive behavior of human experts
- Rules represent a model of actual human behavior

# Intelligent systems – KBS – RBS

---

## Architecture → knowledge base

### □ Rules

#### ■ Examples

- A single cause (antecedent) and more consequences (combined by AND)
  - *IF Ionica works at Facebook, THEN he has many money AND little free time*
- A single cause (antecedent) and more consequences (combined by OR)
  - *IF it is winter THEN it is cold OR it is snow.*
- More causes/antecedents (combined by AND) and a single consequence
  - IF it is winter AND temperature < 0 AND it is wind THEN we can not walk.
- More causes/antecedents (combined by OR) and a single consequence
  - IF it is winter OR temperature < 0 OR it is wind THEN we can not walk.
- More causes/antecedents (combined by ANDOR/) and more consequences
  - IF it is winter AND temperature < 0 OR it is wind THEN the airplanes can not land.

# Intelligent systems – KBS – RBS

---

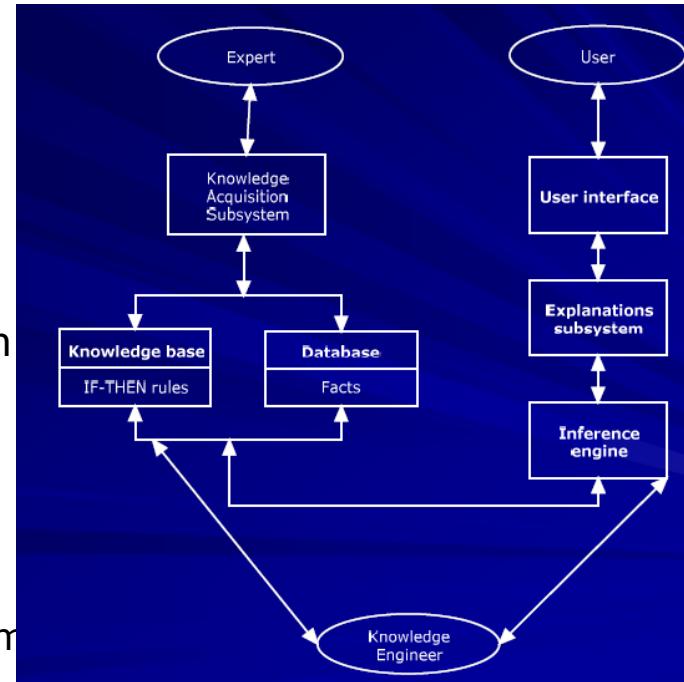
Architecture → knowledge base

- Rules
  - Typology
    - Certainty degree
      - Exact rules - *If you are employed then you have a salary*
      - Uncertain rules – *If it is winter, then temperature < 0*
    - Rules can express
      - Relations – ex. *Dacă studentul are media peste 9.50, atunci el primește bursă*
      - Recommendations – ex. *Dacă plouă, atunci să luăm umbrela*
      - Directives – ex. *Dacă bateria telefonului este gata, atunci trebuie pusă la încărcat*
      - Heuristics – ex. *Dacă lumina telefonului este stinsă, atunci bateria este plină*
  - Rules' advantages
    - Easy to understand (natural form of knowledge)
    - Easy to derive inference and explanations
    - Easy to modify and maintain
  - Rules' limits
    - Complex knowledge requires many rules
    - Search limitations in systems with many rules

# Intelligent systems – KBS – RBS

## Architecture

- ❑ Knowledge base (KB)
  - Specific information
- ❑ Inference engine (IE)
  - Rules utilized for generating new information
- ❑ User interface
  - Allows dialog with users and accessing facts and knowledge from KB (to be added or updated)
  - A facility for the user to interact with the system
- ❑ Knowledge Acquisition Facility
  - An automatic way to acquire knowledge
- ❑ Explanation Facility
  - Explains reasoning of the system to the user



# Intelligent systems – KBS – RBS

---

## Architecture – inference engine

- Content
  - Rules that can generate new information
  - Algorithms
  - RBS's brain – a deduction algorithm based on KB and specific to reasoning method
  - Depends on complexity and type of knowledge
- Aim
  - Helps to explore KB for reasoning → solutions, recommendations or conclusions
- Typology
  - IE with *forward chaining*
  - IE with *backward chaining*

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine with *forward chaining*

### □ Main idea

- starts with the initial facts and keep using the rules to draw new conclusions (or take certain actions) given those facts.
- The rules are of the form
  - left hand side (LHS) ==> right hand side (RHS).
- Data-driven - Start from available information as it becomes available, then try to draw conclusions

### □ Example

- **Question:** *Does employee Popescu get a phone?*
- **Rule:** *If Popescu is an employee, he gets a phone.*
- **Current Fact:** *Popescu is an employee.*
- **Conclusion:** *Popescu gets a phone.*

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine with *forward chaining*

### □ Algorithm

- The execution cycle is
  - Repeat
    - Select a rule whose left hand side conditions match the current state as stored in the working memory.
    - Execute the right hand side of that rule, thus somehow changing the current state.
  - Until there are no rules which apply.
  
- Remarks
  - Facts are represented in a working memory which is continually updated.
  - Rules represent possible actions to take when specified conditions hold on items in the working memory.
  - The conditions are usually patterns that must match items in the working memory,
  - The actions usually involve adding or deleting items from the working memory.

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine with *backward chaining*

- Main idea
  - Starts from a potential conclusion (hypothesis) and search facts that explain it
  - The rules are of the form
    - left hand side (LHS) ==> right hand side (RHS).
  - Goal-driven - Start from a potential conclusion (hypothesis), then seek evidence that supports (or contradicts) it
- Example
  - **Question:** *Does employee John get a computer?*
  - **Statement:** *John gets a computer.*
  - **Current Fact:** *John is a programmer.*
  - **Rule:** *If employee is a programmer, then he gets a computer.*
  - Check the rule base to see what has to be “true” for John to get a computer. A programmer. Is it a fact that john is programmer. If true, then he gets a computer

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine with *backward chaining*

### □ Algorithm

- Execution cycle
  - Start with a goal state
  - System will first check if the goal matches the initial facts given. If it does, the goal succeeds. If it doesn't, the system will look for rules whose conclusions match the goal.
  - One such rule will be chosen, and the system will then try to prove any facts in the preconditions of the rule using the same procedure, setting these as new goals to prove.
- Remarks
  - Needs to keep track of what goals it needs to prove to prove its main hypothesis.

# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

### ❑ Knowledge base

- Facts

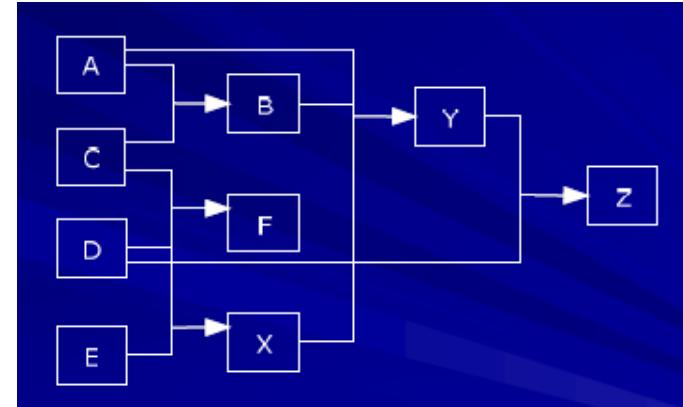
- ❑ A – runny nose
- ❑ B – sinusitis
- ❑ C – headache
- ❑ D - vertigos
- ❑ E – fever
- ❑ F – blood problems
- ❑ X – infection
- ❑ Y – antibiotic
- ❑ Z – bed rest

- Rules

- ❑ R1: *if A is true and C is true, then B is true*
- ❑ R2: *if C is true and D is true then F is true*
- ❑ R3: *if C is true and D is true then E is true*
- ❑ R4: *if A is true and B is true and X is true then Y is true*
- ❑ R5: *if Y is true and D is true then Z is true*

- Goal

- ❑ Fact Z



# Intelligent systems – KBS – RBS

---

## Architecture – inference engine – example

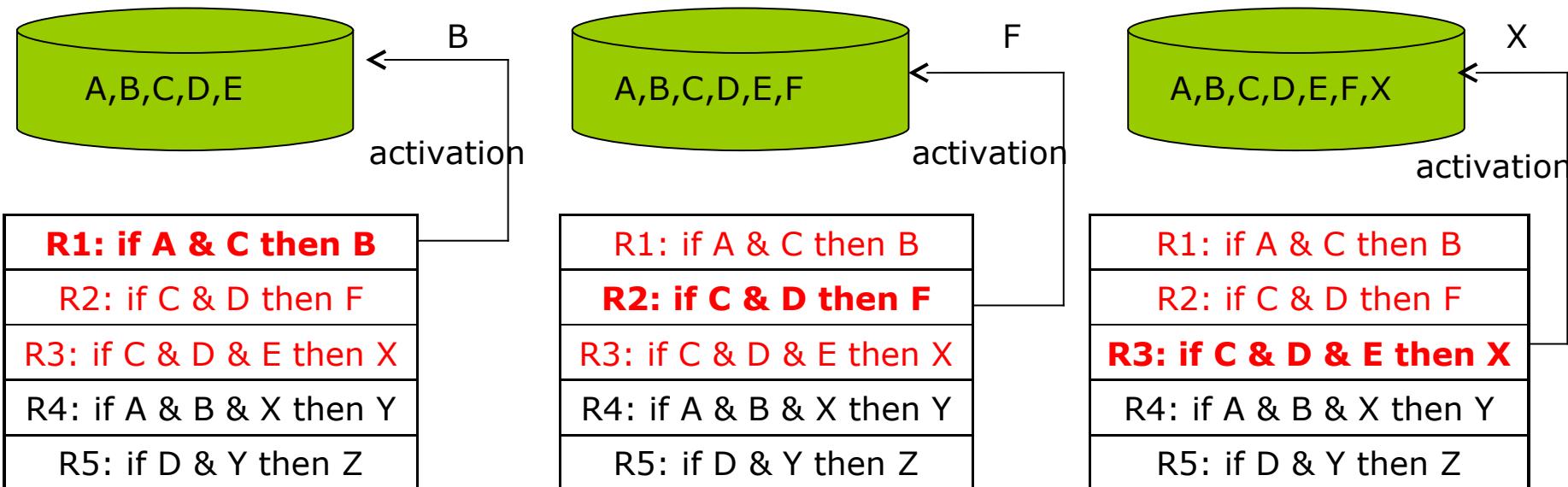
### □ *forward chaining* algorithm

- repeat
  - Select rules that can be applied for facts of KB
    - Rules that contain in LHS facts from KB
  - If for a fact more rules can be applied, choose one of them (that has not been used)
  - Apply the selected rules and add into the KB the new resulted facts
- Until the goal fact or a stop rule is reached

# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

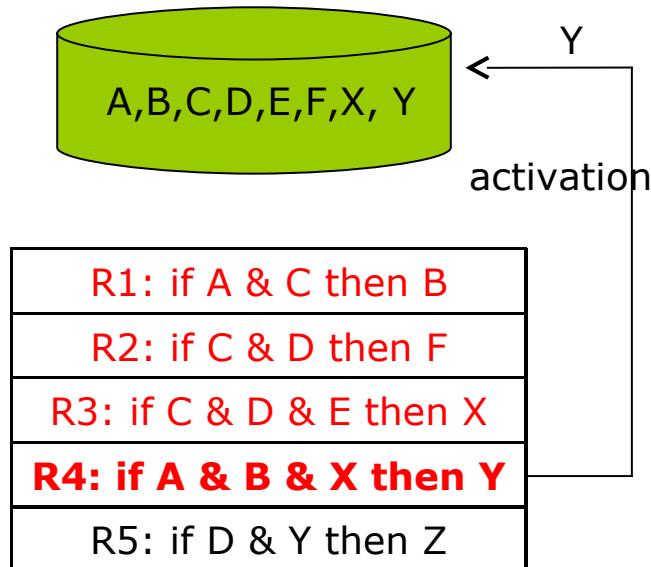
### □ Iteration 1



# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

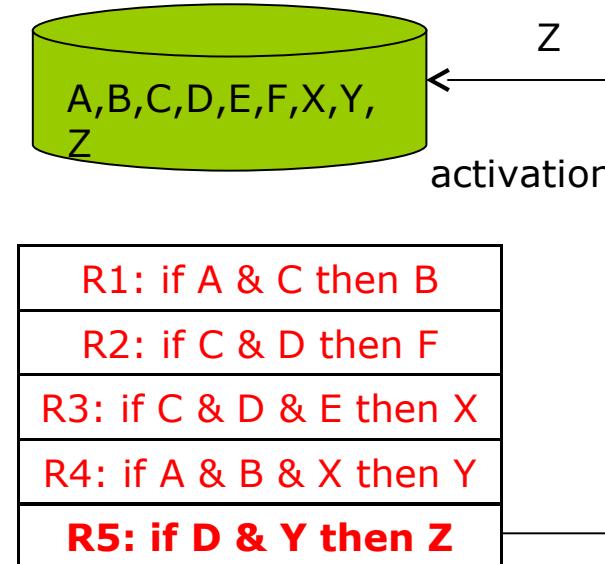
### □ Iteration 2



# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

### □ Iteration 3



# Intelligent systems – KBS – RBS

---

Architecture – inference engine – example

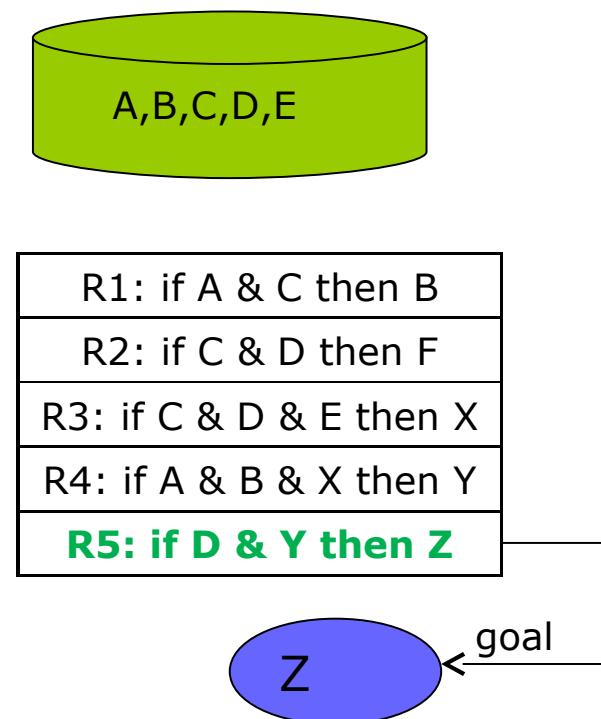
## □ *backward chaining* algorithm

- repeat
  - Select the rules that match the goal
    - Rules that contains in RHS the goal fact
    - If more rules can be applied for a given goal, choose one of them
  - Check the selected rules
    - Replace the goal by causes in the selected rules (these causes become the new goals)
- Until all the goals are true
  - Are known facts (from the KB)
  - Are information provided by user
- Repeat
  - Apply the previous selected rules (in reverse order)
- Until all the rules are applied and the goal is reached (as fact in KB)

# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

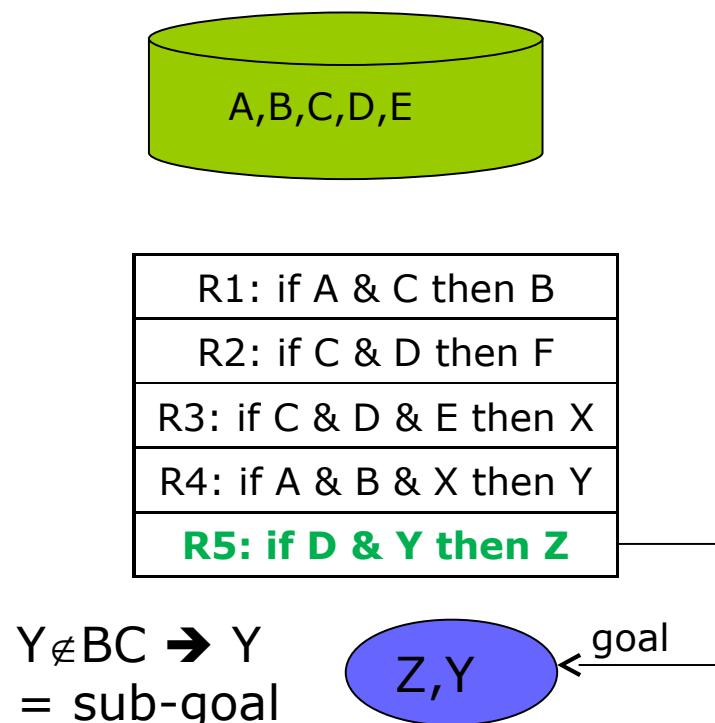
### □ Iteration 1.1



# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

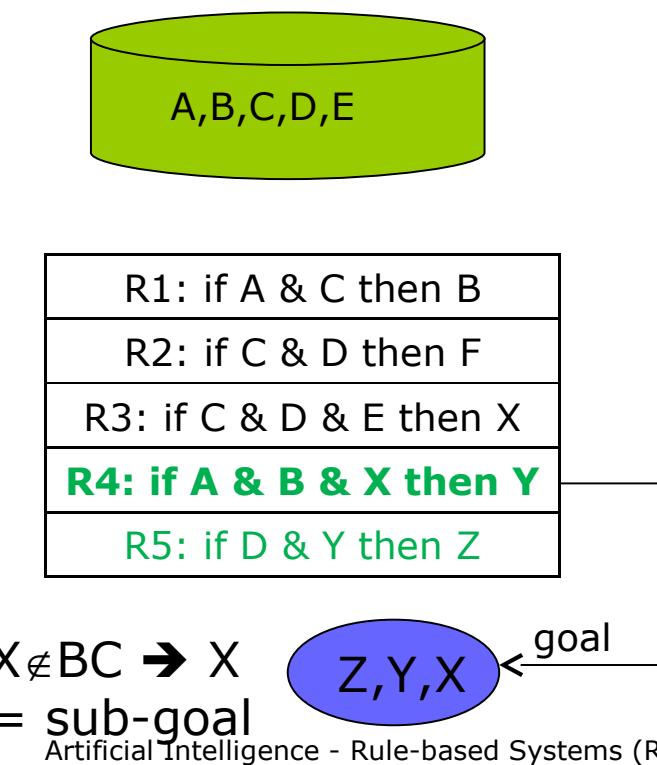
### □ Iteration 1.1



# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

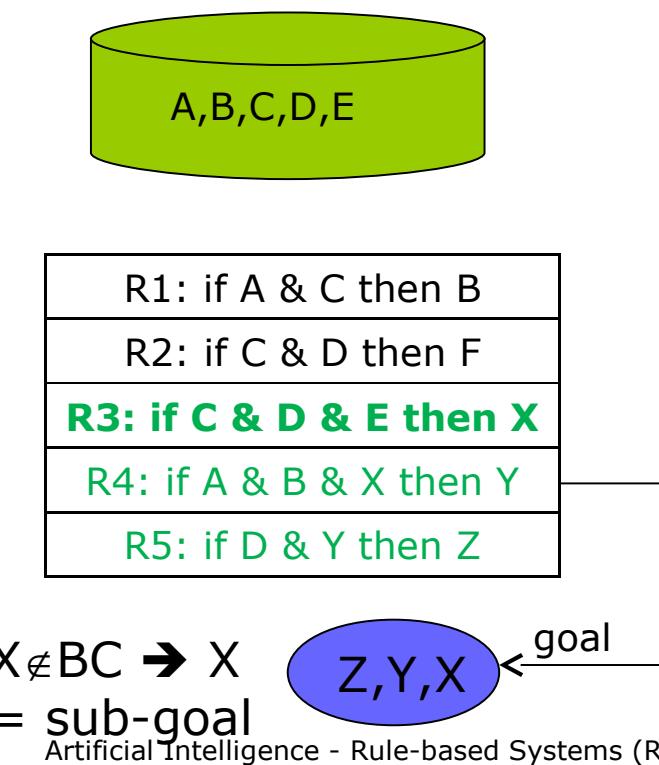
### □ Iteration 1.2



# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

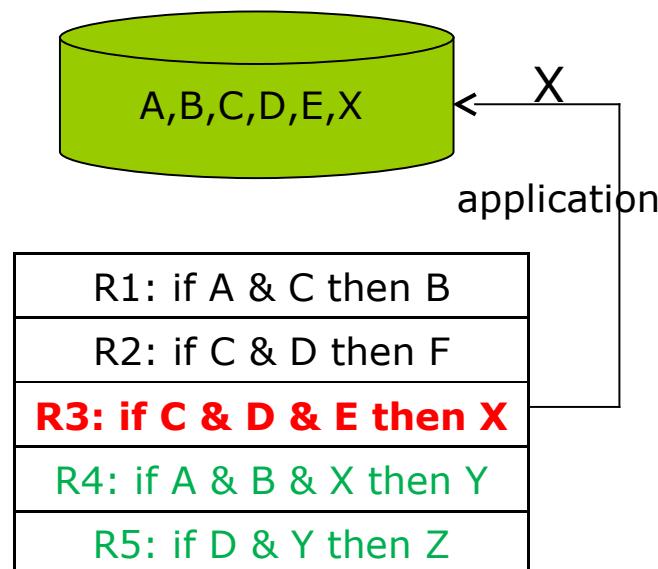
### □ Iteration 1.2



# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

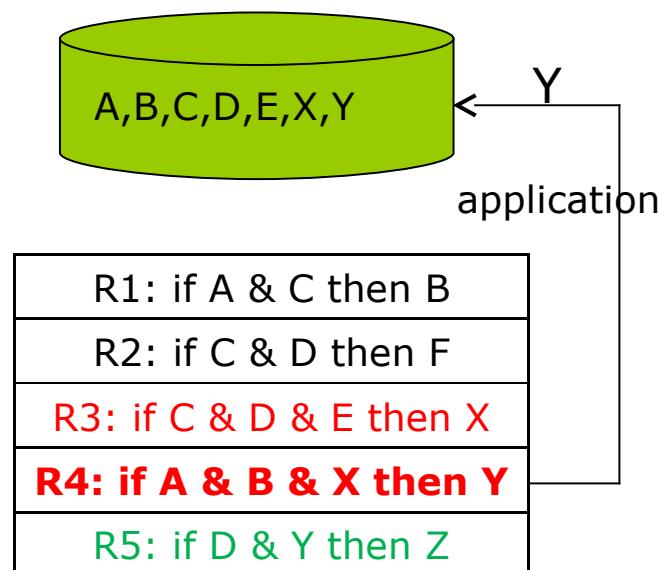
### □ Iteration 2.1



# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

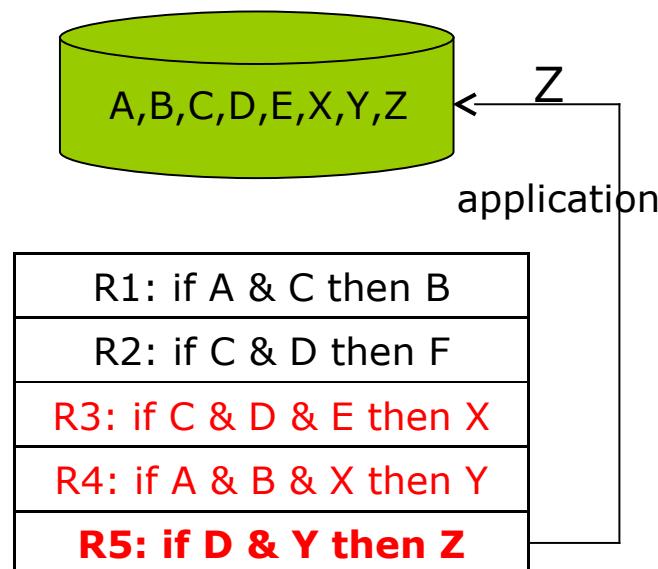
### □ Iteration 2.1



# Intelligent systems – KBS – RBS

## Architecture – inference engine – example

### □ Iteration 2.1



# Intelligent systems – KBS – RBS

---

## Architecture – inference engine

### □ Difficulties

- *Forward Chaining* (FC) or *Backward chaining* (BC)?
- Conflict resolution

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine – difficulties

- *Forward Chaining* (FC) or *Backward chaining* (BC)?
  - FC should be used when:
    - All or most of the data is given in the problem statement;
    - There exist a large number of potential goals but only a few of them are achievable in a particular problem instance.
    - It is difficult to formulate a goal or hypothesis.
  - BC should be used when:
    - A goal or hypothesis is given in the problem statement or can be easily formulated
    - There are a large number of rules that match the facts, producing a large number of conclusions - choosing a goal prunes the search space.
    - Problem data are not given (or easily available)

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine – difficulties

### □ Conflict resolution

- When more than one rule is matched on the facts asserted.

- Eg.

- R1: if color is yellow, then the fruit is apple
    - R2: if color is yellow and shape is longish then the fruit is banana
    - R3: if shape is round then the fruit is apple

- Approaches

- **prima** regulă
  - o regulă **aleatoare**
  - regula **cea mai specifică**
  - **cea mai veche** regulă
  - **cea mai bună** regulă

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine – difficulties

- Conflict resolution

- ***First applicable (First in first serve)***

- If the rules are in a specified order, firing the first applicable one is the easiest way to control the order in which rules fire. From a practical perspective the order can be established by ordering the rules in the knowledge base by placing them in the preferred order
    - Example
      - R1: if color is yellow, then the fruit is apple
      - R2: if color is yellow and shape is longish then the fruit is banana
      - R3: if shape is round then the fruit is apple
    - Remarks
      - Ordering the rules only works for small systems of up to 100 rules).
      - Regulile sunt ordonate doar în sistemele mici

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine – difficulties

### □ Conflict resolution

#### ■ Random:

- A random strategy simply chooses a single random rule to fire from the conflict set. It is also advantageous even though it doesn't provide the predictability or control of the first applicable strategy.

#### □ Example

- R1: if color is yellow, then the fruit is apple
- R2: if color is yellow and shape is longish then the fruit is banana
- R3: if shape is round then the fruit is apple

#### □ Remarks

- Selection can be a good one or a bad one

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine – difficulties

### □ Conflict resolution

#### ■ Most Specific (*Specificity*)

- This strategy is based on the number of conditions of the rules. From the conflict set, the rule with the most conditions is chosen. This is based on the assumption that if it has the most conditions then it has the most relevance to the existing data. It can also be called longest matching strategy and it is based on the assumption that a specific rule process more information than a general one.

#### □ Example

- R1: if color is yellow, then the fruit is apple
- R2: if color is yellow and shape is longish then the fruit is banana
- R3: if shape is round then the fruit is apple

#### □ Remarks

- A specific rule processes more information than a general one  
→*longest matching strategy*

# Intelligent systems – KBS – RBS

---

Architecture – inference engine – difficulties

- Conflict resolution

## ■ Least Recently Used (*Recency*):

- **Each of the rules has a time or step stamp** (or time and date) associated, which marks the last time it was used.
- Example
  - R1: if color is yellow, then the fruit is apple [12.01.2012 – 13.45]
  - R2: if color is yellow and shape is longish then the fruit is banana [7.02.2012 – 21.10]
  - R3: if shape is round then the fruit is apple [10.01.2012 – 10.25]
- Remarks
  - the new rules have been added by an expert whose opinion is less trusted than that of the expert who added the earlier rules. In this case,, it clearly makes more sense to allow the earlier rules priority.

# Intelligent systems – KBS – RBS

---

## Architecture – inference engine – difficulties

- Conflict resolution

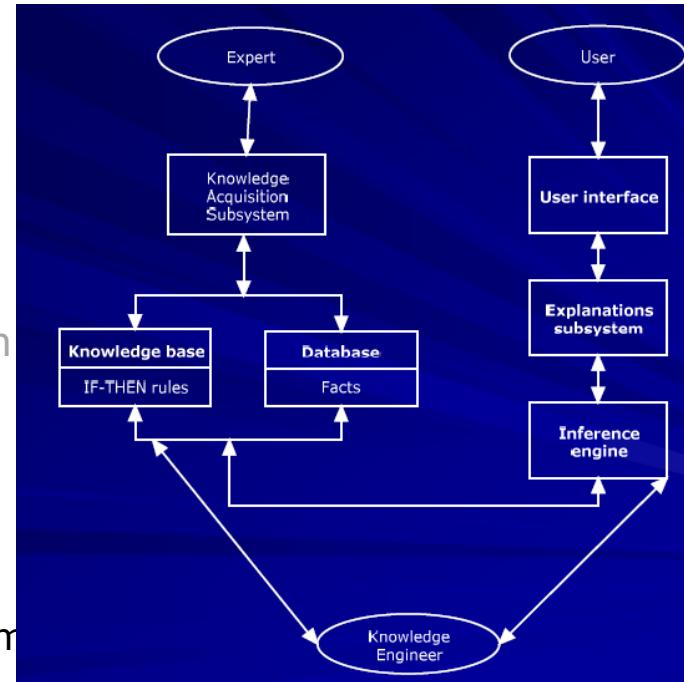
- "Best" rule (**Prioritization**):

- each rule is given a “weight” which specifies how much it should be considered over the alternatives. The rule with the most preferable outcomes is chosen based on this weight
- Example
  - R1: if color is yellow, then the fruit is apple [30%]
  - R2: if color is yellow and shape is longish then the fruit is banana [30%]
  - R3: if shape is round then the fruit is apple [40%]
- Remarks
  - Require human expertise for prioritization

# Intelligent systems – KBS – RBS

## Architecture

- ❑ Knowledge base (KB)
  - Specific information
- ❑ Inference engine (IE)
  - Rules utilized for generating new information
- ❑ User interface
  - Allows dialog with users and accessing facts and knowledge from KB (to be added or updated)
  - A facility for the user to interact with the system
- ❑ Knowledge Acquisition Facility
  - An automatic way to acquire knowledge
- ❑ Explanation Facility
  - Explains reasoning of the system to the user



# Intelligent systems – KBS – RBS

---

## Architecture

### ❑ User interface

- Allows dialog with users and accessing facts and knowledge from KB (to be added or updated)
- A facility for the user to interact with the system
  - ❑ Language processing methods
  - ❑ Menus
  - ❑ Graphical elements

### ❑ Knowledge Acquisition Facility

- An automatic way to acquire knowledge

# Intelligent systems – KBS – RBS

---

## Architecture

- Explanation Facility
  - Explain to users
    - Knowledge of the system
    - Reasoning of the system for obtaining solutions
  - provides the user with a means of understanding the system behavior.
  - This is important because a consultation with a human expert will often require some explanation.
  - Many people would not always accept the answers of an expert without some form of justification.
  - e.g., a medical expert providing a diagnosis/treatment of a patient is expected to explain the reasoning behind his/her conclusions: the uncertain nature of this type of decision may demand a detailed explanation so that the patient concerned is aware of any risks, alternative treatments ,etc.

# Intelligent systems – KBS

---

## ❑ Rule-based systems (RBS)

- Content and objectives
- Design
- Architecture
- Tools and example
- Advantages and limits

# Intelligent systems – KBS – RBS

---

## Tools

- PROLOG
  - Programming language based on *backward chaining*
- ART (Inference Corporation)
  - In 1984, Inference Corporation developed the Automated Reasoning Tool (ART), a forward chaining system.
- ART-IM (Inference Corporation)
  - Following the distribution of NASA's CLIPS, Inference Corporation implemented a forward-chaining only derivative of ART/CLIPS called ART-IM.
- CLIPS –
  - NASA took the forward chaining capabilities and syntax of ART and introduced the "C Language Integrated Production System" (i.e., CLIPS) into the public domain.
- OPS5 (Carnegie Mellon University)
  - OPS5 (Carnegie Mellon University) – First AI language used for Production System (XCON)
- Eclipse (The Haley Enterprise, Inc.)
  - ① Eclipse is the only C/C++ inference engine that supports both Forward and Backward chaining.

# Intelligent systems – KBS – RBS

---

## Examples

- DENDRAL (1965-1983)
  - rule-based expert systems that analyzes molecular structure. Using a plan-generate-test search paradigm and data from mass spectrometers and other sources, DENDRAL proposes plausible candidate structures for new or unknown chemical compounds.
- MYCIN (1972-1980)
  - MYCIN is an interactive program that diagnoses certain infectious diseases, prescribes antimicrobial therapy, and can explain its reasoning in detail
- EMYCIN, HEADMED, CASNET și INTERNIST
  - For medical domain
- PROSPECTOR (1974-1983)
  - Provides advice on mineral exploration
- TEIRESIAS
  - For information retrieval
- XCON (1978-1999)
  - configure computers
- Financial RBSs
  - ExpertTAX, Risk Advisor (Coopers & Lybrand), Loan Probe, Peat/1040 (KPMG), VATIA, Flow Eval (Ernst & Young), Planet, Compas, Comet (Price Waterhouse), Rice (Arthur Andersen), Audit Planning Advisor, World Tax Planner (Deloitte Touche)

# Intelligent systems – KBS – RBS

---

## Advantages and limits

### □ Advantages

- Provide advises to non experts
- Allows the organizations to replicate their very best people. Expert systems carry the intelligence and information found in the intellect of experts and provides this knowledge to other members of the organization.
- Reduce the error due to automation of tedious, repetitive or critical tasks
- Reduce the manpower and time required for system testing and data analysis
- Reduce the costs through acceleration of fault observations
- Eliminate the work that people should not do (such as difficult, time-consuming or error prone tasks, jobs where training needs are large or costly).
- Eliminates work that people would rather not do (such as jobs involving decision making, which does not satisfy everyone; expert systems ensure fair decisions without favoritism in such cases).
- Expert systems perform better than humans in certain situations.
- Perform knowledge acquisition, process analysis, data analysis, system verification
- Increased visibility into the state of the managed system
- Develop functional system requirements
- Coordinate software development
- For simple domains, the rule-base might be simple and easy to verify and validate.
- Expert system shells provide a means to build simple systems without programming.
- Provide consistent answers for repetitive decisions, processes and tasks
- Hold and maintain significant levels of information
- Reduces creating entry barriers to competitors.

### □ Limits

- Narrow domain
- Limited focus
- Instability to learn
- Maintenance problems
- Developmental cost



# Review

---

## ❑ KBSs

- Computational systems that
- Overlap KB and IE

## ❑ Types of KBSs

### ■ LBSs

- Explore knowledge by using methods of formal logic
- Components
  - Language (syntax and semantics) and
  - Deduction method (reasoning)

### ■ RBSs

- Explore knowledge by using methods of human logic
- Can solve problems that do not have a deterministic solution
- Try to simulate a human expert
- Components
  - KB → facts and rules
  - IE → forward and backward chaining

# Next lecture

---

- A. Short introduction in Artificial Intelligence (AI)
- B. Solving search problems
  - A. Definition of search problems
  - B. Search strategies
    - A. Uninformed search strategies
    - B. Informed search strategies
    - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
    - D. Adversarial search strategies
- C. Intelligent systems
  - A. Rule-based systems in certain environments
  - B. Rule-based systems in uncertain environments (Bayes, Fuzzy)**
  - C. Learning systems
    - A. Decision Trees
    - B. Artificial Neural Networks
    - C. Support Vector Machines
    - D. Evolutionary algorithms
  - D. Hybrid systems

# Next lecture – useful information

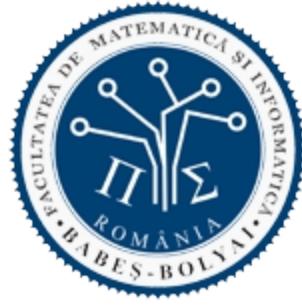
---

- Chapter V of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 3 of *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapters 8 and 9 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

- 
- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean – [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)



BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics



# ARTIFICIAL INTELLIGENCE

**Intelligent systems**

Rule-based systems – uncertainty

# Topics

---

- A. Short introduction in Artificial Intelligence (AI)
- B. Solving search problems
  - A. Definition of search problems
  - B. Search strategies
    - A. Uninformed search strategies
    - B. Informed search strategies
    - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
    - D. Adversarial search strategies
- C. Intelligent systems
  - A. Rule-based systems in certain environments
  - B. Rule-based systems in uncertain environments (Bayes, Fuzzy)**
  - C. Learning systems
    - A. Decision Trees
    - B. Artificial Neural Networks
    - C. Support Vector Machines
    - D. Evolutionary algorithms
  - D. Hybrid systems

# Useful information

---

- Chapter V of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 3 of *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapters 8 and 9 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

# Content

---

## ❑ Intelligent systems

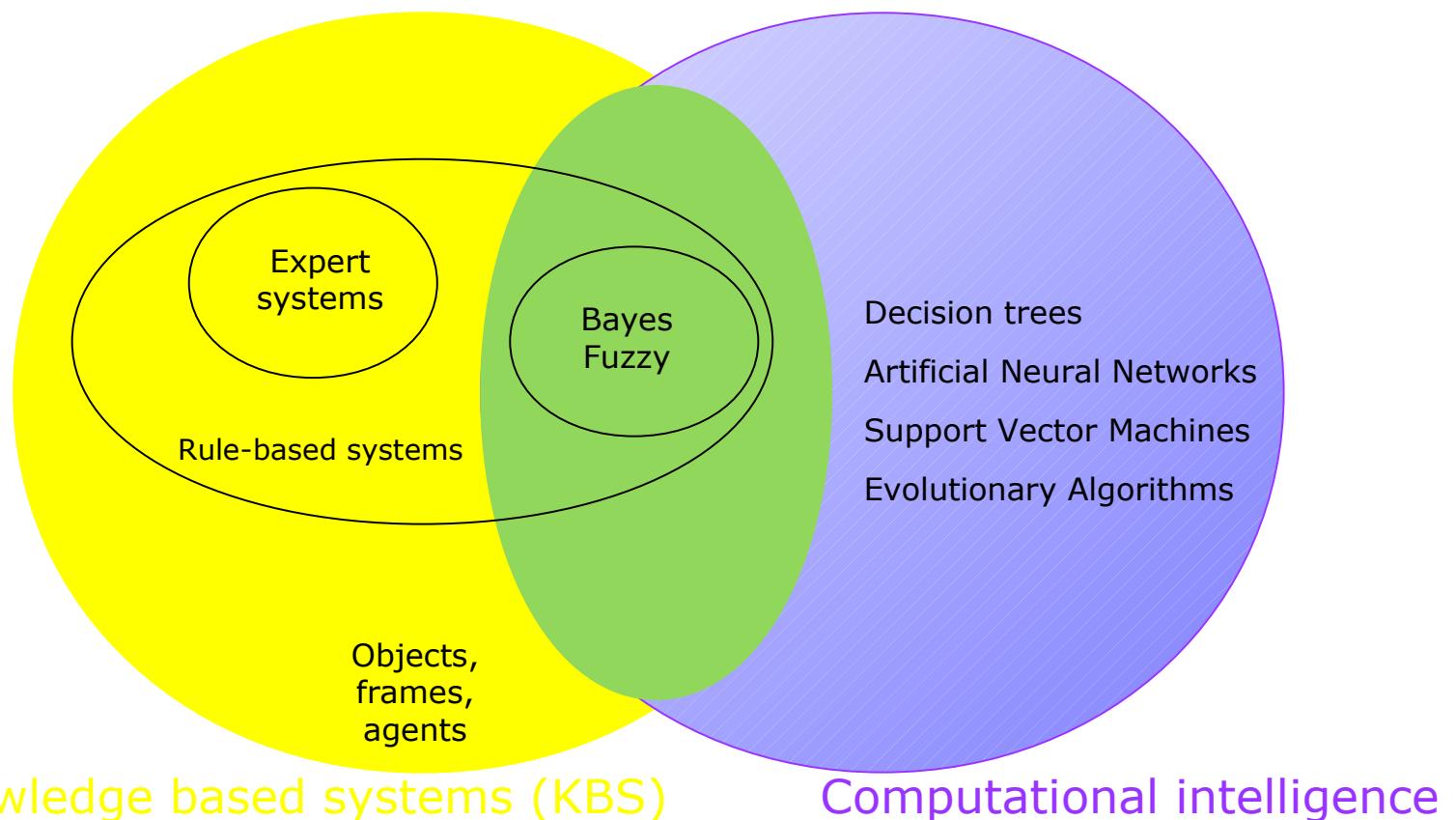
### ■ Knowledge-based systems

#### ❑ Rule-based systems in uncertain environments



# Intelligent systems

---



# Intelligent systems – knowledge-based systems(KBS)

- ❑ Computational systems – composed of 2 parts:
  - Knowledge base (KB)
    - ❑ Specific information of the domain
  - Inference engine (IE)
    - ❑ Rules for generating new information
    - ❑ Domain-independent algorithms

# Intelligent systems - KBS

---

## Knowledge base

### □ Content

- Information (in a given representation) about environment
- Required information for problem solving
- Set of propositions that describe the environment

### □ Typology

- Perfect information
  - Classical logic
    - *IF A is true THEN A is  $\top$  false*
    - *IF B is false THEN B is  $\top$  true*
- Imperfect information
  - Non-exact
  - Incomplete
  - Incommensurable

# Intelligent systems - KBS

---

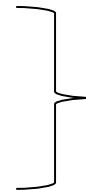
- Sources of uncertainty
  - Imperfection of rules
  - Doubt of rules
  - Using a vague (imprecise) language
- Modalities to express the uncertainty
  - Probabilities
  - Fuzzy logic
  - Bayes theorem
  - Theory of Dempster-Shafer
- Modalities to represent the uncertainty
  - By using a single value → certainty factors, confidence, truth value
    - How sure we are that the given facts are valid
  - By using more values → logic based on ranges
    - Min → lower limit of uncertainty (confidence, necessity)
    - Max → upper limit of uncertainty (plausibility, possibility)

# Intelligent systems - KBS

---

- Reasoning techniques for uncertainty

- Theory of Bayes – probabilistic method
- Theory of certainty
- Theory of possibility (fuzzy logic)



Heuristic  
methods

# Intelligent systems – KBS – certainty factors

## □ Bayes systems

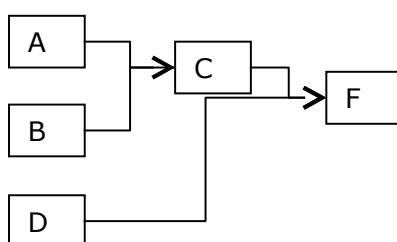
- KBS with probabilistic facts and rules

## □ Systems based on certainty factors

- KBS – facts and rules have associated a certainty factor (confidence factors)
- A kind of Bayes systems with the probabilities replaced by certainty factors

- IF A and B then C

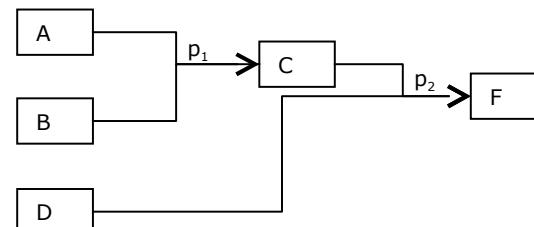
- IF C and D then F



SBR classic

- If A and B then C [with prob  $p_1$ ]

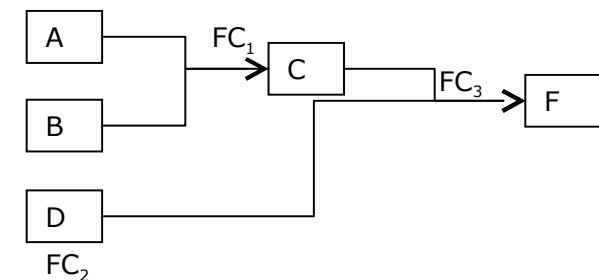
- If C and D then F [with prob  $p_2$ ]



SBR de tip Bayes

- If A and B then C [ $CF_1$ ]

- If C and D [ $CF_2$ ] then F [ $CF_3$ ]



SBR cu FC

# Intelligent systems – KBS – certainty factors

---

## □ Bayes KBSs vs. KBSs based on CFs

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 <i>a priori</i> compute some probabilities	Do not require to <i>a priori</i> compute some probabilities
Hypothesis could be independent or not	Hypothesis are independent

# Intelligent systems – KBS – Bayes systems

---

- Elements of probability theory
- Content and design
- Typology
- Tools
- Advantages and limits

# Intelligent systems – KBS – Bayes systems

---

## Amintim componența unui SBC

- Baza de cunoștințe (BC) → Modalități de reprezentare a cunoștințelor
  - Logica formală (limbaje formale)
    - Definiție
      - Știința principiilor formale de raționament
    - Componente
      - Sintaxă
      - Semantică
      - Metodă de inferență sintactică
    - Tipologie
      - În funcție de numărul valorilor de adevăr:
        - logică duală
        - logică polivalentă
      - În funcție de tipul elementelor de bază:
        - clasică → primitivele = propoziții (predicate)
        - probabilistică → primitivele = variabile aleatoare
      - În funcție de obiectul de lucru:
        - logica propozițională → se lucrează doar cu propoziții declarative, iar obiectele descrise sunt fixe sau unice (Ionică este student)
        - logica predicatelor de ordin I → se lucrează cu propoziții declarative, cu predicate și cuantificări, iar obiectele descrise pot fi unice sau variabile asociate unui obiect unic (Toți studenții sunt prezenți)
    - Reguli
    - Rețele semantice
  - Modulul de control (MC – pentru inferență)

# Intelligent systems – KBS – Bayes systems

---

## Elemente de teoria probabilităților

- ❑ Teorii ale probabilităților
- ❑ Concepte de bază
  - Teoria clasică și teoria modernă
  - Eveniment
  - Probabilitate simplă
  - Probabilitate condiționată
  - Axiome

# Intelligent systems – KBS – Bayes systems

---

## Elemente de teoria probabilităților

### ❑ Teorii ale probabilităților

#### ■ Teoria clasică (*a priori*)

- ❑ Propusă de Pascal și Fermat în 1654
- ❑ Lucrează cu sisteme ideale:
  - toate posibilele evenimente sunt cunoscute
  - toate evenimentele se pot produce cu aceeași probabilitate (sunt uniform distribuite)
- ❑ evenimente discrete
- ❑ metode combinatoriale
- ❑ spațiul rezultatelor posibile este continuu

#### ■ Teoria modernă

- ❑ evenimente continue
- ❑ metode combinatoriale
- ❑ spațiul rezultatelor posibile este cuantificabil

# Intelligent systems – KBS – Bayes systems

---

## Elemente de teoria probabilităților

### □ Concepte de bază

- Considerăm un experiment care poate produce mai multe ieșiri (rezultate)
  - Ex. *Ev1: Aruncarea unui zar poate produce apariția uneia din cele 6 fețe ale zarului (deci 6 rezultate)*
- Eveniment
  - Definiție
    - producerea unui anumit rezultat
    - Ex. *Ev2: Apariția feței cu nr 3*
    - Ex. *Ev3: Apariția unei fețe cu un nr par (2,4,6)*
  - Tipologie
    - Evenimente independente și mutual exclusive
      - Nu se pot produce simultan
      - Ex. *Ev4: Apariția feței 1 la aruncarea unui zar și Ev5: Apariția feței 3 la aruncarea unui zar*
    - Dependente
      - Producerea unor evenimente afectează producerea altor evenimente
      - Ex. *Ev6: Apariția feței 6 la prima aruncare a unui zar și Ev7: Apariția unor fețe a căror numere însumate să dea 8 la 2 aruncări successive ale unui zar*
- Multimea tuturor rezultatelor = *sample space* al experimentului
  - Ex. pentru *Ev1*: (1,2,3,4,5,6)
- Multimea tuturor rezultatelor tuturor evenimentelor posibile = *power set* (multimea părților)

# Intelligent systems – KBS – Bayes systems

---

## Elemente de teoria probabilităților

### □ Concepte de bază

#### ■ Probabilitate simplă $p(A)$

- probabilitatea producerii unui eveniment A independent de alte evenimente (B)
- șansa ca acel eveniment să se producă
- proporția cazurilor de producere a evenimentului în mulțimea tuturor cazurilor posibile
- nr cazurilor favorabile / nr cazurilor posibile
- un număr real în  $[0,1]$ 
  - 0 – imposibilitate absolută
  - 1 – posibilitate absolută
- Ex.  $P(Ev1) = 1/6$ ,  $P(Ev3) = 3/6$

#### ■ Probabilitate condiționată $p(A|B)$

- probabilitatea producerii unui eveniment A dependentă de producerea altor evenimente (B)
- proporția cazurilor de producere a evenimentului A și a evenimentului B în mulțimea tuturor cazurilor producerii evenimentului B
- probabilitatea comună /probabilitatea lui B

$$p(A|B) = \frac{p(A \cap B)}{p(B)}$$

# Intelligent systems – KBS – Bayes systems

---

## Elemente de teoria probabilităților

### □ Concepte de bază

#### ■ Axiome

□  $0 \leq p(E) \leq 1$  pentru orice eveniment  $E$

□  $p(Adevărat) = 1, p(Fals) = 0$

□  $\sum_i p(E_i) = 1$

□ <sup>i</sup>Dacă  $A$  și  $B$  sunt independente

- $p(A \cup B) = p(A) + p(B)$

- $p(A \cap B) = p(A) * p(B)$

□ Dacă  $A$  și  $B$  nu sunt independente

- Dacă  $A$  depinde de  $B$

- $p(A \cup B) = p(A) + p(B) - p(A \cap B)$

- $p(A \cap B) = p(A|B) * p(B)$

- $p(B \cap A) = p(A \cap B)$

- $p(A|B) = \frac{p(B|A)p(A)}{p(B)}$  (b)

- Dacă  $A$  depinde de  $B_1, B_2, \dots, B_n$  (evenimente mutual exclusive)

- $p(A) = \sum_{i=1}^n p(A|B_i)p(B_i)$  (a)

# Intelligent systems – KBS – Bayes systems

---

## Elemente de teoria probabilităților

### □ Concepte de bază

#### ■ Exemplu

- Dacă A depinde de 2 evenimente mutual exclusive (B și  $\neg B$ ), FC ec.

$$p(A) = \sum_{i=1}^n p(A | B_i) p(B_i) \quad \text{avem:}$$

$$\quad \square \quad p(B) = p(B|A)p(A) + p(B|\neg A)p(\neg A)$$

- Înlocuind pe  $p(B)$  în ec.  $p(A | B) = \frac{p(B | A)p(A)}{p(B)}$  se obține ec.:

$$p(A | B) = \frac{p(B | A)p(A)}{p(B | A)p(A) + p(B | \neg A)p(\neg A)} \quad (c)$$

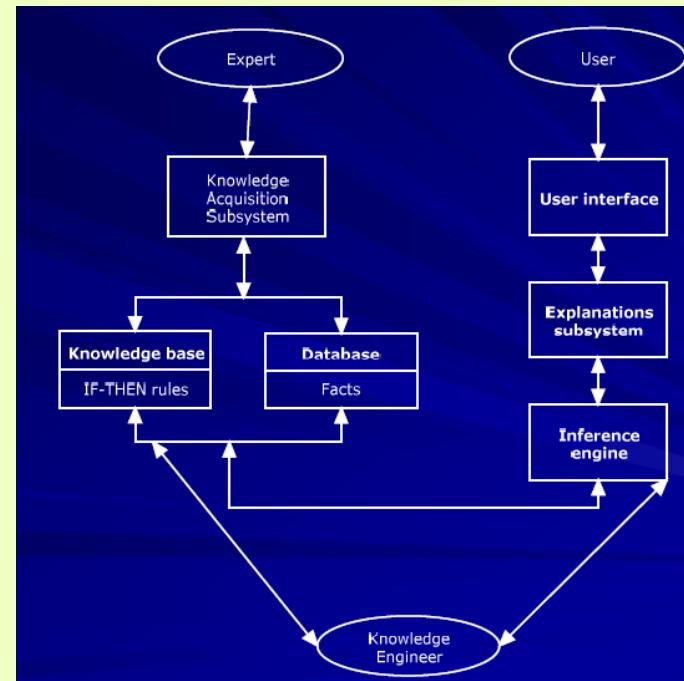
- Ecuația (c) se folosește pentru controlul incertitudinii în sistemele expert

# Intelligent systems – KBS – Bayes systems

## Reamintim ca un SBR are următoarea

### Arhitectură

- ❑ Baza de cunoștințe (BC)
  - Informațiile specifice despre un domeniu
- ❑ Modulul de control (MC)
  - Regulile prin care se pot obține informații noi
- ❑ Interfața cu utilizatorul
  - permite dialogul cu utilizatorii în timpul sesiunilor de consultare, precum și accesul acestora la faptele și cunoștințele din BC pentru adăugare sau actualizare
- ❑ Modulul de îmbogățire a cunoașterii
  - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe.
- ❑ Modulul explicativ
  - are rolul de a explica utilizatorilor atât cunoștințele de care dispune sistemul, cât și raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare. Explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percep și acceptă sistemul



# Intelligent systems – KBS – Bayes systems

---

## Reamintim: SBR – arhitectură

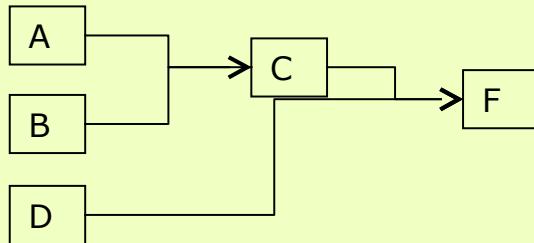
- baza de cunoștințe
  - Conținut
    - ▣ Informațiile specifice despre un domeniu sub forma unor
      - fapte – afirmații corecte
      - reguli - euristici speciale care generează informații (cunoștințe)
  - Rol
    - ▣ stocarea tuturor elementelor cunoașterii (fapte, reguli, metode de rezolvare, euristici) specifice domeniului de aplicație, preluate de la experții umani sau din alte surse
- modulul de control
  - Conținut
    - ▣ regulile prin care se pot obține informații noi
    - ▣ algoritmi independenti de domeniu
    - ▣ creierul SBR – un algoritm de deducere bazat pe BC și specific metodei de raționare
      - un program în care s-a implementat cunoașterea de control, procedurală sau operatorie, cu ajutorul căruia se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii.
    - ▣ depinde de complexitate și tipul cunoștințelor cu care are de-a face
  - Rol
    - ▣ cu ajutorul lui se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii

# Intelligent systems – KBS – Bayes systems

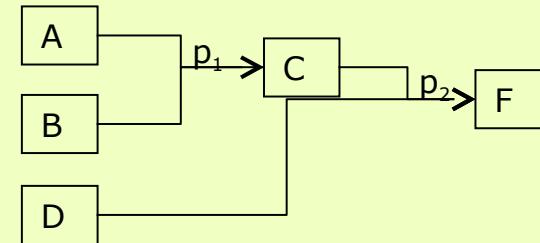
## Conținut și arhitectură

### ❑ Ideea de bază

- SBR (Sisteme expert) în care faptele și regulile sunt probabilistice
- Dacă A și B atunci C
- Dacă C și D atunci F
- Dacă A și B atunci C [cu probabilitatea  $p_1$ ]
- Dacă C și D atunci F [cu probabilitatea  $p_2$ ]



SBR classic



SBR de tip Bayes

# Intelligent systems – KBS – Bayes systems

---

## Conținut și arhitectură

- ❑ Regulile din BC sunt (în general) de forma:
  - Dacă evenimentul (faptul) I este adevărat, atunci evenimentul (faptul) D este adevărat [cu probabilitatea p]
  - Dacă evenimentul I s-a produs, atunci evenimentul D se va produce cu probabilitatea p
    - I – ipoteza (asertiune, concluzie)
    - D – dovada (premisa) care susține ipoteza
- unde:
$$p(I|D) = \frac{p(D|I)p(I)}{p(D|I)p(I) + p(D|\neg I)p(\neg I)} \quad (d)$$
  - $p(I)$  – probabilitatea apriori ca ipoteza I să fie adevărată
  - $p(D|I)$  – probabilitatea ca ipoteza I fiind adevărată să implice dovada D
  - $p(\neg I)$  – probabilitatea apriori ca ipoteza I să fie falsă
  - $p(D|\neg I)$  - probabilitatea găsirii dovezii D chiar dacă ipoteza I este falsă
- ❑ Cum și cine calculează aceste probabilități? → modulul de control

# Intelligent systems – KBS – Bayes systems

---

## Conținut și arhitectură

- Cum calculează MC aceste probabilități într-un SBR?

$$p(I|D) = \frac{p(D|I)p(I)}{p(D|I)p(I) + p(D|\neg I)p(\neg I)} \quad (d)$$

- utilizatorul furnizează informații privind dovezile observate
- experții determină probabilitățile necesare rezolvării problemei
  - Probabilități apriori pentru posibile ipoteze (adevărate sau false) –  $p(I)$  și  $p(\neg I)$
  - Probabilitățile condiționate pentru observarea dovezii D dacă ipoteza I este adevărată  $p(D|I)$ , respectiv falsă  $p(D|\neg I)$
- SBR calculează probabilitatea posteriori  $p(I|D)$  pentru ipoteza I în condițiile dovezilor D furnizate de utilizator
- Actualizare de tip Bayes
  - O tehnică de actualizare a probabilității  $p$  asociate unei reguli care susține o ipoteză pe baza dovezilor (pro sau contra)
  - Inferență (raționament) de tip Bayes

# Intelligent systems – KBS – Bayes systems

---

## Conținut și arhitectură

### ❑ Actualizare de tip Bayes

- O tehnică de actualizare a probabilității  $p$  asociate unei reguli care susține o ipoteză pe baza dovezilor (pro sau contra)
- Actualizarea poate săține cont de:
  - ❑ una sau mai multe ( $m$ ) ipoteze (exclusive și exhaustive)
  - ❑ una sau mai multe ( $n$ ) dovezi (exclusive și exhaustive)
- Cazuri:
  - ❑ Mai multe ipoteze și o singură dovedă
  - ❑ Mai multe ipoteze și mai multe dovezi

$$p(I_i | D) = \frac{p(D | I_i)p(I_i)}{\sum_{k=1}^m p(D | I_k)p(I_k)}$$

$$p(I_i | D_1D_2...D_n) = \frac{p(D_1D_2...D_n | I_i)p(I_i)}{\sum_{k=1}^m p(D_1D_2...D_n | I_k)p(I_k)} = \frac{p(D_1 | I_i)p(D_2 | I_i)...p(D_n | I_i)p(I_i)}{\sum_{k=1}^m p(D_1D_2...D_n | I_k)p(I_k)}$$

# Intelligent systems – KBS – Bayes systems

## Conținut și arhitectură

### ❑ Exemplu numeric

#### ■ Pp. un SBR în care:

- ❑ utilizatorul
  - furnizează 3 dovezi condiționate independente  $D_1$ ,  $D_2$  și  $D_3$
- ❑ expertul
  - crează 3 ipoteze mutual exclusive și exhaustive  $I_1$ ,  $I_2$  și  $I_3$  și stabilește probabilitățile asociate lor –  $p(I_1)$ ,  $p(I_2)$  și  $p(I_3)$
  - determină probabilitățile condiționate pentru observarea fiecărei dovezi pentru toate ipotezele posibile

probabilitatea	Ipotezele		
	$i = 1$	$i = 2$	$i = 3$
$p(I_i)$	0.40	0.35	0.25
$p(D_1 I_i)$	0.30	0.80	0.50
$p(D_2 I_i)$	0.90	0.00	0.70
$p(D_3 I_i)$	0.60	0.70	0.90

# Intelligent systems – KBS – Bayes systems

## Conținut și arhitectură

### ❑ Exemplu numeric

- Presupunem că prima dovardă observată este  $D_3$
- SE calculează probabilitățile posteriori  $p(I_i|D_3)$  pentru toate ipotezele:

$$p(I_1 | D_3) = \frac{0.60 \cdot 0.40}{0.60 \cdot 0.40 + 0.70 \cdot 0.35 + 0.90 \cdot 0.25} = 0.34$$

$$p(I_2 | D_3) = \frac{0.70 \cdot 0.35}{0.60 \cdot 0.40 + 0.70 \cdot 0.35 + 0.90 \cdot 0.25} = 0.34$$

$$p(I_3 | D_3) = \frac{0.90 \cdot 0.25}{0.60 \cdot 0.40 + 0.70 \cdot 0.35 + 0.90 \cdot 0.25} = 0.32$$

- După observarea dovezii  $D_3$ 
  - ❑ Încrederea în ipoteza  $I_2$  este aceeași cu încrederea în ipoteza  $I_1$
  - ❑ Încrederea în ipoteza  $I_3$  crește

# Intelligent systems – KBS – Bayes systems

## Conținut și arhitectură

### ❑ Exemplu numeric

- Presupunem că a doua dovardă observată este  $D_1$
- SE calculează probabilitățile posteriori  $p(I_i|D_1D_3)$  pentru toate ipotezele:

$$p(I_1 | D_1D_3) = \frac{0.30 \cdot 0.60 \cdot 0.40}{0.30 \cdot 0.60 \cdot 0.40 + 0.80 \cdot 0.70 \cdot 0.35 + 0.50 \cdot 0.90 \cdot 0.25} = 0.19$$

$$p(I_2 | D_1D_3) = \frac{0.80 \cdot 0.70 \cdot 0.35}{0.30 \cdot 0.60 \cdot 0.40 + 0.80 \cdot 0.70 \cdot 0.35 + 0.50 \cdot 0.90 \cdot 0.25} = 0.52$$

$$p(I_3 | D_1D_3) = \frac{0.50 \cdot 0.90 \cdot 0.25}{0.30 \cdot 0.60 \cdot 0.40 + 0.80 \cdot 0.70 \cdot 0.35 + 0.50 \cdot 0.90 \cdot 0.25} = 0.29$$

- După observarea dovezii  $D_1$ 
  - ❑ Încrederea în ipoteza  $I_1$  scade
  - ❑ Încrederea în ipoteza  $I_2$  crește (fiind cea mai probabilă de a fi adevărată)
  - ❑ Încrederea în ipoteza  $I_3$  crește

probabilitatea	Ipotezele		
	$i = 1$	$i = 2$	$i = 3$
$p(I_i)$	0.40	0.35	0.25
$p(D_1 I_i)$	0.30	0.80	0.50
$p(D_2 I_i)$	0.90	0.00	0.70
$p(D_3 I_i)$	0.60	0.70	0.90

# Intelligent systems – KBS – Bayes systems

## Conținut și arhitectură

### ❑ Exemplu numeric

- Presupunem că ultima dovardă observată este  $D_2$
- Se calculează probabilitățile posterioare  $p(I_i | D_2 D_1 D_3)$  pentru toate ipotezele:

$$p(I_1 | D_2 D_1 D_3) = \frac{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40}{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40 + 0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35 + 0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25} = 0.45$$

$$p(I_2 | D_2 D_1 D_3) = \frac{0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35}{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40 + 0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35 + 0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25} = 0.00$$

$$p(I_3 | D_2 D_1 D_3) = \frac{0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25}{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40 + 0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35 + 0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25} = 0.55$$

- După observarea dovezii  $D_2$ 
  - ❑ Încrederea în ipoteza  $I_1$  crește
  - ❑ Încrederea în ipoteza  $I_2$  e nulă (ipoteza e falsă)
  - ❑ Încrederea în ipoteza  $I_3$  crește

probabilitatea	Ipotezele		
	$i = 1$	$i = 2$	$i = 3$
$p(I_i)$	0.40	0.35	0.25
$p(D_1   I_i)$	0.30	0.80	0.50
$p(D_2   I_i)$	0.90	0.00	0.70
$p(D_3   I_i)$	0.60	0.70	0.90

# Intelligent systems – KBS – Bayes systems

## Conținut și arhitectură

### ❑ Exemplu practic

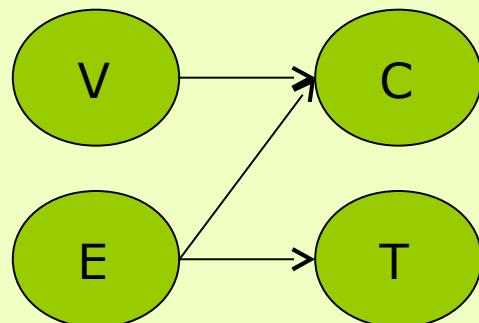
- Presupunem cazul unei mașini care nu pornește când este accelerată, dar scoate fum
  - ❑ Dacă scoate fum, atunci accelerația este defectă [cu probabilitatea 0.7]
  - ❑  $P(I_1|D_1) = 0.7$
- Pe baza unor observări statistice, experții au constatat:
  - ❑ următoarea regulă:
    - Dacă accelerația este defectă, atunci mașina scoate fum [cu probabilitatea 0.85]
  - ❑ probabilitatea ca mașina să pornească din cauză că accelerația este defectă = 0.05 (probabilitate *apriori*)
  - ❑ deci avem
    - 2 ipoteze:
      - $I_1$ : accelerația este defectă
      - $I_2$ : accelerația nu este defectă
    - o dovedă
      - $D_1$ : mașina scoate fum
    - probabilitatea că accelerația este defectă dacă mașina scoate fum
      - $P(I_1|D_1)=P(D_1|I_1)*P(I_1)/(P(D_1|I_1)*P(I_1)+P(D_1|I_2)*P(I_2))$
      - $P(I_1|D_1)=0.23 < 0.7$

	$I_1$	$I_2$
$p(I_i)$	0.05	$1-0.05=0.95$
$P(D_1 I_i)$	0.85	$1-0.85=0.15$

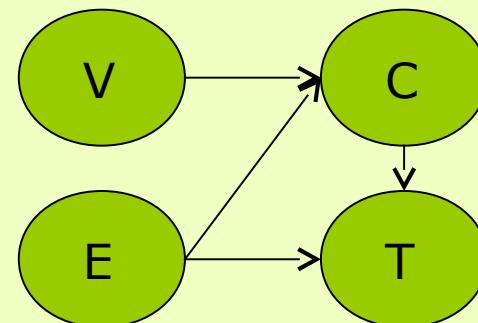
# Intelligent systems – KBS – Bayes systems

## Tipologie

- ❑ Sisteme simple de tip Bayes
  - Consecințele unei ipoteze nu sunt corelate
- ❑ Rețele de tip Bayes
  - Consecințele unei ipoteze pot fi corelate
- ❑ De exemplu, reținem informații despre vârstă (V), educația (E), câștigurile (C) și preferința pentru teatru (T) ale unor persoane



Sistem Bayes simplu (naiv)



Rețea Bayes simplu

# Intelligent systems – KBS – Bayes systems

---

## □ Tool-uri

- MSBNx – [view](#)
- JavaBayes – [view](#)
- BNJ – [view](#)

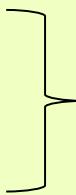
# Intelligent systems – KBS – Bayes systems

---

- Avantaje ale inferenței de tip Bayes
  - Tehnică bazată pe teoreme statistice
  - Probabilitatea dovezilor (simptomelor) în ipotezele (cauzele) date sunt posibil de furnizat
  - Probabilitatea unei ipoteze se poate modifica datorită uneia sau mai multor dovezi
  
- Dezavantaje ale inferenței de tip Bayes
  - Trebuie cunoscute (sau ghicite) probabilitățile apriori ale unor ipoteze

# Intelligent systems – KBS

---

- Tehnici de raționare în medii nesigure
    - Teoria Bayesiana – metodă probabilistică
    - **Teoria certitudinii**
    - Teoria posibilității (logica fuzzy)
- 
- Metode euristice

# Intelligent systems – KBS – certainty factors

---

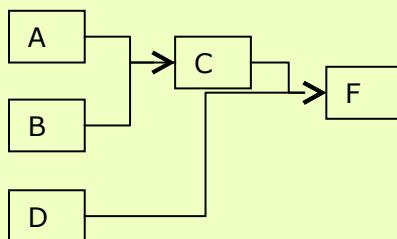
- Conținut și arhitectură
- Tipologie
- Tool-uri
- Avantaje și dezavantaje

# Intelligent systems – KBS – certainty factors

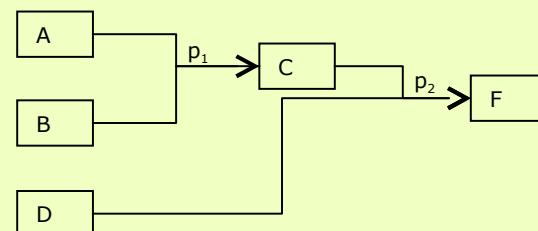
## Conținut și arhitectură

### □ Ideea de bază

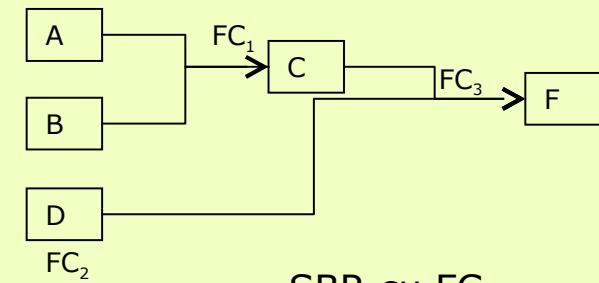
- SBR (sisteme expert) în care faptele și regulile au asociate câte un factor de certitudine (FC)/coeficient de încredere
  - Un fel de sisteme de tip Bayes în care probabilitățile sunt înlocuite cu factori de certitudine
- 
- Dacă A și B atunci C
  - Dacă C și D atunci F
  - Dacă A și B atunci C [cu prob  $p_1$ ]
  - Dacă C și D atunci F [cu prob  $p_2$ ]
  - Dacă A și B atunci C [ $FC_1$ ]
  - Dacă C și D atunci F [ $FC_3$ ]



SBR classic



SBR de tip Bayes



SBR cu FC

# Intelligent systems – KBS – certainty factors

---

Conținut și arhitectură

- ❑ FC măsoară încrederea acordată unor fapte sau reguli
- ❑ Utilizarea FC → alternativă la actualizarea de tip Bayes
- ❑ FC pot fi aplicați
  - faptelor
  - regulilor (concluziei/concluziilor unei reguli)
  - fapte + reguli
- ❑ Într-un SBR (sistem expert) cu factori de certitudine
  - regulile sunt de forma:
    - dacă dovada atunci ipoteza [FC]
    - dacă dovada<sub>[FC]</sub> atunci ipoteza
    - dacă dovada<sub>[FC]</sub> atunci ipoteza [FC]
  - ipotezele susținute de probe sunt independente

# Intelligent systems – KBS – certainty factors

## Conținut și arhitectură

### □ FC – mod de calcul

#### ■ Măsura încrederei (measure of belief – MB)

- măsura creșterii încrederei în ipoteza  $I$  pe baza dovezii  $D$

$$MB(I, D) = \begin{cases} 1, & \text{dacă } p(I) = 1 \\ \frac{p(I | D) - p(I)}{1 - p(I)} & \text{dacă } p(I) < 1 \end{cases}$$

#### ■ Măsura neîncrederei (measure of disbelief – MD)

- măsura creșterii neîncrederei în ipoteza  $I$  pe baza dovezii  $D$

$$MD(I, D) = \begin{cases} 1, & \text{dacă } p(I) = 0 \\ \frac{p(I) - p(I | D)}{p(I)} & \text{dacă } p(I) > 0 \end{cases}$$

#### ■ Pentru evitarea valorilor negative ale MB și MD:

$$MB(I, D) = \begin{cases} 1, & \text{dacă } p(I) = 1 \\ \frac{\max\{p(I | D), p(I)\} - p(I)}{1 - p(I)} & \text{dacă } p(I) < 1 \end{cases}$$

$$MD(I, D) = \begin{cases} 1, & \text{dacă } p(I) = 0 \\ \frac{\min\{p(I | D), p(I)\} - p(I)}{0 - p(I)} & \text{dacă } p(I) > 0 \end{cases}$$

#### ■ FC – încrederea în ipoteza $I$ dată fiind dovada $D$

- Număr din  $[-1, 1]$
- $FC = -1$  dacă se știe că ipoteza  $I$  este falsă
- $FC = 0$  dacă nu se știe nimic despre ipoteza  $I$
- $FC = 1$  dacă se știe că ipoteza  $I$  este adevărată

$$FC(I, D) = \frac{MB(I, D) - MD(I, D)}{1 - \min\{MB(I, D), MD(I, D)\}}$$

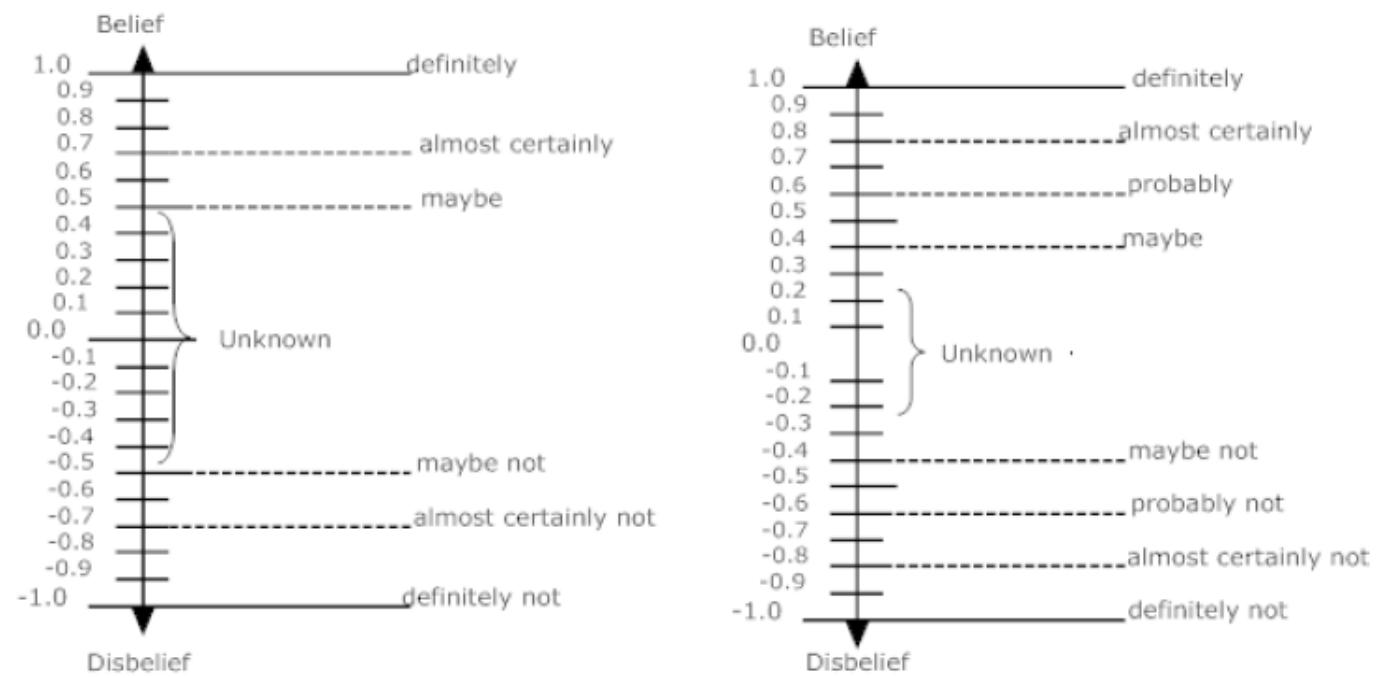
# Intelligent systems – KBS – certainty factors

## Conținut și arhitectură

### ❑ FC – mod de calcul

#### ❑ Încrederea în ipoteza I dată fiind dovada D

- $FC = -1$  dacă se știe că ipoteza este falsă
- $FC = 0$  dacă nu se știe nimic despre ipoteză
- $FC = 1$  dacă se știe că ipoteza este adevărată



# Intelligent systems – KBS – certainty factors

---

## Conținut și arhitectură

### ❑ FC – mod de calcul

❑ Încrederea în ipoteza  $I$  dată fiind dovada  $D$

❑ ipoteza  $I$  poate fi:

- simplă (ex. *Dacă D atunci I*)
- compusă (ex. *Dacă D atunci  $I_1$  și  $I_2$  și ...  $I_n$* )

❑ dovada  $D$  poate fi

- dpdv al compozitiei:
  - simplă (ex. *Dacă D atunci I*)
  - compusă (ex. *Dacă D<sub>1</sub> și D<sub>2</sub> și ... D<sub>n</sub> atunci I*)
- dpdv al incertitudinii (încrederii în dovedă):
  - sigură (ex. *Dacă D atunci I*)
  - nesigură (ex. *D[FC] atunci I*)

# Intelligent systems – KBS – certainty factors

---

## Conținut și arhitectură

- ❑ FC – mod de calcul pentru combinarea încrederii
  - ❑ o doavadă incertă care susține sigur o ipoteză
  - ❑ mai multe dovezi incerte care susțin sigur o singură ipoteză
  - ❑ o doavadă incertă care susține incert o ipoteză
  - ❑ mai multe dovezi incerte care susțin incert o ipoteză

# Intelligent systems – KBS – certainty factors

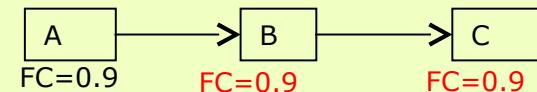
## Conținut și arhitectură

- ❑ FC – mod de calcul pentru combinarea încrederei
  - ❑ O dovadă incertă care susține sigur o ipoteză

$$FC(I) = \begin{cases} FC(D), & \text{dacă } FC(D) > 0 \\ 0, & \text{altfel} \end{cases}$$

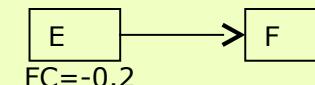
### ❑ Exemplul 1

- ❑  $R_1$ : Dacă  $A_{[FC=0.9]}$  atunci  $B$
- ❑  $R_2$ : Dacă  $B$  atunci  $C$
- ❑  $FC(B)=FC(A)=0.9$
- ❑  $FC(C)=FC(B)=0.9$



### ■ Exemplul 2

- ❑  $R_1$ : Dacă  $E_{[FC=-0.2]}$  atunci  $F$
- ❑  $FC(E$  este adevărat) = -0.2 → dovadă negativă → nu putem spune nimic despre faptul că  $E$  este adevărat → nu se poate spune nimic despre  $F$



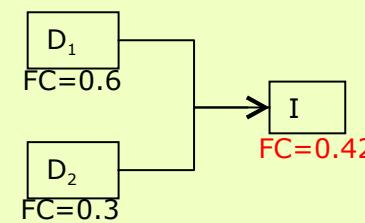
# Intelligent systems – KBS – certainty factors

## Conținut și arhitectură

- FC – mod de calcul pentru combinarea încrederei
  - Mai multe dovezi incerte care susțin sigur o singură ipoteză
    - Dovezi (probe) adunate incremental
    - Mai multe reguli care, pe baza unor dovezi diferite, furnizează aceeași concluzie
    - Aceeași ipoteză (valoare de atribut) I este obținută pe două căi de deducție distincte, cu două perechi diferite de valori pentru  $FC$ ,  $FC[I, D_1]$  și  $FC[I, D_2]$
    - Cele două cai de deducție distincte, corespunzătoare dovezilor (probelor)  $D_1$  și  $D_2$  pot fi:
      - ramuri diferite ale arborelui de căutare generat prin aplicarea regulilor
      - dovezi (probe) indicate explicit sistemului

$$FC(I, D_1 \wedge D_2) = \begin{cases} CF(D_1) + CF(D_2)(1 - CF(D_1)), & \text{dacă } CF(D_1), CF(D_2) > 0 \\ CF(D_1) + CF(D_2)(1 + CF(D_1)), & \text{dacă } CF(D_1), CF(D_2) < 0 \\ \frac{CF(D_1) + CF(D_2)}{1 - \min\{|CF(D_1)|, |CF(D_2)|\}}, & \text{dacă } sign(CF(D_1)) \neq sign(CF(D_2)) \end{cases}$$

- Exemplu
  - $R_1$ : Dacă  $D_1$  [FC=0.6] atunci I
  - $R_2$ : Dacă  $D_2$  [FC=-0.3] atunci I
  - $FC(I, D_1 \wedge D_2) = (0.6 + (-0.3)) / (1 - 0.3) = 0.42$



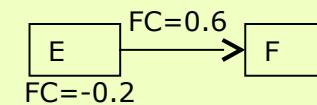
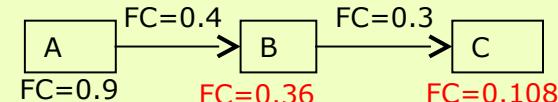
# Intelligent systems – KBS – certainty factors

## Conținut și arhitectură

- ❑ FC – mod de calcul pentru combinarea încrederei
  - ❑ O dovedă incertă care susține incert o ipoteză

$$FC(I) = \begin{cases} FC(D) * FC(\text{regulă}), & \text{dacă } FC(D) > 0 \\ 0, & \text{altfel} \end{cases}$$

- ❑ **Exemplul 1**
  - ❑  $R_1: \text{Dacă } A_{[FC=0.9]} \text{ atunci } B [FC=0.4]$
  - ❑  $R_2: \text{Dacă } B \text{ atunci } C [FC=0.3]$
  - ❑  $FC(B) = FC(A) * FC(R_1) = 0.9 * 0.4 = 0.36$
  - ❑  $FC(C) = FC(B) * FC(R_2) = 0.36 * 0.3 = 0.108$
- **Exemplul 2**
  - ❑  $R_1: \text{Dacă } E_{[FC=-0.2]} \text{ atunci } F [FC=0.6]$
  - ❑  $FC(E \text{ este adevărat}) = -0.2 \rightarrow \text{dovedă negativă} \rightarrow \text{nu putem spune nimic despre faptul că } E \text{ este adevărat} \rightarrow \text{nu se poate spune nimic despre } F$



# Intelligent systems – KBS – certainty factors

## Conținut și arhitectură

- ❑ FC – mod de calcul pentru combinarea încrederii
  - ❑ Mai multe dovezi incerte care susțin incert o ipoteză
    - Dovezile sunt legate prin ŞI logic
  - ❑ Una sau mai multe dintre dovezile incerte care susțin incert o ipoteză
    - Dovezile sunt legate prin SAU logic

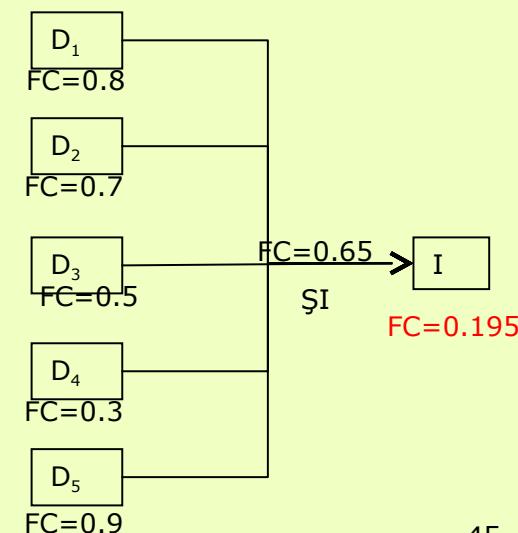
$$CF(I) = \begin{cases} \max\{CF(D_1), CF(D_2), \dots, CF(D_n)\} * CF(\text{regulă}), & \text{dacă } CF(D_i) > 0, i = 1, 2, \dots, n \\ 0, & \text{altfel} \end{cases}$$

### ❑ Exemplul 1

- $R_1: Dacă D_{1[FC = 0.8]} \text{ și } D_{2[FC = 0.7]} \text{ și } D_{3[FC = 0.5]} \text{ și}$

$D_{4[FC = 0.3]} \text{ și } D_{5[FC = 0.9]} \text{ atunci } I [FC = 0.65]$

- $FC(I) = 0.3 * 0.65 = 0.195$



# Intelligent systems – KBS – certainty factors

## Conținut și arhitectură

- ❑ FC – mod de calcul pentru combinarea încrederii
  - ❑ Mai multe dovezi incerte care susțin incert o ipoteză
    - Dovezile sunt legate prin SI logic

$$CF(I) = \begin{cases} \min\{CF(D_1), CF(D_2), \dots, CF(D_n)\} * CF(\text{regulă}), & \text{dacă } CF(D_i) > 0, i = 1, 2, \dots, n \\ 0, & \text{altfel} \end{cases}$$

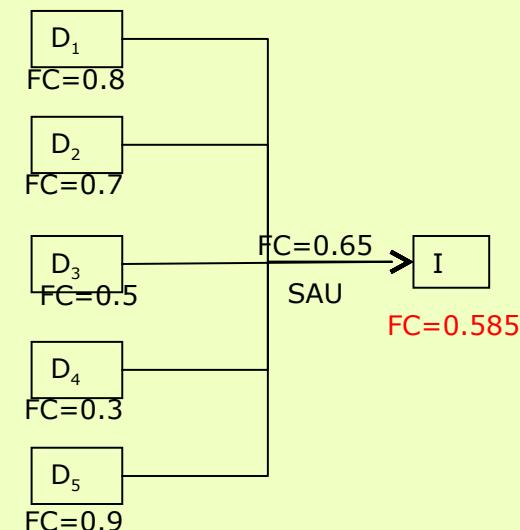
- ❑ Una sau mai multe dintre dovezile incerte susțin incert o ipoteză
  - Dovezile sunt legate prin SAU logic

$$CF(I) = \begin{cases} \max\{CF(D_1), CF(D_2), \dots, CF(D_n)\} * CF(\text{regulă}), & \text{dacă } CF(D_i) > 0, i = 1, 2, \dots, n \\ 0, & \text{altfel} \end{cases}$$

### ❑ Exemplul 2

- $R_1: Dacă D_{1[FC = 0.8]} \text{ sau } D_{2[FC = 0.7]} \text{ sau } D_{3[FC = 0.5]} \text{ sau } D_{4[FC = 0.3]} \text{ sau } D_{5[FC = 0.9]}$   
atunci  $I [FC = 0.65]$

- $FC(I) = 0.9 * 0.65 = 0.585$



# Intelligent systems – KBS – certainty factors

## Exemplu

- Sistem expert pentru diagnosticarea unei răceli

- Fapte în baza de date:

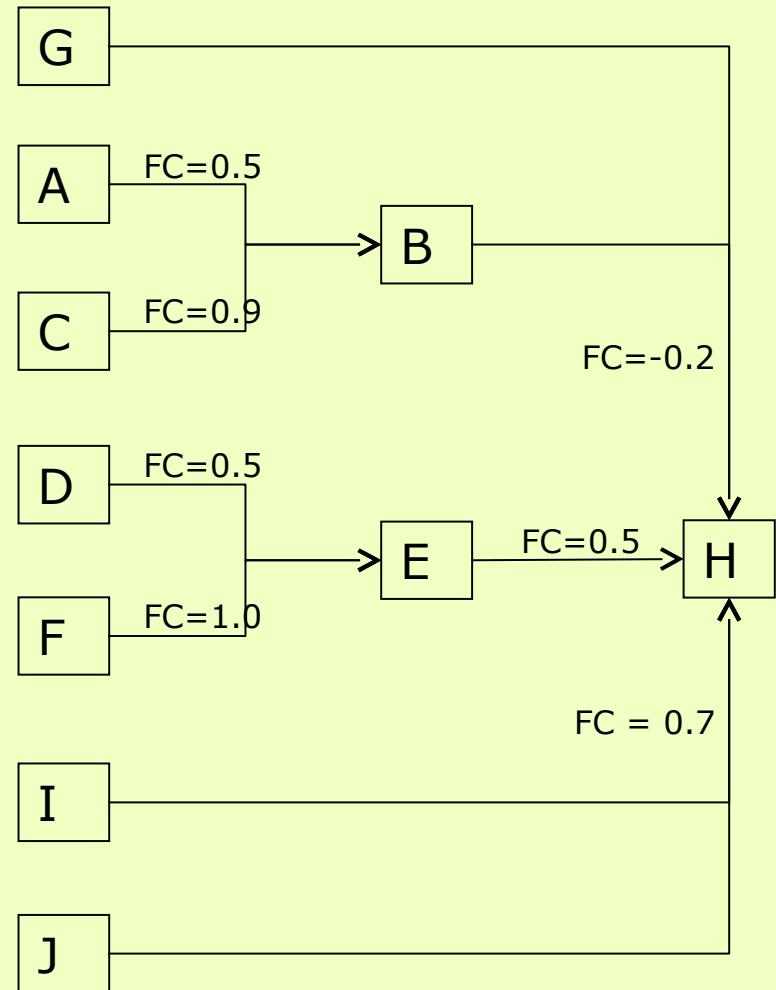
- Febra pacientului 37.4
    - Pacientul tușește de mai puțin de 24 ore
    - Pacientul nu are expectorații
    - Pacientul are o durere de cap cu FC = 0.4
    - Pacientul are nasul înfundat cu FC = 0.5

- Reguli:

- R<sub>1</sub>: Dacă A: febra < 37.5 atunci  
B: simptomele de răceală sunt prezente [FC=0.5]
    - R<sub>2</sub>: Dacă C: febra > 37.5 atunci  
B: simptomele de răceală sunt prezente [FC=0.9]
    - R<sub>3</sub>: Dacă D: tușește > 24 ore atunci  
E: durerea de gât e prezentă [FC=0.5]
    - R<sub>4</sub>: Dacă F: tușește > 48 ore atunci  
E: durerea de gât e prezentă [FC=1.0]
    - R<sub>5</sub>: Dacă B: are simptome de răceală și  
G: nu expectorează atunci H: a răcit [FC=-0.2]
    - R<sub>6</sub>: Dacă E: îl doare gâtul atunci  
H: a răcit [FC=0.5]
    - R<sub>7</sub>: Dacă I: îl doare capul și  
J: are nasul înfundat atunci H: a răcit [FC=0.7]

- Concluzia:

- Pacientul este sau nu răcit?



# Intelligent systems – KBS – certainty factors

## Exemplu

- Sistem expert pentru diagnosticarea unei răceli

- Fapte în baza de date:

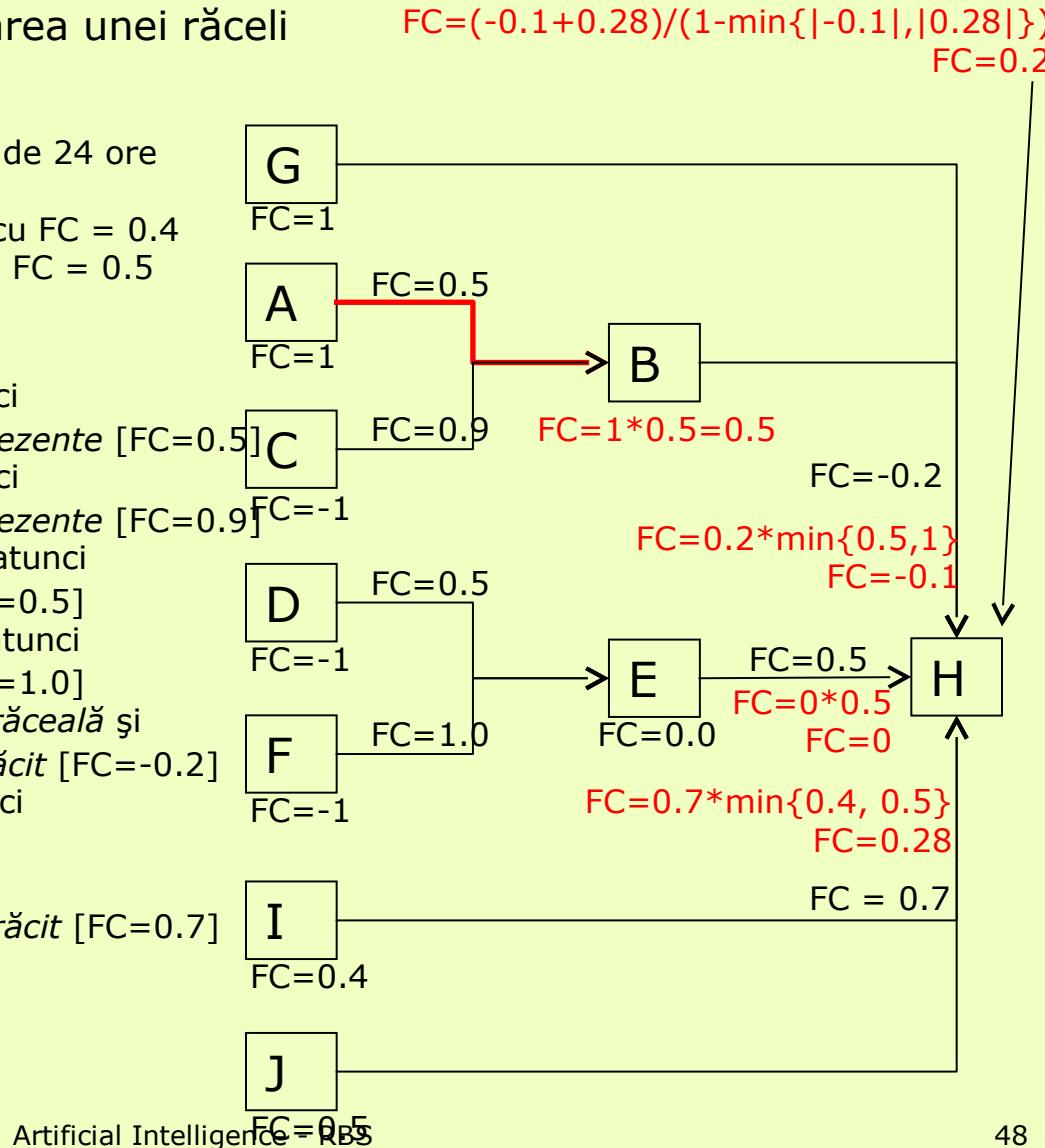
- Febra pacientului 37.4
    - Pacientul tușește de mai puțin de 24 ore
    - Pacientul nu are expectorații
    - Pacientul are o durere de cap cu FC = 0.4
    - Pacientul are nasul înfundat cu FC = 0.5

- Reguli:

- R<sub>1</sub>: Dacă A: febra < 37.5 atunci B: simptomele de răceală sunt prezente [FC=0.5]
    - R<sub>2</sub>: Dacă C: febra > 37.5 atunci B: simptomele de răceală sunt prezente [FC=0.9]
    - R<sub>3</sub>: Dacă D: tușește > 24 ore atunci E: durerea de gât e prezentă [FC=0.5]
    - R<sub>4</sub>: Dacă F: tușește > 48 ore atunci E: durerea de gât e prezentă [FC=1.0]
    - R<sub>5</sub>: Dacă B: are simptome de răceală și G: nu expectorează atunci H: a răcit [FC=-0.2]
    - R<sub>6</sub>: Dacă E: îl doare gâtul atunci H: a răcit [FC=0.5]
    - R<sub>7</sub>: Dacă I: îl doare capul și J: are nasul înfundat atunci H: a răcit [FC=0.7]

- Concluzia:

- Pacientul este sau nu răcit?



# Intelligent systems – KBS – certainty factors

---

- Avantaje
  - Nu este necesar calculul apriori a probabilităților
- Limite
  - ipotezele sustinute de probe sunt independente.
  - exemplu:
    - Fie următoarele fapte:
      - A: Aspersorul a funcționat noaptea trecută
      - U: Iarba este udă dimineață
      - P: Noaptea trecută a plouat.
    - și următoarele două reguli care leagă între ele aceste fapte:
      - $R_1$ : dacă aspersorul a funcționat noaptea trecută atunci există o încredere puternică (0.9) că iarba este udă dimineață
      - $R_2$ : dacă iarba este udă dimineață atunci există o încredere puternică (0.8) că noaptea trecută a plouat
    - Deci:
      - $FC[U,A] = 0.9$  - deci proba aspersor sustine iarba uda cu 0.9
      - $FC[P,U] = 0.8$  - deci iarba uda sustine ploaie cu 0.8
      - $FC[P,A] = 0.8 * 0.9 = 0.72$  - deci aspersorul sustine ploaia cu 0.72

# Intelligent systems – KBS – certainty factors

---

## □ SBR de tip Bayes vs. SBR cu FC

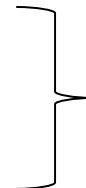
Bayes	FC
Teorie probabilităților este veche și fundamentată matematic	Teoria FC este nouă și fără demonstrații matematice
Necesită existența unor informații statistice	Nu necesită existența unor date statistice
Propagarea încrederei crește în timp exponențial	Informația circulă repede și eficient în SBR

# Intelligent systems - KBS

---

## □ Reasoning techniques for uncertainty

- Theory of Bayes – probabilistic method
- Theory of certainty
- Theory of possibility (fuzzy logic)



Heuristic  
methods

# Intelligent systems – KBS – Fuzzy systems

---

- Theory of possibility
- Content and design
- Typology
- Tools
- Advantages and limits

# Intelligent systems – KBS – Fuzzy systems

---

## Teoria posibilității (logica fuzzy)

- ▣ Why fuzzy?
  - Problem: translate in C++ code the following sentences:
    - ▣ Georgel is tall.
    - ▣ It is cold outside.
  
- ▣ When fuzzy is important?
  - Natural queries
  - Knowledge representation for a KBS
  - Fuzzy control – then we deal by imprecise phenomena  
(noisy phenomena)

# Intelligent systems – KBS – Fuzzy systems

---

## Remember the components of a KBS

- Knowledge base → knowledge representation
  - Formal logic (formal languages)
    - Definition
      - Science of formal principles for rationing
    - Components
      - Syntax – atomic symbols used by language and the constructing rules of the language
      - Semantic – associates a meaning to each symbol and a truth value (true or false) to each rule
      - Syntactic inference – rules for identifying a subset of logic expressions → theorems (for generating new expressions)
    - Typology
      - True value
        - Dual logic
        - Polyvalent logic
      - Basic elements
        - Classic → primitives = sentences (predicates)
        - Probabilistic → primitives = random variables
      - Working manner
        - Propositional logic → declarative propositions and fix or unique objects (Ionica is student)
        - First-order logic → declarative propositions, predicates and quantified variables, unique objects or variables associated to a unique object
  - Rules
  - Semantic nets
- Inference engine

# Intelligent systems – KBS – Fuzzy systems

---

## Theory of possibility – a little bit of history

- Parminedes (400 B.C.)
- Aristotle
  - "Law of the Excluded Middle" – every sentence must be True or False
- Plato
  - A third region, between True and False
  - Forms the basis of fuzzy logic
- Lukasiewicz (1900)
  - Has proposed an alternative and systematic approach related to bi-valent logic of Aristotle – trivalent logic: true, false or possible
- Lotfi A. Zadeh (1965)
  - Mathematical description of fuzzy set theory and fuzzy logic: truth functions takes values in  $[0,1]$  (instead of {True, False})
    - He has proposed new operations in fuzzy logic
    - He has considered the fuzzy logic as a generalisation of the classic logic
  - He has written the first paper about fuzzy sets

# Intelligent systems – KBS – Fuzzy systems

---

## Theory of possibility

- ▣ Fuzzy logic
  - Generalisation of Boolean logic
  - Deals by the concept of partial truth
    - Classical logic – all things are expressed by binary elements
      - 0 or 1, white or black, yes or no
    - Fuzzy logic – gradual expression of a truth
      - Values between 0 and 1
  
- ▣ Logic vs. algebra
  - Logical operators are expressed by using mathematical terms (George Boole)
    - Conjunction = minimum  $\rightarrow a \wedge b = \min(a, b)$
    - Disjunction = maximum  $\rightarrow a \vee b = \max(a, b)$
    - Negation = difference  $\rightarrow \neg a = 1 - a$

# Intelligent systems – KBS – Fuzzy systems

---

## Remember: KBS - design

- Knowledge base
  - Content
    - Specific information
      - Facts – correct affirmations
      - Rules – special heuristics that generate knowledge
  - Aim
    - Store all the information (facts, rules, solving methods, heuristics) about a given domain (taken from some experts)
- Inference engine
  - Content
    - Rules for generating new information
    - Domain-independent algorithms
    - Brain of a KBS
  - Aim
    - Help to explore the KB by reasoning for obtaining solutions, recommendations or conclusions

# Intelligent systems – KBS – Fuzzy systems

---

## Content and design

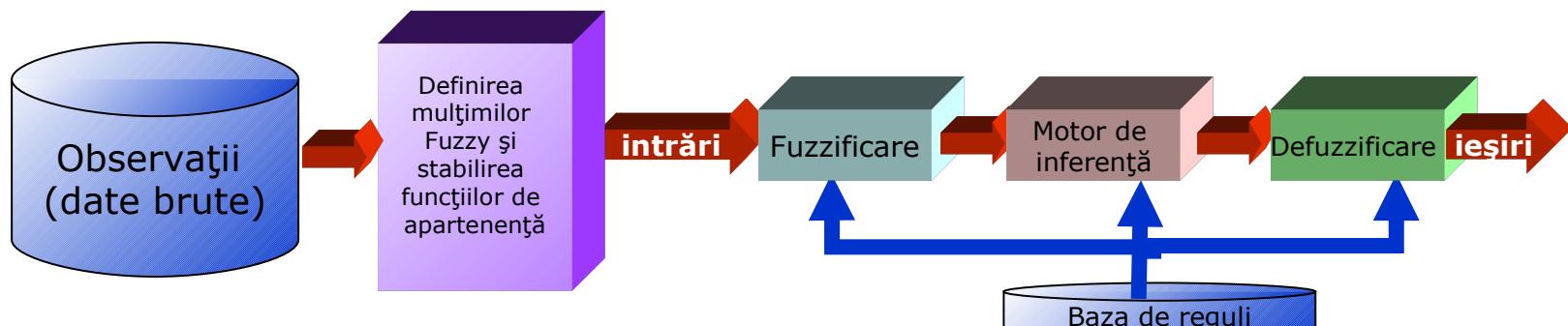
- ❑ Main idea
  - Cf. to certainty theory:
    - *Popescu is tall*
  - Cf. to uncertainty theory
    - Cf. to probability theory
      - *There is 80% chance that Popescu is young*
    - Cf. fuzzy logic
  - Cf. teoriei informațiilor certe
    - *Popescu este Tânăr*
  - Cf. teoriei informațiilor incerte
    - Cf. teoriei probabilităților:
      - *Există 80% șanse ca Popescu să fie Tânăr*
    - Cf. logicii fuzzy:
      - *Popescu's degree of membership to the group of young people is 0.80*
- ❑ Necessity
  - Real phenomena involve fuzzy sets
  - Example
    - *The room's temperature can be:*
      - *low,*
      - *Medium or*
      - *high*
    - These sets of possible temperatures can overlap
      - A temperature can belong to more classes (groups) depends on the person that evaluates that temperature

# Intelligent systems – KBS – Fuzzy systems

## Content and design

### □ Steps for constructing a fuzzy system

- Define the inputs and the outputs – by an expert
  - Raw inputs and outputs
  - Fuzzification of inputs and outputs
    - Fix the fuzzy variables and fuzzy sets based on membership functions
- Construct a base of rules – by an expert
  - Decision matrix
- Evaluate the rules
  - Inference – transform the fuzzy inputs into fuzzy outputs by applying all the rules
- Aggregate the results
- Defuzzificate the result
- Interpret the result



# Intelligent systems – KBS – Fuzzy systems

---

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic)
  - Fuzzy facts (fuzzy sets)
    - Definition
    - Representation
    - Operations – complements, containment, intersection, reunion, equality, algebraic product, algebraic sum
    - Properties – associativity, commutativity, distributivity, transitivity, idempotency, identity, involution
    - Hedges
  - Fuzzy variables
    - Definition
    - Properties
- Establish the fuzzy variables and the fuzzy sets based on membership functions

# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **definition**
  - Set definition – 2 possibilities:
    - By enumeration of elements
      - Ex. Set of students = {Ana, Maria, Ioana}
    - By specifying a property of elements
      - Ex. Set of even numbers = {x | x = 2n, where n = 2k}
  - Characteristic function  $\mu$  for a set
    - Let X a universal set and x an element of this set ( $x \in X$ )
    - Classical logic
      - Let R a sub-set of X:  $R \subset X$ , R – regular set
      - Every element x belongs to set R
    - $\mu_R : X \rightarrow \{0, 1\}$ , where  $\mu_R(x) = \begin{cases} 1, & x \in R \\ 0, & x \notin R \end{cases}$
  - Fuzzy logic
    - Let F a sub-set of X (a univers) :  $F \subset X$ , F – fuzzy set
    - Every element x belongs to F by a given degree of membership  $\mu_F(x)$
    - $\mu_F : X \rightarrow [0, 1]$ ,  $\mu_F(x) = g$ , where  $g \in [0, 1]$  – membership degree of x to F
    - $g = 0 \rightarrow$  not-belong
    - $g = 1 \rightarrow$  belong
    - A fuzzy set = a pair  $(F, \mu_F)$ , where  $\mu_F(x) = \begin{cases} 1, & \text{if } x \text{ is totally in } F \\ 0, & \text{if } x \text{ is not in } F \\ \in (0,1) & \text{if } x \text{ is part of } F (x \text{ is a fuzzy number}) \end{cases}$

# Intelligent systems – KBS – Fuzzy systems

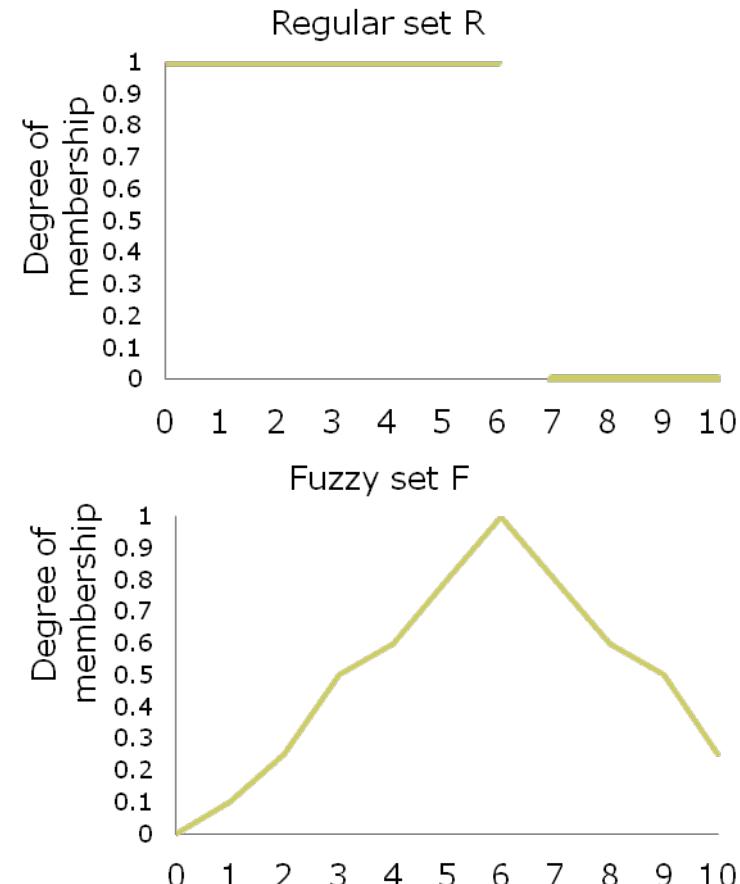
Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **definition**

- Example 1

- X – set of natural numbers < 11
    - R – set of natural numbers < 7
    - F – set of natural numbers that are neighbours of 6

x	$\mu_R(x)$	$\mu_F(x)$
0	1	0
1	1	0.1
2	1	0.25
3	1	0.5
4	1	0.6
5	1	0.8
6	1	1
7	0	0.8
8	0	0.6
9	0	0.5
10	0	0.25



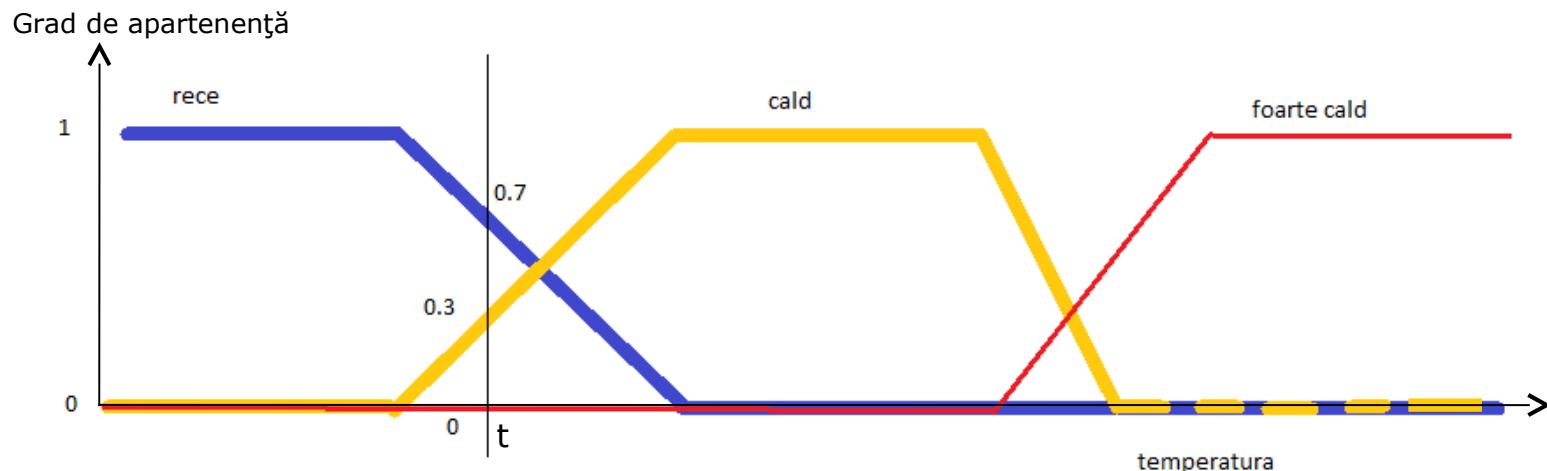
# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **definition**

- Example 2

- A temperature  $t$  can have 3 truth values:
      - Red (0): is not hot
      - Orange (0.3): warm
      - Blue (0.7): cold



# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **representation**

- Regular sets

- Exact limits → Venn diagrams

- Fuzzy sets

- Gradual limits → representations based on membership functions

- Singular

- $\mu(x) = s$ , where  $s$  is a scalar

- Triangular

- $$\mu(x) = \max \left\{ 0, \min \left\{ \frac{x-a}{b-a}, 1, \frac{c-x}{c-b} \right\} \right\}$$

- Trapezoidal

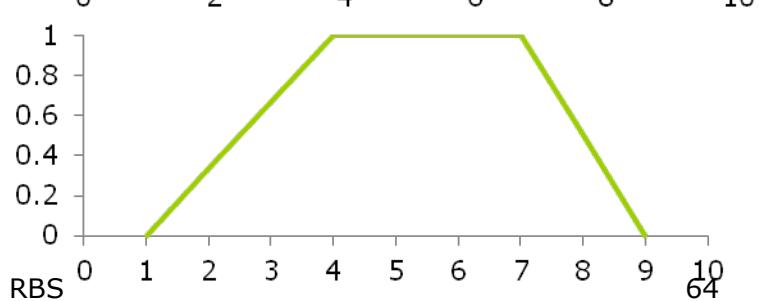
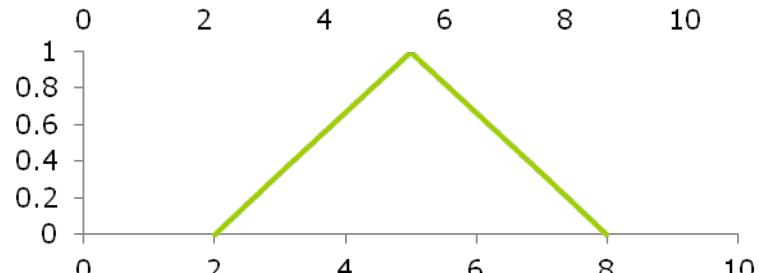
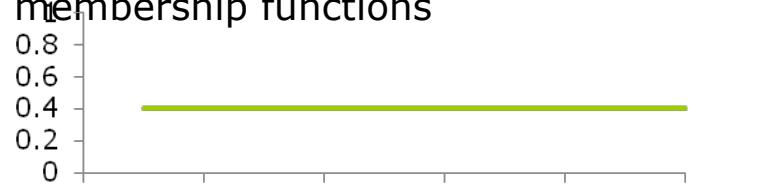
- $$\mu(x) = S(x) = \max \left\{ 0, \min \left\{ \frac{x-a}{b-a}, 1, \frac{d-x}{d-c} \right\} \right\}$$

- Z function

- $$\mu(x) = 1 - S(x)$$

- $\Pi$  function

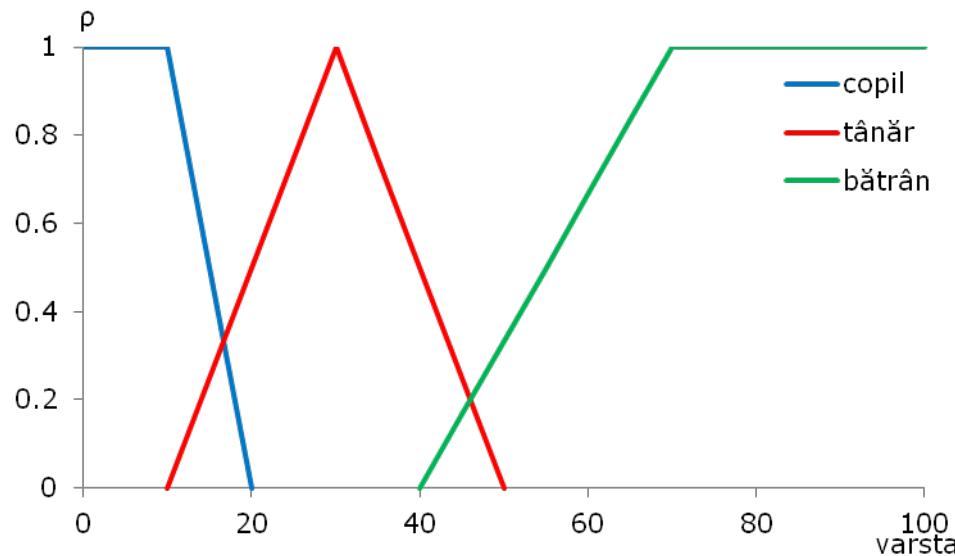
- $$\mu(x) = \Pi(x) = \begin{cases} S(x), & \text{if } x \leq c \\ Z(x), & \text{if } x > c \end{cases}$$



# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **representation**
  - Example
    - *Age of a person*



# Intelligent systems – KBS – Fuzzy systems

---

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **operations**
  - complement
  - Containment
  - Intersection
  - Union
  - Equality
  - Algebraic product
  - Algebraic sum

# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **operations**

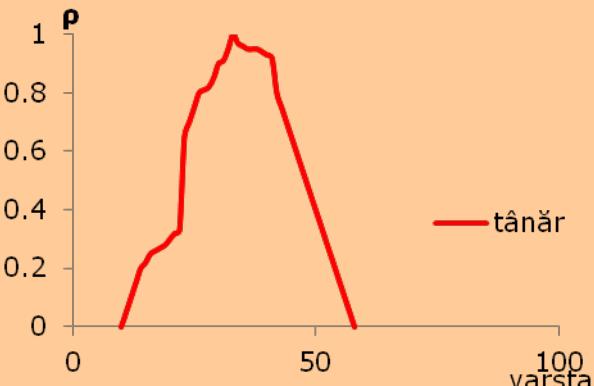
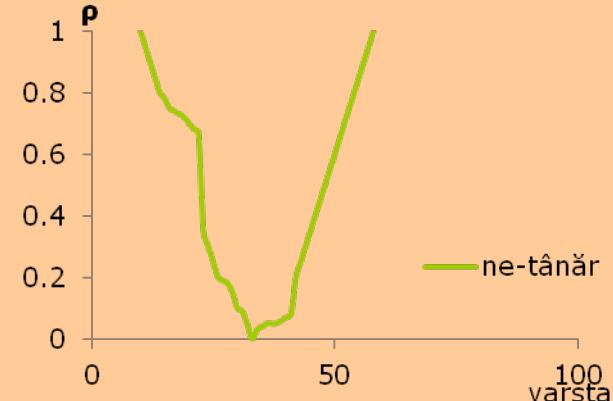
- Complement

- $X$  – a universe
    - $A$  – a fuzzy set (with universe  $X$ )
    - $B$  – a fuzzy set (with universe  $X$ )

- $B$  is complement of  $A$  ( $B = \neg A$ ) if:
      - $\mu_B(x) = \mu_{\neg A}(x) = 1 - \mu_A(x)$  for all  $x \in X$

- Example:

- *Old persons (based on their age)*
      - $A = \{(30, 0), (40, 0.2), (50, 0.4), (60, 0.6), (70, 0.8), (80, 1)\}$
    - *Young persons (that are not old) (based on their age)*
      - $\neg A = \{(30, 1), (40, 0.8), (50, 0.6), (60, 0.4), (70, 0.2), (80, 0)\}$



# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

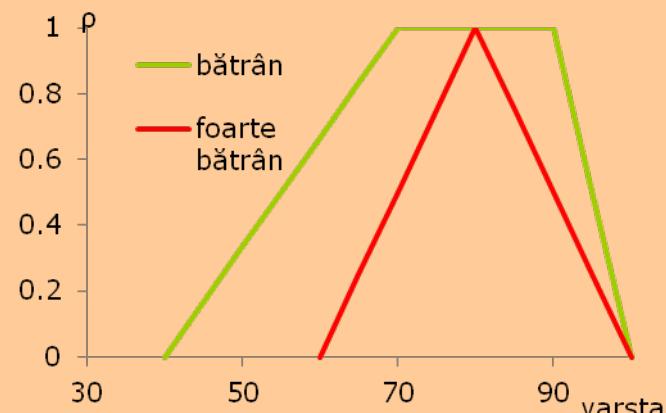
- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **operations**

- Containment

- $X$  – a universe
    - $A$  – a fuzzy set (with universe  $X$ )
    - $B$  – a fuzzy set (with universe  $X$ )

- $B$  is a subset of  $A$  ( $B \subset A$ ) if:
      - $\mu_B(x) \leq \mu_A(x)$  for all  $x \in X$

- Example
    - *Old persons (based on their age)*
      - $A = \{(60, 0.6), (65, 0.7), (70, 0.8), (75, 0.9), (80, 1)\}$
    - *Very old persons (based on their age)*
      - $B = \{(60, 0.6), (65, 0.67), (70, 0.8), (75, 0.8), (80, 0.95)\}$



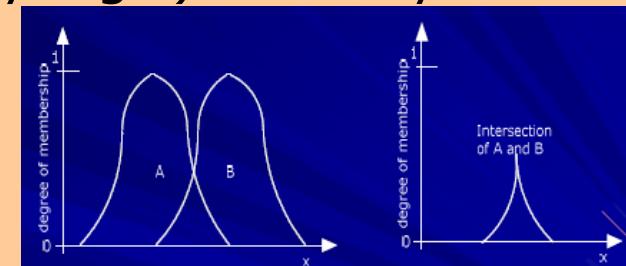
# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **operations**

- intersection

- X – a universe
  - A – a fuzzy set (with universe X)
  - B – a fuzzy set (with universe X)
  - C – a fuzzy set (with universe X)
  - **C** is an intersection of A and B if:
    - $\mu_C(x) = \mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \cap \mu_B(x)$  for all  $x \in X$



- Example

- *Old persons (based on their age)*
      - $A = \{(30, 0) (40, 0.1) (50, 0.2) (60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)\}$
    - *Middle-age persons*
      - $B = \{(30, 0.1) (40, 0.2) (50, 0.6) (60, 0.5), (65, 0.2) (70, 0.1), (75, 0), (80, 0)\}$
    - *Old and middle age persons*
      - $C = \{(30, 0) (40, 0.1) (50, 0.2) (60, 0.5), (65, 0.2) (70, 0.1), (75, 0), (80, 0)\}$

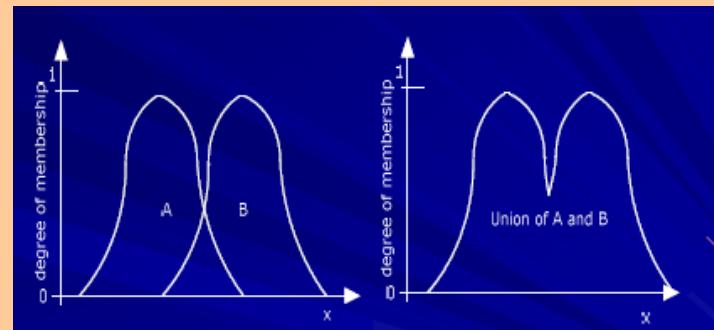
# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **operations**

- union

- X – a universe
    - A – a fuzzy set (with universe X)
    - B – a fuzzy set (with universe X)
    - C – a fuzzy set (with universe X)
    - C is the union of A nad B if:
      - $\mu_C(x)=\mu_{A \cup B}(x)=\max\{\mu_A(x), \mu_B(x)\}=\mu_A(x) \cup \mu_B(x)$  for all  $x \in X$



- Example

- *Old persons (based on their age)*
      - $A=\{(30,0) (40, 0.1) (50, 0.2) (60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)\}$
    - *Middle-age persons*
      - $B=\{(30,0.1) (40, 0.2) (50, 0.6) (60, 0.5), (65, 0.2) (70, 0.1), (75, 0), (80, 0)\}$
    - *Old or middle-age persons*
      - $C=\{(30,0.1) (40, 0.2) (50, 0.6) (60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)\}$

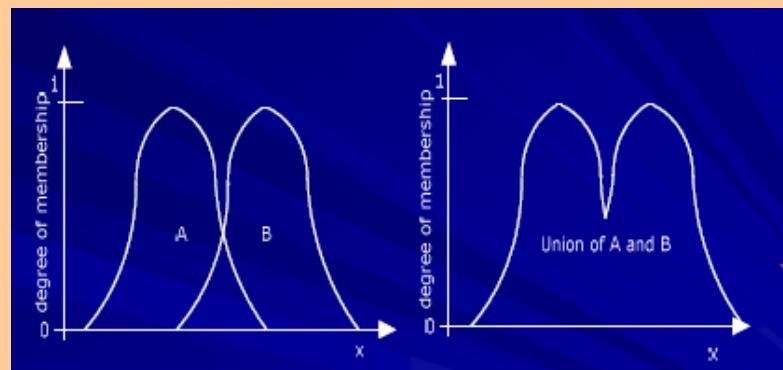
# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **operations**

- Equality, product and algebraic sum

- X – a universe
    - A – a fuzzy set (with universe X)
    - B – a fuzzy set (with universe X)
    - C – a fuzzy set (with universe X)
  - B is equal to A ( $B=A$ ) if:
    - $\mu_B(x)=\mu_A(x)$  for all  $x \in X$
  - C is the product of A and B ( $C=A*B$ ) if:
    - $\mu_C(x)=\mu_A(x)\mu_B(x)=\mu_A(x)*\mu_B(x)$  for all  $x \in X$
  - C is the sum of A and B ( $C=A+B$ ) if:
    - $\mu_C(x)=\mu_A(x)+\mu_B(x)=\mu_A(x)+\mu_B(x)$  for all  $x \in X$



# Intelligent systems – KBS – Fuzzy systems

---

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **properties**
  - Asociativity
  - Commutativity
  - Distributivity
  - Transitivity
  - Idem potency
  - Identity
  - Involution

# Intelligent systems – KBS – Fuzzy systems

---

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy facts (fuzzy sets) → **hedges**

- Main idea

- Modifiers, adjectives or adverbs that change the truth values of sentences
  - Ex. *Very, less, much, more, close*, etc.
- Change the shape of fuzzy sets
- Can act on
  - Fuzzy numbers
  - Truth values
  - Membership functions
- Heuristics

- Utility

- Closer to the natural language → subjectivism
- Evaluation of linguistic variables

# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → **Fuzzy facts** (fuzzy sets) → **hedges**

- Typology

- *Hedges* that reduce the truth value

(produce a concentration)

- *Very*  $\mu_{A\_very}(x) = (\mu_A(x))^2$
      - *Extremely*  $\mu_{A\_extremly}(x) = (\mu_A(x))^3$
      - *Very very*  $\mu_{A\_very\_very}(x) = (\mu_{A\_foarte}(x))^2 = (\mu_A(x))^4$

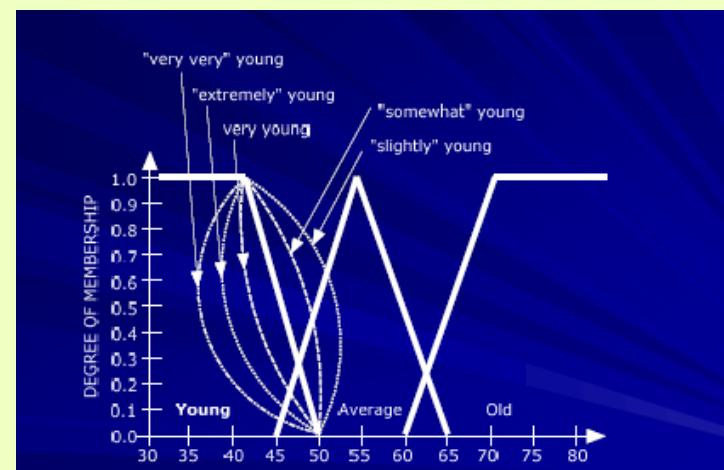
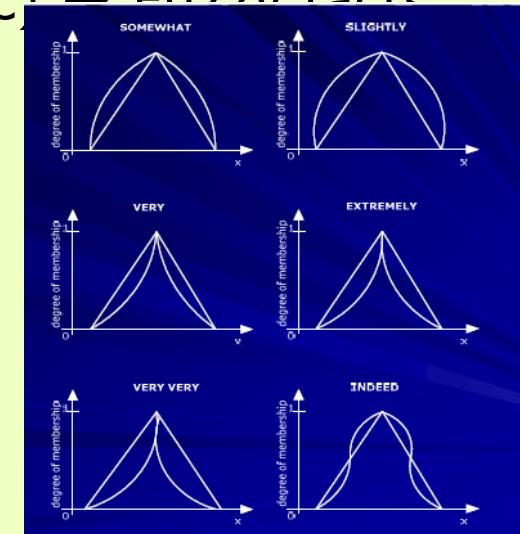
- *Hedges* that increase the truth value

(produce a dilatation)

- *Somewhat*  $\mu_{A\_somewhat}(x) = (\mu_A(x))^{1/2}$
      - *slightly*  $\mu_{A\_slightly}(x) = (\mu_A(x))^{1/3}$

- *Hedges* that intensify the truth value

- *indeed*  
$$\mu_{A\_indeed}(x) = \begin{cases} 2(\mu_A(x))^2, & \text{if } 0 \leq \mu_A(x) \leq 0.5 \\ 1 - 2(1 - \mu_A(x))^2, & \text{if } 0.5 \leq \mu_A(x) \leq 1 \end{cases}$$



# Intelligent systems – KBS – Fuzzy systems

---

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic)
  - Fuzzy facts (fuzzy sets)
    - Definition
    - Representation
    - Operations – complements, containment, intersection, reunion, equality, algebraic product, algebraic sum
    - Properties – associativity, commutativity, distributivity, transitivity, idempotency, identity, involution
    - Hedges
  - **Fuzzy variables**
    - **Definition**
    - **Properties**
  - Establish the fuzzy variables and the fuzzy sets based on membership functions

# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

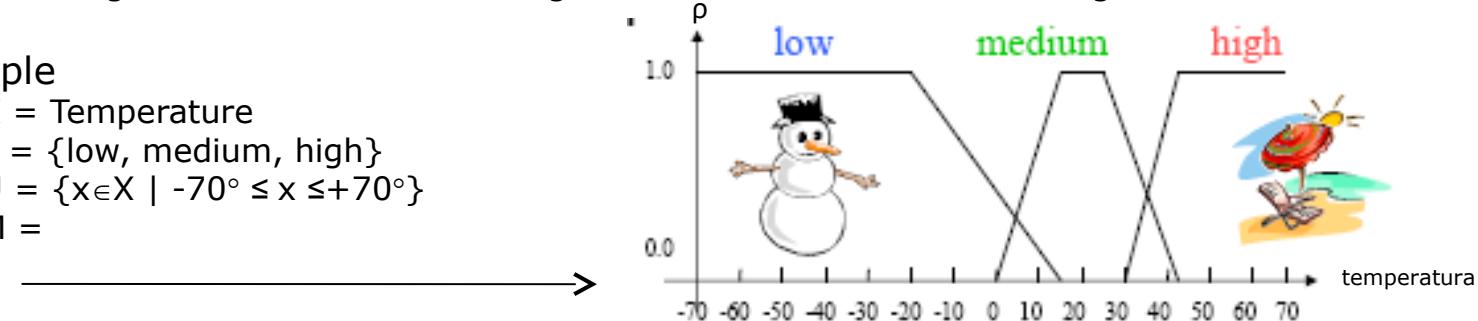
## ❑ Elements from probability theory (fuzzy logic) → Fuzzy variables → **definition**

- A fuzzy variable is defined by  $V = \{x, l, u, m\}$ , where:
  - $x$  – name of symbolic variable
  - $L$  – set of possible labels for variable  $x$
  - $U$  – universe of the variable
  - $M$  – semantic regions that define the meaning of labels from  $L$  (membership functions)

- Membership functions
  - Subjective assessment
    - The shape of functions is defined by experts
  - Ad-hoc assessment
    - Simple functions that can solve the problem
  - Assessment based on distributions and probabilities of information extracted from measurements
  - Adapted assessment
    - By testing
  - Automated assessment
    - Algorithms utilised for defining functions based on some training data

## ■ Example

- $X$  = Temperature
- $L = \{\text{low, medium, high}\}$
- $U = \{x \in X \mid -70^\circ \leq x \leq +70^\circ\}$
- $M =$



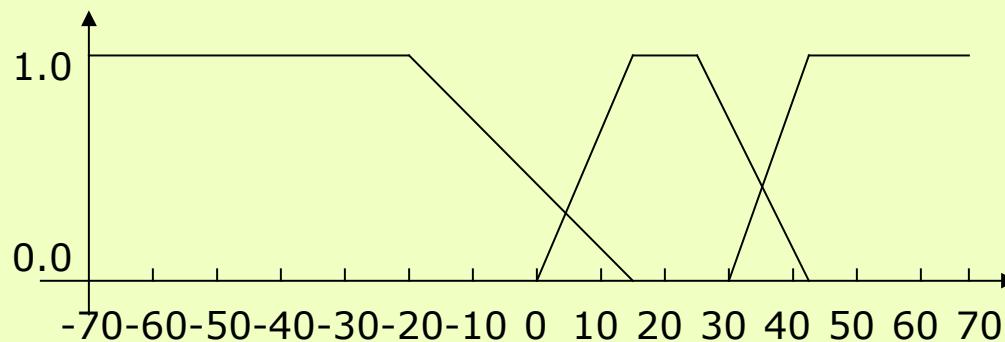
# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

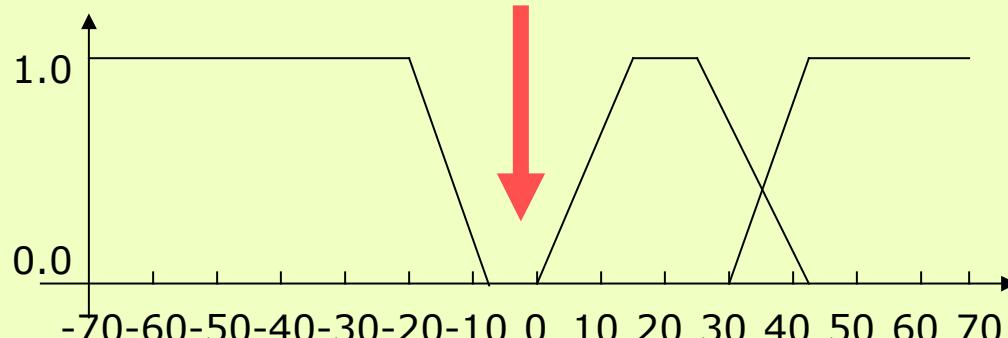
- Elements from probability theory (fuzzy logic) → Fuzzy variables → **properties**

- Completeness

- A fuzzy variable  $V$  is complete if for all  $x \in X$  there is a fuzzy set  $A$  such as  $\mu_A(x) > 0$



Complete



Incomplete

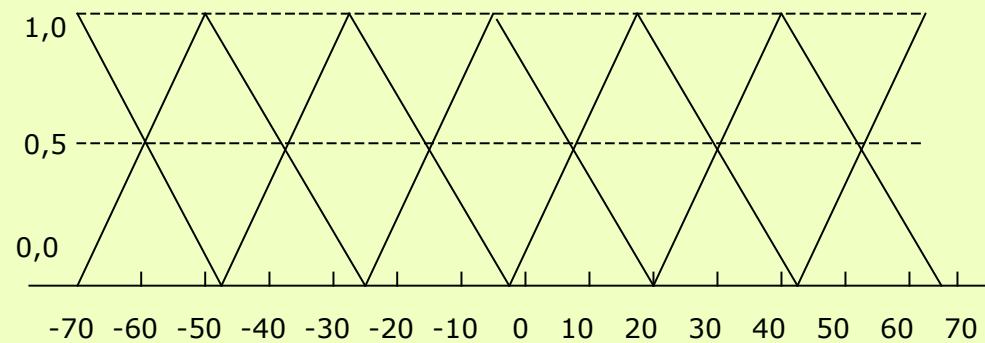
# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

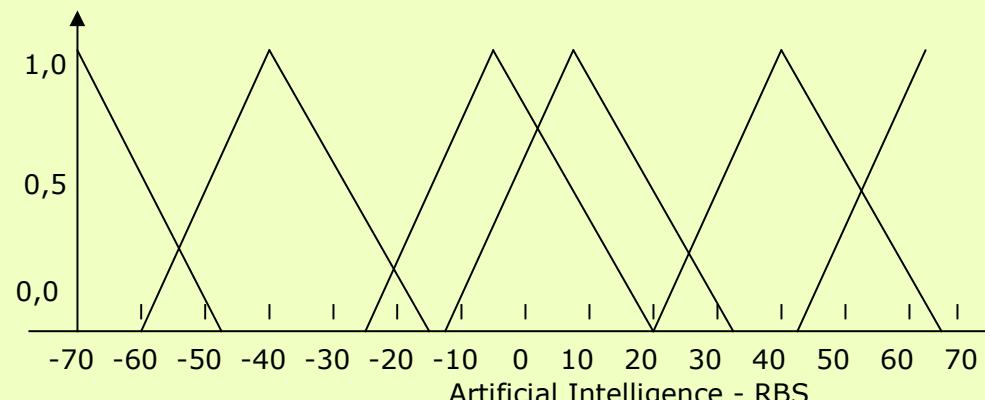
- Elements from probability theory (fuzzy logic) → Fuzzy variables → **properties**

- Unit partition

- A fuzzy variable V forms a unit partition if for all input values x we have  $\sum_{i=1}^p \mu_{A_i}(x) = 1$
    - where p is the number of sets that x belongs to
    - There are no rules for defining 2 neighbour sets
      - Usually, the overlap is between 25% și 50%



Unit partition



Non-unit partition

# Intelligent systems – KBS – Fuzzy systems

---

Content and design → fuzzification of input data

- Elements from probability theory (fuzzy logic) → Fuzzy variables → **properties**
  - Unit partition
    - A complete fuzzy variable can be transformed into a unit partition:

$$\mu_{\hat{A}_i}(x) = \frac{\mu_{A_i}(x)}{\sum_{j=1}^p \mu_{A_j}(x)} \text{ for } i = 1, \dots, p$$

# Intelligent systems – KBS – Fuzzy systems

---

Content and design → fuzzification of input data

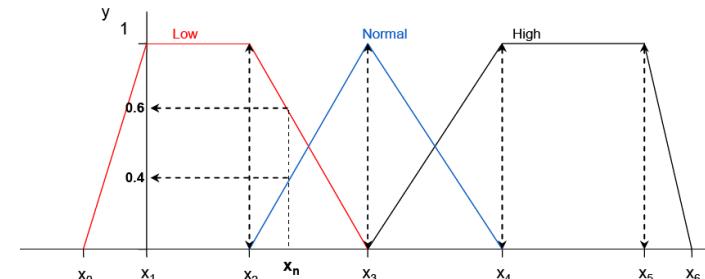
- Elements from probability theory (fuzzy logic)
  - Fuzzy facts (fuzzy sets)
    - Definition
    - Representation
    - Operations – complements, containment, intersection, reunion, equality, algebraic product, algebraic sum
    - Properties – associativity, commutativity, distributivity, transitivity, idempotency, identity, involution
    - Hedges
  - Fuzzy variables
    - Definition
    - Properties
- **Establish the fuzzy variables and the fuzzy sets based on membership functions**

# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

## □ Mechanism

- Establish the raw (input and out[put] data of the system)
- Define membership functions for each input data
  - Each membership function has associated a quality label – linguistic variable
  - A raw variable can have associated one or more linguistic variables
  - Example
    - Raw variable: temperature T
    - Linguistic variable: low → A<sub>1</sub>, medium → A<sub>2</sub>, high → A<sub>3</sub>
- Transform each raw input data into a linguistic data → fuzzification
  - Establish the fuzzy set of that raw input data
  - How?
    - For a given raw input determine the membership degree for each possible set
  - Example
    - $T (=x_n) = 5^\circ$
    - $A_1 \rightarrow \mu_{A_1}(T) = 0.6$
    - $A_2 \rightarrow \mu_{A_2}(T) = 0.4$

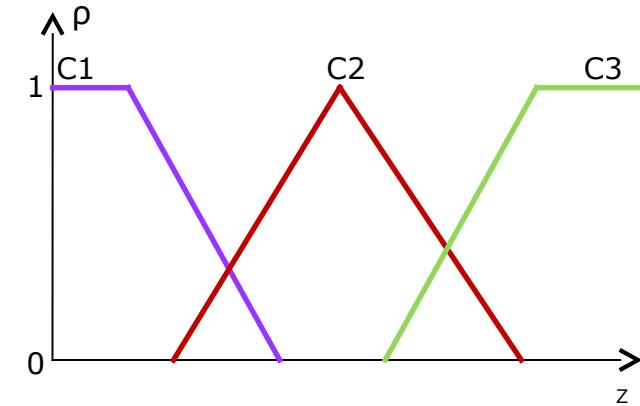
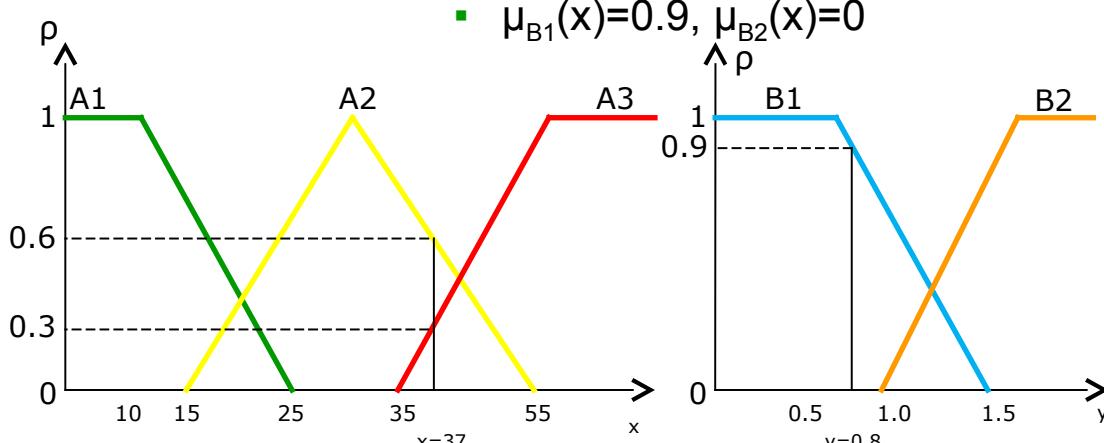


# Intelligent systems – KBS – Fuzzy systems

Content and design → fuzzification of input data

## ❑ Mechanism

- Example - air conditioner device
  - ❑ Inputs :
    - $x$  (temperature – cold, normal, hot) and
    - $y$  (humidity – small, large)
  - ❑ Outputs:
    - $z$  (machine power – low, medium, high)
  - ❑ Input data:
    - Temperature  $x = 37$ 
      - $\mu_{A1}(x)=0, \mu_{A2}(x)=0.6, \mu_{A3}(x)=0.3$
    - Humidity  $y = 0.8$ 
      - $\mu_{B1}(y)=0.9, \mu_{B2}(y)=0$

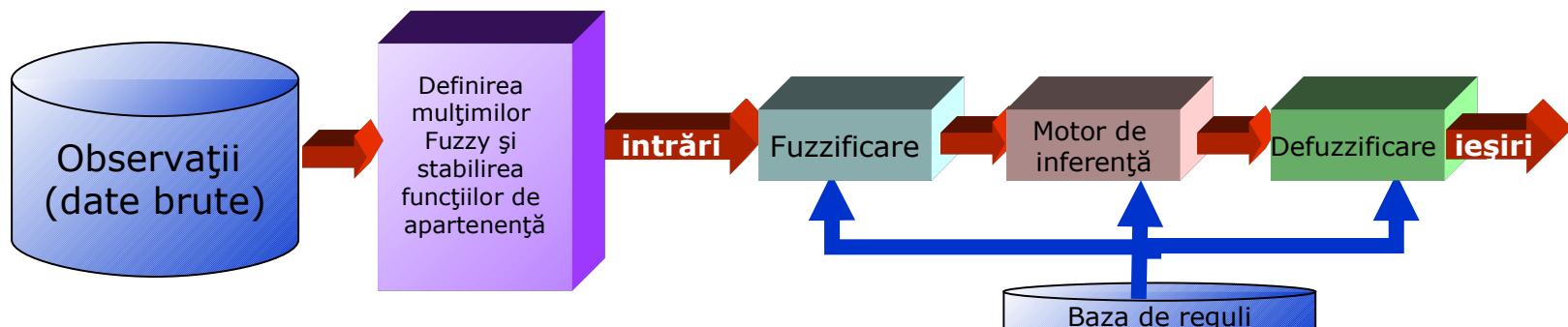


# Intelligent systems – KBS – Fuzzy systems

## Content and design

### □ Steps for constructing a fuzzy system

- Define the inputs and the outputs – by an expert
  - Raw inputs and outputs
  - Fuzzification of inputs and outputs
    - Fix the fuzzy variables and fuzzy sets based on membership functions
- **Construct a base of rules – by an expert**
  - **Decision matrix**
- Evaluate the rules
  - Inference – transform the fuzzy inputs into fuzzy outputs by applying all the rules
- Aggregate the results
- Defuzzificate the result
- Interpret the result



# Intelligent systems – KBS – Fuzzy systems

---

Content and design → Construct a base of rules – by an expert

## □ Rules

- Definition
  - Linguistic constructions
    - Affirmative sentences: A
    - Conditional sentences: if A then B
  - Where A and B are (collections of) sentences that contain linguistic variables
    - A – premise of the rule
    - B – consequence of the rule
- Typology
  - Non-conditional
    - x is (in)  $A_i$
    - Eg. *Save the energy*
  - Conditional
    - If x is (in)  $A_i$  then z is (in)  $C_k$
    - If x is (in)  $A_i$  and y is (in)  $B_j$ , then z is (in)  $C_k$
    - If x is (in)  $A_i$  or y is (in)  $B_j$ , then z is (in)  $C_k$

# Intelligent systems – KBS – Fuzzy systems

---

Content and design → Construct a base of rules – by an expert

- Rules
- Example

	Rules of classical logic	Rules of fuzzy logic
R <sub>1</sub>	<i>If temperature is -5, then is cold</i>	<i>If temperature is law, then is cold</i>
R <sub>2</sub>	<i>If temperature is 15, then is warm</i>	<i>If temperature is medium, then is warm</i>
R <sub>3</sub>	<i>If temperature is 35, then is hot</i>	<i>If temperature is high, then is hot</i>

# Intelligent systems – KBS – Fuzzy systems

---

Content and design → Construct a base of rules – by an expert

- Rules
- Database of fuzzy rules

- $R_{11}$ : if  $x$  is  $A_1$  and  $y$  is  $B_1$  then  $z$  is  $C_u$
- $R_{12}$ : if  $x$  is  $A_1$  and  $y$  is  $B_2$  then  $z$  is  $C_v$
- ...
- $R_{1n}$ : if  $x$  is  $A_1$  and  $y$  is  $B_n$  then  $z$  is  $C_x$
  
- $R_{21}$ : if  $x$  is  $A_2$  and  $y$  is  $B_1$  then  $z$  is  $C_x$
- $R_{22}$ : if  $x$  is  $A_2$  and  $y$  is  $B_2$  then  $z$  is  $C_z$
- ...
- $R_{2n}$ : if  $x$  is  $A_2$  and  $y$  is  $B_n$  then  $z$  is  $C_v$
  
- ...
  
- $R_{m1}$ : if  $x$  is  $A_m$  and  $y$  is  $B_1$  then  $z$  is  $C_x$
- $R_{m2}$ : if  $x$  is  $A_m$  and  $y$  is  $B_2$  then  $z$  is  $C_v$
- ...
- $R_{mn}$ : if  $x$  is  $A_m$  and  $y$  is  $B_n$  then  $z$  is  $C_u$

# Intelligent systems – KBS – Fuzzy systems

---

Content and design → Construct a base of rules – by an expert

- Rules
- Properties
  - ▢ Completeness
    - A database of fuzzy rules is complete
      - If all input values have associated a value between 0 and 1
      - If all fuzzy variable are complete
      - If used fuzzy sets have a non-compact support
  - ▢ Consistency
    - A set of fuzzy rules is inconsistent if two rules have the same premises and different consequences
      - If  $x$  in A and  $y$  in B then  $z$  in C
      - If  $x$  in A and  $y$  in B then  $z$  in D
- Problems of the database
  - ▢ Rule's explosion
    - #of rules increases exponential whit the # of input variables
    - # of input set combinations is
      - Where the  $i^{th}$  variable is composed by  $p_i$  sets

$$P = \prod_{i=1}^n p_i$$

# Intelligent systems – KBS – Fuzzy systems

Content and design → Construct a base of rules – by an expert

- Decision matrix of the knowledge database

- Example – air conditioner device

- Inputs :

- x (temperature – cold, normal, hot) and
    - y (humidity – small, large)

- Outputs:

- z (machine power – law, constant, high)

- Rules:

- *If temperature is normal and humidity is small then the power is constant*

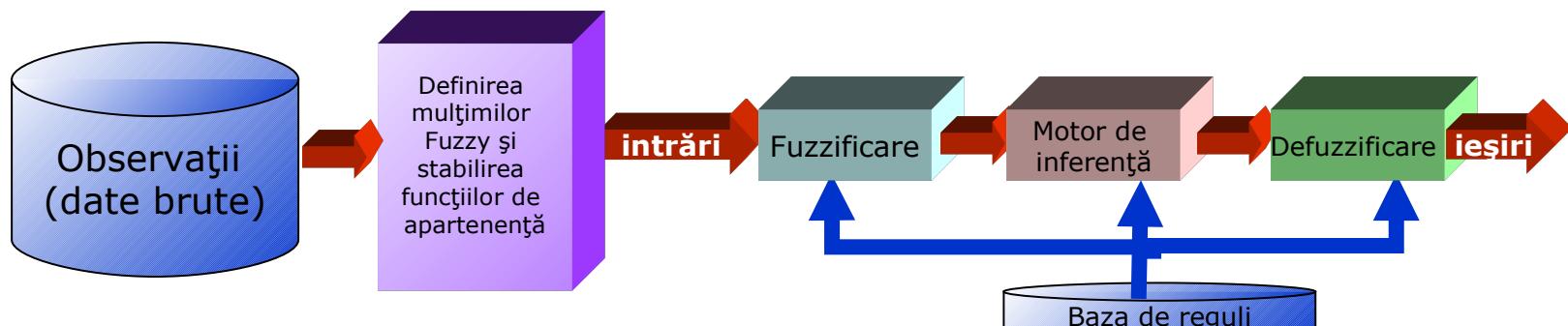
		Input data y	
		Small	Large
Input data x	Cold	Law	Constant
	Normal	Constant	High
	Hot	High	High

# Intelligent systems – KBS – Fuzzy systems

## Content and design

### □ Steps for constructing a fuzzy system

- Define the inputs and the outputs – by an expert
  - Raw inputs and outputs
  - Fuzzification of inputs and outputs
    - Fix the fuzzy variables and fuzzy sets based on membership functions
- Construct a base of rules – by an expert
  - Decision matrix
- **Evaluate the rules**
  - Inference – transform the fuzzy inputs into fuzzy outputs by applying all the rules
- Aggregate the results
- Defuzzificate the result
- Interpret the result



# Intelligent systems – KBS – Fuzzy systems

---

Content and design → rule evaluation (fuzzy inference)

- Which rules are firstly evaluated?

- Fuzzy inference
    - Rules are evaluated in **parallel**, each rules contributing to the shape of the final result
    - Resulted fuzzy sets are de-fuzzified **after all the rules** have been evaluated

**Remember**

- Forward inference
    - For a given state of problem, collect the required information and apply the possible rules
  - Backward inference
    - Identify the rules that determine the final state and apply only that rules (if it is possible)

- How the rules are evaluated?

- Evaluation of causes
  - Evaluation of consequences

# Intelligent systems – KBS – Fuzzy systems

---

Content and design → rule evaluation (fuzzy inference)

## □ Evaluation of causes

- For each premise of a rule (*if s is (in) A*) establish the membership degree of raw input data to all fuzzy sets
- A rule can have more premises linked by logic operators *AND*, *OR* or *NOT* → use fuzzy operators
  - Operator *AND* → intersection (minimum) of 2 sets
    - $\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$
  - Operator *OR* → union (maximum) of 2 sets
    - $\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$
  - Operator *NOT* → negation (complement) of a set
    - $\mu_{\neg A}(x) = 1 - \mu_A(x)$
- The result of premise's evaluation
  - Degree of satisfaction
  - Other names:
    - Rule's firing strength
    - Degree of fulfillment

# Intelligent systems – KBS – Fuzzy systems

---

Content and design → rule evaluation (fuzzy inference)

- Evaluation of consequences

- Determine the results
  - Establish the membership degree of variables (involved in the consequences) to different fuzzy sets
- Each output region must be de-fuzzified in order to obtain crisp value
- Based on the consequence's type
  - Mamdani model – consequence of rule: "output variable belongs to a fuzzy set"
  - Sugeno model – consequence of rule: "output variable is a crisp function that depends on inputs"
  - Tsukamoo model – consequence of rule: "output variable belongs to a fuzzy set following a monotone membership function"

# Intelligent systems – KBS – Fuzzy systems

---

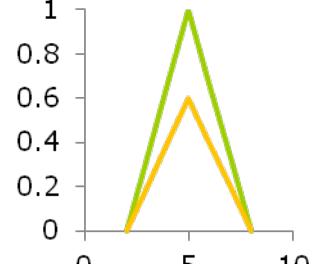
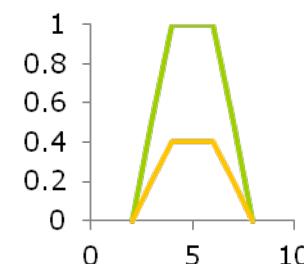
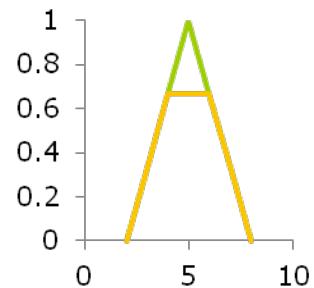
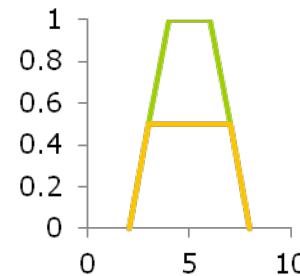
Content and design → rule evaluation (fuzzy inference) →  
**Evaluation of consequences**

- Mamdani model
  - Main idea:
    - consequence of rule: “output variable belongs to a fuzzy set”
    - Result of evaluation is applied for the membership function of the consequence
    - Example
      - ***if x is in A and y is in B, then z is in C***
  - Typology (based on how the results is applied on the membership function of the consequence)
    - Clipped fuzzy sets
    - Scaled fuzzy sets

# Intelligent systems – KBS – Fuzzy systems

Content and design → rule evaluation (fuzzy inference) →  
**Evaluation of consequences**

- Mamdani model
  - Typology (based on how the results is applied on the membership function of the consequence)
    - Clipped fuzzy sets
      - Membership function of the consequence is cut at the level of the result's truth value
      - Advantage → easy to compute
      - Disadvantage → some information are lost
    - Scaled fuzzy sets
      - Membership function of the consequence is adjusted by scaling (multiplication) at the level of the result's truth value
      - Advantage → few information is lost
      - Disadvantage → complicate computing



# Intelligent systems – KBS – Fuzzy systems

- Content and design → rule evaluation (fuzzy inference) →  
**Evaluation of consequences** → Mamdani model

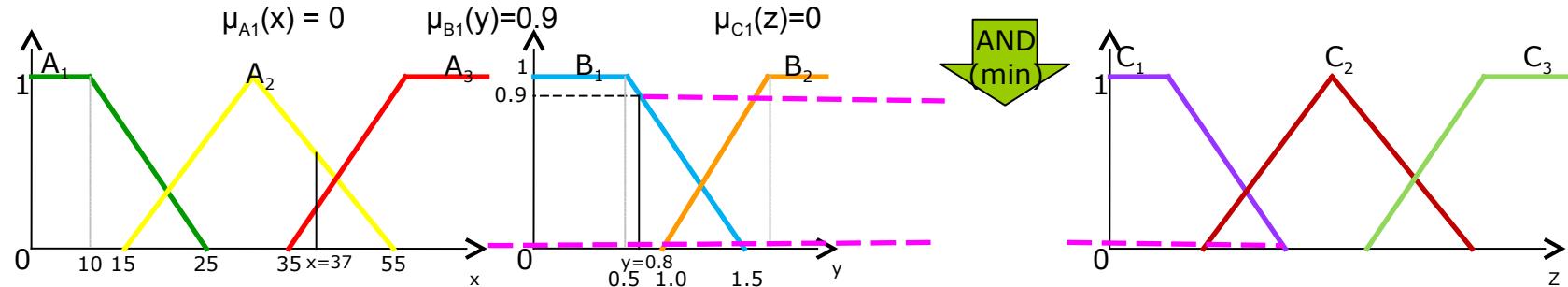
- Example – air conditioner device
  - Inputs :
    - x (temperature – cold, normal, hot) and
    - y (humidity – small, large)
  - Outputs:
    - z (machine power – law, constant, high)
  - Input data:
    - Temperature x = 37
      - $\mu_{A1}(x)=0, \mu_{A2}(x)=0.6, \mu_{A3}(x)=0.3$
    - Humidity y = 0.8
      - $\mu_{B1}(x)=0.9, \mu_{B2}(x)=0$

		Input data y	
		Small	Large
Input data x	Cold	Law	Constant
	Normal	Constant	High
	Hot	High	High

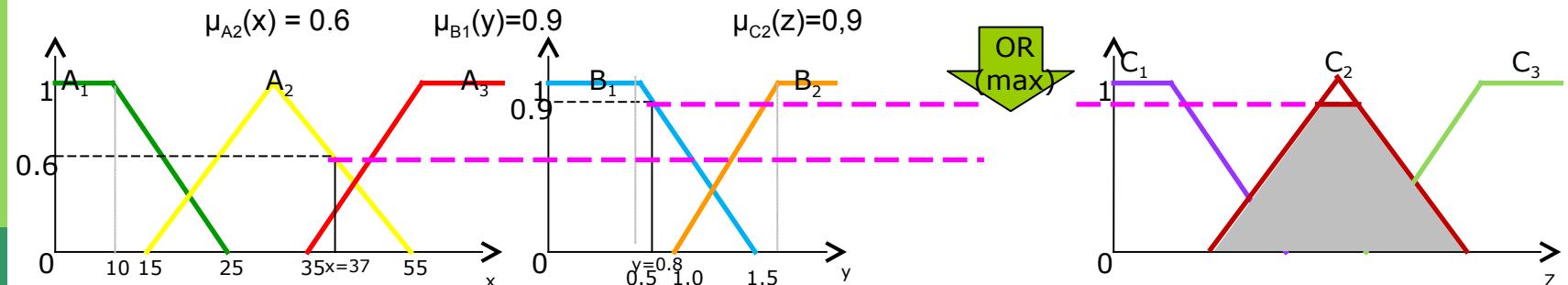
# Intelligent systems – KBS – Fuzzy systems

Content and design → rule evaluation → Evaluation of consequences → Mamdani model

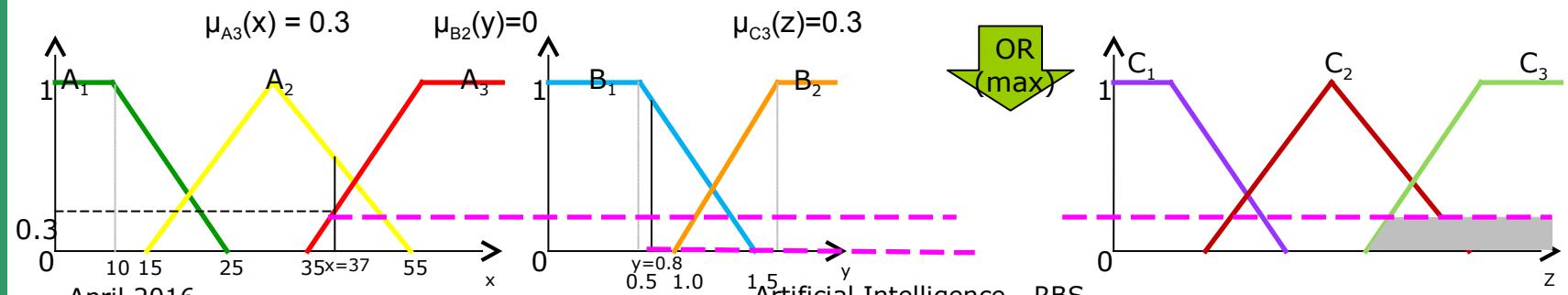
R1: if  $x$  is in  $A_1$  and  $y$  is in  $B_1$  then  $z$  is in  $C_1$



R2: if  $x$  is in  $A_2$  or  $y$  is in  $B_1$  then  $z$  is in  $C_2$



R3: if  $x$  is in  $A_3$  or  $y$  is in  $B_2$  then  $z$  is in  $C_3$



# Intelligent systems – KBS – Fuzzy systems

---

Content and design → rule evaluation (fuzzy inference) →

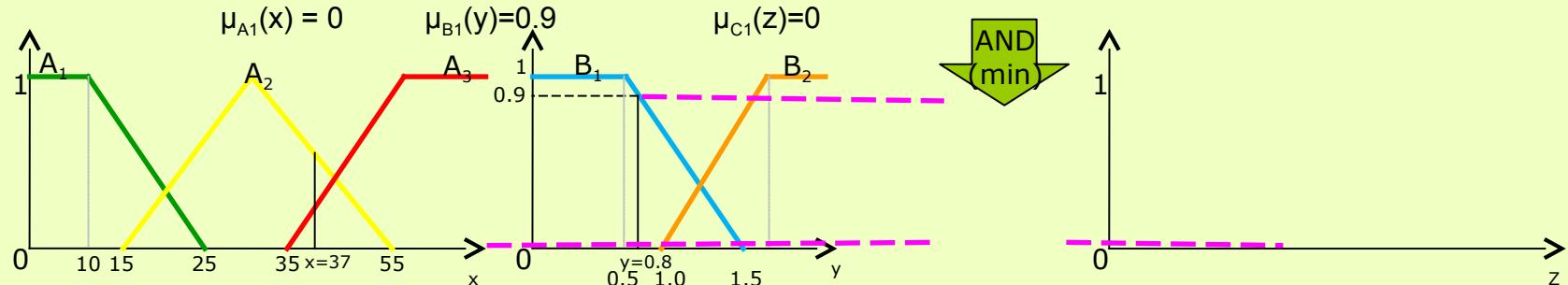
## Evaluation of consequences

- Sugeno model
  - Main idea
    - consequence of rule: “output variable is a crisp function that depends on inputs”
    - Example
      - If x is in A and y is in B then z is f(x,y)***
    - Typology (based on characteristics of  $f(x,y)$ )
      - Sugeno model of degree 0 → if  $f(x,y) = k$  – constant (membership function of the consequences are singleton – a fuzzy set whose membership functions have value 1 for a single (unique) point of the universe and 0 for all other points)
      - Sugeno model of degree 1 → if  $f(x,y) = ax + by + c$

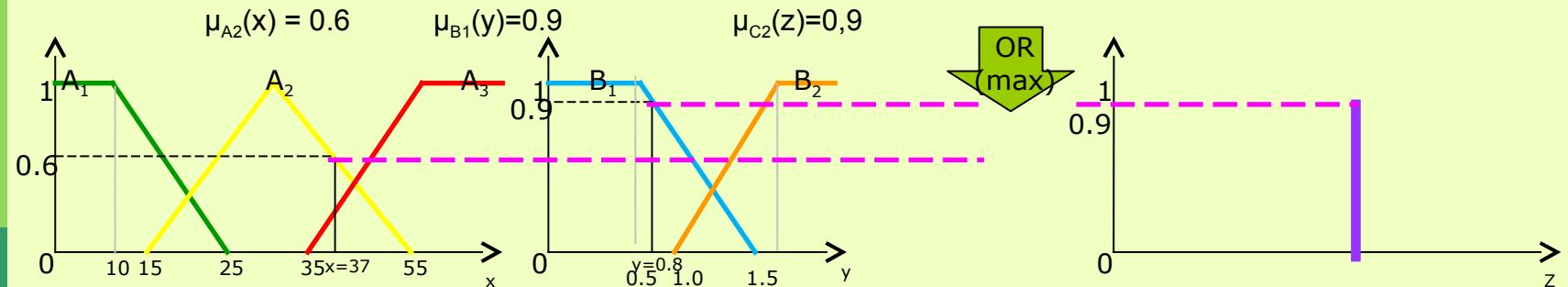
# Intelligent systems – KBS – Fuzzy systems

Content and design → rule evaluation → Evaluation of consequences → Sugeno model

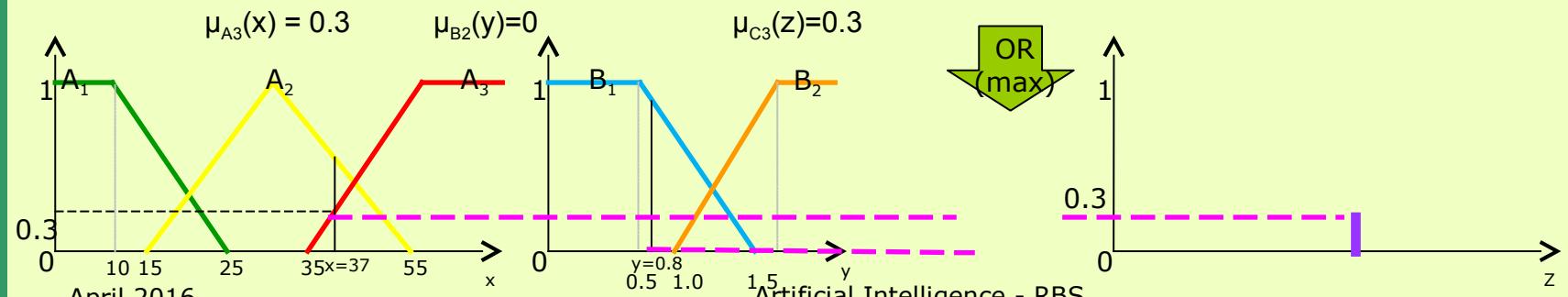
R1: if  $x$  is in  $A_1$  and  $y$  is in  $B_1$  then  $z$  is in  $C_1$



R2: if  $x$  is in  $A_2$  or  $y$  is in  $B_1$  then  $z$  is in  $C_2$



R3: if  $x$  is in  $A_3$  or  $y$  is in  $B_2$  then  $z$  is in  $C_3$



# Intelligent systems – KBS – Fuzzy systems

---

Content and design → rule evaluation (fuzzy inference) →

## **Evaluation of consequences**

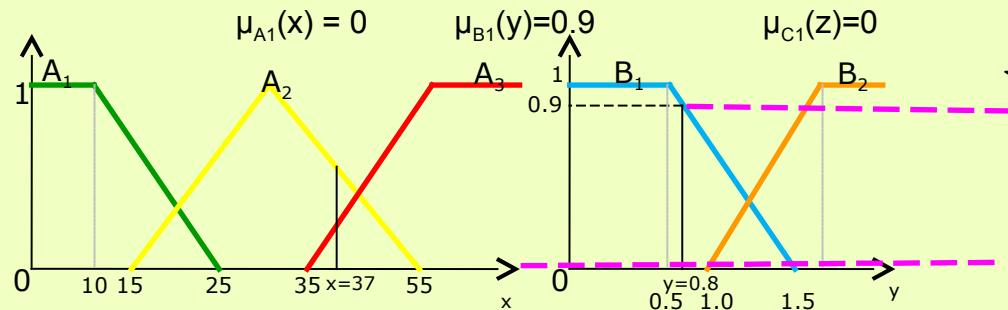
- Tsukamoto model

- Main idea
    - consequence of rule: “output variable belongs to a fuzzy set following a monotone membership function”
      - A crisp value is obtained as output → *rule’s firing strength*

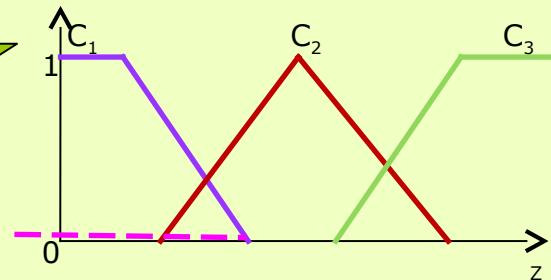
# Intelligent systems – KBS – Fuzzy systems

Content and design → rule evaluation → Evaluation of consequences → Tsukamoto model

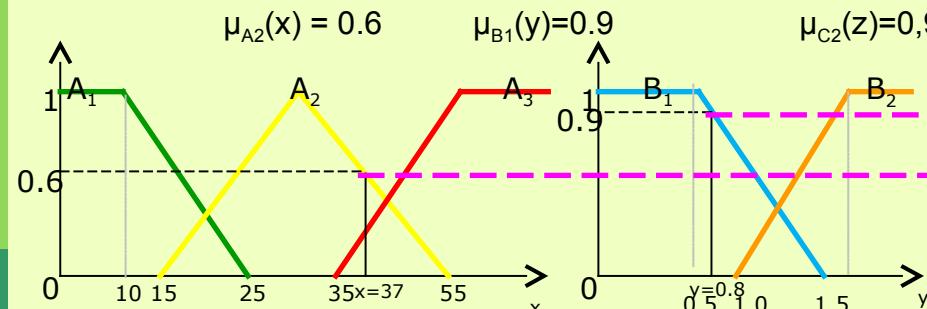
R1: if  $x$  is in  $A_1$  and  $y$  is in  $B_1$  then  $z$  is in  $C_1$



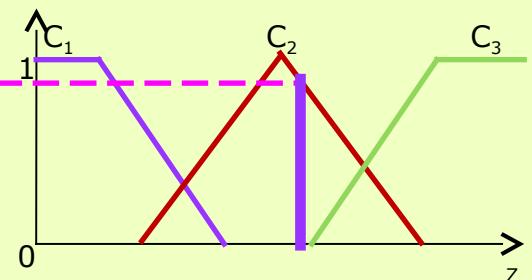
AND  
(min)



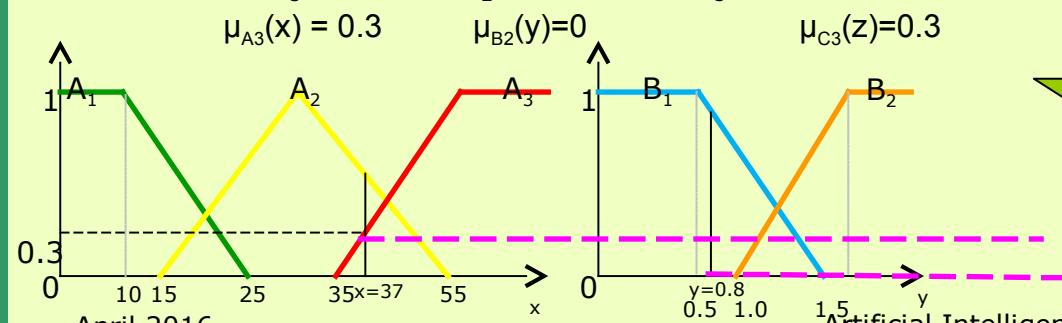
R2: if  $x$  is in  $A_2$  or  $y$  is in  $B_1$  then  $z$  is in  $C_2$



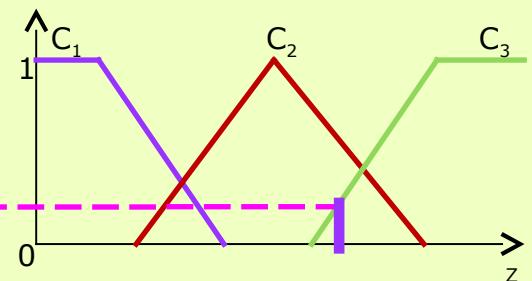
OR  
(max)



R3: if  $x$  is in  $A_3$  or  $y$  is in  $B_2$  then  $z$  is in  $C_3$



OR  
(max)

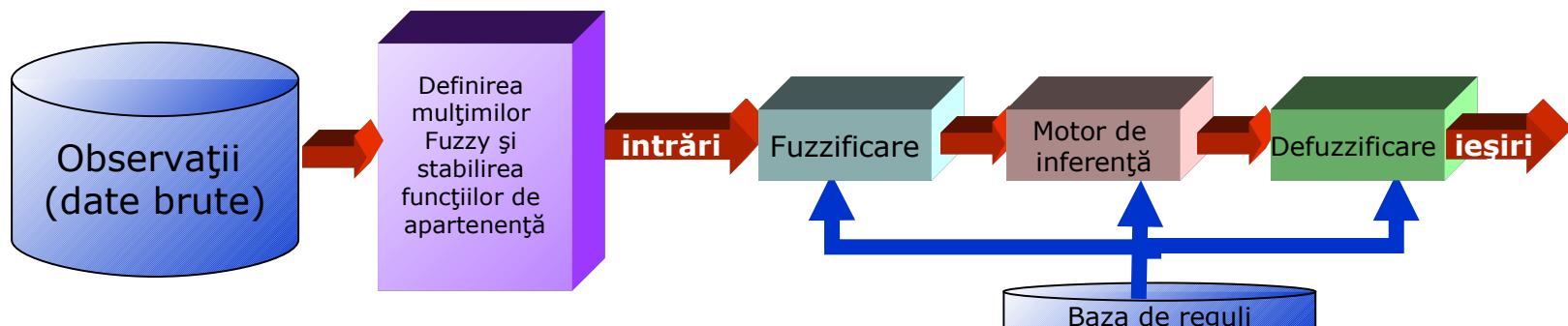


# Intelligent systems – KBS – Fuzzy systems

## Content and design

### □ Steps for constructing a fuzzy system

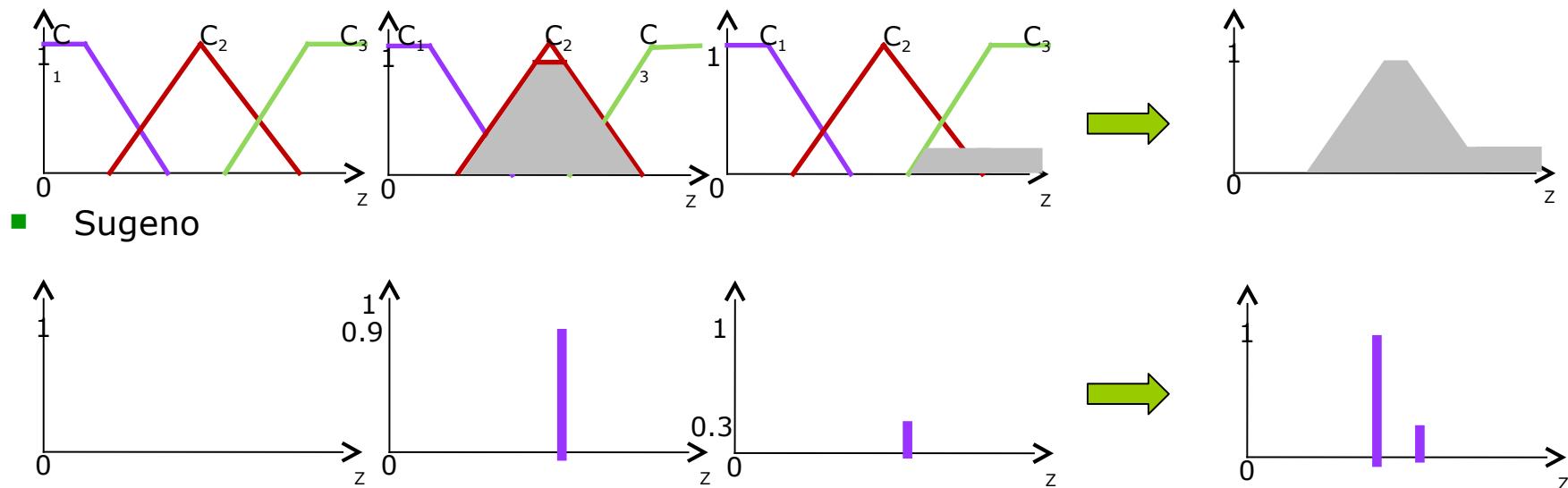
- Define the inputs and the outputs – by an expert
  - Raw inputs and outputs
  - Fuzzification of inputs and outputs
    - Fix the fuzzy variables and fuzzy sets based on membership functions
- Construct a base of rules – by an expert
  - Decision matrix
- Evaluate the rules
  - Inference – transform the fuzzy inputs into fuzzy outputs by applying all the rules
- **Aggregate the results**
- Defuzzificate the result
- Interpret the result



# Intelligent systems – KBS – Fuzzy systems

## Content and design → **Aggregate the results**

- ❑ Union of outputs for all the applied rules
- ❑ Consider the membership functions for all the consequences and combine them into a single fuzzy set (a single result)
- ❑ Aggregation process have as
  - Inputs → membership functions (clipped or scaled) of the consequences
  - Outputs → a fuzzy set of the output variable
- ❑ Example
  - Mamdani

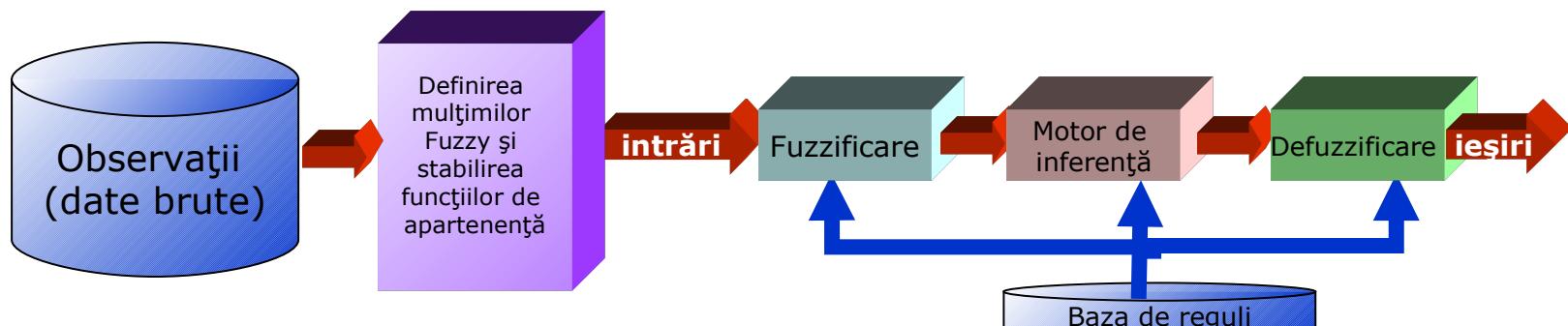


# Intelligent systems – KBS – Fuzzy systems

## Content and design

### □ Steps for constructing a fuzzy system

- Define the inputs and the outputs – by an expert
  - Raw inputs and outputs
  - Fuzzification of inputs and outputs
    - Fix the fuzzy variables and fuzzy sets based on membership functions
- Construct a base of rules – by an expert
  - Decision matrix
- Evaluate the rules
  - Inference – transform the fuzzy inputs into fuzzy outputs by applying all the rules
- Aggregate the results
- **Defuzzificate the result**
- Interpret the result



# Intelligent systems – KBS – Fuzzy systems

---

Content and design → defuzzification

## □ Main idea

- Transform the fuzzy result into a crisp (raw) value
- Inference → obtain some fuzzy regions for each output variable
- Defuzzification → transform each fuzzy region into a crisp value

## □ Methods

- Based on the gravity center
  - COA – Centroid Area
  - BOA – *Bisector of area*
- Based on maximum of membership function
  - MOM - *Mean of maximum*
  - SOM - *Smallest of maximum*
  - LOM - *Largest of maximum*

# Intelligent systems – KBS – Fuzzy systems

Content and design → defuzzification → methods

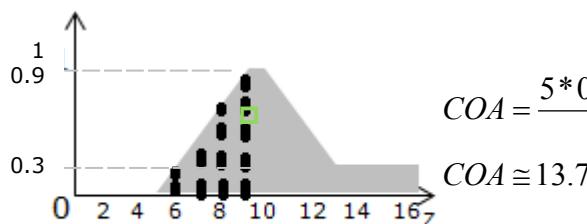
- COA – Centroid Area

- Identify the z point from the middle of aggregated set

$$COG = \frac{\sum_{i=0}^n x_i \mu_A(x_i)}{\sum_{i=0}^n \mu_A(x_i)} \text{ sau } COG = \frac{\int x_i \mu_A(x_i) dx}{\int \mu_A(x_i) dx}$$

- Example

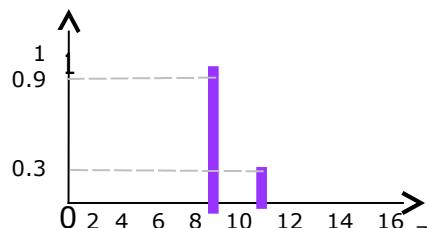
- Mamdani model → estimation of COA by using a sample of n points ( $x_i$ ,  $i = 1, 2, \dots, n$ ) of the resulted fuzzy set



$$COA = \frac{5*0 + 6*0.3 + 7*0.5 + 8*0.7 + 9*0.9 + 10*0.9 + 11*0.7 + 12*0.5 + 13*0.3 + 14*0.3 + 15*0.3 + 16*0.3}{0 + 0.3 + 0.5 + 0.7 + 0.9 + 0.9 + 0.7 + 0.5 + 0.3 + 0.3 + 0.3 + 0.3}$$

$$COA \approx 13.7$$

- Sugeno or Tsukamoto model → COA becomes a weighted average of m crisp values obtained by applying all m rules



$$COA = \frac{9*0.9 + 11*0.3}{0.9 + 0.3}$$

$$COA \approx 9.5$$

# Intelligent systems – KBS – Fuzzy systems

---

Content and design → defuzzification → methods

- BOA – Bisector of area

- Identify the point  $z$  that determine the splitting of aggregated set in 2 parts of equal area

$$BOA = \int_{\alpha}^z \mu_A(x)dx = \int_{z}^{\beta} \mu_A(x)dx,$$

where  $\alpha = \min\{x \mid x \in A\}$  and  $\beta = \max\{x \mid x \in A\}$

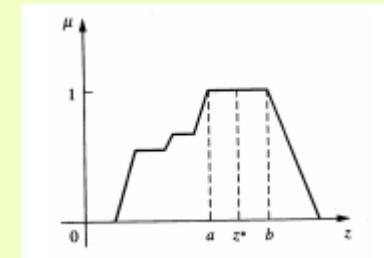
# Intelligent systems – KBS – Fuzzy systems

Content and design → defuzzification → methods

- MOM - *Mean of maximum*

- Identify the point  $z$  that represents the mean of that points (from the aggregated set) that have a maximum membership function

$$MOM = \frac{\sum_{x_i \in \max \mu} x_i}{|\max \mu|}, \text{ where } \max \mu = \mu^* = \{x \mid x \in A, \mu(x) = \max\}$$



- SOM - *Smallest of maximum*

- Identify the smallest point  $z$  (from the aggregated set) that have a maximum membership function

- LOM - *Largest of maximum*

- Identify the largest point  $z$  (from the aggregated set) that have a maximum membership function

# Intelligent systems – KBS – Fuzzy systems

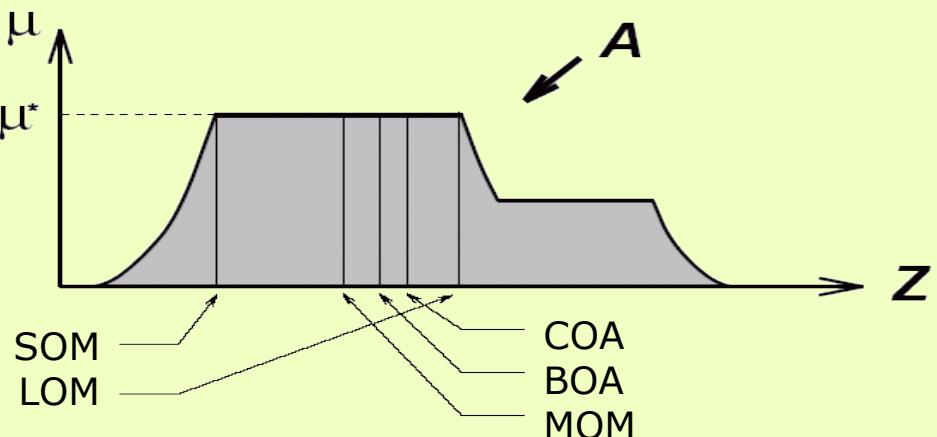
Content and design → defuzzification

## □ Main idea

- Transform the fuzzy result into a crisp (raw) value
- Inference → obtain some fuzzy regions for each output variable
- Defuzzification → transform each fuzzy region into a crisp value

## □ Methods

- Based on the gravity center
  - COA – Centroid Area
  - BOA – *Bisector of area*
- Based on maximum of membership function
  - MOM - *Mean of maximum*
  - SOM - *Smallest of maximum*
  - LOM - *Largest of maximum*

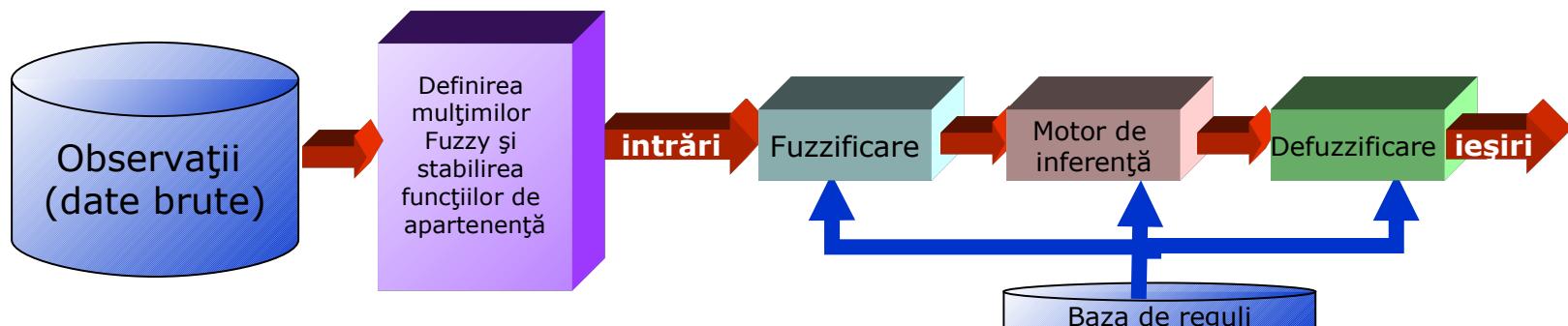


# Intelligent systems – KBS – Fuzzy systems

## Content and design

### □ Steps for constructing a fuzzy system

- Define the inputs and the outputs – by an expert
  - Raw inputs and outputs
  - Fuzzification of inputs and outputs
    - Fix the fuzzy variables and fuzzy sets based on membership functions
- Construct a base of rules – by an expert
  - Decision matrix
- Evaluate the rules
  - Inference – transform the fuzzy inputs into fuzzy outputs by applying all the rules
- Aggregate the results
- Defuzzificate the result
- **Interpret the result**

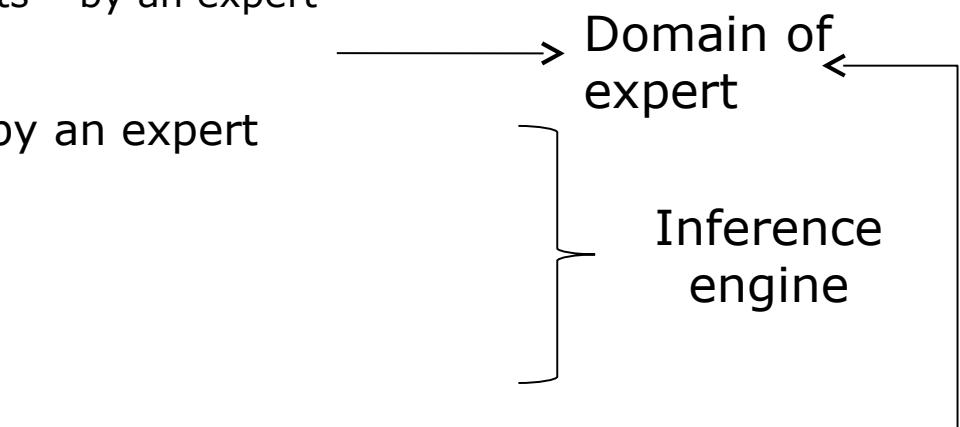


# Intelligent systems – KBS – Fuzzy systems

## Content and design

### □ Steps for constructing a fuzzy system

- Define the inputs and the outputs – by an expert
  - Raw inputs and outputs
  - Fuzzification of inputs and outputs
- Construct a base of rules – by an expert
- Evaluate the rules
- Aggregate the results
- Defuzzificate the result
- Interpret the result



# Intelligent systems – KBS – Fuzzy systems

---

## □ Advantages

- Imprecision and real-world approximations can be expressed through some rules
- Easy to understand, to test and to maintain
- Robustness → can operate when rules are not so clear
- Require few rules than other KBSs
- Rules are evaluated in parallel

## □ Disadvantages

- Require many simulations and tests
- Do not automatically learn
- It is difficult to identify the most correct rules
- There is not mathematical model

# Intelligent systems – KBS – Fuzzy systems

---

## Applications

- Space control
  - Altitude of satellites
  - Setting the planes
- Auto-control
  - Automatic transmission, traffic control, anti-breaking systems
- Business
  - Decision systems, personal evaluation, fond management, market predictions, etc
- Industry
  - Energy exchange control, water purification control
  - pH control, chemical distillation, polymer production, metal composition
- Electronic devices
  - Camera exposure, humidity control. Air conditioner, shower setting
  - Freezer setting
  - Washing machine setting

# Intelligent systems – KBS – Fuzzy systems

---

## Applications

- ❑ Nourishment
  - Cheese production
- ❑ Military
  - Underwater recognition, infrared image recognition, vessel traffic decision
- ❑ Navy
  - Automatic drivers, route selection
- ❑ Medical
  - Diagnostic systems, pressure control during anesthesia, modeling the neuropathology results of Alzheimer patients
- ❑ Robotics
  - Kinematics (arms)

# Review

---



## ❑ KBSs

- Computation systems where knowledge database and inference engine overlap

## ❑ KBSs can work

- In certainty environment
  - ❑ LBS
  - ❑ RBS
- In uncertainty environments
  - ❑ Bayes systems
    - Rules have associated some probabilities
  - ❑ Systems based on certainty factors
    - Fact and rules have associated certainty factors
  - ❑ Fuzzy systems
    - Fact have associated degree of membership to some sets

# Next lecture

---

- A. Short introduction in Artificial Intelligence (AI)
- B. Solving search problems
  - A. Definition of search problems
  - B. Search strategies
    - A. Uninformed search strategies
    - B. Informed search strategies
    - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
    - D. Adversarial search strategies
- C. Intelligent systems
  - A. Rule-based systems in certain environments
  - B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
- C. Learning systems
  - A. Decision Trees
  - B. Artificial Neural Networks
  - C. Support Vector Machines
  - D. Evolutionary algorithms
- D. Hybrid systems

# Next lecture – useful information

---

- Chapter VI (18 and 19) of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 8 of *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapters 10, 11, 12 and 13 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter V of *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Chapters 3 and 4 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

- 
- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean – [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)

# Intelligent Systems

## Automatic learning systems (ALS)

T. Mihoc<sup>1</sup>

<sup>1</sup>Department of Computer Science  
Babeş Bolyai University

## Machine Learning

Problem Definition  
and Applications

Design

Automatic learning

Supervised learning

Unsupervised learning

Active learning;

Reinforcement  
learning

# Machine Learning

Problem Definition and Applications

Design

Automatic learning

Supervised learning

Unsupervised learning

Active learning; Reinforcement learning

## Machine Learning

Problem Definition  
and Applications

Design

Automatic learning

Supervised learning

Unsupervised  
learning

Active learning;  
Reinforcement  
learning

# Machine Learning

## Problem Definition and Applications

Design

Automatic learning

Supervised learning

Unsupervised learning

Active learning; Reinforcement learning

# Machine Learning

- ▶ Problem:
  - ▶ "How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes? "
- ▶ Applications:
  - ▶ Image and voice recognition
    - ▶ Handwritten recognition
    - ▶ Face detection
  - ▶ Computer vision
    - ▶ Obstacle detection
    - ▶ Footprint recognition
  - ▶ Bio-surveillance
  - ▶ Robot control
  - ▶ Predictions
  - ▶ Medical diagnostic
  - ▶ Fraud detection

## Machine Learning

Problem Definition  
and Applications

Design

Automatic learning

Supervised learning

Unsupervised  
learning

Active learning;  
Reinforcement  
learning

## Machine Learning

Problem Definition  
and Applications

Design

Automatic learning

Supervised learning

Unsupervised  
learning

Active learning;

Reinforcement

learning

- ▶ Definition:
  - ▶ Arthur Samuel (1959): *field of study that gives computers the ability to learn without being explicitly programmed*
  - ▶ Herbert Simon (1970): *learning is any process by which a system improves performance from experience*
  - ▶ Ethem Alpaydin (2010): *programming computers to optimize a performance criterion using example data or past experience*
- ▶ Necessity:
  - ▶ Better computational systems
    - ▶ To difficult or too expensive to be constructed manually
    - ▶ Systems that automatically adapt (spam filters,  
Systems that discover information in large databases  
→ *data mining, financial analysis, text/image analysis*)
  - ▶ Understanding the biological systems

## Machine Learning

Problem Definition  
and Applications

### Design

Automatic learning

Supervised learning

Unsupervised  
learning

Active learning;  
Reinforcement  
learning

# Machine Learning

Problem Definition and Applications

## Design

Automatic learning

Supervised learning

Unsupervised learning

Active learning; Reinforcement learning

# Design

- ▶ Improve task T
  - ▶ establish the goal (what has to be learn) objective function and its representation
  - ▶ select a learning algorithms to perform the inference of the goal based on experience
- ▶ Respect a performance metric P
  - ▶ Evaluation of the algorithms performances
- ▶ Based on experience E
  - ▶ Select an experience database

## Examples

- ▶ T: Playing checkers  
P: percent of winning games  
E: playing the game
- ▶ T: handwritten recognition  
P: percent of correct recognized words  
E: database of images with different words
- ▶ T: separate the spams  
P: percent of correct classified emails  
E: databases with annotated emails

## Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
Unsupervised  
learning  
Active learning;  
Reinforcement  
learning

- ▶ identify the function that must be learned  
Example for checkers game: a function that selects the next move/evaluates a move
- ▶ Representation of objective function
  - ▶ Different representations
    - ▶ Table
    - ▶ Symbolic rules
    - ▶ Numeric functions
    - ▶ Probabilistic functions
  - ▶ There is a trade-off between:
    - ▶ Expressiveness of representation
    - ▶ Facile learning
  - ▶ Computation of the objective function:
    - ▶ polynomial time
    - ▶ non-polynomial time

# Design

## Selecting an algorithm

- ▶ By using the training data
- ▶ Induce the hypothesis definition that:
  - ▶ Match the data
  - ▶ Generalize unseen data
- ▶ Main principle – error minimization (cost function – loss function)

## Evaluation of a learning system

- ▶ Experimental
  - ▶ comparing different methods on different data (cross-validation)
  - ▶ collect data based on performances (accuracy, training time, testing time)
  - ▶ statistical analyze of the differences
- ▶ Theoretic – mathematical analyzes of algorithms and theorems proving
  - ▶ Computational complexity
  - ▶ Ability to match the training data
  - ▶ Complexity of the most relevant sample for learning

## Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
Unsupervised  
learning  
Active learning;  
Reinforcement  
learning

- ▶ Comparing the performances of 2 algorithms for solving a given problem
  - ▶ Performance measures
  - ▶ Parameters of a statistic series
  - ▶ Proportion (percent) computed for a statistical series (ex. Accuracy)
- ▶ Comparing based on confidence intervals
  - ▶ one problem, two solving algorithms  $p_1, p_2$
  - ▶ confidence intervals  $I_1 = [p_1 - \Delta_1, p_1 + \Delta_1]$  and  $I_2 = [p_2 - \Delta_2, p_2 + \Delta_2]$
  - ▶ if  $I_1 \cap I_2 = \emptyset$  algorithm 1 works better than algorithm 2 (for the given problem)
  - ▶ if  $I_1 \cap I_2 \neq \emptyset$  can't be decided

## Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
Unsupervised  
learning  
Active learning;  
Reinforcement  
learning

# Design: choose the training database

Based on:

- ▶ direct experience: pairs (in, out) that are useful for the objective function
  - Ex.: board annotated with correct or incorrect move
- ▶ indirect experience: useful feedback (unlike i/o pairs) for the objective function
  - Ex.: sequences of moves and the final score of the game

Data sources:

- ▶ random generated examples (positive and negative examples);
- ▶ positive examples collected by a learner;
- ▶ real examples

## Machine Learning

Problem Definition  
and Applications

### Design

Automatic learning

Supervised learning

Unsupervised  
learning

Active learning;  
Reinforcement  
learning

### Content:

- ▶ training data
- ▶ test data

### Characteristics

- ▶ Independent data (otherwise → collective learning)
- ▶ Training and testing data must respect the same distribution law (otherwise → transfer learning/inductive transfer)
  - ▶ vehicle recognition, truck recognition, text analyses, spam filters

## Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
Unsupervised  
learning  
Active learning;  
Reinforcement  
learning

# Design: choose the training database

Characteristics extracted (attributes) from raw data

- ▶ Quantitative characteristics: → nominal or rational scale
  - ▶ Continuous values → weight
  - ▶ Discrete values → no. of computers
  - ▶ Range values → event times
- ▶ Qualitative characteristics
  - ▶ Nominal → colour
  - ▶ Ordinal → sound intensity (low, medium, high)
  - ▶ Structured → Trees root is a generalization of children  
(vehicle → car, bus, tractor, truck)

## Data transformation

- ▶ Standardization → numerical attributes
  - ▶ Remove the scale effect (different scale and units)
  - ▶ Raw values are transformed in z scores
$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$
 where  $x_{ij}$  is the value of the  $j^{th}$  attribute of  $i^{th}$  instance,  $\mu_j$  ( $\sigma_j$ ) is the mean (respectively the standard deviation) of the  $j^{th}$  attribute of all instances
- ▶ Selection of some attributes

# Machine Learning – Typology

Based on their aim (goal)

## ► ISs for prediction

- ▶ Aim: predict the output for a new input based on a previously learned model
- ▶ Eg. predicting sales of a product for a time in the future based on price, calendar month, region, average income

## ► ISs for regression

- ▶ Aim: estimation of the (uni or multi - variable) function shape based on a previously learned model
- ▶ Eg.: estimate the function that models the edge of a surface

## ► ISs for classification

- ▶ Aim: classify an object into one or more known or unknown - categories based on their characteristics
- ▶ Eg.: diagnostic systems for cancer: malign or benign or normal

## ► ISs for planning

- ▶ Aim: generate a sequence of optimal actions for performing a task
- ▶ Eg.: planning the moves of a robot from a position to a source of energy

# Machine Learning – Typology

Based on the experience learned during training process

- ▶ ISs with supervised learning
- ▶ ISs with unsupervised learning
- ▶ ISs with active learning
- ▶ ISs with reinforcement learning

## Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
Unsupervised  
learning  
Active learning;  
Reinforcement  
learning

## Machine Learning

Problem Definition  
and Applications  
Design

### Automatic learning

Supervised learning  
Unsupervised  
learning  
Active learning;  
Reinforcement  
learning

## Machine Learning

Problem Definition and Applications

Design

### Automatic learning

Supervised learning

Unsupervised learning

Active learning; Reinforcement learning

# Supervised learning

## Aim

To provide a correct output for a new entry.

## Definition

Consider a training data set of  $N$  elements, organized in pairs  $(entrydata_i, output_i)$  where  $i = \overline{1, N}$  and a testing data set of  $n$  similar pairs.

- ▶  $entrydata_i = (attr_{i_1}, attr_{i_2}, \dots, attr_{i_m})$ ,  $m$  the number of attributes for one data entry.
- ▶  $output_i$  can be:
  - ▶ a category from  $k$  classes (an element from a predefined set of  $k$  elements) → classification problem
  - ▶ a real number → regression problem

We denote by *Supervised learning* the process of finding a function (unknown) that maps the given entries to the outputs.

# Supervised learning

## Other names

classification (regression), inductive learning

Process → 2 steps

- ▶ Training – learning, using a certain algorithm, the model from the training data
- ▶ Testing – testing the detected model using the test data (unseen data)

## Characteristics

Features an experimental adnotated Data Base for learning.

# Supervised learning

## Problems' types

- ▶ regression
  - ▶ Aim: prediction of an output for a new input
  - ▶ Continuous output (a real number)
  - ▶ Ex. prediction of prices
- ▶ classification
  - ▶ Aim: classifying a new input
  - ▶ Discrete output (a label from a predefined set)
  - ▶ Ex. detection of malignant tumors

## Example of problems

- ▶ recognizing hand writing
- ▶ image recognition
- ▶ predicting the weather
- ▶ spam detection

# Supervised learning

## The quality of learning

### Definition

The *quality of learning* is a measure of the algorithm's performance. This measure is determined during the training faze and the testing faze.

Example: Accuracy = the number of correct classified entries / the total number of entries.

# Supervised learning

## The quality of learning

### Evaluation methods

- ▶ disjoint sets of training and testing
  - ▶ the training set is divided in two parts, one for learning one for testing
  - ▶ adequate for large data sets
- ▶ cross-validation with many ( $h$ ) equal sub-sets
  - ▶ dividing the data set in  $h$  parts ( $h - 1$  for training 1 for testing)
  - ▶ the performance is the average on the  $h$  runs
  - ▶ adequate for small data sets
- ▶ leave-one-out cross-validation
  - ▶ similar with cross-validation but  $h =$  the number of elements from the data set

### Machine Learning

Problem Definition and Applications

Design

Automatic learning

Supervised learning

Unsupervised learning

Active learning; Reinforcement learning

# Supervised learning

## The quality of learning

Difficulties: over-fitting → very good performance for the learning data set and very poor on the test data.

## Performance measures

- ▶ statistical measures
- ▶ efficiency (building and testing the model)
- ▶ scaling (efficiency for large data sets)
- ▶ compactness
- ▶ robustness (coping with noises and missing data)

# Supervised learning

## Statistical measures

### ► Accuracy

- ▶ no. of correct classified / total no. of examples
- ▶ the opposite of error
- ▶ determined on the training set and on the testing set

### ► Precision (P)

- ▶ no. of correct positive classified / total no. of examples classified as positive
- ▶ the probability that a positive classified example is relevant
- ▶  $TP/(TP + FP)$

### ► Rappel (R)

- ▶ no. of correct positive classified / total no. of positive examples
- ▶ the probability that a positive example is correctly classified
- ▶  $TP/(TP + FN)$
- ▶ confusion matrix – real results versus computed results

### ► Score

- ▶ combines  $P$  and  $R$
- ▶  $2PR/(P + R)$

## Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
**Unsupervised  
learning**

Active learning;  
Reinforcement  
learning

# Unsupervised learning

## Aims:

- ▶ finding a model or a structure in a data set
- ▶ dividing a set of unlabeled examples in disjoint sets (clustering) such as:
  - ▶ the elements of a cluster are very similar
  - ▶ the elements of different clusters are very different

## Definition

Consider a data set consisting of  $N$  elements. We define by *unsupervised learning* the process of determining a function (unknown) that groups the elements from the data set in several classes. The number of classes can be predefined  $k$  or unknown. The elements from one class are similar according to some criteria.

# Unsupervised learning

## Distance versus Similarity

- ▶ distance → min
- ▶ similarity → max

## Examples of distances

- ▶ Euclidean:  $d(p, q) = \sqrt{\sum_{j=1}^m (p_j - q_j)^2}$
- ▶ Manhattan:  $d(p, q) = \sum_{j=1}^m |p_j - q_j|$
- ▶ Mahalanobis:  $d(p, q) = \sqrt{((p - q)^T S^{-1} (p - q))}$ ,  
where  $S$  is the co-variance matrix
- ▶ Internal product:  $d(p, q) = \sum_{j=1}^m p_j q_j$
- ▶ Cosine:  

$$d(p, q) = \sum_{j=1}^m p_j q_j / (\sqrt{\sum_{j=1}^m p_j^2}) * \sqrt{\sum_{j=1}^m q_j^2})$$
- ▶ Hamming: the number of differences between  $p$  and  $q$
- ▶ Levenshtein: the minimum no. of operations in order to transform  $p$  in  $q$

## Machine Learning

Problem Definition  
and Applications

Design

Automatic learning  
Supervised learning

**Unsupervised  
learning**

Active learning;  
Reinforcement  
learning

## Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
**Unsupervised  
learning**

Active learning;  
Reinforcement  
learning

# Unsupervised learning

## Other names

clustering

Process → 2 steps

- ▶ Training – learning, using a certain algorithm, the clusters
- ▶ Testing – testing the detected model by placing a new element in a cluster

## Characteristics

Features non ad-notated Data Base for learning.

# Unsupervised learning

## The quality of learning

- ▶ Internal criteria – high similarity inside the clusters and reduced similarity between elements from different clusters
  - ▶ distances inside the clusters
  - ▶ distances between the clusters
  - ▶ David-Bouldin index
  - ▶ Dunn index
- ▶ External criteria – using benchmark data sets that are previously grouped for testing
  - ▶ comparing with known data – almost impossible in real applications
  - ▶ precision
  - ▶ recall
  - ▶ F-measure

## Machine Learning

Problem Definition  
and Applications

Design

Automatic learning

Supervised learning

Unsupervised  
learning

Active learning;  
Reinforcement  
learning

# Unsupervised learning

## Internal criteria

### Machine Learning

Problem Definition  
and Applications

Design

Automatic learning

Supervised learning

**Unsupervised  
learning**

Active learning;  
Reinforcement  
learning

- ▶ distances inside the cluster  $c_j$  that has  $n_j$  instances
  - ▶ average distance between points  

$$D_a(c_j) = \sum_{x_{i_1}, x_{i_2} \in c_j} \|x_{i_1} x_{i_2}\| / (n_j * (n_j - 1))$$
  - ▶ average of distances between the closest two points  

$$D_{nn}(c_j) = \sum_{x_{i_1} \in c_j} \min_{x_{i_2} \in c_j} \|x_{i_1} x_{i_2}\| / n_j$$
  - ▶ distance between centroids  $D_c(c_j) = \sum_{x_i \in c_j} \|x_i \mu_j\| / n_j$ ,  
 where  $\mu_j = 1/n_j \sum_{x_i \in c_j} x_i$
- ▶ distances between two clusters  $c_{j_1}$  and  $c_{j_2}$ 
  - ▶ simple link:  $d_s(c_{j_1}, c_{j_2}) = \min_{x_{i_1} \in c_{j_1}, x_{i_2} \in c_{j_2}} \{\|x_{i_1} x_{i_2}\|\}$
  - ▶ complete link:  $d_{co}(c_{j_1}, c_{j_2}) = \max_{x_{i_1} \in c_{j_1}, x_{i_2} \in c_{j_2}} \{\|x_{i_1} x_{i_2}\|\}$
  - ▶ average link:  

$$d_a(c_{j_1}, c_{j_2}) = \sum_{x_{i_1} \in c_{j_1}, x_{i_2} \in c_{j_2}} \{\|x_{i_1} x_{i_2}\|\} / (n_{j_1} * n_{j_2})$$
  - ▶ centroid link:  $d_{ce}(c_{j_1}, c_{j_2}) = \|\mu_{j_1} \mu_{j_2}\|$

# Unsupervised learning

## Internal criteria

- ▶ David-Bouldin index → min → compact clusters

$$DB = 1/nc * \sum_{i=1}^{nc} \max_{j=\overline{1,nc}, j \neq i} ((\sigma_i + \sigma_j)/d(\mu_i, \mu_j))$$

- ▶  $nc$  - the no. of clusters,
- ▶  $\mu_i$  the centroid of cluster  $i$ ,
- ▶  $\sigma_i$  the average distance between the elements from cluster  $i$  to the centroid of the cluster,
- ▶  $d$  the distance between the centroids of two clusters

- ▶ Dunn index

- ▶ identifies compact and well separated clusters
- ▶  $D = D_{min}/D_{max}$
- ▶  $D_{min}$  minimum distance between two different clusters
- ▶  $D_{max}$  maximum distance between two elements from the same clusters

# Unsupervised learning

## Classification criteria: the clustering determination

- ▶ hierachic

- ▶ a taxonomic tree is built:
  - \* cluster creation → recursive
  - \*  $k$  is unknown (no. of clusters)
- ▶ agglomerating → from small clusters towards big ones
- ▶ divisive → from large clusters to small ones

- ▶ non-hierachic

- ▶ partitioning → dividing the data set → all clusters simultaneous
- ▶ optimizes an objective function (local or global defined)
  - ▶ square error, graph based, probabilistic models, closest neighbour
- ▶  $k$  is predefined (no. of clusters)

# Unsupervised learning

Classification criteria: the clustering determination

## Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
**Unsupervised  
learning**

Active learning;  
Reinforcement  
learning

- ▶ based on data density
  - ▶ the clusters' creation is based on the data density in some region
  - ▶ the clusters' creation is based on the data connectivity in some region
  - ▶ one attempt to model the data distribution
  - ▶ advantage: the clusters can have any shape
- ▶ grid based
  - ▶ not exactly a new method
  - ▶ the space is segmented in regular zones
  - ▶ the objects are placed in a multidimensional grid (ex: ACO)

# Unsupervised learning

Classification criteria: typology

- ▶ considering the algorithm behaviour
  - ▶ agglomerating
    - 1 every element is considered a cluster
    - 2 compute the distance between each two clusters
    - 3 the closest two clusters merge
    - 4 repeat steps 2 and 3 until a stop condition
  - ▶ divisive
    - 1 consider randomly from all elements centers for  $k$  clusters
    - 2 divide all the elements to the  $k$  clusters
    - 3 recompute the new centers for the clusters
    - 4 repeat steps 2 and 3 until the algorithm converges (no changes in the elements distribution occurred)
- ▶ considering the attributes
- ▶ considering the type of membership of elements to clusters
  - hard clustering versus fuzzy clustering

## Machine Learning

Problem Definition and Applications

Design

Automatic learning

Supervised learning

Unsupervised learning

Active learning;  
Reinforcement learning

## Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
Unsupervised  
learning

Active learning;  
Reinforcement  
learning

## Active learning

The algorithm receives supplementary information during learning faze in order to improve its performances.

## Reinforcement learning

The main feature is the interaction with the environment  
(actions lead to rewards)

# Automatic learning

Classification criteria: the learned model

- ▶ Least Squares method
- ▶ Descending Gradient methods
- ▶ Evolutionary algorithms
- ▶ kNN
- ▶ Decision Trees
- ▶ Support vector machines
- ▶ Artificial neural networks
- ▶ Genetic programming
- ▶ Hidden Markov Models

Machine Learning

Problem Definition  
and Applications  
Design

Automatic learning  
Supervised learning  
Unsupervised  
learning

Active learning;  
Reinforcement  
learning

## Machine Learning

Problem Definition  
and Applications

Design

Automatic learning

Supervised learning

Unsupervised  
learningActive learning;  
Reinforcement  
learning

# Least Squares method

Consider a regression problem:

- ▶ input data  $x \in R^d$
- ▶ output data  $y \in R$
- ▶ determine a model  $f$  that maps  $x$  in  $y$
- ▶  $f(x) = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \dots + \beta_d * x_d$

we will define a function:

$$\text{Loss} = \sum_{i=1,d} (y_i - f(x_i))^2$$

if we minimize it we obtain the optimal values for  $\beta$

## Modeling the $\beta$ coefficients

- ▶ iteration 0: random values (or zero)
- ▶ iteration  $t + 1$ :

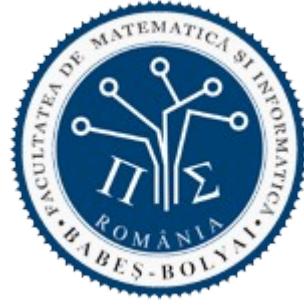
$$\beta_k(t+1) = \beta_k(t) - \text{learning rate} * \text{error}(t) * x, \text{ where } k = \overline{1, d}$$

$$\beta_0(t+1) = \beta_0(t) - \text{learning rate} * \text{error}(t),$$

where  $\text{error} = \text{computed} - \text{realOutput}$



BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics



# ARTIFICIAL INTELLIGENCE

**Intelligent systems**

Machine learning

Support Vector Machines

K-means

# Useful information

---

- Chapter 15 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter 9 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- Documents from *svm* folder

# Intelligent systems – Machine Learning

---

## □ Typology

- Experience criteria:

- Supervised learning
  - Unsupervised learning
  - Active learning
  - Reinforcement learning

- Algorithm criteria

- Decision trees
  - Artificial Neural Networks
  - Evolutionary Algorithms
  - **Support Vector Machines**
  - Hidden Markov Models
  - K-means

# Intelligent Systems – Support Vector Machines

---

## ❑ Support Vector Machines (SVMs)

- Definition
- Solved problems
- Advantages
- Difficulties
- Tools

# Intelligent Systems – Support Vector Machines

---

## □ Definition

- Developed by Vapnik in 1970
- Popularised after 1992
- Linear classifiers that identify the hyper-plane that separates the positive and negative classes
- Have a theoretical foundation
- Work very well for large data (text mining, image analysis)

## ■ Remember

- Supervised learning problem – a data set:
  - $(x^d, t^d)$ , with:
  - $x^d \in \mathbf{R}^m \rightarrow x^d = (x_{1^d}, x_{2^d}, \dots, x_{m^d})$
  - $t^d \in \mathbf{R} \rightarrow t^d \in \{1, -1\}$ , 1 → positive class, -1 → negative class
  - where  $d = 1, 2, \dots, n, n+1, n+2, \dots, N$
- First  $n$  data ( $x^d$  and  $t^d$  are known) are used as training data
- Last  $N-n$  data ( $x^d$  is known,  $t^d$  is unknown) are used as testing data

# Intelligent Systems – Support Vector Machines

---

## □ Definition

- SVM finds a linear function  $f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$ , ( $\mathbf{w}$  -weight vector) such as

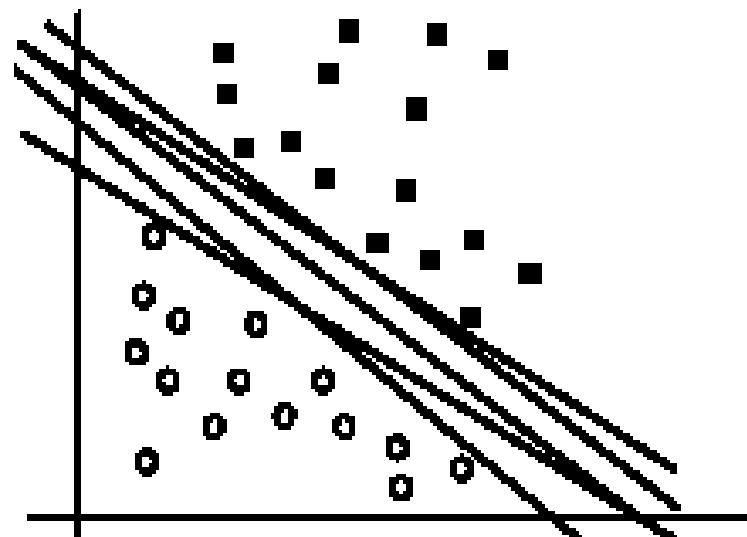
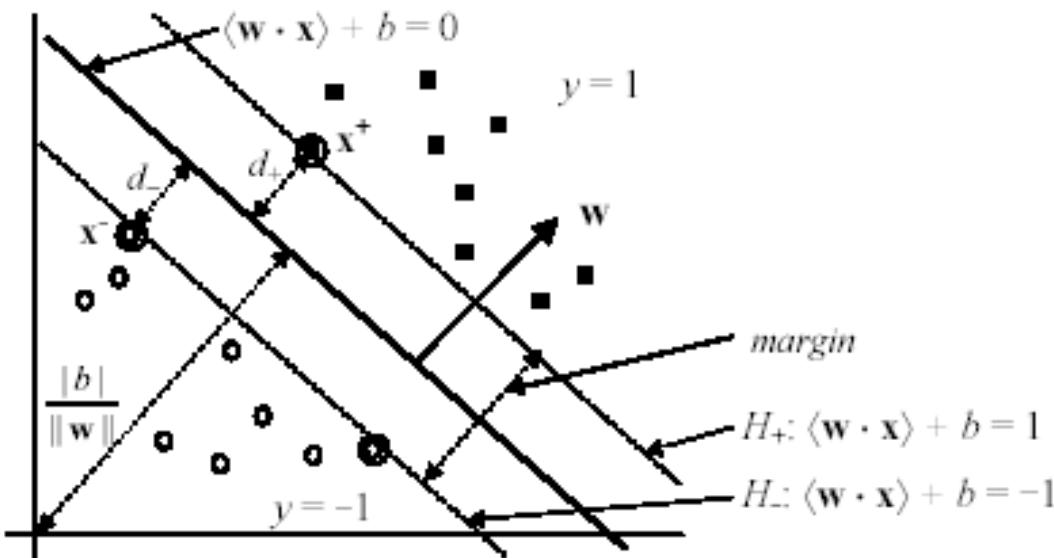
$$y_i = \begin{cases} 1 & \text{if } \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1 & \text{if } \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases}$$

- $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0 \rightarrow$  decision hyper-plane that separates the two classes

# Intelligent Systems – Support Vector Machines

## □ Definition

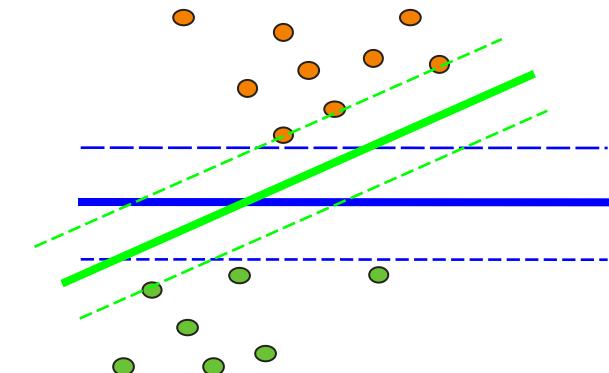
- There are more hyper-planes
  - Which is the best hyper-plane?
- SVM searches the hyper-plane with the largest margin (that minimises the generalisation error)
  - SMO (*Sequential minimal optimization*) algorithm



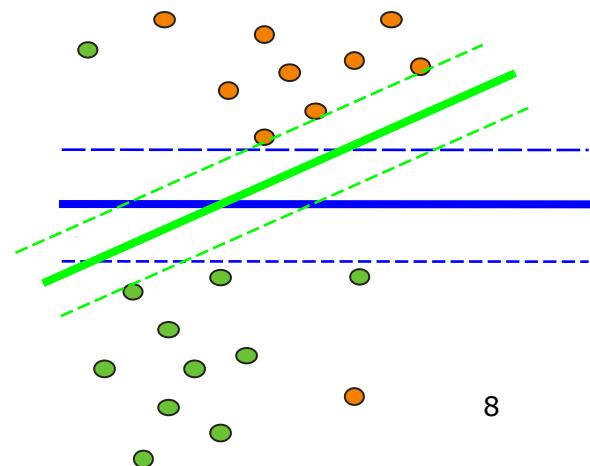
# Intelligent Systems – Support Vector Machines

## ❑ Solved problems

- Classification problems → more cases  
(based on the data type):
  - ❑ Linear separable

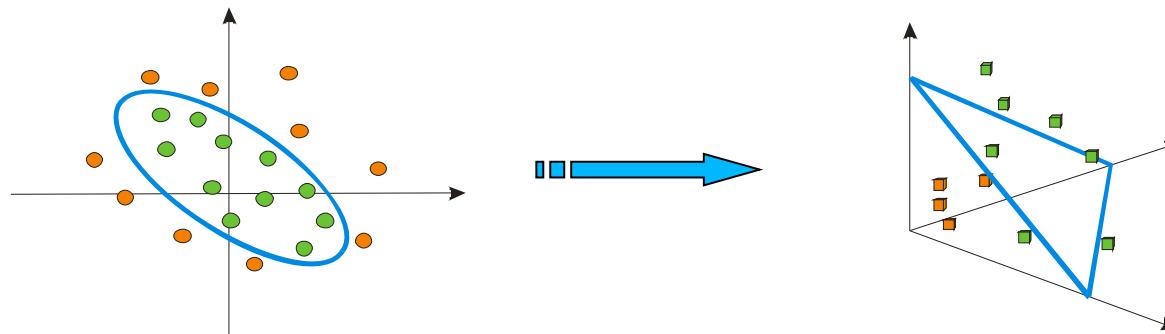


- Separable
  - Error = 0
- Non-separable
  - Constraints are relaxed → some error are allowed
  - C – penalisation coefficient



# Intelligent Systems – Support Vector Machines

- Solved problems → classification problems → data cases:
  - Non-linear separable
    - Input space is transformed (mapped) into a space of more dimensions (feature space) by using kernel function – in this new space the data becomes linear separable
    - In SVMs the kernel function computes the distance among 2 points
      - → kernel ~ similarity function



# Intelligent Systems – Support Vector Machines

---

- Solved problems → classification problems → data cases:

- Non-linear separable → possible kernels
  - Classic kernels
    - Polynomial kernel:  $K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = (\mathbf{x}^{d1}, \mathbf{x}^{d2} + 1)^p$
    - RBF kernel:  $K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = \exp(- ||\mathbf{x}^{d1} - \mathbf{x}^{d2}||^2 / 2\sigma^2)$
  - Multiple Kernels
    - Linear :  $K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = \sum w_i K_i(\mathbf{x}^{d1}, \mathbf{x}^{d2})$
    - Non-linear
      - Without coefficients:  $K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = K_1(\mathbf{x}^{d1}, \mathbf{x}^{d2}) + K_2(\mathbf{x}^{d1}, \mathbf{x}^{d2}) * \exp(K_3(\mathbf{x}^{d1}, \mathbf{x}^{d2}))$
      - With coefficients:  $K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = K_1(\mathbf{x}^{d1}, \mathbf{x}^{d2}) + c_1 * K_2(\mathbf{x}^{d1}, \mathbf{x}^{d2}) \exp(c_2 + K_3(\mathbf{x}^{d1}, \mathbf{x}^{d2}))$
  - Kernels for strings
  - Kernels for images
  - Kernels for graphs

# Intelligent Systems – Support Vector Machines

---

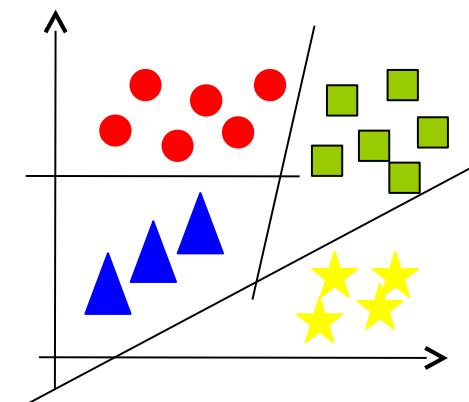
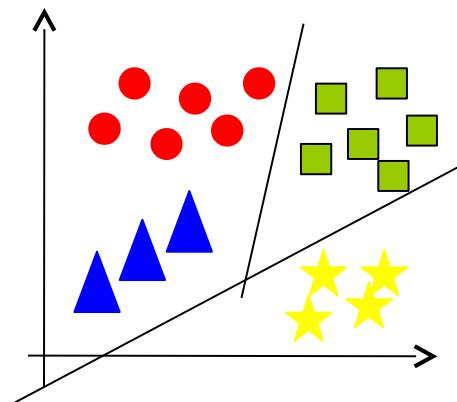
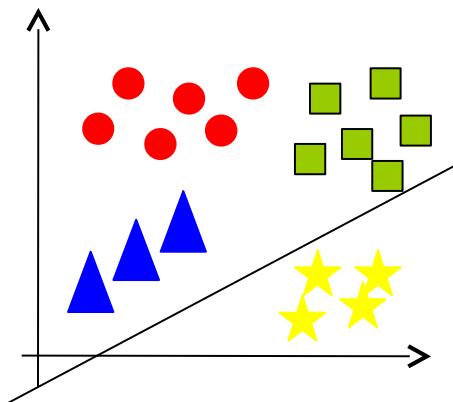
## ❑ SVM setting

### ■ SVM's parameters

- ❑ Penalisation coefficient C
  - C- small → slowly convergence
  - C – large → fast convergence
- ❑ Kernel parameters (what kernel and what parameters)
  - If m (# of attributes) is larger than n (# of data)
    - SVM by a linear kernel (SVM without kernel) →  
 $K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = \mathbf{x}^{d1} \cdot \mathbf{x}^{d2}$
  - If m (# of attributes) is large and n (# of data) is medium
    - SVM with Gaussian kernel  $K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = \exp(-||\mathbf{x}^{d1} - \mathbf{x}^{d2}||^2/2\sigma^2)$
    - $\sigma$  – dispersion of training data
    - Attributes must be normalised (scalled to (0,1))
  - If m (# of attributes) is small and n (# of data) is large
    - Ad new attributes and than SVM with linear kernel

# Intelligent Systems – Support Vector Machines

- ❑ SVM for multi-class classification problems (more than 2 classes)
  - one vs. all



# Intelligent Systems – Support Vector Machines

## □ Structured SVMs

### ■ Machine Learning

- Simple SVM  $f: \mathcal{X} \rightarrow \mathbb{R}$

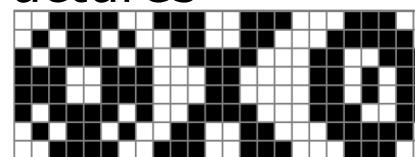
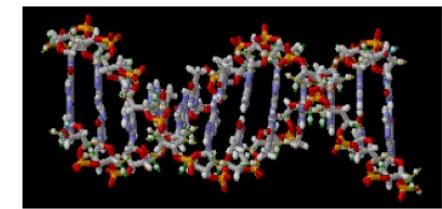
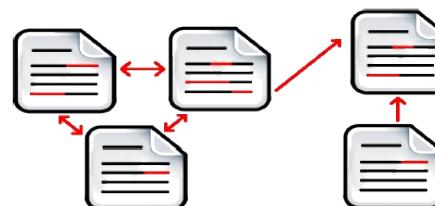
- Any type of inputs
  - Numerical outputs (natural numbers, integers, real numbers)

- Structured SVM:  $\mathcal{X} \rightarrow \mathcal{Y}$

- Any type of inputs
  - Any type of outputs (numerical or structured outputs)

### ■ Structured information

- Texts and hyper-texts
- Molecules and molecular structures
- Images



# Intelligent Systems – Support Vector Machines

---

## ■ Structured SVMs

### ■ Applications

- Natural Language Processing
  - Automatic translation (outputs → sentences)
  - Syntactic and/or morphologic analysis of sentences (outputs → syntactic and/or morphologic tree)
- Bioinformatic
  - Prediction of secondary structures (outputs → bi=partite graphs)
  - Prediction of enzyme function (outputs → paths in trees)
- Speech processing
  - Automatic transcriptions (outputs → sentences)
  - Transformation of texts in voice (outputs → audio signal)
- Robotics
  - Planning (outputs → sequences of actions)

# Intelligent Systems – Support Vector Machines

---

## ❑ Advantages

- Can work with any type of data (linear or non-linear separable, uniform distributed or not, with known or unknown distribution)
  - Kernel function that creates new attributes (features) → hidden layers of an ANN
- If the problem is convex SVM finds a unique solution → global optima
  - ANNs can associate more solutions → local optima
- Automatic selection of the learnt model (by support vectors)
  - In ANNs hidden layers have to be configured a priori
- Avoid over fitting
  - ANNs have over fitting problems even the cross-validation is involved

## ❑ Difficulties

- Real attributes only
- Binary classification problems only
- Difficult mathematical background

## ❑ Tools

- LibSVM → <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- Weka → SMO
- SVMLight → <http://svmlight.joachims.org/>
- SVMTorch → <http://www.torch.ch/>
- <http://www.support-vector-machines.org/>

# Intelligent systems – Machine Learning

---

## □ Typology

- Experience criteria:

- Supervised learning
  - Unsupervised learning
  - Active learning
  - Reinforcement learning

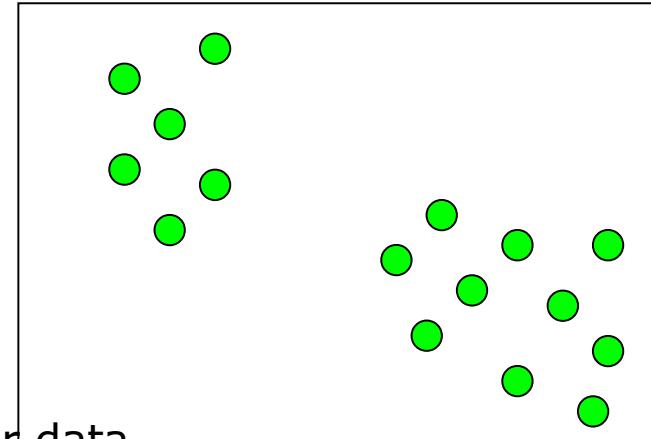
- Algorithm criteria

- Decision trees
  - Artificial Neural Networks
  - Evolutionary Algorithms
  - Support Vector Machines
  - Hidden Markov Models
  - **K-means**

# Unsupervised learning

---

- Aim
  - Finds a model or a structure of data
- Solved problems
  - Identification of groups (clusters)
    - Analysis of genes
    - Image processing
    - Analysis of social networks
    - Market segmentation
    - Analysis of astronomic data
    - Clusters of computers
  - Dimension reduction
  - Identification of causes (explanations) for data
  - Modelling the data densities
- Specific
  - Data are not annotated (labelled)



# Unsupervised learning – definition

---

Separates the un-labeled examples in disjoint sub-sets (clusters) such as:

- ❑ Examples of the same cluster are similar
- ❑ Examples of different clusters are different

## Definition

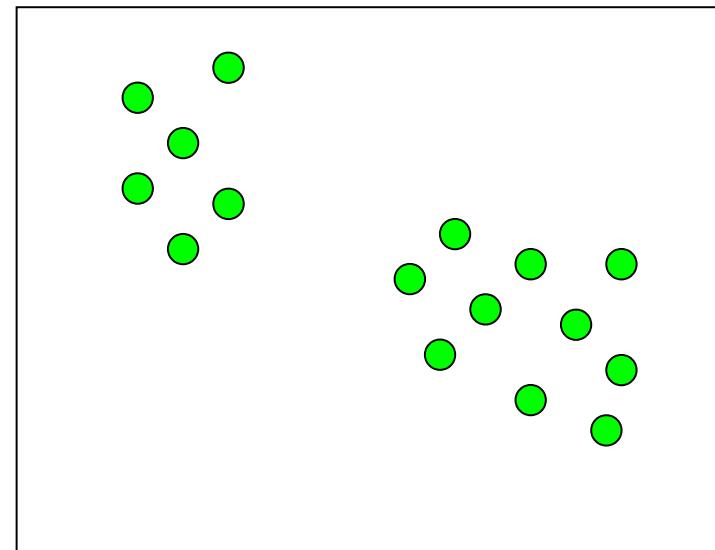
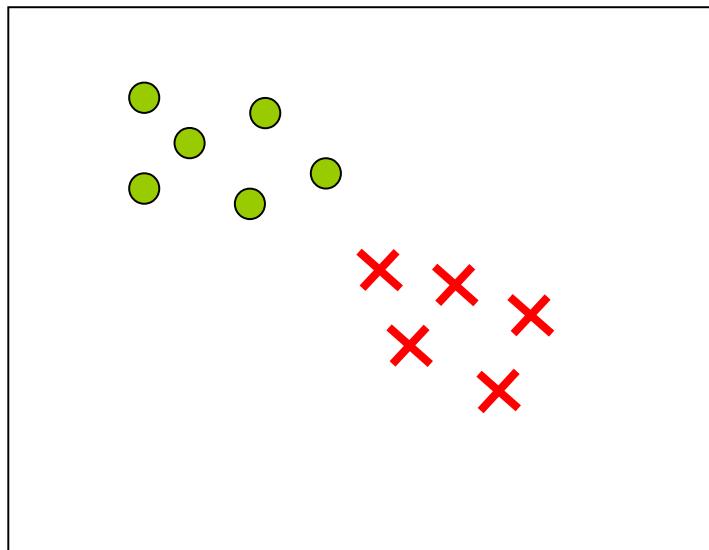
- ❑ Given
  - A set of data (examples, instances, cases)
    - ❑ Training data
      - As **attribute\_data<sub>i</sub>**, where
        - $i = 1, N$  ( $N = \#$  of training data)
        - **attribute\_data<sub>i</sub>** =  $(atr_{i1}, atr_{i2}, \dots, atr_{im})$ ,  $m = \#$  of attributes (characteristics, properties) of data
    - ❑ Testing data
      - As  $(attribute\_data_i), i = 1, n$  ( $n = \#$  of testing data)
- ❑ Determine
  - An unknown function that groups the training data in more classes
    - ❑ # of classes can be pre-defined ( $k$ ) or un-known
    - ❑ Data of the same class are similar
  - The class of a new testing data by using the learnt grouping (on training data)

## Other names

- ❑ Clustering

# Unsupervised learning – definition

## □ Supervised vs. un-supervised



# Unsupervised learning – definition

---

- Distance between 2 elements  $\mathbf{p}$  and  $\mathbf{q} \in R^m$ 
  - Euclid distance
    - $d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{j=1,2,\dots,m} (p_j - q_j)^2}$
  - Manhattan distance
    - $d(\mathbf{p}, \mathbf{q}) = \sum_{j=1,2,\dots,m} |p_j - q_j|$
  - Mahalanobis distance
    - $d(\mathbf{p}, \mathbf{q}) = \sqrt{(\mathbf{p} - \mathbf{q}) S^{-1} (\mathbf{p} - \mathbf{q})^T}$ ,
      - Where S is the covariance matrix ( $S = E[(\mathbf{p} - E[\mathbf{p}])(\mathbf{q} - E[\mathbf{q}])^T]$ )
  - Internal product
    - $d(\mathbf{p}, \mathbf{q}) = \sum_{j=1,2,\dots,m} p_j q_j$
  - Cosine distance
    - $d(\mathbf{p}, \mathbf{q}) = \sum_{j=1,2,\dots,m} p_j q_j / (\sqrt{\sum_{j=1,2,\dots,m} p_j^2} * \sqrt{\sum_{j=1,2,\dots,m} q_j^2})$
  - Hamming distance
    - # of differences between  $\mathbf{p}$  and  $\mathbf{q}$
  - Levenshtein distance
    - Minimal # of operations required for transforming  $\mathbf{p}$  in  $\mathbf{q}$
- Distance vs. similarity
  - Distance → minimisation
  - Similarity → maximisation

# Unsupervised learning – definition

## Application

- Gene clustering
- Market segmentation (for client clustering)
- news.google.com

# Unsupervised learning – process

## Process

- ❑ 2 steps:
  - Learning
    - ❑ Determine (learn), by using an algorithm, the existing clusters
  - Testing
    - ❑ Include a new data in one of the identified (during training) clusters

## Learning quality (clustering validation)

- ❑ Internal criteria
  - Large similarity inside the cluster and reduce similarity between clusters
- ❑ External criteria
  - Using benchmarks composed of *apriori* grouped data

# Unsupervised learning – evaluation

---

## Performance measures

### □ Internal criteria

- Distance inside the cluster
- Distance between clusters
- Davies-Bouldin index
- Dunn index

### □ External criteria

- Comparison with known data – impossible in real-world applications
- Precision
- Recall
- F-measure

# Unsupervised learning – typology

- How the clusters are forming
  - Hierarchic clustering
  - Non-hierarchic (partitioned) clustering
  - Clustering based on data density
  - Clustering based on a grid

# Unsupervised learning – typology

- How the clusters are forming
  - Hierarchic clustering
    - Creates a dendrogram (taxonomic tree)
      - Creates the clusters (recursively)
      - $k$  (# of clusters) is un-known
    - Agglomerativ clustering (bottom-up) → small clusters to large clusters
    - Divisiv clustering (top-down) → large clusters to small clusters
    - Eg.
      - Clustering ierarhic aglomerativ

# Unsupervised learning – typology

---

- How the clusters are formed
  - Non-hierarchic
    - Partition → determine a data separation → all the clusters in the same time
    - Optimises an objective function defined
      - Locally – by using some features only
      - Globally – by using all attributes
  - that can be:
    - squared error – sum of squared distances between data and the cluster's centroid → min
      - Ex. *K-means*
    - Graph-based
      - Ex. Clustering based in minimum spanning tree
    - Based on probabilistic models
      - Ex. Identify the data distribution → expectation maximisation
    - Based on the nearest neighbour
- Required to fix k apriori → fix the initial clusters
  - Algorithm is run more times with different parameters and the most efficient version is selected
- Ex. *K-means, ACO*

# Unsupervised learning – typology

- How the clusters are forming
  - Based on data densities
    - Data density and data connectivity
      - Cluster formation is based on data density from a given region
      - Cluster formation is based on data connectivity from a given region
    - Function of data density
      - Tries to model the data distribution
    - Advantage:
      - Modeling of clusters of any shape

# Unsupervised learning – typology

- How the clusters are forming
  - Based on a grid
    - Is not a distinct approach
      - Can be hierachic, partitional or density-based
    - Involves data space segmentation in regular areas
    - Objects are placed on a multi-dimensional grid
    - Eg. ACO

# Unsupervised learning – typology

---

- How the algorithms work
  - Agglomerative clustering
    1. Initially, each instance form a cluster
    2. Compute the distance between any 2 clusters
    3. Reunion the closest 2 clusters
    4. Repeat steps 3 and 4 until a single cluster is obtained or other stop criterion is satisfied
  - Divisive clustering
    1. Establish the number of clusters ( $k$ )
    2. Initialise the centre of each cluster
    3. Determine a data separation
    4. Re-compute the center of each cluster
    5. Repeat steps 3 and 4 until the partition is unchanged (algorithm converges)
- How the algorithm takes into account the attributes (features)
  - Monotonic – attributes are taken into account one-by-one
  - Poly-tonic – attributes are simultaneous taken into

# Unsupervised learning – typology

## □ How the data belong to clusters

### ■ Exact clustering (hard clustering)

- Each instance  $x_i$  has associated a label (class)  $c_j$

### ■ Fuzzy clustering

- Each instance has associated a membership degree (probability)  $f_{ij}$  to a given class  $c_j \rightarrow$  an instance  $x_i$  can belongs to more clusters
- Asociază fiecarei intrări  $\mathbf{x}_i$  un grad (probabilitate) de apartenență  $f_{ij}$  la o anumită clasă  $c_j \rightarrow$  o instanță  $\mathbf{x}_i$ , poate apartine mai multor clusteri

# Unsupervised learning – algorithms

- Agglomerative hierarchical clustering
- K-means
- AMA
- Probabilistic models
- Nearest neighbor
- Fuzzy
- Artificial Neural Network
- Evolutionary algorithms
- ACO

# Unsupervised learning – algorithms

## Agglomerate hierarchical clustering

- Consider a distance between 2 instances  
 $d(x_{i1}, x_{i2})$
- Form  $N$  clusters, each of them containing an instance
- Repeat
  - Determine the closest 2 clusters
  - Reunion the 2 clusters → a cluster
- Until a single cluster is obtained (that contains all the instances)

# Unsupervised learning – algorithms

---

## Agglomerate hierarchical clustering

- Distance between 2 clusters  $c_i$  and  $c_j$ :
  - Simple link → minimal distance between the objects of 2 clusters
    - $d(c_i, c_j) = \max_{x_{i1} \in c_i, x_{i2} \in c_j} sim(\mathbf{x}_{i1}, \mathbf{x}_{i2})$
  - Complete link → maximal distance between the objects of 2 clusters
    - $d(c_i, c_j) = \min_{x_{i1} \in c_i, x_{i2} \in c_j} sim(\mathbf{x}_{i1}, \mathbf{x}_{i2})$
  - Average link → mean of distances between the objects of 2 clusters
    - $d(c_i, c_j) = 1 / (n_i * n_j) \sum_{x_{i1} \in c_i} \sum_{x_{i2} \in c_j} d(\mathbf{x}_{i1}, \mathbf{x}_{i2})$
  - Average link over group → distance between the means (centroids) of 2 clusters
    - $d(c_i, c_j) = \rho(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j), \rho - distance, \boldsymbol{\mu}_j = 1/n_j \sum_{x_{iecf}} \mathbf{x}_i$

# Unsupervised learning – algorithms

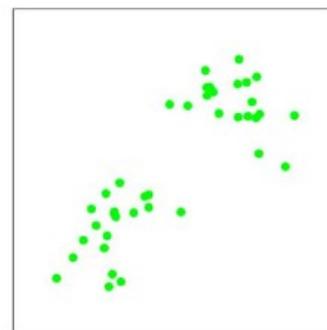
---

K-means (Lloyd algorithm / Voronoi iteration)

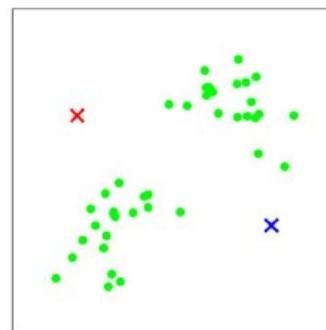
- Suppose that  $k$  clusters will form
- Initialize  $k$  centroids  $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_k$ 
  - A centroid  $\boldsymbol{\mu}_j$  ( $j=1, 2, \dots, k$ ) is a vector of  $m$  values ( $m$  - # of features)
- Repeat until convergence
  - Associated to each instance the nearest centroid  $\rightarrow$  for each instance  $\mathbf{x}_i$ ,  $i = 1, 2, \dots, N$ 
    - $c_i = \arg \min_{j=1, 2, \dots, k} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2$
  - Re-compute the centroids by moving them in the mean of instances associated to it  $\rightarrow$  for each cluster  $c_j$ ,  $j = 1, 2, \dots, k$ 
    - $$\boldsymbol{\mu}_j = \frac{\sum_{i=1, 2, \dots, N} \mathbf{x}_i}{\sum_{i=1, 2, \dots, N} 1_{c_i=j}}$$

# Unsupervised learning – algorithms

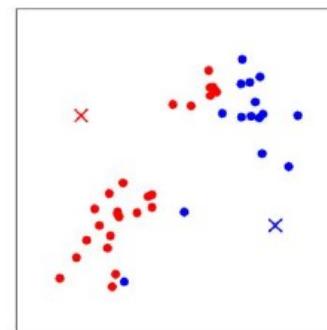
## □ K-means



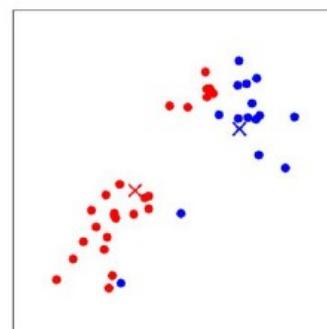
(a)



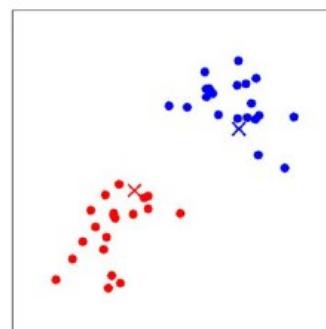
(b)



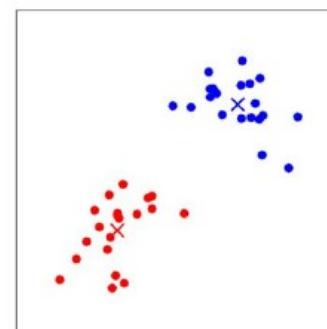
(c)



(d)



(e)



(f)

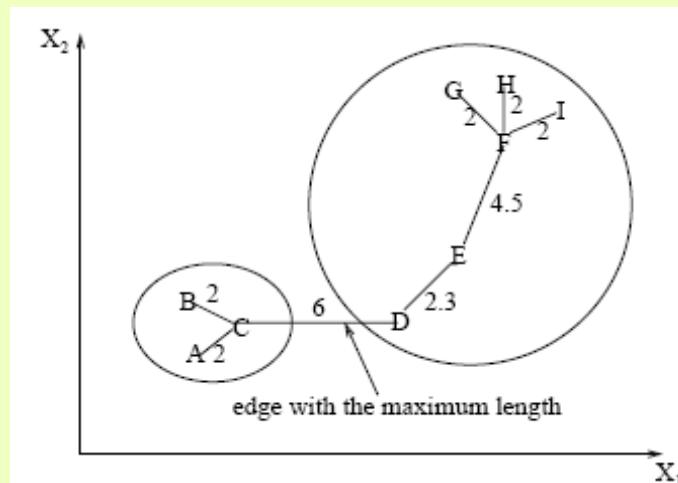
# Unsupervised learning – algorithms

## K-means

- Initialisation of  $k$  centroids  $\mu_1, \mu_2, \dots, \mu_k$ 
  - With random values (in the definition domain of the problem)
  - With  $k$  instances of  $N$  (randomly selected)
- Does algorithm converge always?
  - Yes, because of distortion function  $J$ 
    - $J(c, \mu) = \sum_{i=1,2,\dots,N} ||\mathbf{x}_i - \mu_{c_i}||^2$  which is decreasing
  - Converges in a local optima
  - Finding the global optima  $\rightarrow$  NP-difficult problem

## Clustering based on minimum spanning tree

- ❑ Construct the minimum spanning tree of data
- ❑ Eliminate from the tree the longest edges and form clusters



# Învățare ne-supervizată – algoritmi

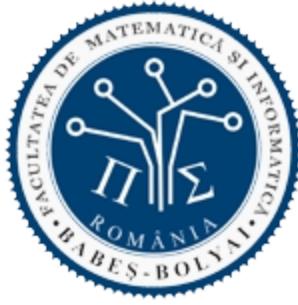
## Modele probabilistice

---

- <http://www.gatsby.ucl.ac.uk/~zoubin/courses04/ul.pdf>
- <http://learning.eng.cam.ac.uk/zoubin/nipstudent.pdf>



BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics



# ARTIFICIAL INTELLIGENCE

**Intelligent systems**

Machine learning

Artificial Neural Networks

# Topics

---

- A. Short introduction in Artificial Intelligence (AI)
- B. Solving search problems
  - A. Definition of search problems
  - B. Search strategies
    - A. Uninformed search strategies
    - B. Informed search strategies
    - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
    - D. Adversarial search strategies
- C. Intelligent systems
  - A. Rule-based systems in certain environments
  - B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
  - C. Learning systems
    - A. Decision Trees
    - B. Artificial Neural Networks**
    - C. Support Vector Machines
    - D. Evolutionary algorithms
  - D. Hybrid systems

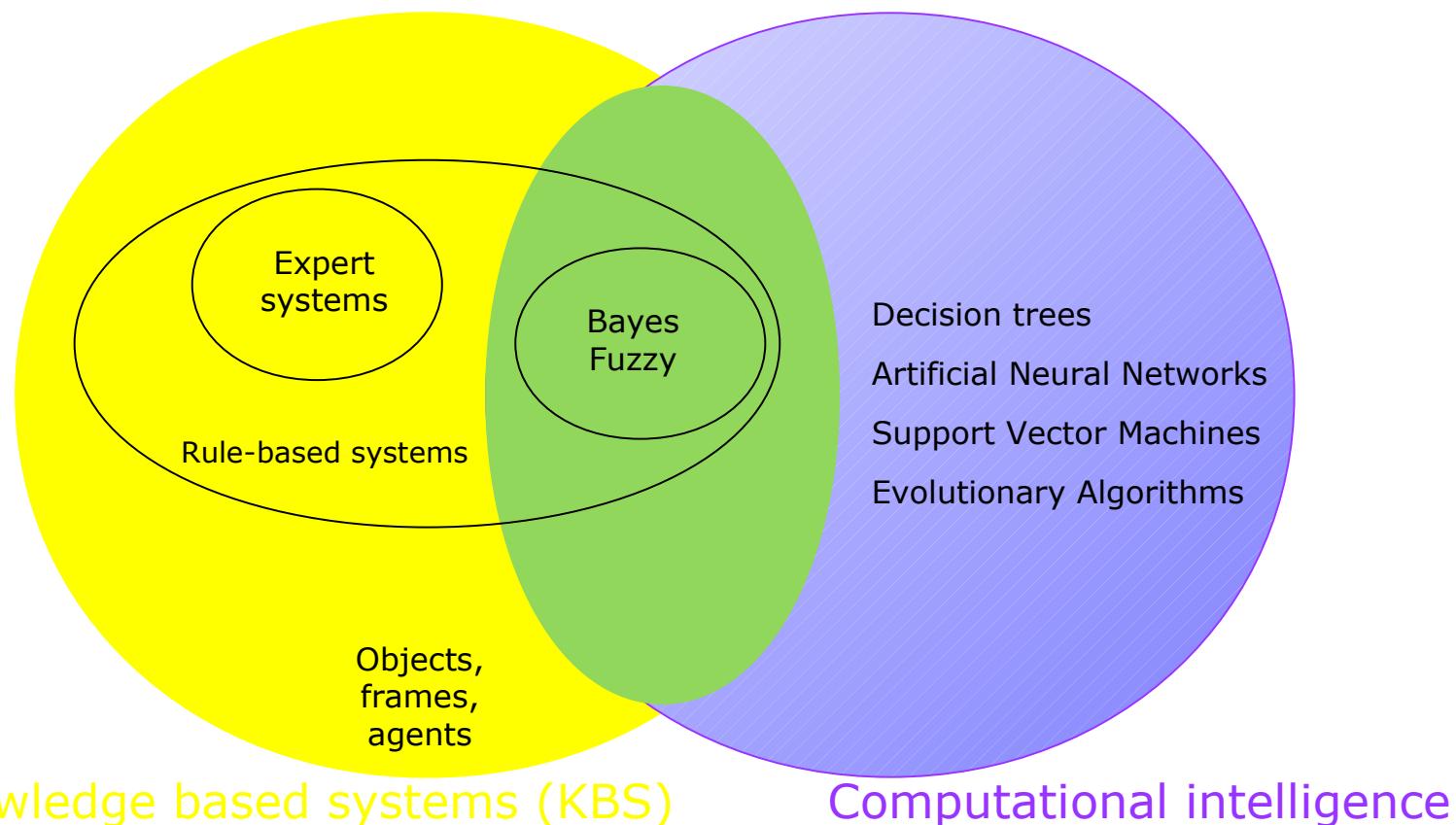
# Useful information

---

- Chapter VI (19) of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 8 of *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapters 12 and 13 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter V of *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Chapter 4 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

# Intelligent systems

---



# Intelligent systems – Machine Learning

---

## □ Typology

- Experience criteria:
  - Supervised learning
  - Unsupervised learning
  - Active learning
  - Reinforcement learning
  
- Algorithm criteria
  - Decision trees
  - **Artificial Neural Networks**
  - Evolutionary Algorithms
  - Support Vector Machines
  - Hidden Markov Models

# Intelligent Systems - ANNs

---

## □ Artificial Neural Networks (ANNs)

- Aim
- Definition
- Solved problems
- Characteristics
- Example
- Design
- Evaluation
- Typology

# Intelligent Systems - ANNs

---

## □ Aim

- Binary classification for any input data (discrete or continuous)

- Data can be separated by:

- A line  $\rightarrow ax + by + c = 0$  (if m = 2)
    - A plan  $\rightarrow ax + by + cz + d = 0$  (if m = 3)
    - A hyperplan  $\sum a_i x_i + b = 0$  (if m > 3)

- How do we identify the optimal values of a, b, c, d,  $a_i$ ?
    - Artificial Neural Networks (ANNs)
    - Support Vector Machines (SVMs)

- Why ANN?

- Who does the brain learn?

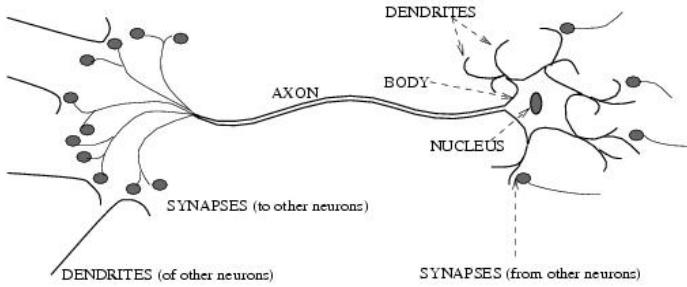
# Intelligent Systems - ANNs

---

- Aim → why ANN?
  - Some tasks can be easily done by humans, but they are difficult to be encoded as algorithms
    - Shape recognition
      - Old friends
      - Handwritten
      - Voice
    - Rational processes
      - Car driving
      - Piano playing
      - Basketball playing
      - Swimming
  - Such tasks are too difficult to be formalized and done by a rational process

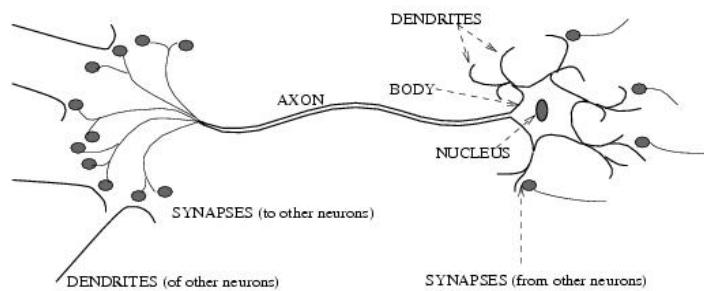
# Intelligent Systems - ANNs

- Aim → how does the brain learn
  - Human brain – components
    - ~10.000.000.000 of neurons connected through **synapses**
    - Each neuron
      - Has a body (soma), an axon and more dendrites
      - Can be in a given state
        - Active state – if the input information is over a given stimulation threshold
        - Passive state – otherwise
  - Synapse
    - Link between the axon of a neuron and the dendrites of other neurons
    - Take part to information exchange between neurons
    - 5.000 connections/neuron (average)
  - During a life, new connections can appear



# Intelligent Systems - ANNs

- Aim → how does the brain learn?
  - How does learning (information processing) take place?
    - ▣ Useful connections become permanent (others are eliminated)
    - ▣ The brain is interested about news
    - ▣ Models for information processing
      - Learning
      - Storing
      - Memorizing
  - ▣ Memory
    - Typology
      - Short time memory
      - Immediately → 30 sec.
      - Working memory
      - Long term memory
    - Capacity
      - Increasing along life
      - Limited → learning a poetry strophe by strophe
    - Influenced by emotional states
  - ▣ Brain
    - Neuron network
    - Complex system, non-linear and parallel that processes information
  - ▣ Information is stored and processed by the entire network, not only by a part of network → global information and processing
  - ▣ Basic characteristic of a neural network → learning → artificial neural network



# Intelligent Systems - ANNs

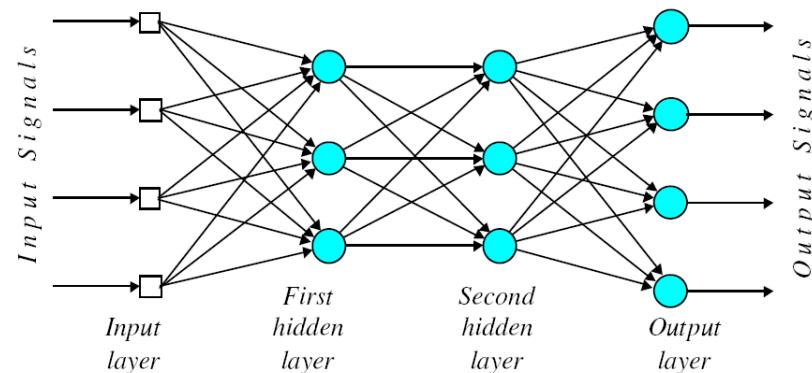
---

## □ Definition

- What is an ANN?
- Biological NN vs. artificial NN
- How does network learn?

# Intelligent Systems - ANNs

- Definition → what is an ANN?
  - A structure similar to a biological NN
  - A set of nodes (units, neurons, processing elements) located in a graph with more layers
    - Nodes
      - Have inputs and outputs
      - Perform a simple computing through an activation function
      - Connected by weighted links
        - Links between nodes give the network structure
        - Links influence the performed computations
    - Layers
      - Input layer
        - Contains  $m$  nodes ( $m$  - # of attributes of a data)
      - Output layer
        - Contains  $r$  nodes ( $r$  - # of outputs)
      - Intermediate layers
        - Different structures
        - Different sizes



# Intelligent Systems - ANNs

## □ Definition → biological NNs vs. artificial NNs

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

## □ Definition → how does network learn?

- A training data set of n data

$$((x_{p1}, x_{p2}, \dots, x_{pm}, y_{p1}, y_{p2}, \dots, y_{pr},))$$

with  $p = 1, 2, \dots, n$ , m – #attributes, r – #outputs

- Form an ANN with m input nodes, r output nodes and an internal structure
  - Some hidden layers, each layer having a given number of nodes
  - With weighted connections between every 2 nodes of consecutive layers
- Determine the optimal weights by minimising the error
  - Difference between the real output y and the output computed by the network

# Intelligent Systems - ANNs

---

- Problems solved by an ANN
  - Problem data can be represented by pairs (attribute-value)
  - Objective function can be:
    - Single or multi-criteria
    - Discrete or continuous (real values)
  - Training data can be noised
  - A large training time

# Intelligent Systems - ANNs

---

## □ Design

- ANN construction (for solving problem P)
- Initialisation of ANN's parameters
- ANN training
- ANN testing

# Intelligent Systems - ANNs

---

## □ Design

### ■ ANN construction (for solving problem P)

#### □ For a classification problem:

- $(x^d, t^d)$ , cu:
- $x^d \in \mathbf{R}^m \rightarrow x^d = (x_{1}^d, x_{2}^d, \dots, x_m^d)$
- $t^d \in \mathbf{R}^R \rightarrow t^d = (t_{1}^d, t_{2}^d, \dots, t_R^d)$ ,
- with  $d = 1, 2, \dots, n, n+1, n+2, \dots, N$

#### □ First n data are used for training

#### □ Last N-n data are used for testing

#### □ Construct the ANN:

- Input layer has m nodes (each node reads an attribute of an input data –  $x_{1}^d, x_{2}^d, \dots, x_m^d$ )
- Output layer can contain R nodes (each node provides an output attribute –  $t_{1}^d, t_{2}^d, \dots, t_R^d$ )
- One or more hidden layers with one or more neurons on each layer

# Intelligent Systems - ANNs

---

## □ Design

- ANN construction (for solving problem P)
- **Initialisation of ANN's parameters**
- **ANN training**
- ANN testing

# Intelligent Systems - ANNs

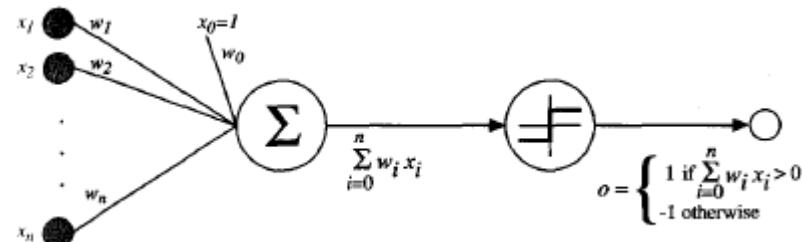
---

## □ Design

- Initialisation of ANN's parameters
  - Initialisation of weights (between nodes from consecutive layers)
  - Establish the activation function of each neuron of hidden layers
- ANN's training
  - Aim
    - Identify the optimal weights
  - Algorithm
    - Search the optimal weights by minimising the errors (difference between the real output and the computed output)
  - How does ANN learn?
    - Network = set of primitive computational units
      - Network learning =  $\cup$  primitive unit learning
    - Primitive computational units
      - Perceptron
      - Linear unit
      - Sigmoidal unit

# Intelligent Systems - ANNs

- Design → ANN's training → How does ANN learn?
  - Neuron as a simple computing element
    - Neuron structure
      - Each node has inputs and outputs
      - Each node performs a simple computation
    - Neuron processing
      - Information is transmitted to the neuron
      - Neuron processes the information
      - The answer of neuron is read
    - Neuron learning – algorithm of learning the weights that correctly process the information
      - Start by some initial weights
      - While not stopCondition
        - Process the information and establish the quality of current weights
        - Modify the weights such to obtain better results



# Intelligent Systems - ANNs

---

- Design → ANN's training → How does ANN learn?
  - Neuron as simple computing element
    - Neuron structure
      - Each node has inputs and outputs
      - Each node performs a simple computation through an activation function
    - Neuron processing  $net = \sum^n x_i w_i$ 
      - Information is transmitted to the neuron → compute the weighted sum of inputs
      - Neuron processes the information → by using an activation function
        - Constant function
        - Step function
        - Slope function
        - Linear function
        - Sigmoid function
        - Gaussian function

# Intelligent Systems - ANNs

## □ Design → ANN's training → How does ANN learn?

### ■ Activation function of a neuron

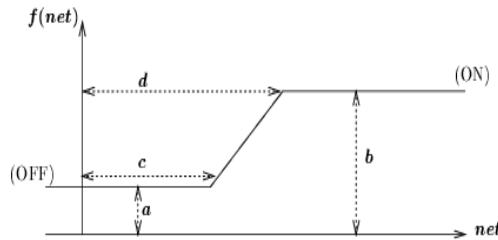
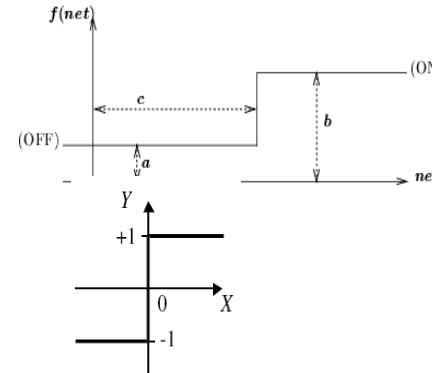
- Constant function  $f(\text{net}) = \text{const}$
- Step function ( $c$  - threshold)

$$f(\text{net}) = \begin{cases} a, & \text{if } \text{net} < c \\ b, & \text{if } \text{net} > c \end{cases}$$

- For  $a=+1$ ,  $b=-1$  and  $c=0 \rightarrow$  sign function
- Discontinuous function

### □ Slope function

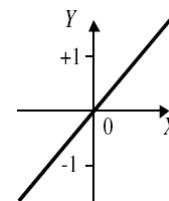
$$f(\text{net}) = \begin{cases} a, & \text{if } \text{net} \leq c \\ b, & \text{if } \text{net} \geq d \\ a + \frac{(\text{net}-c)(b-a)}{d-c}, & \text{otherwise} \end{cases}$$



### □ Funcția liniară

$$f(\text{net}) = a * \text{net} + b$$

- For  $a = 1$  and  $b = 0 \rightarrow$  identity function  $f(\text{net})=\text{net}$
- Continuous function



# Intelligent Systems - ANNs

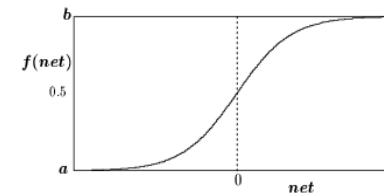
## □ Design → ANN's training → How does ANN learn?

### ■ Activation function of a neuron

#### □ Sigmoid function

- Shape of S
- Continuous and differentiable
- Rotational symmetric to a given point ( $net = c$ )

$$\lim_{net \rightarrow -\infty} f(net) = a \quad \lim_{net \rightarrow \infty} f(net) = b$$



- Examples:

$$f(net) = z + \frac{1}{1 + \exp(-x \cdot net + y)}$$

$$f(net) = \tanh(x \cdot net - y) + z$$

$$\text{where } \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

- for  $y=0$  and  $z = 0 \Rightarrow a=0, b = 1, c=0$
- For  $y=0$  and  $z = -0.5 \Rightarrow a=-0.5, b = 0.5, c=0$
- The  $x$  is greater, the curve is steeper

# Intelligent Systems - ANNs

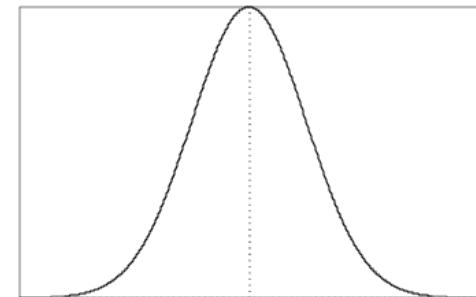
---

## □ Design → ANN's training → How does ANN learn?

### ■ Activation function of a neuron

- Gaussian function
  - Bell shape
  - Continuous

$$\lim_{net \rightarrow \infty} f(net) = a$$



- Has a single optimum point (maximum) – for net =  $\mu$
- Example

$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{net - \mu}{\sigma}\right)^2\right]$$

# Intelligent Systems - ANNs

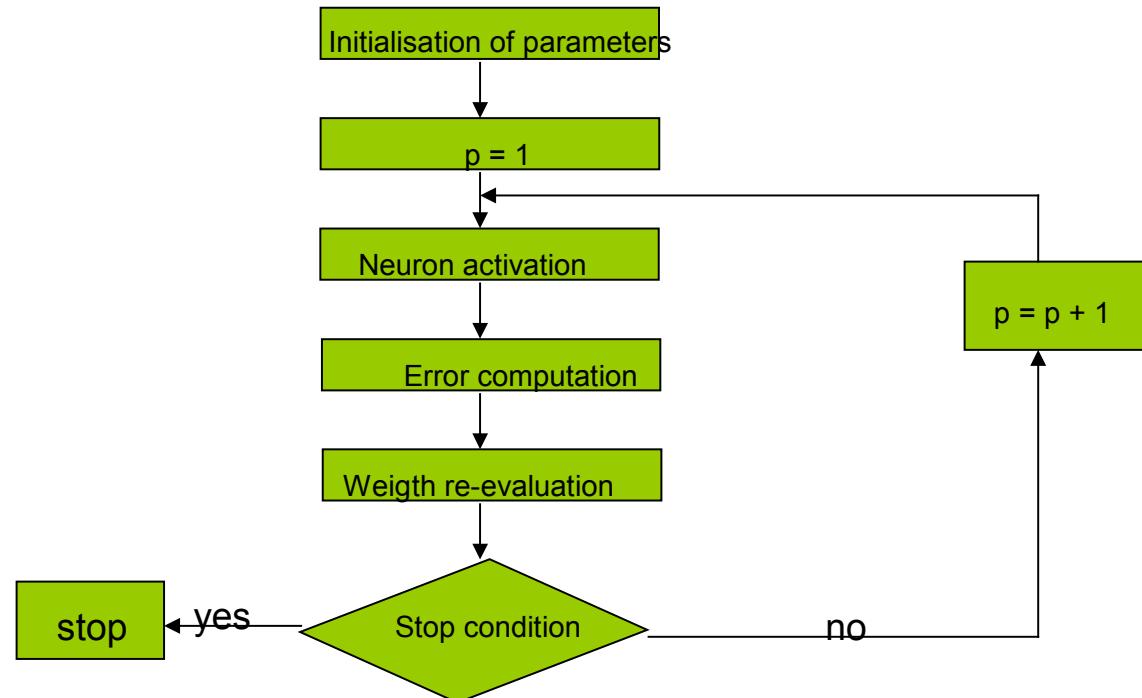
---

- Design → ANN's training → How does ANN learn?
  - Neuron as simple computation element
    - Neuron structure
    - Neuron processing
      - Information is transmitted to the neuron → compute the weighted sum of inputs
$$net = \sum_{i=1}^n x_i w_i$$
  - Neuron processes the information → using an activation function
    - Constant function
    - Step function
    - Slope function
    - Linear function
    - Sigmoid function
    - Gaussian function
  - Read the neuron's answer → determine if the computed result is the same with the real one

# Intelligent Systems - ANNs

- Design → ANN's training → How does ANN learn?

- Neuron as simple computation element
  - Neuron structure
  - Neuron processing
  - Neuron learning
    - Algorithm



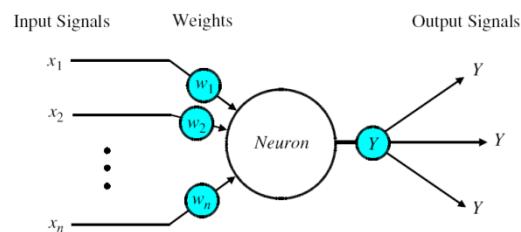
# Intelligent Systems - ANNs

---

- Design → ANN's training → How does ANN learn?
  - Neuron learning
    - 2 basic rules
      - Perceptron's rule → perceptron's algorithm
        1. Start by some random weights
        2. Determine the quality of the model create for these weights for a single input data
        3. Re-compute the weights based on the model's quality
        4. Repeat (from step 2) until a maximum quality is obtained
      - Delta's rule → algorithm of gradient descent
        1. Start by some random weights
        2. Determine the quality of the model create for these weights for all input data
        3. Re-compute the weights based on the model's quality
        4. Repeat (from step 2) until a maximum quality is obtained
      - Similar to perceptron's rule, but the model's quality is established based on all data (all training data)

# Intelligent Systems - ANNs

- Design → ANN's training → How does ANN learn?
  - Neuron learning
    - Suppose we have a train data set:
      - $(x^d, t^d)$ , cu:
        - $x^d \in \mathbb{R}^m \Rightarrow x^d = (x_{1^d}, x_{2^d}, \dots, x_{m^d})$
        - $t^d \in \mathbb{R}^R \Rightarrow t^d = (t_{1^d}, t_{2^d}, \dots, t_{R^d})$ , and  $R = 1 (\Rightarrow t^d = (t_{1^d}))$
        - With  $d = 1, 2, \dots, n$
      - ANN = elementary computational primitive (a neuron) → a network:
        - $m$  output nodes
        - linked to the computing neuron through weights  $w_i$ ,  $i = 1, 2, \dots, m$  and
        - an output node



# Intelligent Systems - ANNs

---

- Design → ANN's training → How does ANN learn?
  - Neuron learning
    - Perceptron's algorithm
      - Based on error minimisation associated to an instance of train data
      - Modify the weights based on error associated to an instance of train data

Initialisation of network weights

$$w_i = \text{random}(a,b), \text{ where } i=1,2,\dots,m$$

$$d = 1$$

While there are incorrect classified examples

    Activate the neuron and determine the output

    Perceptron → sign activation function

$$o^d = \text{sign}(\mathbf{wx}) = \text{sign}\left(\sum_{i=1}^m w_i x_i\right)$$

$$\Delta w_i = \eta(t^d - o^d)x_i^d, \text{ unde } i = 1,2,\dots,m$$

Determine the weight modification

$$\text{where } \eta - \text{learning rate}$$
$$w_i^* = w_i + \Delta w_i$$

Modify the weights

if  $d < n$  then  $d++$

otherwise  $d = 1$

EndWhile

# Intelligent Systems - ANNs

- Design → ANN's training → How does ANN learn?
  - Neuron learning
    - Gradient descent algorithm
      - Based on the error associated to the entire set of train data
      - Modify the weights in the direction of the steepest slope of error reduction  $E(\mathbf{w})$  for the entire set of train data
    - How the steepest slope is determined? → derive  $E$  based on  $w$  (establish the gradient of error  $E$ )
$$\nabla E(\mathbf{w}) = \left( \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_m} \right)$$
    - Error's gradient is computed based on activation function of neuron (that must be differentiable → continuous)
      - Linear function  $f(\text{net}) = \sum_{i=1}^m w_i x_i^d$
      - Sigmoid function  $f(\text{net}) = \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}}} = \frac{1}{1 + e^{-\sum_{i=1}^m w_i x_i^d}}$
    - How the weights are modified?

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \text{ where } i = 1, 2, \dots, m$$

# Intelligent Systems - ANNs

- Design → ANN's training → How does ANN learn?
  - Neuron learning
    - Descent gradient algorithm → error's gradient computation
      - Linear function

$$f(\text{net}) = \sum_{i=1}^m w_i x_i^d$$
$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d=1}^n (t^d - o^d)^2 = \frac{1}{2} \sum_{d=1}^n \frac{\partial (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n 2(t^d - o^d) \frac{\partial (t^d - o^d)}{\partial w_i}$$
$$\frac{\partial E}{\partial w_i} = \sum_{d=1}^n (t^d - o^d) \frac{\partial (t^d - w_1 x_1^d - w_2 x_2^d - \dots - w_m x_m^d)}{\partial w_i} = \sum_{d=1}^n (t^d - o^d)(-x_i^d)$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d=1}^n (t^d - o^d) x_i^d$$

- Sigmoid function

$$f(\text{net}) = \frac{1}{1 + e^{-\text{wx}}} = \frac{1}{1 + e^{-\sum_{i=1}^m w_i x_i^d}}$$

$$y = s(z) = \frac{1}{1 + e^{-z}} \Rightarrow \frac{\partial s(z)}{\partial z} = s(z)(1 - s(z))$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d=1}^n (t^d - o^d)^2 = \frac{1}{2} \sum_{d=1}^n \frac{\partial (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n 2(t^d - o^d) \frac{\partial (t^d - \text{sig}(\text{wx}^d))}{\partial w_i} = \sum_{d=1}^n (t^d - o^d)(1 - o^d)o^d(-x_i^d)$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d=1}^n (t^d - o^d)(1 - o^d)o^d x_i^d$$

# Intelligent Systems - ANNs

- Design → ANN's training → How does ANN learn?
  - Neuron learning
    - Gradient descent algorithm (GDA)

Simple GDA	Stochastic GDA
<p>Initialisation of network weights <math>w_i = \text{random}(a,b)</math>, where <math>i=1,2,\dots,m</math></p> <p>While not stop condition</p> <p><math>\Delta w_i = 0</math>, unde <math>i=1,2,\dots,m</math></p> <p>For each train example <math>(x_d, t_d)</math>, where <math>d=1,2,\dots,n</math></p> <ul style="list-style-type: none"><li>Activate the neuron and determine the output <math>o_d</math></li><li>Linear activation <math>\rightarrow o_d = wx_d</math></li><li>Sigmoid activation <math>\rightarrow o_d = \text{sig}(wx_d)</math></li></ul> <p>For each weight <math>w_i</math>, where <math>i=1,2,\dots,m</math></p> <ul style="list-style-type: none"><li>Determine the weight modification</li></ul> $\Delta w_i = \Delta w_i - \eta \frac{\partial E}{\partial w_i}$ <p>where <math>\eta</math> - learning rate</p> <p>For each weight <math>w_i</math>, where <math>i=1,2,\dots,m</math></p> <ul style="list-style-type: none"><li>Modify the weights <math>w_i</math></li></ul> $w_i = w_i + \Delta w_i$ <p>EndWhile</p>	<p>Initialisation of network weights <math>w_i = \text{random}(a,b)</math>, where <math>i=1,2,\dots,m</math></p> <p>While not stop condition</p> <p><math>\Delta w_i = 0</math>, unde <math>i=1,2,\dots,m</math></p> <p>For each train example <math>(x_d, t_d)</math>, where <math>d=1,2,\dots,n</math></p> <ul style="list-style-type: none"><li>Activate the neuron and determine the output <math>o_d</math></li><li>Linear activation <math>\rightarrow o_d = wx_d</math></li><li>Sigmoid activation <math>\rightarrow o_d = \text{sig}(wx_d)</math></li></ul> <p>For each weight <math>w_i</math>, where <math>i=1,2,\dots,m</math></p> <ul style="list-style-type: none"><li>Determine the weight modification</li></ul> $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$ <p>where <math>\eta</math> - learning rate</p> <p>Modify the weights <math>w_i</math></p> $w_i = w_i + \Delta w_i$ <p>EndWhile</p>

# Intelligent Systems - ANNs

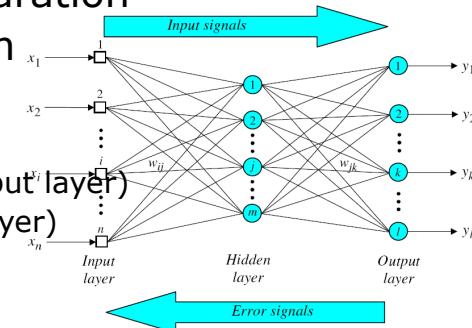
Design → ANN's training → How does ANN learn?

Neuron learning

Differences	Perceptron's algorithm	Gradient descent algorithm (delta rule)
What does $o^d$ represent?	$o^d = \text{sign}(\mathbf{w}\mathbf{x}^d)$	$o^d = \mathbf{w}\mathbf{x}^d$ or $o^d = \text{sig}(\mathbf{w}\mathbf{x}^d)$
Convergence	After a finite # of steps (until the perfect separation)	Asymptotic (to minimum error)
Solved problems	With linear separable data	Any data (linear separable or non-linear)
Neuron's output	Discrete and with threshold	Continuous and without threshold

# Intelligent Systems - ANNs

- Design → ANN's training
  - Network learning
    - Network = set of primitive computational units that are interconnected →
      - Net learning = ∪ primitive learning
    - Net with neurons located on one or more layers → ANN is able of learning a complex model (not a linear one only) for data separation
    - Algorithm for learning the weights → backpropagation
      - Based on descent gradient algorithm
      - Improvements
        - Information is forward propagated (from input layer to output layer)
        - Error is backward propagated (from output layer to input layer)



Initialisation of network weights

While not stop condition

For each train example  $(x^d, t^d)$ , where  $d=1,2,\dots,n$

Activate the neuron and determine the output  $o^d$

forward propagate the information and determine the output of each neuron

Modify the weights

Establish and backward propagate the error

Establish the errors of neurons from the output layer

Backward propagate the errors in the entire network → distribute the errors on all connections of the network

Modify the weights

EndWhile

# Intelligent Systems - ANNs

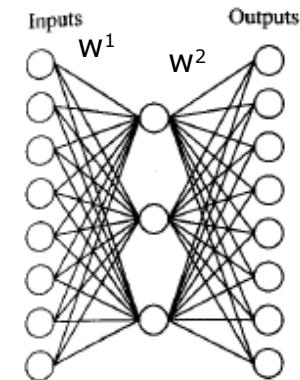
- Design → ANN's training
  - How does network learn?

- Suppose a train data:

- $(x^d, t^d)$ , with:
      - $x^d \in \mathbb{R}^m \rightarrow x^d = (x_{1^d}, x_{2^d}, \dots, x_{m^d})$
      - $t^d \in \mathbb{R}^R \rightarrow t^d = (t_{1^d}, t_{2^d}, \dots, t_{R^d})$
      - With  $d = 1, 2, \dots, n$

- Consider 2 ANNs

- An ANN with a single hidden layer with  $H$  neurons →  $\text{ANN}_1$ 
      - $m$  neurons on the input layer
      - $R$  neurons on the output layer
      - $H$  neurons on the hidden layer
      - Weights between input and hidden layers  $w_{ih_1}^1$  with  $i=1, 2, \dots, m$ ,  $h_1 = 1, 2, \dots, H$
      - Weights between hidden and output layers  $w_{h_1r}^1$  with  $h_1 = 1, 2, \dots, H$ ,  $r = 1, 2, \dots, R$
    - An ANN with  $P$  hidden layers, each layer having  $H_i$  ( $i = 1, 2, \dots, P$ ) neurons →  $\text{ANN}_p$ 
      - $m$  neurons on the input layer
      - $R$  neurons on the output layer
      - $P$  hidden layers
      - $H_p$  neurons on the  $p^{\text{th}}$  hidden layer,  $p = 1, 2, \dots, P$
      - Weights between input layer and first hidden layer  $w_{ih_1}^1$  with  $i=1, 2, \dots, m$ ,  $h_1 = 1, 2, \dots, H_1$
      - Weights between first hidden layer and second hidden layer  $w_{h_1h_2}^2$  with  $h_1 = 1, 2, \dots, H_1$ ,  $h_2 = 1, 2, \dots, H_2$
      - Weights between second hidden layer and third hidden layer  $w_{h_2h_3}^3$  with  $h_2 = 1, 2, \dots, H_2$ ,  $h_3 = 1, 2, \dots, H_3$
      - ...
      - Weights between  $(P-1)^{\text{th}}$  hidden layer and last hidden layer  $w_{h_{p-1}h_p}^{P-1}$  with  $h_{p-1} = 1, 2, \dots, H_{p-1}$  și  $h_p = 1, 2, \dots, H_p$
      - Weights between last hidden layer and output layer  $w_{h_p r}^P$  with  $h_p = 1, 2, \dots, H_p$ ,  $r = 1, 2, \dots, R$



# Intelligent Systems - ANNs

- Design → ANN's training -> How does network learn?
  - Backpropagation algorithm for ANN<sub>1</sub>

Initialisation of network weights  $w_{ih}^1$  and  $w_{hr}^2$  with  $i=1,2,\dots,m$ ,  $h = 1,2,\dots,H$ ,  $r = 1,2,\dots,R$

While not stop condition

For each train example ( $x^d, t^d$ ), where  $d=1,2,\dots,n$

Activate the neuron and determine the output  $o^d$

Forward propagate the information and determine the output of each neuron

$$o_h^d = \sum_{i=1}^m w_{ih}^1 x_i^d \text{ sau } o_h^d = \text{sig}\left(\sum_{i=1}^m w_{ih}^1 x_i^d\right), \text{ cu } h = 1,2,\dots,H$$

$$o_r^d = \sum_{h=1}^H w_{hr}^2 o_h^d \text{ sau } o_r^d = \text{sig}\left(\sum_{h=1}^H w_{hr}^2 o_h^d\right), \text{ cu } r = 1,2,\dots,R$$

Modify the weights

Establish and backward propagate the error

Establish the errors of neurons from the output layer

$$\delta_r^d = t_r^d - o_r^d \text{ or } \delta_r^d = o_r^d(1-o_r^d)(t_r^d - o_r^d), \text{ with } r = 1,2,\dots,R$$

Modify the weights between hidden layer and output layer

$$w_{hr}^2 = w_{hr}^2 + \eta \delta_r^d o_h^d, \text{ where } h = 1,2,\dots,H, r = 1,2,\dots,R$$

Backward propagate the errors in the entire network → distribute the errors on all connections of the network

$$\delta_h^d = \sum_{r=1}^R w_{hr}^2 \delta_r^d \text{ sau } \delta_h^d = o_h^d(1-o_h^d) \sum_{r=1}^R w_{hr}^2 \delta_r^d$$

Modify the weights between input layer and hidden layer

$$w_{ih}^1 = w_{ih}^1 + \eta \delta_h^d x_i^d, \text{ where } i = 1,2,\dots,m, h = 1,2,\dots,H$$

EndWhile

# Intelligent Systems - ANNs

- Design → ANN's training -> How does network learn?
  - Backpropagation algorithm for ANN<sub>P</sub>

Initialisation of network weights  $w_{ih_1}^1, w_{h_1 h_2}^2, \dots, w_{h_{p-1} h_p}^p, w_{h_p r}^{p+1}$

While not stop condition

For each train example  $(x^d, t^d)$ , where  $d=1,2,\dots,n$

Activate the neuron and determine the output

Forward propagate the information and determine the output of each neuron

$$o_{h_1}^d = \sum_{i=1}^m w_{ih_1}^1 x_i^d \quad \text{or} \quad o_{h_1}^d = \text{sig}\left(\sum_{i=1}^m w_{ih_1}^1 x_i^d\right), \text{ with } h_1 = 1, 2, \dots, H_1$$

$$o_{h_2}^d = \sum_{h_1=1}^{H_1} w_{h_1 h_2}^2 o_{h_1}^d \quad \text{or} \quad o_{h_2}^d = \text{sig}\left(\sum_{h_1=1}^{H_1} w_{h_1 h_2}^2 o_{h_1}^d\right), \text{ with } h_2 = 1, 2, \dots, H_2$$

...

$$o_{h_p}^d = \sum_{h_{p-1}=1}^{H_{p-1}} w_{h_{p-1} h_p}^p o_{h_{p-1}}^d \quad \text{or} \quad o_{h_p}^d = \text{sig}\left(\sum_{h_{p-1}=1}^{H_{p-1}} w_{h_{p-1} h_p}^p o_{h_{p-1}}^d\right), \text{ with } h_p = 1, 2, \dots, H_p$$

$$o_r^d = \sum_{h_p=1}^{H_p} w_{h_p r}^{p+1} o_{h_p}^d \quad \text{or} \quad o_r^d = \text{sig}\left(\sum_{h_p=1}^{H_p} w_{h_p r}^{p+1} o_{h_p}^d\right), \text{ with } r = 1, 2, \dots, R$$

# Intelligent Systems - ANNs

---

- Design → ANN's training -> How does network learn?
  - Backpropagation algorithm for ANN<sub>P</sub>

Initialisation of network weights  $w_{ih_1}^1, w_{h_1 h_2}^2, \dots, w_{h_{p-1} h_p}^p, w_{h_p r}^{p+1}$

While not stop condition

For each train example  $(x^d, t^d)$ , where  $d=1,2,\dots,n$

Activate the neuron and determine the output

Forward propagate the information and determine the output of each neuron

Modify the weights

Establish and backward propagate the error

Establish the errors of neurons from output layer

$$\delta_r^d = t_r^d - o_r^d \text{ or } \delta_r^d = o_r^d(1 - o_r^d)(t_r^d - o_r^d), \text{ with } r=1,2,\dots,R$$

Modify the weights between the last hidden layer and the output layer

$$w_{h_p r}^{p+1} = w_{h_p r}^{p+1} + \eta \delta_r^d o_{h_p}^d, \text{ where } h_p = 1,2,\dots,H_p, r = 1,2,\dots,R$$

# Intelligent Systems - ANNs

- Design → ANN's training -> How does network learn?
  - Backpropagation algorithm for ANN<sub>P</sub>

Initialisation of network weights       $w_{ih_1}^1, w_{h_1 h_2}^2, \dots, w_{h_{p-1} h_p}^p, w_{h_p r}^{p+1}$

While not stop condition

For each train example ( $x^d, t^d$ ), where  $d=1,2,\dots,n$

Activate the neuron and determine the output

Forward propagate the information and determine the output of each neuron

Modify the weights

Establish and backward propagate the error

Establish the errors of neurons from output layer

Modify the weights between the last hidden layer and the output layer

Backward propagate (on each layer) the errors in the entire network → distribute the errors on all connections proportional to the weights and modify the weights

$$\delta_{h_p}^d = \sum_{r=1}^R w_{h_p r}^{p+1} \delta_r^d \text{ or } \delta_{h_p}^d = o_{h_p}^d (1 - o_{h_p}^d) \sum_{r=1}^R w_{h_p r}^{p+1} \delta_r^d$$

$$w_{h_p r}^{p+1} = w_{h_p r}^{p+1} + \eta \delta_r^d o_{h_p}^d, \text{ where } h_p = 1, 2, \dots, H_p, r = 1, 2, \dots, R$$

$$\delta_{h_{p-1}}^d = \sum_{h_p=1}^{H_p} w_{h_{p-1} h_p}^p \delta_{h_p}^d \text{ or } \delta_{h_{p-1}}^d = o_{h_{p-1}}^d (1 - o_{h_{p-1}}^d) \sum_{h_p=1}^{H_p} w_{h_{p-1} h_p}^p \delta_{h_p}^d$$

$$w_{h_{p-1} h_p}^p = w_{h_{p-1} h_p}^p + \eta \delta_{h_p}^d o_{h_{p-1}}^d, \text{ where } h_{p-1} = 1, 2, \dots, H_{p-1} \text{ and } h_p = 1, 2, \dots, H_p$$

...

$$\delta_{h_1}^d = \sum_{h_2=1}^{H_2} w_{h_1 h_2}^2 \delta_{h_2}^d \text{ or } \delta_{h_1}^d = o_{h_1}^d (1 - o_{h_1}^d) \sum_{h_2=1}^{H_2} w_{h_1 h_2}^2 \delta_{h_2}^d$$

$$w_{h_1 h_2}^2 = w_{h_1 h_2}^2 + \eta \delta_{h_2}^d o_{h_1}^d, \text{ where } i = 1, 2, \dots, m, h_1 = 1, 2, \dots, H_1$$

# Intelligent Systems - ANNs

---

- Design → ANN's training -> How does network learn?
  - Backpropagation algorithm
    - Stop conditions
      - Error is 0
      - After a given number of iterations
        - During an iteration, a single example is processed
        - n iterations = an epoch

# Intelligent Systems - ANNs

---

## □ Design

- ANN construction (for solving problem P)
- Initialisation of ANN's parameters
- ANN training
- **ANN testing**

# Intelligent Systems - ANNs

---

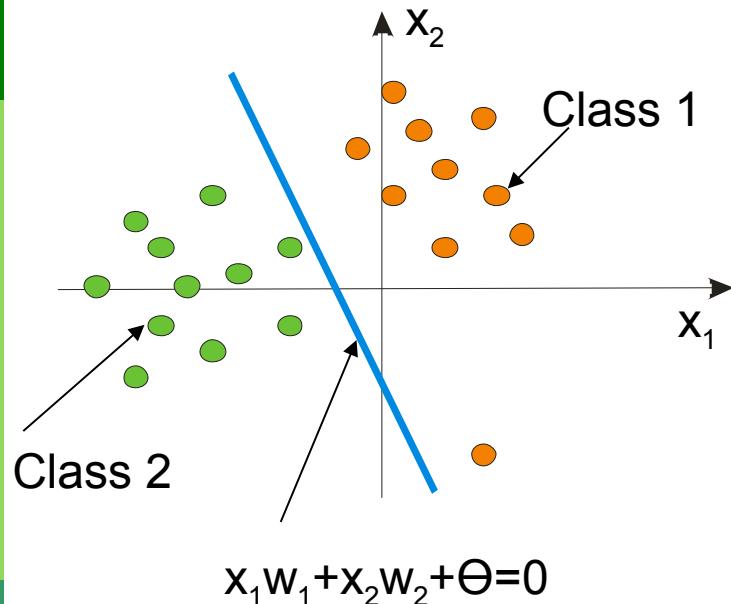
## □ Design

### ■ ANN testing

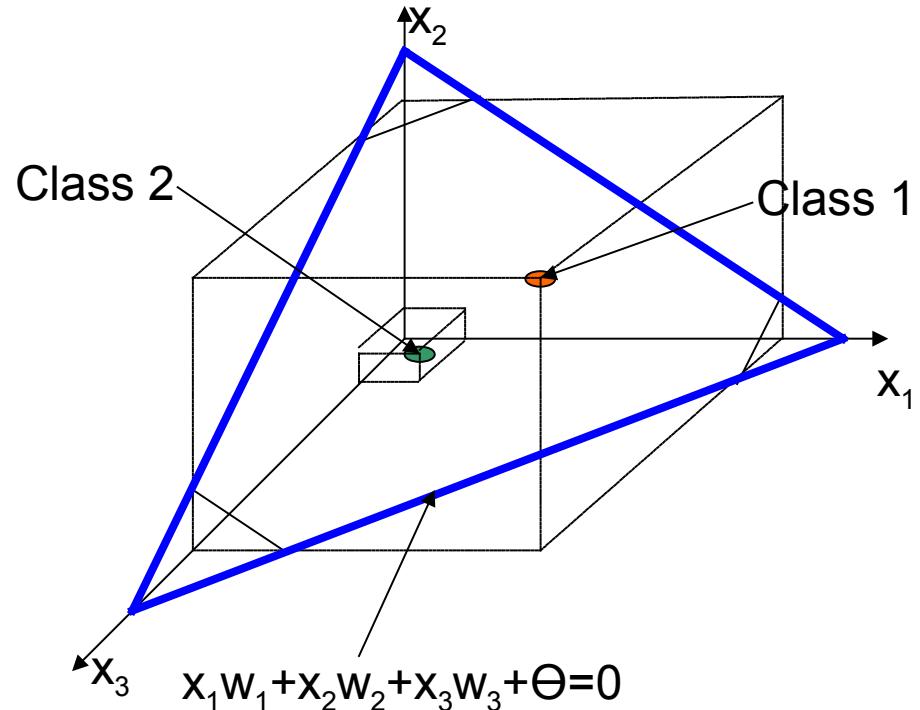
- Decode the model learnt by ANN
  - By combining the weights and inputs
  - Taking into account the activation function of neurons and the structure of the network

# Intelligent Systems - ANNs

## □ Example



Binary classification with  $m=2$   
input attributes



Binary classification with  $m=3$   
input attributes

# Intelligent Systems - ANNs

## □ Example

### ■ Perceptron for solving *logic And* problem

Epoch	Inputs		Desired output $Y_d$	Initial weights		Actual output $Y$	Error $e$	Final weights	
	$x_1$	$x_2$		$w_1$	$w_2$			$w_1$	$w_2$
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	1	1	1	0.2	-0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	-1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	-1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	-1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

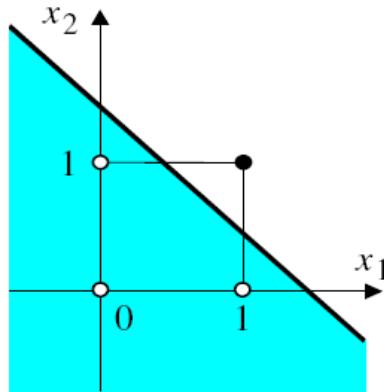
Threshold:  $\theta = 0.2$ ; learning rate:  $\alpha = 0.1$

# Intelligent Systems - ANNs

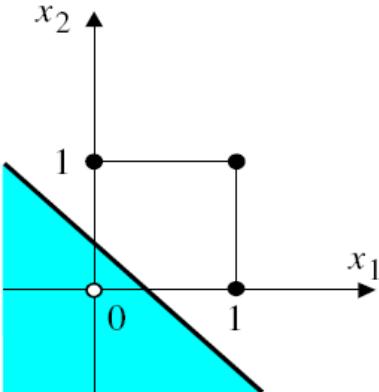
## □ Example

### ■ Perceptron – limits

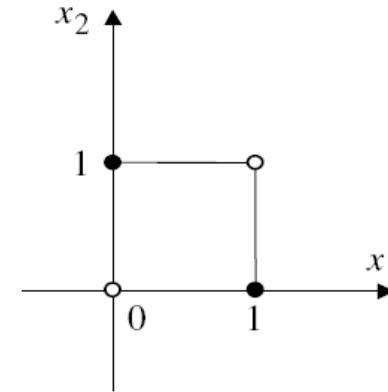
- A perceptron can learn And and OR operations, but it can not learn XOR operation (it is not linear separable)



(a)  $AND (x_1 \cap x_2)$



(b)  $OR (x_1 \cup x_2)$



(c)  $Exclusive\text{-}OR$   
 $(x_1 \oplus x_2)$

- Non-linear separable data can not be classified
  - Solutions
    - Neuron with continuous threshold
    - More neurons

# Intelligent Systems - ANNs

---

## □ Typology

- feed-forward ANNs
  - Information is processed and passed from a layer to another layer
  - Node connections do not form cycles
  - Utilised for supervised learning, especially
  - Activation function → linear, sigmoid, Gaussian
- Recurrent ANNs (with feedback)
  - Can contain connections between nodes of the same layer
  - Node connections can form cycles
  - Jordan ANNs
  - Elman ANNs
  - Hopfield ANNs
  - Self-organised ANNs → for unsupervised learning
    - Hebbian ANNs
    - Kohonen ANNs (*Self organised maps*)

} for supervised learning

# Intelligent Systems - ANNs

---

## □ Advantages

- Can solve supervised and unsupervised learning problems
- Can identify dynamic and non-linear relations among data
- Can solve multi-class classification problems
- Can compute very fast (parallel and distribute computing)

## □ Limits

- ANNs have over-fitting problems (even if cross-validation is performed)
- ANNs can find, sometimes, local optima only (without identifying the global optimum)

# Review



## □ Automatic learning systems

### ■ Artificial Neural Networks

- Computational models inspired by biological neural networks
- Special graphs with nodes located in layers
  - Input layer → read the input data of the problem
  - Output layer → provides results
  - Hidden layer(s) → perform computations
- Nodes (neurons)
  - Have weighted inputs
  - Have activation functions (linear, sigmoid, etc)
  - Require training → algorithms like:
    - Perceptron
    - Gradient descent
- Training algorithm for an entire ANN → Backpropagation
  - Information is forward propagated
  - Errors are backward propagated

# Next lecture

---

- A. Short introduction in Artificial Intelligence (AI)
- B. Solving search problems
  - A. Definition of search problems
  - B. Search strategies
    - A. Uninformed search strategies
    - B. Informed search strategies
    - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
    - D. Adversarial search strategies
- C. Intelligent systems
  - A. Rule-based systems in certain environments
  - B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
  - C. Learning systems
    - A. Decision Trees
    - B. Artificial Neural Networks
    - C. **Support Vector Machines**
    - D. Evolutionary algorithms
  - D. Hybrid systems

# Next lecture – useful information

---

- Chapter 15 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter 9 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- Documents from *12\_svm* and *13\_GP folders*

- 
- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean – [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)

# Deep learning

April 2021

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Outline

Introduction

Introduction

Tensors

Tensors

Convolution

Convolution

Kernels

Kernels

Convolutional Network - structure

Convolutional Network - structure

Convolutional Layer

Convolutional Layer

Feature learning

Feature learning  
Training the Convolution Layer

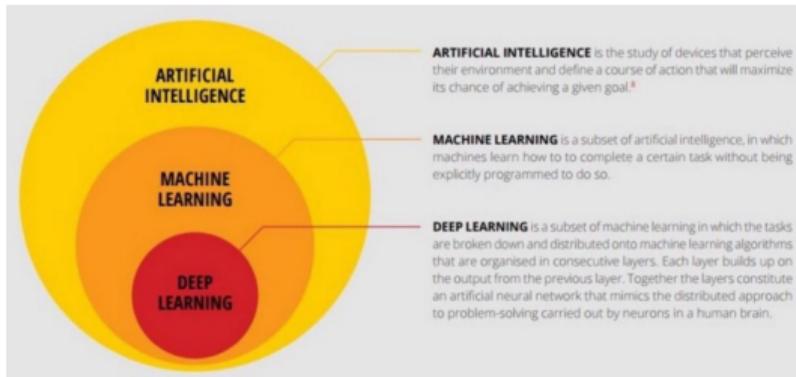
Training the Convolution Layer

Pooling Layer

Pooling Layer

# What is deep learning?

- ▶ a sub-field of machine learning dealing with algorithms inspired by the structure and function of the brain called artificial neural networks



- ▶ it mirrors the functioning of our brains
- ▶ similar to how nervous system structured where each neuron connected each other and passing information

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Deep Convolutional Neural Networks

(*ConvNets*) are deep artificial neural networks used:

- ▶ to classify images (e.g. name what they see)
- ▶ cluster them by similarity (photo search)
- ▶ perform object recognition within scenes

algorithms that can identify faces, individuals, street signs, tumors, perform optical character recognition (OCR).

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Deep Convolutional Neural Networks

- ▶ pre-processing required in a ConvNet is much lower as compared to other classification algorithms
- ▶ architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain - Visual Cortex
- ▶ a ConvNet captures the spatial and temporal dependencies in an image

Introduction

Tensors

Convolution

Kernels

Convolutional Network - structure

Convolutional Layer

Feature learning  
Training the Convolution Layer

Pooling Layer

# Tensors

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

An  $n^{th}$  - rank tensor in  $m$ -dimensional space is a mathematical object that has  $n$  indices and  $m^n$  components and obeys certain transformation rules.

- generalizations of scalars, vectors, and matrices to an arbitrary number of indices.
- used in physics such as elasticity, fluid mechanics, and general relativity.

# Notations for tensors

- ▶  $a_{ijk\dots}$ ,  $a^{ijk\dots}$ ,  $a_i^{jk\dots}$ , etc., may have an arbitrary number of indices
  - ▶ a tensor (rank  $r + s$ ) may be of mixed type  $(r, s)$ :  
 $r$  "contravariant" (upper) indices and  $s$  "covariant" (lower) indices.
- the positions of the slots in which contravariant and covariant indices are placed are significant!

$$a_{\mu\nu}^{\lambda} \neq a_{\mu}^{\nu\lambda}$$

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

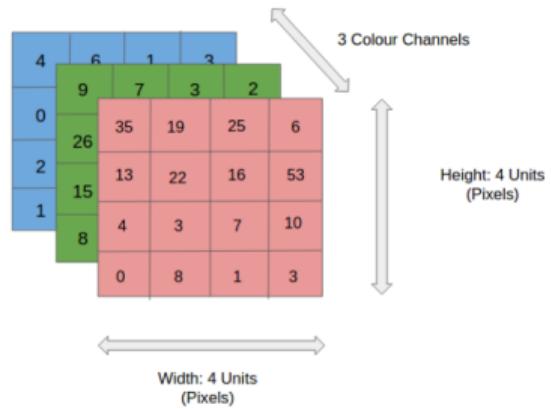
Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Images as tensors

ConvNets ingest and process images as tensors.



- reduce the images → form easy to process (without losing critical features)

Introduction  
Tensors  
Convolution  
Kernels  
Convolutional Network - structure  
Convolutional Layer  
Feature learning  
Training the Convolution Layer

Pooling Layer

# Convolution operation

an operation on two functions  $x(t)$  (**input**) and  $w(t)$  (**kernel**)  
of a real - valued argument

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

notation:

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

if  $t$  is discrete the integral turns into a sum

$$s(t) = (x \circledast w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Convolution operation

in practice we have two tensors:

- ▶ the input - a multidimensional array of data
  - ▶ the kernel - a multidimensional array of parameters  
(adapted by the learning algorithm)
- stored separately → that these functions are zero everywhere but in the finite set of points
  - we can implement the infinite summation as a summation over a finite number of array elements
  - convolutions can be over more than one axis at a time

$$S(i, j) = (I \circledast K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

Introduction

Tensors

Convolution

Kernels

Convolutional Network - structure

Convolutional Layer

Feature learning  
Training the Convolution Layer

Pooling Layer

# Using a kernel

we want apply a filter over the image

AIM: **extract the high-level features** (edges, color, gradient orientation)

Example: *Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)*

Kernel / filter:

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Using a kernel

Introduction

Tensors

Convolution

Kernels

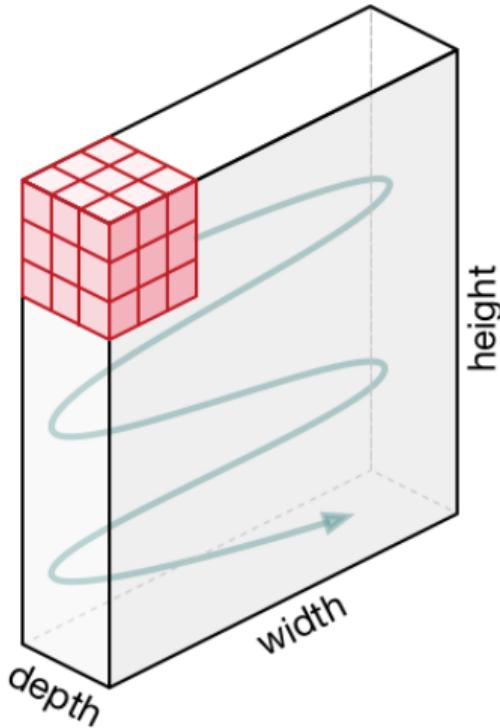
Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Using a Kernel - movement of the Kernel



- ▶ kernel shifts
- ▶ every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering

Introduction  
Tensors  
Convolution  
Kernels  
Convolutional Network - structure  
Convolutional Layer  
Feature learning  
Training the Convolution Layer  
Pooling Layer

# Using a kernel - 3 channels

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Layers of a convolutional network

Typical three stages:

- ▶ first: several convolutions
- ▶ second: a nonlinear activation function (ex: rectified linear activation function) - **detector stage**
- ▶ third: **pooling** function to modify the output

Examples: max pooling (Zhou and Chellappa, 1988 - maximum output within a rectangular neighborhood), the average of a rectangular neighborhood,  $L^2$  norm of a rectangular neighborhood, a weighted average based on the distance from the central pixel.

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Training the network

- ▶ similar with ANN
- ▶ after computing the error, a gradient descent method is applied to all layers

Introduction

Tensors

Convolution

Kernels

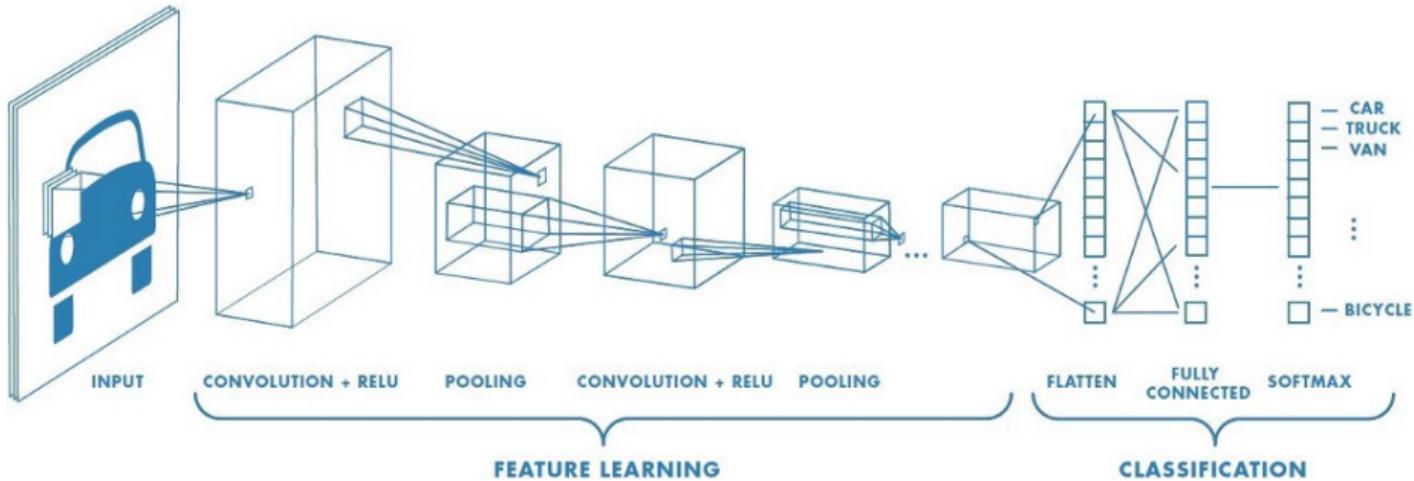
Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Feature learning



# Feature learning

Applying this filter to an image will result in a feature map that only contains vertical lines. It is a vertical line detector.

0.0	1.0	0.0
0.0	1.0	0.0
0.0	1.0	0.0

Table: A 3x3 element filter for detecting vertical lines

Dragging this filter systematically across pixel values in an image can only highlight vertical line pixels.

# Feature learning

0.0	0.0	0.0
1.0	1.0	1.0
0.0	0.0	0.0

Table: A horizontal line detector

- ▶ combining the filters' results (combining both feature maps, will result in all of the lines in an image being highlighted)
- ▶ a suite of tens or even hundreds of other small filters can be designed to detect other features in the image
- ▶ the values of the filter are weights to be learned during the training of the network

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Feature learning

training under gradient descent

- ▶ the network will learn what types of features to extract from the input
- ▶ the extracted features will be those that minimize the loss function (e. g. extract features that are the most useful for classifying images as dogs or cats)

Introduction

Tensors

Convolution

Kernels

Convolutional Network - structure

Convolutional Layer

Feature learning  
Training the Convolution Layer

Pooling Layer

# Gradient descent on Convolution Layer

- ▶ the input  $X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}$
- ▶ the filter  $F = \begin{bmatrix} f_{1,1} & f_{1,2} \\ f_{2,1} & f_{2,2} \end{bmatrix}$
- ▶ the output is  $O = X \circledast F$
- ▶  $\frac{\partial E}{\partial O}$  is the gradient of loss from previous layer

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

# Gradient descent on Convolution Layer

[Introduction](#)[Tensors](#)[Convolution](#)[Kernels](#)[Convolutional Network - structure](#)[Convolutional Layer](#)[Feature learning  
Training the Convolution Layer](#)[Pooling Layer](#)

After we apply the convolution on  $X$  and  $F$  we have:

$$O_{1,1} = x_{1,1} * F_{1,1} + x_{1,2} * F_{1,2} + x_{2,1} * F_{2,1} + x_{2,2} * F_{2,2}$$

$$O_{1,2} = x_{1,2} * F_{1,1} + x_{1,3} * F_{1,2} + x_{2,2} * F_{2,1} + x_{2,3} * F_{2,2}$$

$$O_{2,1} = x_{2,1} * F_{1,1} + x_{2,2} * F_{1,2} + x_{3,1} * F_{2,1} + x_{3,2} * F_{2,2}$$

$$O_{2,2} = x_{2,2} * F_{1,1} + x_{2,3} * F_{1,2} + x_{3,2} * F_{2,1} + x_{3,3} * F_{2,2}$$

# Gradient descent on Convolution Layer

we compute the partial derivative of  $O$  with respect of  $F$

$$\frac{\partial O_{1,1}}{\partial F_{1,1}} = x_{1,1}, \quad \frac{\partial O_{1,1}}{\partial F_{1,2}} = x_{1,2}, \quad \frac{\partial O_{1,1}}{\partial F_{2,1}} = x_{2,1}, \quad \frac{\partial O_{1,1}}{\partial F_{2,2}} = x_{2,2}$$

similar we compute for  $O_{1,2}$ ,  $O_{2,1}$ , and  $O_{2,2}$

the gradient to update the filter will be

$$\frac{\partial E}{\partial F} = \frac{\partial E}{\partial O} * \frac{\partial O}{\partial F} \tag{1}$$

Introduction  
Tensors  
Convolution  
Kernels  
Convolutional Network - structure  
Convolutional Layer  
Feature learning  
Training the Convolution Layer  
Pooling Layer

# Gradient descent on Convolution Layer

if we expand Equation 1

$$\begin{aligned}\frac{\partial E}{\partial F_{1,1}} &= \frac{\partial E}{\partial O_{1,1}} * \frac{\partial O_{1,1}}{\partial F_{1,1}} + \frac{\partial E}{\partial O_{1,2}} * \frac{\partial O_{1,2}}{\partial F_{1,1}} \\ &\quad + \frac{\partial E}{\partial O_{2,1}} * \frac{\partial O_{2,1}}{\partial F_{1,1}} + \frac{\partial E}{\partial O_{2,2}} * \frac{\partial O_{2,2}}{\partial F_{1,1}} \\ &= \frac{\partial E}{\partial O_{1,1}} * x_{1,1} + \frac{\partial E}{\partial O_{1,2}} * x_{1,2} + \frac{\partial E}{\partial O_{2,1}} * x_{2,1} \\ &\quad + \frac{\partial E}{\partial O_{2,2}} * x_{2,2}\end{aligned}$$

Introduction

Tensors

Convolution

Kernels

Convolutional Network - structure

Convolutional Layer

Feature learning  
Training the Convolution Layer

Pooling Layer

# Gradient descent on Convolution Layer

[Introduction](#)[Tensors](#)[Convolution](#)[Kernels](#)[Convolutional Network - structure](#)[Convolutional Layer](#)[Feature learning  
Training the Convolution Layer](#)[Pooling Layer](#)

we observe that this is actually a convolution product between the input and the loss gradient

$$\begin{bmatrix} \frac{\partial E}{\partial F_{1,1}} & \frac{\partial E}{\partial F_{1,2}} \\ \frac{\partial E}{\partial F_{2,1}} & \frac{\partial E}{\partial F_{2,2}} \end{bmatrix} = X \circledast \begin{bmatrix} \frac{\partial E}{\partial O_{1,1}} & \frac{\partial E}{\partial O_{1,2}} \\ \frac{\partial E}{\partial O_{2,1}} & \frac{\partial E}{\partial O_{2,2}} \end{bmatrix} \quad (2)$$

# Gradient descent on Convolution Layer

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

Feature learning  
Training the  
Convolution Layer

Pooling Layer

similar we get for the derivative of  $E$  with respect of  $X$

$$\begin{bmatrix} \frac{\partial E}{\partial x_{1,1}} & \frac{\partial E}{\partial x_{1,2}} & \frac{\partial E}{\partial x_{1,3}} \\ \frac{\partial E}{\partial x_{2,1}} & \frac{\partial E}{\partial x_{2,2}} & \frac{\partial E}{\partial x_{2,3}} \\ \frac{\partial E}{\partial x_{3,1}} & \frac{\partial E}{\partial x_{3,2}} & \frac{\partial E}{\partial x_{3,3}} \end{bmatrix} = \begin{bmatrix} F_{2,2} & F_{2,1} \\ F_{1,2} & F_{1,1} \end{bmatrix} \circledast \begin{bmatrix} \frac{\partial E}{\partial O_{1,1}} & \frac{\partial E}{\partial O_{1,2}} \\ \frac{\partial E}{\partial O_{2,1}} & \frac{\partial E}{\partial O_{2,2}} \end{bmatrix}$$

observe that matrix  $F$  is flipped over 180° degrees in this formula.

# Pooling

helps to make the representation approximately invariant to small translations of the input

- useful property if we care more about whether some feature is present than exactly where it is
- essential for handling inputs of varying size
- Some guidance as to which kinds of pooling one should use in various situations : Boureau et al., 2010.

Introduction

Tensors

Convolution

Kernels

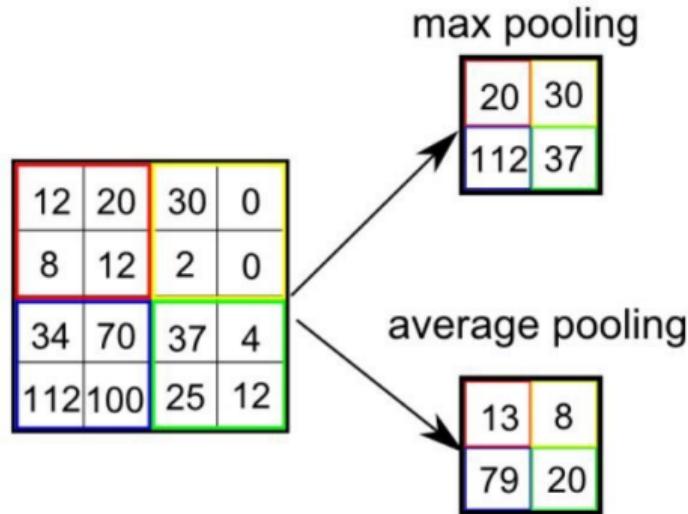
Convolutional Network - structure

Convolutional Layer

Feature learning  
Training the Convolution Layer

Pooling Layer

# Pooling



- ▶ Max Pooling - returns the maximum value from the portion of the image covered by the Kernel
- ▶ Average Pooling returns the average of all the values from the portion of the image covered by the Kernel

Introduction  
Tensors  
Convolution  
Kernels  
Convolutional Network - structure  
Convolutional Layer  
Feature learning  
Training the Convolution Layer  
Pooling Layer

# Training the Pooling layer

Introduction

Tensors

Convolution

Kernels

Convolutional  
Network -  
structure

Convolutional  
Layer

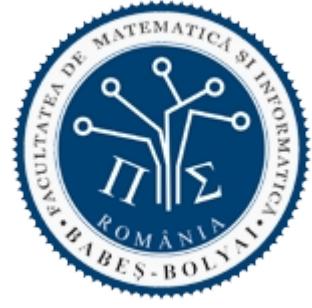
Feature learning  
Training the  
Convolution Layer

Pooling Layer

in an  $N \times N$  pooling block, forward propagation of error is reduced to a single value - value of the “winning unit”



BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics



# ARTIFICIAL INTELLIGENCE

**Intelligent systems**

Machine learning

Genetic Programming

# Topics

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

- A. Definition of search problems
- B. Search strategies
  - A. Uninformed search strategies
  - B. Informed search strategies
  - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
  - D. Adversarial search strategies

### C. Intelligent systems

- A. Rule-based systems in certain environments
- B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
- C. Learning systems
  - A. Decision Trees
  - B. Artificial Neural Networks
  - C. **Evolutionary algorithms**
    - Support Vector Machines
- D. Hybrid systems

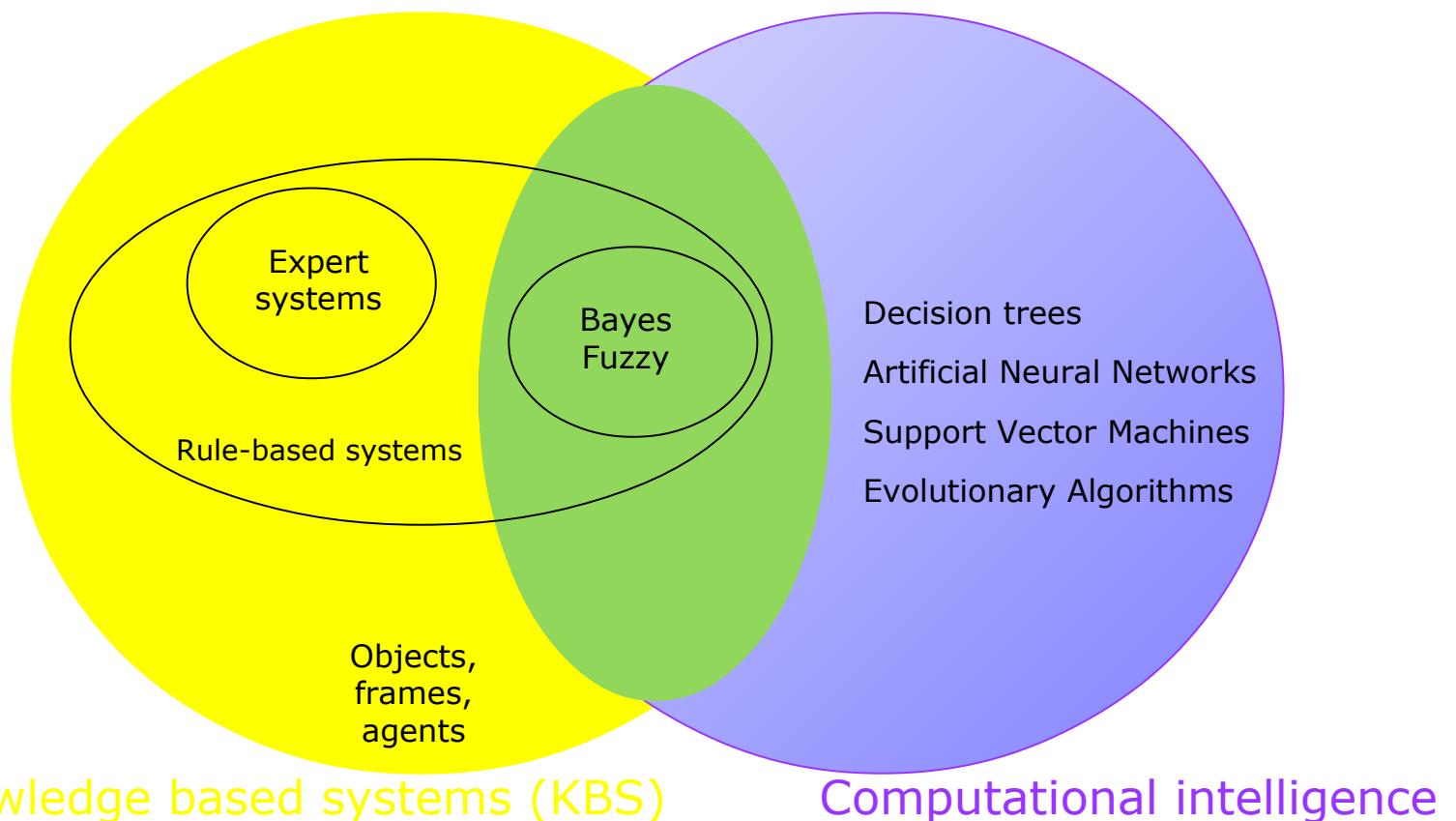
# Useful information

---

- Chapter 15 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter 9 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- Documents from *12\_svm* and *13\_GP folders*

# Intelligent systems

---



# Intelligent systems – Machine Learning

---

## □ Typology

- Experience criteria:

- Supervised learning
  - Unsupervised learning
  - Active learning
  - Reinforcement learning

- Algorithm criteria

- Decision trees
  - Artificial Neural Networks
  - **Evolutionary Algorithms**
  - Support Vector Machines
  - Hidden Markov Models

# Intelligent systems – machine learning

---

## ❑ Genetic programming (GP)

- Definition
- Design
- Advantages
- Limits
- Versions

# Intelligent systems - GP

---

## Remember

- Supervised learning → regression problem (study of relations among variables)
  - For a set of n data (examples, instances, cases)
    - Training data – as pairs ((attribute\_data<sub>i</sub>, output<sub>i</sub>), where
      - $i = 1, n$  ( $n = \#$  of training data)
      - **attribute\_data<sub>i</sub>** =  $(atr_{i1}, atr_{i2}, \dots, atr_{im})$ ,  $m = \#$  of attributes (characteristics, properties) of an example
      - *output<sub>i</sub>* – a real number
    - Testing data
      - $(attribute\_data_i), i = n+1, N$  ( $N-n = \#$  of testing data)
  - Determine
    - An (unknown) function that maps the attributes info outputs on training data
    - Output (value) of a (new) test data by using the learnt function (on training data)
- How we find the function (expression)?
  - Evolutionary algorithms → genetic programming

# Intelligent systems - GP

---

## Remember

- Evolutionary algorithms
  - Nature-inspired (bio-inspired)
  - Iterative
  - Based on
    - Populations of potential solutions
    - Random search guided by
      - Natural selection operation
      - Crossover and mutation operations
  - Parallel processing of more solutions
- Evolutionary metaphor

Natural evolution	Problem solving
Individual	Possible solution
Population	Set of possible solutions
Chromosome	Solution coding (representation)
Gene	Part of representation
Fitness (adaptation measure)	Quality
Crossover and mutation	Search operators
Environment	Problem search space

# Intelligent systems - GP

## Remember

- ❑ Evolutionary algorithm

Initialisation  $P(0)$

Evaluation  $P(0)$

$g := 0$ ; //generation

while (not stop\_condition) execute

    Repeat

        Select 2 parents  $p_1$  and  $p_2$  from  $P(g)$

        Crossover( $p_1, p_2$ )  $\rightarrow o_1$  and  $o_2$

        Mutation( $o_1$ )  $\rightarrow o_1^*$

        Mutation( $o_2$ )  $\rightarrow o_2^*$

        Evaluation( $o_1^*$ )

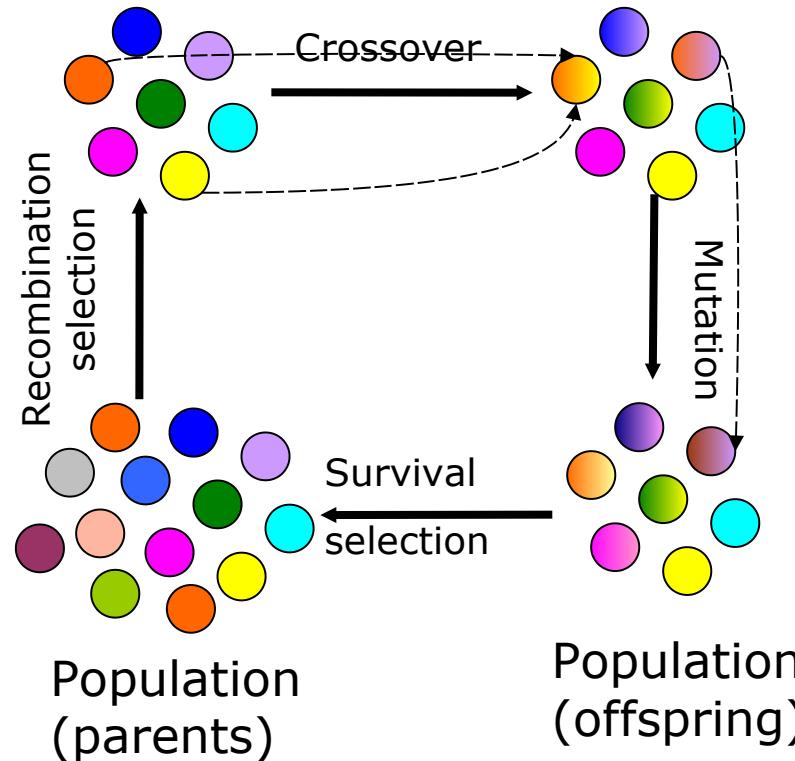
        Evaluation( $o_2^*$ )

        Add  $o_1^*$  and  $o_2^*$  in  $P(g+1)$

    until  $P(g+1)$  is complete

$g := g + 1$

EndWhile



# Intelligent systems - GP

---

## □ Definition

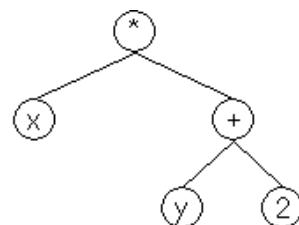
- Proposed by Koza in 1988
- <http://www.genetic-programming.org/>
- A special case of evolutionary algorithms
- Chromosome
  - As trees that encode small programs
- Fitness of a chromosome
  - Performance of the program encoded by the chromosome
- GP's aim
  - Evolving computer programs
  - Gas evolve solutions for particular problems only

# Intelligent systems - GP

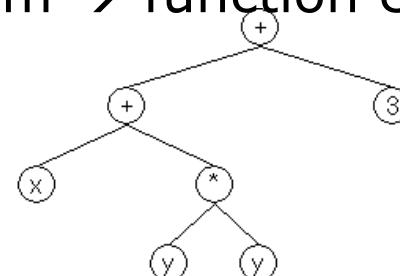
## □ Design

### ■ Chromosome representation

- Very important, but a difficult task
- Chromosome = tree with nodes of type
  - Function → (mathematical) operators  
 $(+,-,\cdot,/,sin,log, if, \dots)$
  - Terminal → attributes of data or constants  
 $(x,y,z,a,b,c, \dots)$
- that encodes the mathematical expression of the program (regression problem → function expression)



$$x(y+2)$$



$$x+y^2+3$$

# Intelligent systems - GP

## □ Design

### ■ Fitness

- Prediction error – difference between what we want to obtain and what we actually obtain
  - For a regression problem with input data (2 attributes and an output) and 2 chromosomes:
    - $c_1 = 3x_1 - x_2 + 5$
    - $c_2 = 3x_1 + 2x_2 + 2$
- $f^*(x_1, x_2) = 3x_1 + 2x_2 + 1$  – unknown

$x_1$	$x_2$	$f^*(x_1, x_2)$	$f_1(x_1, x_2)$	$f_2(x_1, x_2)$	$ f^* - f_1 $	$ f^* - f_2 $
1	1	6	7	7	1	1
0	1	3	4	4	1	1
1	0	4	8	5	4	1
-1	1	0	1	1	1	1
					$\Sigma = 7$	$\Sigma = 4$

→  $c_2$  is better than  $c_1$

# Intelligent systems - GP

## □ Design

### ■ Fitness

- Prediction error – difference between what we want to obtain and what we actually obtain
- For a classification problem with input data (2 attributes and an output) and 2 chromosomes:
  - $c_1 = 3x_1 - x_2 + 5$
  - $c_2 = 3x_1 + 2x_2 + 2$

$x_1$	$x_2$	$f^*(x_1, x_2)$	$f_1(x_1, x_2)$	$f_2(x_1, x_2)$	$ f^*-f_1 $	$ f^*-f_2 $
1	1	Yes	Yes	Yes	0	0
0	1	No	Yes	No	1	0
1	0	Yes	No	No	1	1
-1	1	Yes	No	yes	1	0
					$\Sigma=3$	$\Sigma= 1$

→  $c_2$  is better than  $c_1$

# Intelligent systems - GP

---

## ❑ Design

### ■ Chromosome initialisation

- ❑ Random generation of correct trees → valid programs (valid mathematical expressions)
- ❑ Establish a maximal depth of the trees  $D_{\max}$
- ❑ 3 initialisation methods
  - *Full* → each branch of the root has depth  $D_{\max}$ 
    - Nodes of depth  $d < D_{\max}$  are initialised by a function from  $F$
    - Nodes of depth  $d = D_{\max}$  are initialised by a terminal from  $T$
  - *Grow* → each branch of the root has a depth  $< D_{\max}$ 
    - Nodes of depth  $d < D_{\max}$  are initialised by an element from  $F \cup T$
    - Nodes of depth  $d = D_{\max}$  are initialised by a terminal from  $T$
  - *Ramped half and half* →  $\frac{1}{2}$  of population is initialised by using *full* method and  $\frac{1}{2}$  of population is initialised by *grow* method

# Intelligent systems - GP

---

## ❑ Design

- Genetic operators → recombination selection
  - Similar to other EAs
  - Advise → proportional selection
  - over-selection → for very large populations
    - Sort the population based on fitness and consider 2 groups:
      - Group 1 contains the best  $x\%$  chromosomes from population
      - Group 2 contains  $(100-x)\%$  chromosome from population
      - For populations with 1000, 2000, 4000 or 8000 chromosomes,  $x$  can be 32%, 16%, 8% and 4% respectively
    - 80% of selection operators will choose chromosomes from the first group and 20% of them from the second group

# Intelligent systems - GP

---

## ❑ Design

- Genetic operators → survival selection
  - ❑ Sketches
    - Generational
    - Steady-state
  - ❑ Problems
    - *Bloat* → the fattest individual survives (size of chromosomes increases during evolution)
    - Solutions
      - Block the variation operators that produce to fat offsprings
      - parsimony pressure - to give a penalty in the cost function (or fitness function) to long programs or program with many non-coding parts

# Intelligent systems - GP

---

## □ Design

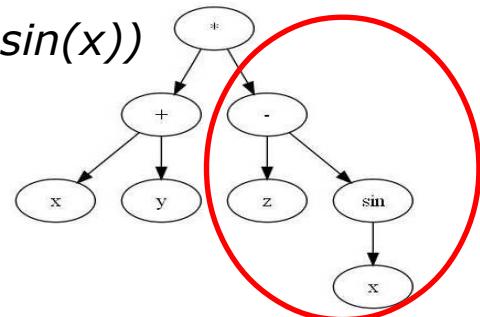
- Genetic operators → crossover and mutation
  - Parameters
    - A probability  $p$  of choosing between XO and mutation
      - $p = 0$  (cf. Koza) or  $p = 0.05$  (cf. Banzhaf)
    - Two probabilities  $p_c$  and  $p_m$  for establishing the part of chromosome(s) that must be changed
  - Size of offsprings can be different to size of parents

# Intelligent systems - GP

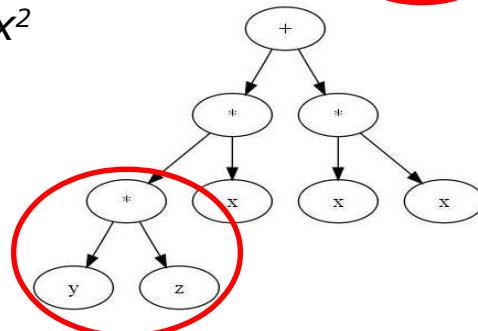
## □ Design

- Genetic operators → crossover
  - By cutting point
    - sub-trees are exchanged
    - Cutting point is randomly generated

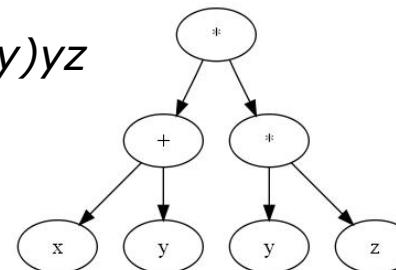
$$p_1 = (x+y)*(z-\sin(x))$$



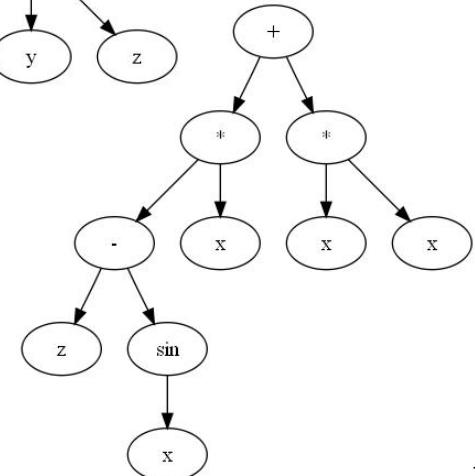
$$p_2 = xyz + x^2$$



$$f_1 = (x+y)yz$$



$$f_2 = (z-\sin(x))x + x^2$$



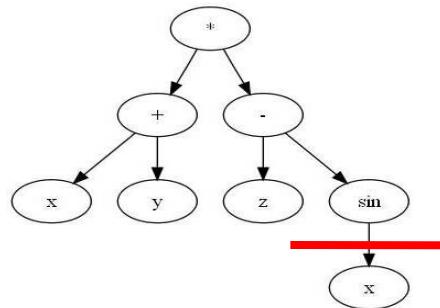
# Intelligent systems - GP

## □ Design

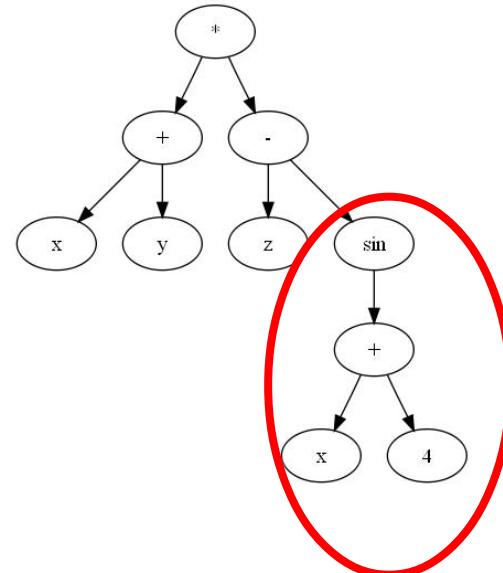
### ■ Genetic operators → mutation

- *Grow* mutation → replace a leaf by a new sub-tree

$$p = (x+y)*(z-\sin(x))$$



$$f = (x+y)*(z-\sin(x+4))$$



# Intelligent systems - GP

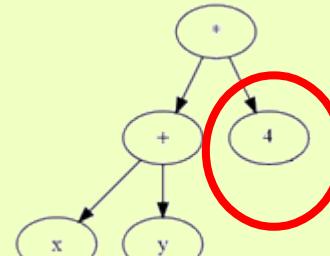
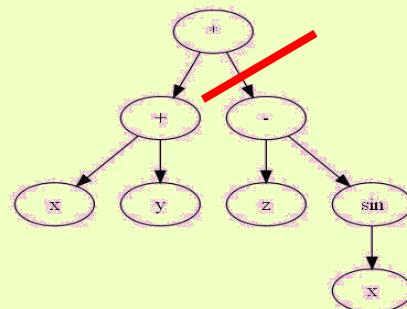
## □ Design

### ■ Genetic operators → mutation

- *Shrink* mutation → replace a sub-tree by a leaf

$$p = (x+y)*(z-\sin(x))$$

$$f = (x+y)*4$$



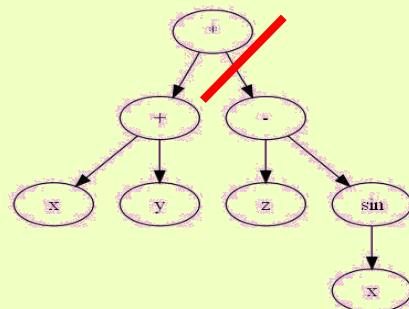
# Intelligent systems - GP

## □ Design

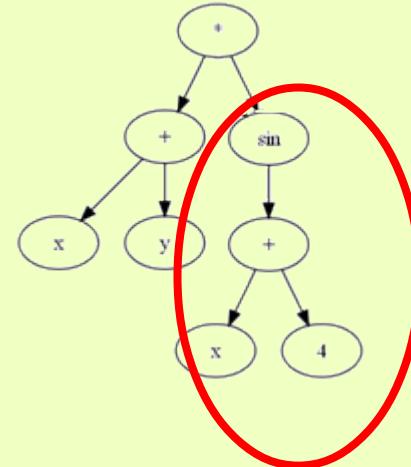
### ■ Genetic operators → mutation

- ▣ Koza mutation → replace a node (leaf or internal) by a sub-tree

$$p = (x+y)*(z-\sin(x))$$



$$f = (x+y)*\sin(x+4)$$



# Intelligent systems - GP

---

## □ Design

- Genetic operators → mutation
  - *Switch* mutation
    - Select an internal node and re-order its sub-trees
  - *Cycle* mutation
    - Select a node and replace it by a new node of the same type (internal node with a function, leaf node with a terminal)

# Intelligent systems - GP

---

## ❑ GAs vs. GP

- Chromosome's shape
  - ❑ GAs – linear chromosomes
  - ❑ GP – non-linear chromosomes
- Chromosome's size
  - ❑ GAs – fix size
  - ❑ GP – variable size (depth or width)
- Offspring generation
  - ❑ GAs – XO and mutation
  - ❑ GP – XO or mutation

# Intelligent systems - GP

---

## □ Advantages

- GP finds solutions for problems without an optimal solution
  - A program for car driving → there are more solution
    - Some solutions → safe but slow driving
    - Other solutions → dangerous, but fast driving
    - Car driving ↔ trade-off large speed and safety
  - GP is useful for problems whose variables are frequently changed
    - Car driving on a highway
    - Car driving on a forest road

## □ Limits

- Large time required for evolving the solution

# Intelligent systems - GP

---

## □ GP versions

- Linear GP (Cramer, Nordin)
- Gene Expression Programming (Ferreira)
- Multi Expression Programing (Oltean)
- Gramatical Evolution (Ryan, O'Neill)
- Cartesian Genetic Programming (Miller)

# Intelligent systems - GP

## □ Linear GP

- Evolving programs written in an imperative language (fitness computation does not require interpretation) → works fast
- Representation
  - Vector of statements, each statement being (in the case of a maximal arrity of n for a function)
    - Index\_op, out\_register, in<sub>1</sub>\_register, in<sub>2</sub>\_register,..., in<sub>n</sub>\_register
      - $v_i = v_j * v_k$  // instruction operating on two registers
      - $v_i = v_j * c$  // instruction operating on one register and one constant
      - $v_i = \sin(v_j)$  // instruction operating on one register

```
void LGP_program (double v[11])
{
    ...
    v[8] = v[0] - 10;
    v[6] = v[2] * v[0];
    v[5] = v[8] * 7;
    v[4] = v[2] - v[0];
    v[10] = v[1]/v[4];
    v[3] = sin(v[1]);
    v[1] = v[8] - v[6];
    v[7] = v[10] * v[3];
    v[9] = v[0] + v[7];
    v[2] = v[7] + 3;
    ...
}
```

```
void LGP_effective_program (double v[11])
{
    ...
    v[4] = v[2] - v[0];
    v[10] = v[1]/v[4];
    v[3] = sin(v[1]);
    v[7] = v[10] * v[3];
    v[9] = v[0] + v[7];
    ...
}
```

# Intelligent systems - GP

---

## ❑ Linear GP

- Initialisation
  - ❑ Random
  - ❑ Constraints
    - Initial length of chromosome (# of statements)
- Variation genetic operators
  - ❑ Crossover – 2-cutting points
  - ❑ Mutation
    - Micro-mutation → change an operand or operator (without modifying the size of chromosome)
    - Macro-mutation → insert or eliminate a statement (modifying the size of chromosome)

# Intelligent systems - GP

---

## □ Linear GP

- Advantages
  - Evolution into a low-level language
- Disadvantages
  - # of required registers (# of problem's attributes)
- Resources
  - Register Machine Learning Technologies <http://www.aimlearning.com>
  - *Peter Nordin's home page* <http://fy.chalmers.se/~pnordin>
  - *Wolfgang Banzhaf's home page* <http://www.cs.mun.ca/~banzhaf>
  - *Markus Brameier's home page* <http://www.daimi.au.dk/~brameier>

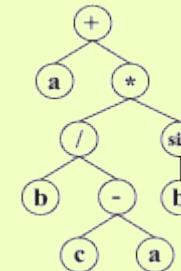
# Intelligent systems - GP

## □ Gene Expression Programming (GEP)

### ■ Main idea

- ▣ Linear representation of expressions that can be encoded in a tree (by breadth-first traversing procedure)

$$C = +a * /Sb - bcacabbc$$



### ■ Representation

- ▣ A chromosome is composed by more genes
  - Linked by + or \*
- ▣ Each gene is composed by
  - Head
    - Contains functions and terminals
  - Tail
    - Contains t terminals only, where t = (n-1)\*h+1, with n – maximal arrity of a function from F

# Intelligent systems - GP

---

## □ GEP

- Initialisation
  - Randomly, by elements from F and T (cf. to previous rules)
- Variation genetic operators
  - Crossover
    - At allele level
      - One cutting point
      - Two cutting points
    - At gene level
      - Chromosomes exchange (between them) some genes (located on corresponding positions)
  - Mutation
    - At allele level
      - Change an element from head or tail (following the previous initialisation rules)
  - Transpositions
    - the introduction of an insertion sequence somewhere in a chromosome

# Intelligent systems - GP

---

## □ GEP

- Advantages
  - Coding into chromosomes some correct programs due to gene splitting in head and tail
- Disadvantages
  - Multi-gene chromosomes
    - How many genes?
    - How to link the genes?
- Resources
  - Gene Expression Programming website, <http://www.gepsoft.com>
  - *Heitor Lopes's home page*  
<http://www.cpgei.cefetpr.br/~hslopes/index-english.html>
  - *Xin Li's home page* <http://www.cs.uic.edu/~xli1>
  - *GEP in C#* <http://www.c-sharpcorner.com/Code/2002/Nov/GEPAlgorithm.asp>

# Intelligent systems - GP

---

## ❑ Multi Expression Programming (MEP)

### ■ Main idea

- ❑ Chromosome is composed by more genes, each gen being a 3 address code
  - Similarly to GEP, but faster

### ■ Representation

- ❑ Linear
- ❑ A gene contains a (binary or unary) function and pointers to its arguments
- ❑ Chromosome encodes more possible solutions → each solution corresponds to a gene
  - Quality of a solution (gene) = sum (over training data) of differences between what we want to obtain and what we obtain
  - Fitness = quality of the best gene

# Intelligent systems - GP

---

## □ MEP

- Initialisation
  - First gene must be a terminal
  - Other genes can contain
    - A terminal or
    - A (unary or binary) function and pointers to its arguments
      - Arguments of a function located in the  $i^{\text{th}}$  gene must be located in genes of index  $< i$
- Variation genetic operators
  - Crossover → exchange some genes between parents
    - 1-cutting point
    - 2-cutting points
    - Uniform
  - Mutation → modify a gene
    - First gene → randomly generate a new terminal
    - Other genes → randomly generate a terminal or a function (function symbol and its arguments)

# Intelligent systems - GP

---

## □ MEP

- Advantages
  - Dynamic output for each chromosome
    - Complexity of the search program (expression)
    - Programs (expression) of variable length obtained without special operators
    - Program of exponential length encoded into chromosomes of polynomial length
- Disadvantages
  - Complexity of decoding for unknown training data → evolving game's strategies
- Resources
  - Mihai Oltean's home page <http://www.cs.ubbcluj.ro/~>
  - Crina Grossan's home page <http://www.cs.ubbcluj.ro/~cgrosan>
  - MEP web page <http://www.mep.cs.ubbcluj.ro>
  - MEP in C# <http://www.c-sharpcorner.com>

# Intelligent systems - GP

## Grammatical Evolution (GE)

### Main idea

- Evolving programs in Backus-Naur form (program expressed as a grammar with terminal symbols, non-terminals, start symbol and rules)

### Representation

- Binary strings of codons (groups of 8 bits) → rule that must be applied
- Example

- $G = \{N, T, S, P\}$ ,  $N = \{+, -, *, /, \sin, (, )\}$ ,  $T = \{\text{expr}, \text{op2}, \text{op1}\}$ ,  $S = \langle \text{expr} \rangle$ ,  $P$  is:
  - $\langle \text{expr} \rangle ::= a | b | c | \langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle | (\langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle) | \langle \text{op1} \rangle \langle \text{expr} \rangle$
  - $\langle \text{op2} \rangle ::= + | - | * | /$ ,
  - $\langle \text{op1} \rangle ::= \sin$
- $C^*_{GE} = (9 \ 12 \ 12 \ 3 \ 15 \ 7 \ 11 \ 4 \ 2 \ 5 \ 0 \ 6 \ 11 \ 0 \ 1 \ 7 \ 12)$
- $S = \langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle \rightarrow a \langle \text{op2} \rangle \langle \text{expr} \rangle \rightarrow a + \langle \text{expr} \rangle \rightarrow a + \langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle \rightarrow a + \langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle \rightarrow a + b \langle \text{op2} \rangle \langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle$

$$C_{GE} = (00001001 \ 00001100 \ 00001100 \ 00000011 \ 00001111 \ 00000111 \\ 00001011 \ 00000100 \ 00000010 \ 00000101 \ 00000000 \ 00000110 \\ 00001011 \ 00000000 \ 00000001 \ 00000111 \ 00001100)$$

$a + b / \langle \text{op2} \rangle \langle \text{expr} \rangle$   
 $a + b / (\langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle) \langle \text{op2} \rangle \langle \text{expr} \rangle$   
 $a + b / (c \langle \text{op2} \rangle \langle \text{expr} \rangle) \langle \text{op2} \rangle \langle \text{expr} \rangle$   
 $a + b / (c - \langle \text{expr} \rangle) \langle \text{op2} \rangle \langle \text{expr} \rangle$   
 $a + b / (c - a) \langle \text{op2} \rangle \langle \text{expr} \rangle$   
 $a + b / (c - a) * \langle \text{expr} \rangle$   
 $a + b / (c - a) * \langle \text{op1} \rangle \langle \text{expr} \rangle$   
 $a + b / (c - a) * \sin \langle \text{expr} \rangle$

$$E = a + b / (c - a) * \sin(b)$$

# Intelligent systems - GP

---

## □ GE

### ■ Initialisation

- Binary string is randomly initialised by 0 or 1 (without constraints) → valid programs
- Decoding ends when a complete program is obtained
  - If the codons end and the program is incomplete, restart the codons from beginning → wrapping

### ■ Variation genetic operators

- Crossover
  - Cutting point XO
- Mutation
  - Probabilistic change of a bit into its opponent
- Duplication
  - A sequence of genes is copied to the end of chromosome
- Pruning
  - Elimination of unused genes

# Intelligent systems - GP

---

## □ GE

### ■ Advantages

- Evolving programs written in languages whose statements can be expressed as BNF rules
- Representation can be changed by changing the grammar

### ■ Disadvantages

- Infinite wrapping → limit the repetitions and penalise the chromosomes that overpass a given threshold of repetitions

### ■ Resources

- Grammatical Evolution web page, <http://www.grammatical-evolution.org>
- *Conor Ryan's home page*, <http://www.csis.ul.ie/staff/conorryan>
- *Michael O'Neill's home page*, <http://ncra.ucd.ie/members/oneillm.html>
- *John James Collins's home page*, <http://www.csis.ul.ie/staff/jjcollins>
- *Maarten Keijzer's home page*, <http://www.cs.vu.nl/~mkeijzer>
- *Anthony Brabazon's home page* <http://ncra.ucd.ie/members;brabazont.html>

# Intelligent systems - GP

## □ Cartesian Genetic Programming (CGP)

### ■ Main idea

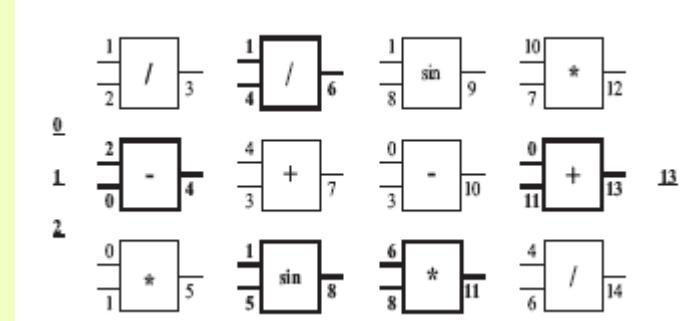
- ▢ Chromosomes as graphs (matrix) → more complex programs

### ■ Representation

- ▢ Cartesian system (matrix of nodes)

- ▢ A node has associated
  - A function
  - Inputs
  - Outputs

- ▢ Chromosome output
  - Output of any node



$$C = (1, 2, 3, \underline{2}, \underline{0}, \underline{1}, 0, 1, 2, \underline{1}, \underline{4}, \underline{3}, 4, 3, 0, \underline{1}, \underline{5}, \underline{4}, 1, 8, 4, 0, 3, 1, \underline{6}, \underline{8}, \underline{2}, 10, 7, 2, 0, 11, 0, 4, 6, 3, 13)$$

# Intelligent systems - GP

---

## □ CGP

- Initialisation
  - Randomly
  - Inputs of any node must be nodes from previous columns
    - Nodes of the first column has as inputs the problem attributes
- Variation genetic operators
  - Crossover
    - Is not applied
  - Mutation
    - Modify the elements of a node

# Intelligent systems - GP

---

## □ CGP

### ■ Advantages

- Evolving the index of node that provides the output of the program encoded into the chromosome
- Evolved program can have one or more outputs

### ■ Disadvantages

- number of columns influences the results

### ■ Resources

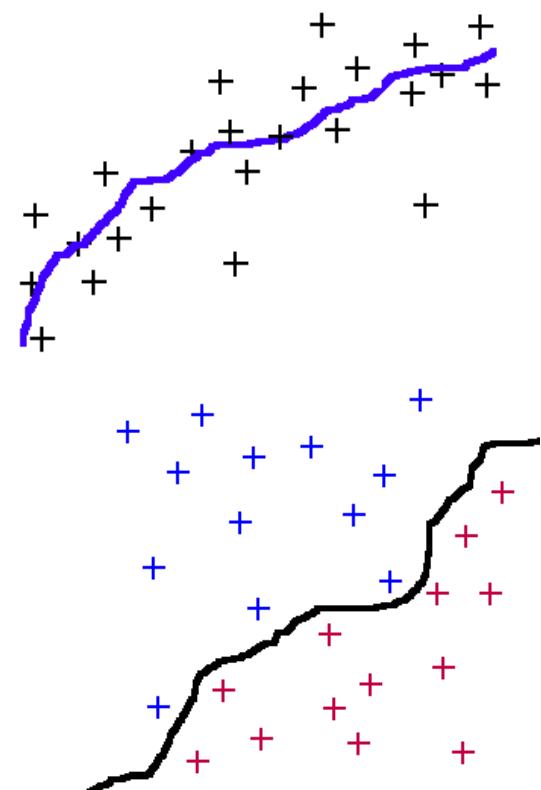
- Julian. F. Miller's home page <http://www.elec.york.ac.uk/intsys/users/jfm7>
- Lukás Sekanina's home page <http://www.fit.vutbr.cz/~sekanina/>

# Intelligent systems - GP

---

## ❑ Applications

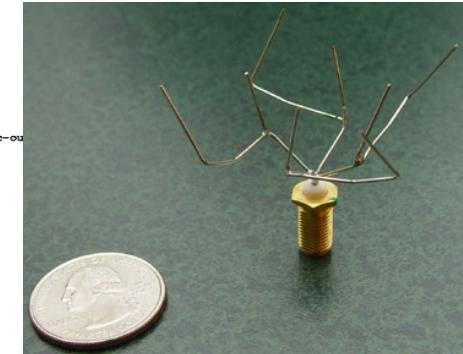
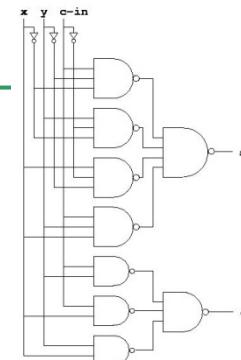
- Problems with relations between inputs and outputs
- Regression problems
- Classification problems



# Intelligent systems - GP

- Applications
  - Design problems

- Evolving digital circuits
- Evolving antenna
  - [http://idesign.ucsc.edu/projects/evo\\_antenna.html](http://idesign.ucsc.edu/projects/evo_antenna.html)
- Evolving programs
- Evolving pictures and music
  - <http://www.cs.vu.nl/~gusz/>
- Others
  - <http://www.genetic-programming.com/humancompetitive.html>



# Review

---



## ❑ Machine learning

- Genetic programming (GP)
  - ❑ Evolutionary algorithms with chromosomes as trees
  - ❑ Chromosomes
    - Trees
    - Matrix
    - Linear
  - ❑ Encode potential solutions
    - Mathematical expressions → regression/classification problems
    - Boolean expressions → Even Parity problems or digital circuits design
    - Programs → evolving source codes for problem solving

# Next lecture

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

- A. Definition of search problems
- B. Search strategies
  - A. Uninformed search strategies
  - B. Informed search strategies
  - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
  - D. Adversarial search strategies

### C. Intelligent systems

- A. Rule-based systems in certain environments
- B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
- C. Learning systems
  - A. Decision Trees
  - B. Artificial Neural Networks
  - C. Evolutionary algorithms
  - **Support Vector Machines**
- D. Hybrid systems

# Next lecture – useful information

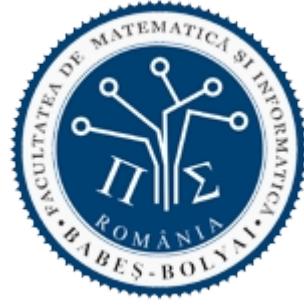
---

- Chapter 15 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter 9 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- Documents from *svm* folder

- 
- ❑ Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean - [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)



BABEŞ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics



# ARTIFICIAL INTELLIGENCE

**Intelligent systems**

Machine learning

Decision trees

# Useful information

---

- Chapter VI (18) of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapters 10 and 11 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter V of *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Chapters 3 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

# Content

---

## ❑ Intelligent systems

- Automatic learning systems (ALS)
  - Machine Learning - ML
    - Problem
    - Design
    - Typology
      - » Supervised learning
      - » Unsupervised learning
      - » Reinforcement learning
      - » Learning theory
  - Systems
    - Decision trees

# Intelligent systems – Machine Learning (ML)

---

- Problem
  - “How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?”
  
- Applications
  - Image and voice recognition
    - Handwritten recognition
    - Face detection
  - Computer vision
    - Obstacle detection
    - Footprint recognition
  - Bio-surveillance
  - Robot control
  - Predictions
  - Medical diagnostic
  - Fraud detection

# Intelligent systems – Machine Learning (ML)

---

## □ Definition

- Arthur Samuel (1959)
  - ▣ “field of study that gives computers the ability to learn without being explicitly programmed”
- Herbert Simon (1970)
  - ▣ “Learning is any process by which a system improves performance from experience.”
- Tom Mitchell (1998)
  - ▣ “a well-posed learning problem is defined as follows: He says that a computer program is set to learn from an experience E with respect to some task T and some performance measure P if its performance on T as measured by P improves with experience E”
- Ethem Alpaydin (2010)
  - ▣ Programming computers to optimize a performance criterion using example data or past experience.

## □ Necessity

- Better computational systems
  - ▣ To difficult or too expensive to be constructed manually
    - Systems that automatically adapt
      - Spam filters
    - Systems that discover information in large database → data mining
      - Financial analysis
      - Text/image analyses
- Understanding the biological systems



# Intelligent systems – Machine Learning (ML)

---

## □ Design

- Improve of task T
  - Establish the goal (what has to be learn) – objective function – and its representation
  - Select a learning algorithms to perform the inference of the goal based on experience
- Respect a performance metric P
  - Evaluation of the algorithm's performances
- Based on experience E
  - Select an experience database
- Example
  - T: playing checkers
  - P: percent of winning games
  - E: playing the game
  - T: handwritten recognition
  - P: percent of correct recognized words
  - E: database of images with different words
  - T: separate the spams
  - P: percent of correct classified emails
  - E: databases with annotated emails

# Intelligent systems – Machine Learning (ML)

---

- Design → choose the objective function
  - Which is the function that must be learn?
    - Ex.: checkers game → a function that:
      - Selects the next move
      - Evaluates a move
    - In order to identify the best move
  - Representation of objective function
    - Different representations
      - Tabel
      - Symbolic rules
      - Numeric functions
      - Probabilistic functions
    - Ex. Checkers game
      - A linear combinations of # white pieces, # black pieces, # of white compromised pieces, # black compromised pieces
    - There is a trade-off between
      - Expressiveness of representation
      - Easy of learning
    - Objective function computation
      - Polynomial time
      - Non-polynomial time

# Intelligent systems – Machine Learning (ML)

---

- ▣ Design → select a learning algorithm
  - Algorithm
    - ▣ By using the training data
    - ▣ Induce the hypothesis definition that
      - Match the data
      - Generalize the un - seen data
  - Main principle
    - ▣ Error minimisation (cost function – loss function)
- ▣ Design → evaluation of a learning system
  - Experimental
    - ▣ By comparing different methods on different data (cross-validation)
    - ▣ Collect data based on performances
      - Accuracy, training time, testing time
    - ▣ Statistical analyse of the differences
  - Theoretic
    - ▣ Mathematical analyse of algorithms and theorem proving
      - Computational complexity
      - Ability to match the training data
      - Complexity of the most relevant sample for learning

# Intelligent systems – Machine Learning (ML)

- ▣ Design → evaluation of a learning system
  - Comparing the performances of 2 algorithms for solving a given problem
    - ▢ Performance measures
      - Parameters of a statistic series
      - Proportion (percent) computed for a statistical series (ex. Accuracy)
    - ▢ Comparing based on confidence intervals
      - For a problem and 2 solving algorithms with performances  $p_1$  and  $p_2$
      - Confidence intervals  $I_1 = [p_1 - \Delta_1, p_1 + \Delta_1]$  și  $I_2 = [p_2 - \Delta_2, p_2 + \Delta_2]$
      - If  $I_1 \cap I_2 = \emptyset \rightarrow$  algorithm 1 works better than algorithm 2 (for the given problem)
      - if  $I_1 \cap I_2 \neq \emptyset \rightarrow$  impossible to decide
  - ▢ Confidence interval for the mean (average)
    - For a statistical series of  $n$  data, with computed mean  $m$  and dispersion  $\sigma$ , determine the confidence interval of the mean  $\mu$
    - $P(-z \leq (m-\mu)/(\sigma/\sqrt{n}) \leq z) = 1 - \alpha \rightarrow \mu \in [m - z\sigma/\sqrt{n}, m + z\sigma/\sqrt{n}]$
    - $P = 95\% \rightarrow z = 1.96$
  - ▢ Confidence interval for accuracy
    - For an accuracy  $p$  computed for  $n$  data, determine the confidence interval of accuracy
    - $p \in [p - z(p(1-p)/n)^{1/2}, p + z(p(1-p)/n)^{1/2}]$
    - $P = 95\% \rightarrow z = 1.96$

$P=1-\alpha$	$z$
99.9%	3.3
99.0%	2.577
98.5%	2.43
97.5%	2.243
95.0%	1.96
90.0%	1.645
85.0%	1.439
75.0%	1.151

# Intelligent systems – Machine Learning (ML)

---

- Design → choose the training database
  - Based on
    - Direct experience
      - Pairs (in, out) that are useful for the objective function
      - Eg. Checkers game → board game annotated by correct or incorrect move
    - Indirect experience
      - Useful feedback (unlike i/o pairs) for the objective function
      - Eg. Checkers game → sequences of moves and the final score of the game
  - Data sources
    - Random generated examples
      - Positive and negative examples
    - Positive examples collected by a learner
    - Real examples
  - Content
    - Training data
    - Test data
  - Characteristics
    - Independent data
      - Otherwise → collective learning
    - Training and testing data must respect the same distribution law
      - Otherwise → *transfer learning/inductive transfer*
        - Vehicle recognition → truck recognition
        - Text analyses
        - Spam filters

# Intelligent systems – Machine Learning (ML)

---

- ▣ Design → choose the training database
  - Characteristics extracted (attributes) from raw data
    - ▣ Quantitative characteristics → nominal or rational scale
      - Continuous values → weight
      - Discrete values → # of computers
      - Range values → event times
    - ▣ Qualitative characteristics
      - Nominal → colour
      - Ordinal → sound intensity (low, medium, high)
    - ▣ Structured
      - Trees – root is a generalisation of children (vehicle → car, bus, tractor, truck)
  - Data transformation
    - ▣ Standardisation → numerical attributes
      - Remove the scale effect (different scale and units)
      - Raw values are transformed in z scores
        - $Z_{ij} = (x_{ij} - \mu_j)/\sigma_j$ , where  $x_{ij}$  – value of  $j^{th}$  attribute of  $i^{th}$  instance,  $\mu_j$  ( $\sigma_j$ ) is the mean (standard deviation) of  $j^{th}$  attribute for all instances
    - ▣ Selection of some attributes

# Intelligent systems – Machine Learning (ML)

## □ Typology

### ■ Based on their aim (goal)

#### ▢ ISs for prediction

- Aim: predict the output for a new input based on a previously learned model
- Eg. predicting sales of a product for a time in the future based on price, calendar month, region, average income

#### ▢ ISs for regression

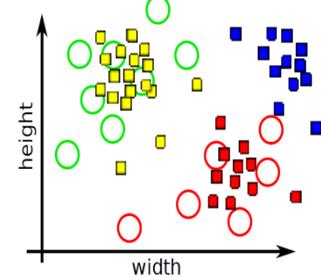
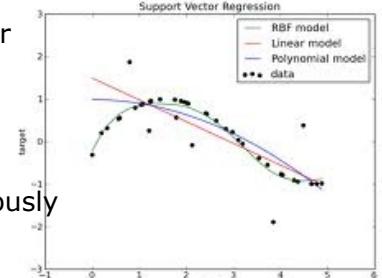
- Aim: estimation of the (uni or multi - variable) function shape based on a previously learned model
- Eg.: estimate the function that models the edge of a surface

#### ▢ ISs for classification

- Aim: classify an object into one or more – known or unknown - categories based on their characteristics
- Eg.: diagnostic systems for cancer: malign or benign or normal

#### ▢ ISs for planning

- Aim: generate a sequence of optimal actions for performing a task
- Eg.: planning the moves of a robot from a position to a source of energy



# Intelligent systems – Machine Learning (ML)

## □ Typology

- Based on the experience learned during training process

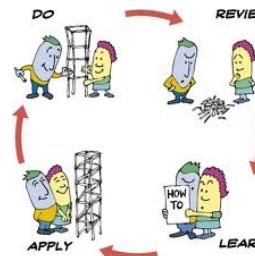
- ISs with supervised learning



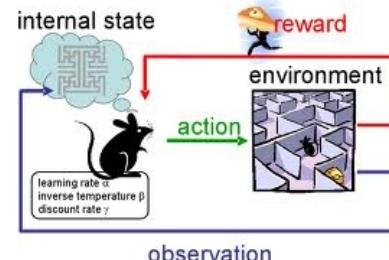
- ISs with unsupervised learning



- ISs with active learning



- ISs with reinforcement learning



# Intelligent systems – Machine Learning (ML)

---

- Typology

- Based on algorithm
  - **Decision trees**
  - Artificial Neural Networks
  - Evolutionary algorithms
  - Support Vector Machines
  - Hidden Markov Models

# Intelligent systems – decision trees (DT)

---

## □ Decision trees (DT s)

- Aim
- Definition
- Solved problems
- Example
- Process
- Tools
- Advantages and limits

# Intelligent systems – decision trees (DT)

---

## □ Aim

- Divide a collection of articles in smaller sets by successively applying some decision rules → asking more questions
  - Each question is addressed based on the answer of the previous question
- Elements are characterized by non-metric information

## □ Definition

- Decision tree
  - A special graph → bi-colour and oriented tree
  - Contains three node types:
    - Decision nodes → possibilities of decider (a test on an attribute of item that must be classified)
    - Hazard nodes → random events outside the control of decider (exam results, therapy consequences)
    - Result nodes → final states that have a utility or a label
  - Decision and hazard nodes alternate on the tree levels
  - Result nodes → leaf (terminal nodes)
  - (oriented) Edges of the tree consequences of decisions (can be probabilistic)
- Each internal node corresponds to an attribute
- Each branch under a node (attribute) corresponds to the value of that attribute
- Each leaf corresponds to a class

# Intelligent systems – decision trees (DT)

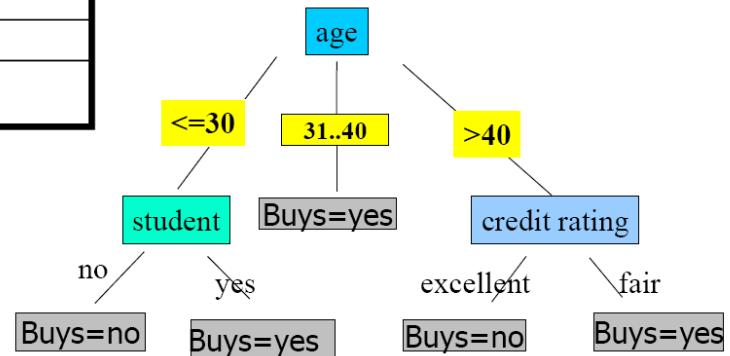
---

- Problems solved by DTs
  - Problem's instances are represented by a fixed number of attributes, each attribute having a finite number of values
  - Objective function takes discrete values
  - DT represents a dis-junction of more conjunctions, each conjunction being "attribute  $a_i$  has value  $v_j$ "
  - Training data could contain errors
  - Training data could be incomplete
    - Some data have not all attributes
- Classification problem
  - Binary classification
    - Instances are  $[(\text{attribute}_{ij}, \text{value}_{ij}), \text{class}_i, i=1,2,\dots,n, j=1,2,\dots,m, \text{class}_i \text{ taking 2 values}]$
  - Multi-class ( $k$ -class)
    - Instances are  $[(\text{attribute}_{ij}, \text{value}_{ij}), \text{class}_i, i=1,2,\dots,n, j=1,2,\dots,m, \text{class}_i \text{ taking } k \text{ values}]$
- Regression problems
  - DTs are constructed in a similar manner to those of classification problems, but instead to label each node by the label of a class, each node has associated a real value or a function that depends on the inputs of that node
  - Input space is split in decision regions by parallel cuttings to  $Ox$  and  $Oy$
  - Discrete outputs are transformed in continuous functions
  - Quality of problem solving
    - Prediction error (square or absolute)

# Intelligent systems – decision trees (DT)

## □ Example

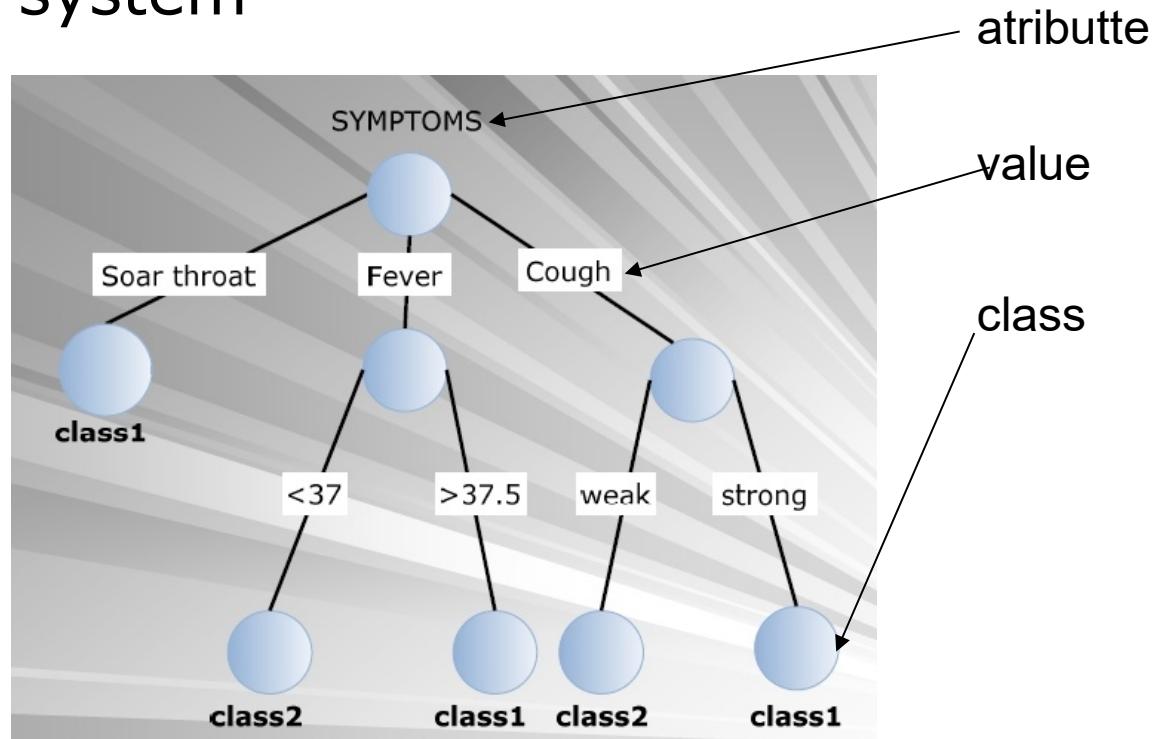
rec	Age	Income	Student	Credit_rating	Buys_computer(CLASS)
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31...40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No



# Intelligent systems – decision trees (DT)

## □ Example

### ■ Medical system



# Intelligent systems – decision trees (DT)

## □ Example

### ■ Credits

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

# Intelligent systems – decision trees (DT)

---

- Process
  - Tree construction (induction)
    - Based on training data
    - Works bottom-up or top-down (splitting)
  - Using the tree as a problem solver
    - All decisions performed along a path from the root to a leaf form a rule
    - Rules from DT are used for labeling new data
  - Pruning
    - Identify and move/eliminate branches that reflect noise or exceptions

# Intelligent systems – decision trees (DT)

---

- Process → Tree construction (induction)
  - Split the training data into subsets based on the characteristics of data
    - ▣ A node → Question related to a property
    - ▣ Branches of a node → possible answers to the question of the node
    - ▣ Initially, all examples are located in the root
      - An attribute gives the root and its values give the branches
    - ▣ On next levels, examples are partitioned based on their attributes → order of attributes
      - For each node, an attribute is (recursively) chosen – its values → branches
    - ▣ Splitting → greedy decision making
  - Iterative process
    - ▣ Stop conditions
      - All examples from a node belong to the same class → node is a leaf and is labeled by *class*,
      - There are no examples → node becomes a leaf and is labeled by the majority class of training data
      - There are no attributes

# Intelligent systems – decision trees (DT)

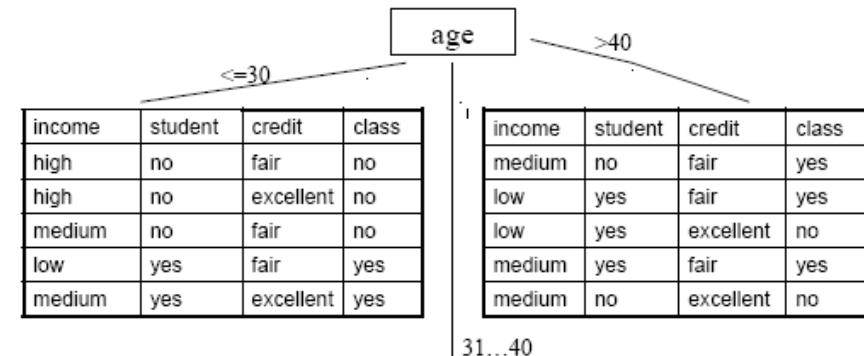
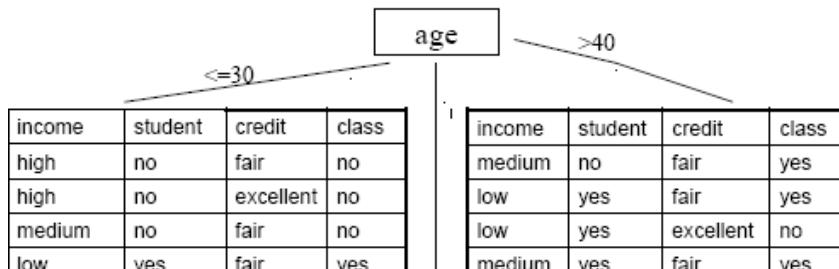
---

- Process → Tree construction (induction)
  - Example

rec	Age	Income	Student	Credit_rating	Buy_computer(CLASS)
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31...40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No

# Intelligent systems – decision trees (DT)

- Process → Tree construction (induction)
  - Example
    - Attribute *age* is selected for the root



income	student	credit	class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

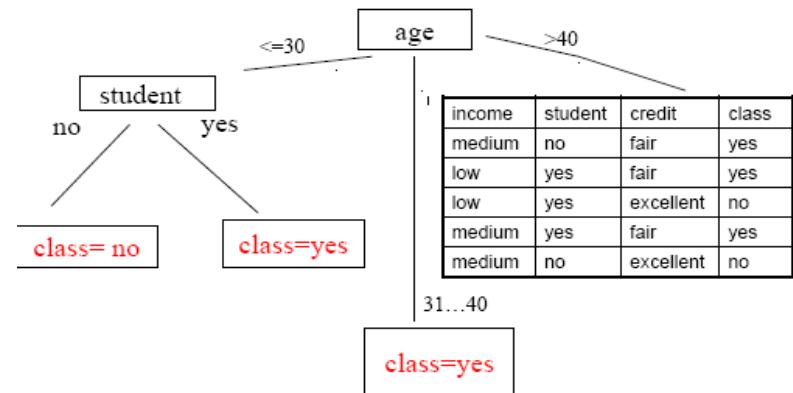
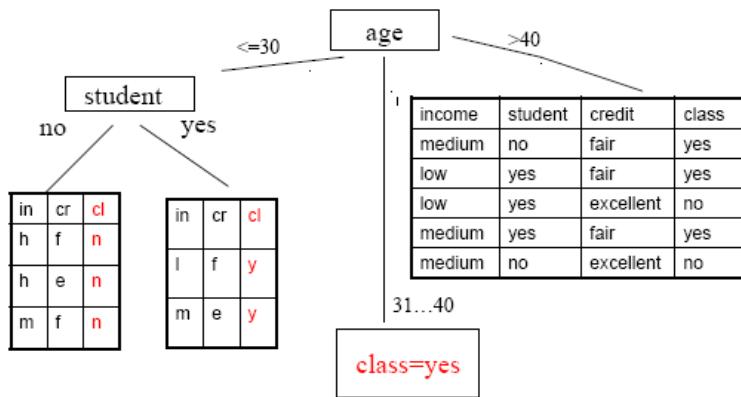
class=yes

# Intelligent systems – decision trees (DT)

## □ Process → Tree construction (induction)

### ■ Example

- Attribute *age* is selected for the root
- Attribute *student* is selected on branch *age*  $\leq 30$

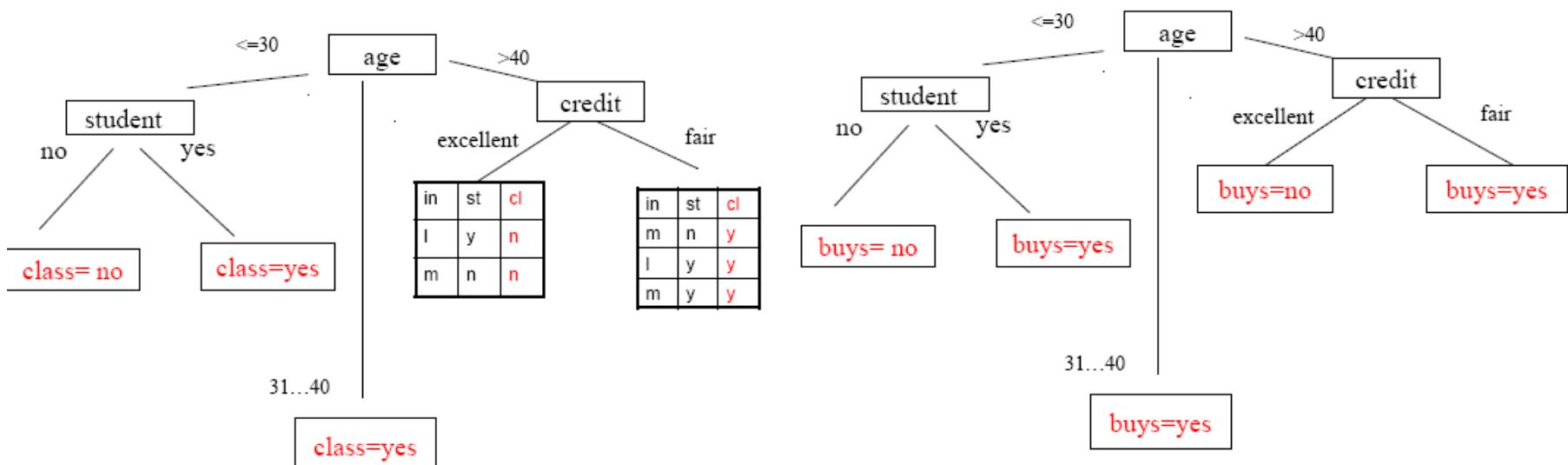


# Intelligent systems – decision trees (DT)

## □ Process → Tree construction (induction)

### ■ Example

- Attribute *age* is selected for the root
- Attribute *student* is selected on branch *age*  $\leq 30$
- Attribute *credit* is selected on branch *age*  $> 40$



# Intelligent systems – decision trees (DT)

---

- Process → tree construction → ID3/C4.5 algorithm
  - Greedy, recursive, top-down, divide-and-conquer

```
generate(D, A){ //D – a partitioning of training data, A – list of attributes
    create a new node N
    if examples from D belong to a single class C then
        node N becomes a leaf and is labeled by C
        return node N
    else
        if A=∅ then
            node N becomes a leaf and is labeled by majority class of D
            return node N
        else
            separation_attribute = AttributeSelection(D, A)
            label node N by separation_attribute
            for all possible values vj of separation_attribute
                let Dj – set of examples from D that have separation_attribute=vj
                if Dj =∅ then
                    add a leaf (to node N) labeled by majority class of D
                else
                    add a node (to node N) return by generate(Dj, A–separation_attribute)
            return node N
}
```

# Intelligent systems – decision trees (DT)

---

- Process → tree construction → ID3/C4.5 algorithm
  - AttributeSelection(D,A) → select the attribute that corresponds to a node (root or internal node)
    - Random
    - Attribute with the fewest/most values
    - Based on a pre-established order
      - Information gain
      - Gain rate
      - Gini index
      - Distance between partitions created by the attribute

# Intelligent systems – decision trees (DT)

---

- Process → tree construction → ID3/C4.5 algorithm → Attribute Selection
  - Information gain
    - An impurity measure
      - 0 (minim) – if all examples belong to the same class
      - 1 (maxim) – if examples are uniformly distributed over classes
    - Based on data entropy
      - Expected number of bits required by coding the class of an element from data
      - Binary classification (2 classes):  $E(S) = - p_+ \log_2 p_+ - p_- \log_2 p_-$ , where
        - $p_+$  – proportion of positive examples in dataset S
        - $p_-$  – proportion of negative examples in dataset S
      - Multi-class classification:  $E(S) = \sum_{i=1, 2, \dots, k} p_i \log_2 p_i$  – data entropy related to target attribute (output attribute), where
        - $p_i$  – proportion of examples from class  $i$  in dataset S
    - Information gain of an attribute
      - How the elimination of attribute  $a$  reduces the dataset's entropy
      - $Gain(S, a) = E(S) - \sum_{v \in \text{valori}(a)} |S_v| / |S| E(S_v)$
      - $\sum_{v \in \text{valori}(a)} |S_v| / |S| E(S_v)$  – expected information

# Intelligent systems – decision trees (DT)

Process → tree construction → ID3/C4.5 algorithm → Attribute Selection

- Information gain
  - Example

	a1	a2	a3	Clasa
d1	mare	roșu	cerc	clasa 1
d2	mic	roșu	pătrat	clasa 2
d3	mic	roșu	cerc	clasa 1
d4	mare	albastru	cerc	clasa 2

$$S = \{d1, d2, d3, d4\} \rightarrow p_+ = 2/4, p_- = 2/4 \rightarrow E(S) = -p_+\log_2 p_+ - p_-\log_2 p_- = 1$$

$$S_{v=mare} = \{d1, d4\} \rightarrow p_+ = 1/2, p_- = 1/2 \rightarrow E(S_{v=mare}) = 1$$

$$S_{v=mic} = \{d2, d3\} \rightarrow p_+ = 1/2, p_- = 1/2 \rightarrow E(S_{v=mic}) = 1$$

$$S_{v=rosu} = \{d1, d2, d3\} \rightarrow p_+ = 2/3, p_- = 1/3 \rightarrow E(S_{v=rosu}) = 0.923$$

$$S_{v=albastru} = \{d4\} \rightarrow p_+ = 0, p_- = 1 \rightarrow E(S_{v=albastru}) = 0$$

$$S_{v=cerc} = \{d1, d3, d4\} \rightarrow p_+ = 2/3, p_- = 1/3 \rightarrow E(S_{v=cerc}) = 0.923$$

$$S_{v=patrat} = \{d2\} \rightarrow p_+ = 0, p_- = 1 \rightarrow E(S_{v=patrat}) = 0$$

$$\text{Gain}(S, a) = E(S) - \sum_{v \in \text{values}(a)} |S_v| / |S| E(S_v)$$

$$\text{Gain}(S, a_1) = 1 - (|S_{v=mare}| / |S| E(S_{v=mare}) + |S_{v=mic}| / |S| E(S_{v=mic})) = 1 - (2/4 * 1 + 2/4 * 1) = 0$$

$$\text{Gain}(S, a_2) = 1 - (|S_{v=rosu}| / |S| E(S_{v=rosu}) + |S_{v=albastru}| / |S| E(S_{v=albastru})) = 1 - (3/4 * 0.923 + 1/4 * 0) = 0.307$$

$$\text{Gain}(S, a_3) = 1 - (|S_{v=cerc}| / |S| E(S_{v=cerc}) + |S_{v=patrat}| / |S| E(S_{v=patrat})) = 1 - (3/4 * 0.923 + 1/4 * 0) = 0.307$$

# Intelligent systems – decision trees (DT)

---

- Process → tree construction → ID3/C4.5 algorithm → Attribute Selection
  - Gain rate
    - Penalises an attribute by integrating a new term – *split information* – that depends on spreading degree and on uniformity degree of separation
      - *Split information* – entropy related to possible values of attribute a
      - $S_v$  – proportion of examples from dataset S that have attribute a with value v

# Intelligent systems – decision trees (DT)

---

## □ Process

- Tree construction
- Using the tree as a problem solver
  - Main idea
    - Extract the rules from the constructed tree
      - IF  $age = "<=30"$  AND  $student = "no"$  THEN  $buys\_computer = "no"$
      - IF  $age = "<=30"$  AND  $student = "yes"$  THEN  $buys\_computer = "yes"$
      - IF  $age = "31...40"$  THEN  $buys\_computer = "yes"$
      - IF  $age = ">40"$  AND  $credit\_rating = "excellent"$  THEN  $buys\_computer = "no"$
      - IF  $age = ">40"$  AND  $credit\_rating = "fair"$  THEN  $buys\_computer = "yes"$
    - Use the rules for classifying the test data (new data)
      - Let  $x$  a data without class → rules can be written as predicates
      - IF  $age(x, <=30)$  AND  $student(x, no)$  THEN  $buys\_computer(x, no)$
      - IF  $age(x, <=30)$  AND  $student(x, yes)$  THEN  $buys\_computer(x, yes)$

# Intelligent systems – decision trees (DT)

---

## □ Process

- Tree construction
- Using the tree as a problem solver
  - Difficulties
    - *Underfitting* → DT constructed on training data is too simple → large classification error during training and testing
    - *Overfitting* → DT constructed on training data matches the training data, but it cannot generalize new data
  - Solutions
    - Pruning → remove some branches (un-useful, redundant) → small tree
    - Cross-validation

# Intelligent systems – decision trees (DT)

---

## ❑ Process

- Tree construction
- Using the tree as a problem solver
- Pruning

### ❑ Why?

- After the DT is constructed, classification rules are extracted in order to represent the knowledge as if-then rules (easy to understand)
- A rule is created by traversing the DT from root to a leaf
- Each pair (attribute, value) – (node, edge) – is a conjunction in the premise of the rule (if part), except the last node of the path that is a leaf and represents the consequence (output, then part) of the rule

### ❑ Typology

- *pre-pruning*
  - Increasing the tree is stopped during construction by stopping the division of nodes that become leaf labeled by majority class of examples from that node
- *post-pruning*
  - After the DT is constructed, eliminate the branches of some nodes that become leaf → classification error reduces (on testing data)

# Intelligent systems – decision trees (DT)

---

## ❑ Tools

- <http://webdocs.cs.ualberta.ca/~aixplore/learning/DecisionTrees/Applet/DecisionTreeApplet.html>
- WEKA → J48
- <http://id3alg.altervista.org/>
- <http://www.rulequest.com/Personal/c4.5r8.tar.gz>

## ❑ Biblio

- <http://www.public.asu.edu/~kirkwood/DAStuff/decisiontrees/index.html>

# Intelligent systems – decision trees (DT)

---

## □ Advantages

- Easy to understand and interpret
- Can use nominal or categorized data
- Decision logic can be easily followed (rules are visible)
- Works better with large data

## □ Disadvantages

- Instability → change the training data
- Complexity → representation
- Difficult to use
- The DT construction is expensive
- The DT construction requires a lot of information

# Intelligent systems – decision trees (DT)

---

## □ Difficulties

- There can be more trees
  - To small
  - With a better accuracy (easy to be read and with good performances)
  - Identify the best tree → NP-problem
- Select the best tree
  - Heuristic algorithms
  - ID3 → the smallest tree
    - Occam theorem: “always choose the simplest explanation”
- Continuous attributes
  - Range splitting
    - How many intervals?
    - How large intervals?
- To large trees
  - Pre - pruning → stops to construct the tree earlier
  - Post-pruning → remove some branches

# Review



## □ Automatic learning systems

### ■ Machine Learning – ML

- Supervised learning → annotated train data (by label from a predefined set) and test data have to be annotated by using the learnt model (by one of the known labels)
- Unsupervised learning → not-annotated train data; a labeling model has to be learnt in order to annotate the test data; the set of labels is unknown before training

### ■ Systems

- Decision trees
  - Each internal node → attribute
  - Each branch of a node (attribute) → value of that attribute
  - Each leaf → class (label) – contains all data from that class

# Next lecture

---

## A. Short introduction in Artificial Intelligence (AI)

### A. Solving search problems

- A. Definition of search problems
- B. Search strategies
  - A. Uninformed search strategies
  - B. Informed search strategies
  - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
  - D. Adversarial search strategies

## C. Intelligent systems

- A. Rule-based systems in certain environments
- B. Rule-based systems in uncertain environments (Bayes, Fuzzy)

### C. Learning systems

- A. Decision Trees
- Artificial Neural Networks
  - A. Support Vector Machines
  - Evolutionary algorithms

- D. Hybrid systems

# Next lecture – useful information

---

- Chapter VI (19) of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 8 of *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapters 12 and 13 of *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter V of *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Chapter 4 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

- 
- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean - [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)