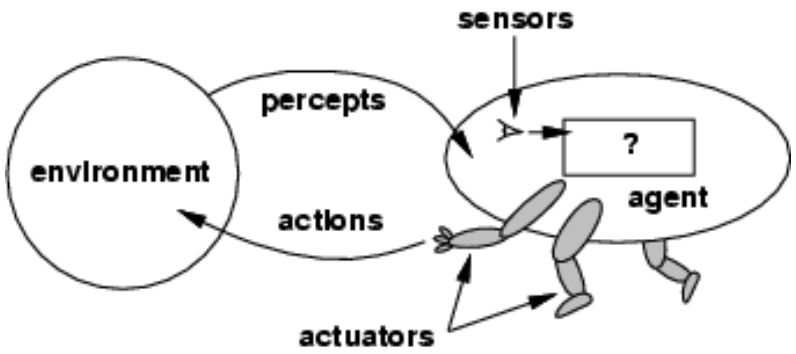
Artificial Intelligence -> Intelligent Agents
aima.cs.berkeley.edu

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
- Human agent:
  - eyes, ears, and other organs for sensors;
  - hands, legs, mouth, and other body parts for actuators
- Robotic agent:
  - cameras and infrared range finders for sensors;
  - various motors for actuators

Environment/Agent



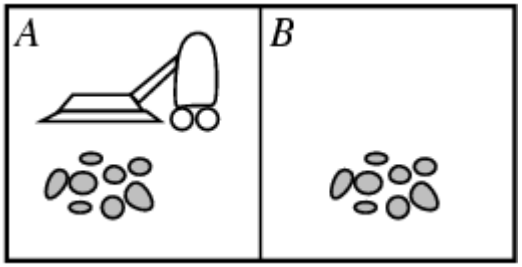
- The agent function maps from percept histories to actions:

[f: $P^* \rightarrow A$]

- The agent program runs on the physical architecture to produce f

- agent = architecture + program

Vacuum-cleaner world



- Percepts: location and contents, e.g., [A,Dirty];
- Actions: *Left*, *Right*, *Suck*, *NoOp*;
- Table of Percepts-> Actions

Rational agent

- Rational Agent: For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

PEAS

- PEAS: Performance measure, Environment, Actuators, Sensors
- Must first specify the setting for intelligent agent design;
- Consider, e.g., the task of designing an automated taxi driver:
  - Performance measure: Safe, fast, legal, comfortable trip, maximize profits;
  - Environment: Roads, other traffic, pedestrians, customers;
  - Actuators: Steering wheel, accelerator, brake, signal, horn;
  - Sensors: Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard;

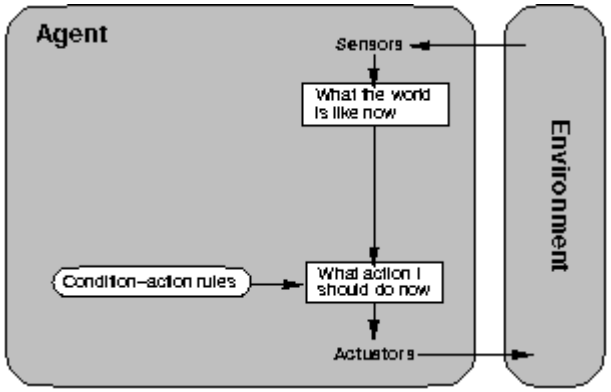
- An agent is completely specified by the agent function mapping percept sequences to actions
- One agent function is rational;
- Aim: find a way to implement the rational agent function concisely;

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

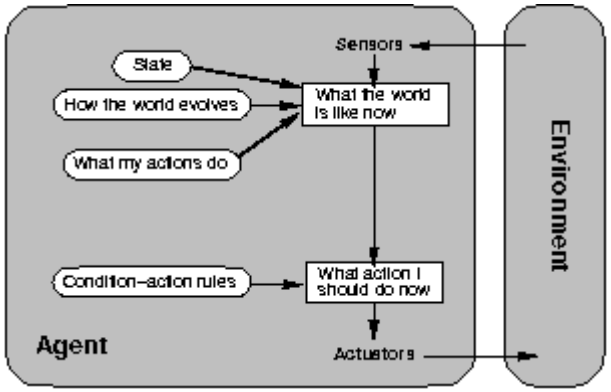
Figure 2.4 PEAS description of the task environment for an automated taxi.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, minimize costs, lawsuits	Patient, hospital, staff	Display questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display categorization of scene	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Maximize purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Maximize student's score on test	Set of students, testing agency	Display exercises, suggestions, corrections	Keyboard entry

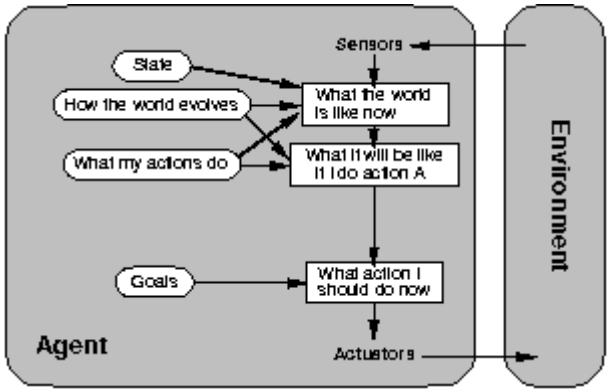
Figure 2.5 Examples of agent types and their PEAS descriptions.



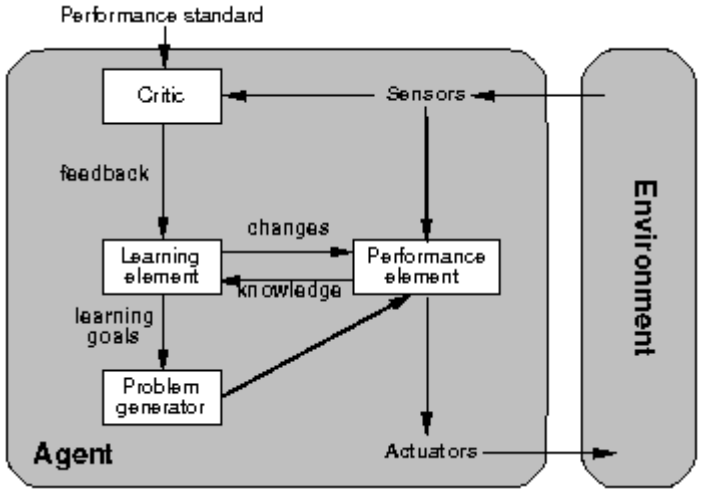
Simple reflex agent



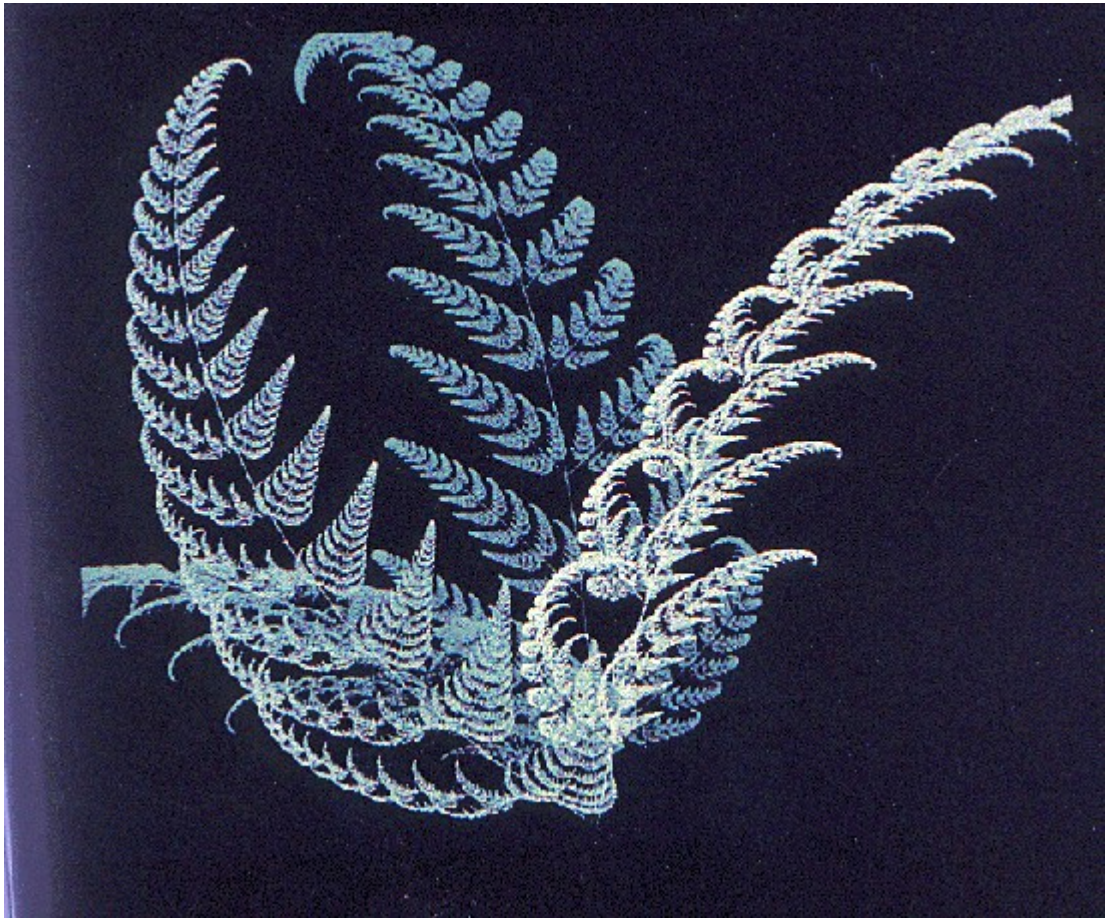
Reflex +state agent



Goal-based agent



Learning agent



http://www.mathcurve.com/fractals/fougere/fougere.shtml

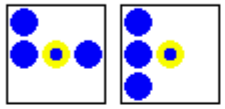
Quelle est la vraie fougère et la fougère fractale en 3D?

Le jeu de la vie

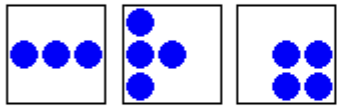
http://math.com/students/wonders/life/life.html

To apply one step of the rules, we count the number of live neighbors for each cell. What happens next depends on this number.

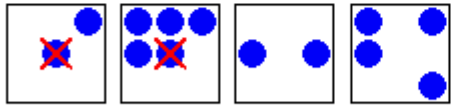
- A dead cell with exactly three live neighbors becomes a live cell (birth).



- A live cell with two or three live neighbors stays alive (survival).



- In all other cases, a cell dies or remains dead (overcrowding or loneliness).



Note: The number of live neighbors is always based on the cells *before* the rule was applied. In other words, we must first find all of the cells that change before changing any of them. Sounds like a job for a computer!

The most common still life is called the block. It is simply a 2x2 square of live cells:

You will see it appear many times as you run the R-pentomino. Every live cell has exactly three neighbors, but no dead cell has more than two neighbors.

Some other still lifes you will see are:


beehive


boat


ship


loaf

If you've been following the R-pentomino in the applet, you've probably noticed that some of the objects are moving. This was one of the most exciting early discoveries in Life. These common moving patterns, called gliders, consist of just 5 cells:

Run 6 Gliders

(Click **Go** to start the pattern) Follow this pattern along for four steps in the applet. Or, to really understand what's going on, apply the rules by hand as Conway did. You will see that after four steps, it looks just like it did when it started. The only difference is that it is shifted along a diagonal. Repeat this process, and you have a moving Life object, which in general we call a spaceship. Remember, the rules said nothing about movement; moving patterns just appear. This is a simple but convincing demonstration of emergent complexity.

The Queen Bee Shuttle

A very important pattern makes a brief appearance after running the R-pentomino for 774 steps:

Different kinds of agent we deal with :

- Searching Agent
- Problem-solving agent
- Logical/Knowledge-based agent -> Probabilistic Agent
- Learning agent

Searching Agent

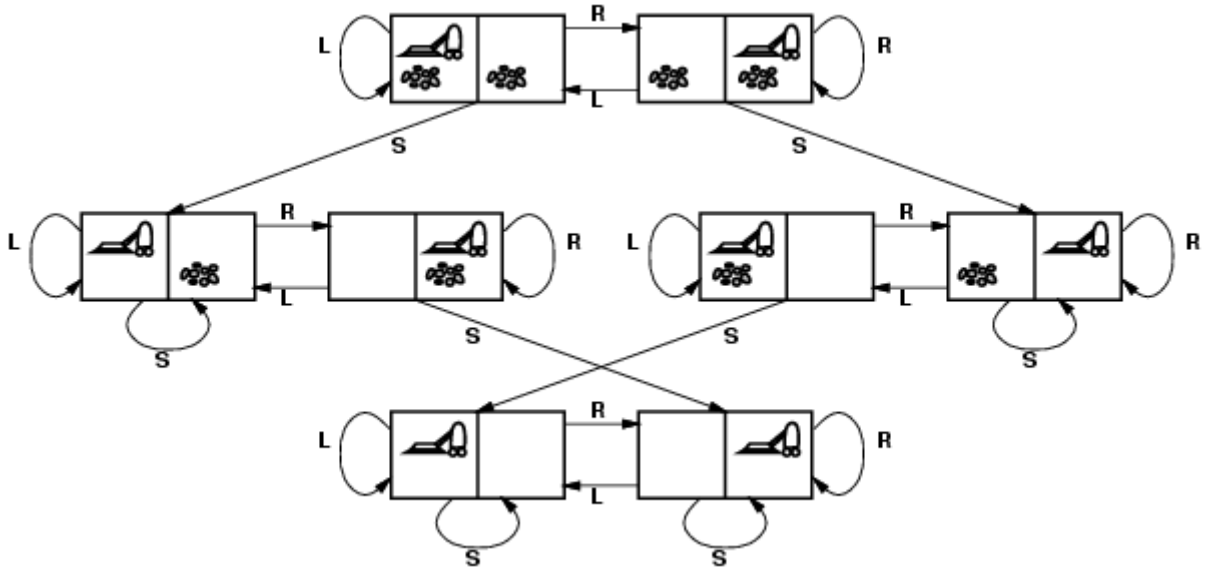
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabytes
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Figure 3.11 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 10,000 nodes/second; 1000 bytes/node.

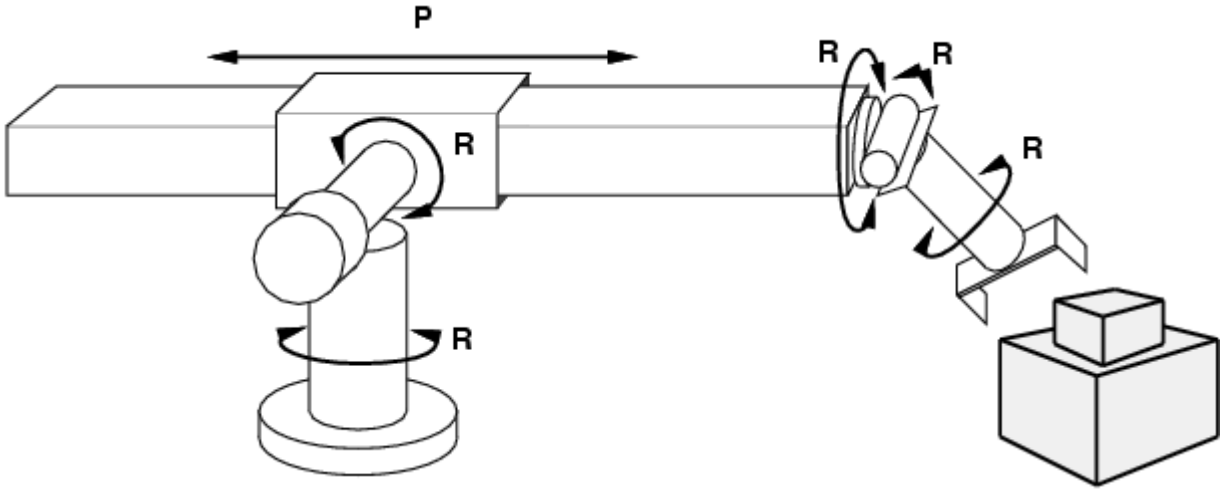
Problem-solving Agent

Tree-search and Vacuum world state space graph



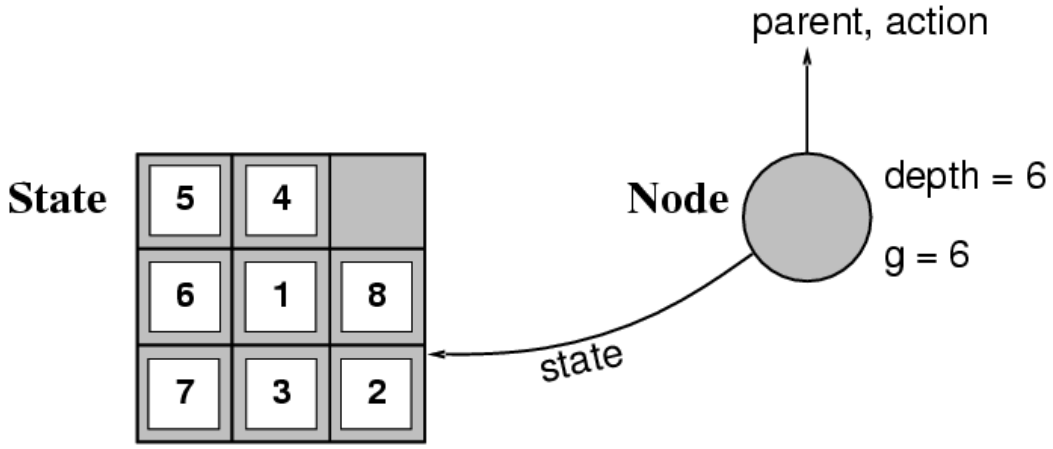
- states? integer dirt and robot location
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations
- path cost? 1 per action

Robotic assembly



- states?: real-valued coordinates of robot joint angles parts of the object to be assembled
- actions?: continuous motions of robot joints
- goal test?: complete assembly
- path cost?: time to execute

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost *g(x)*, depth



- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

Breadth-first search

- Complete? Yes (if b is finite);
- Time? $1+b+b^2+b^3+\ldots+b^d+b(b^d-1) = O(b^{d+1})$;
- Space? $O(b^{d+1})$ (keeps every node in memory);
- Optimal? Yes (if cost = 1 per step);

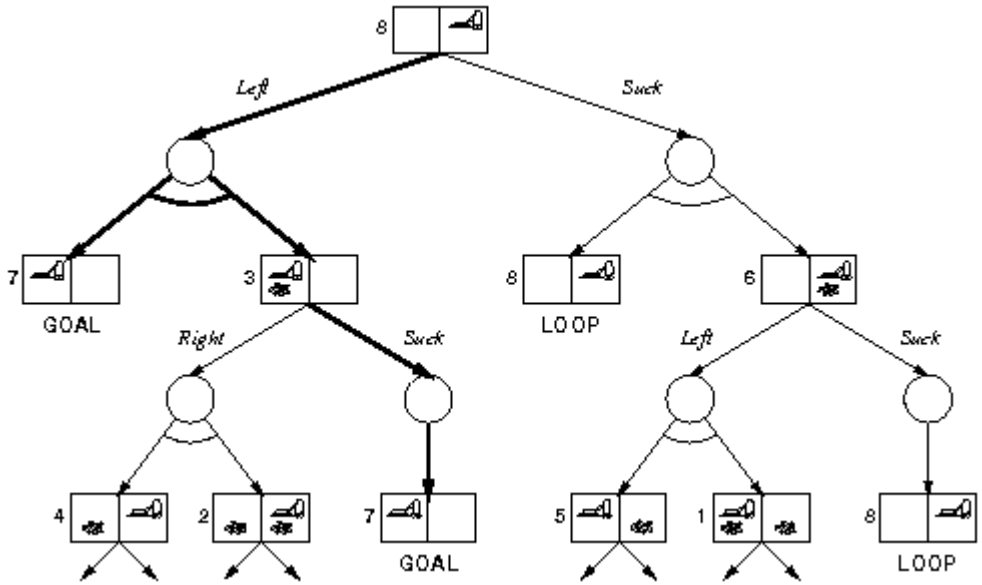
- Space is the bigger problem (more than time);

Depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path;
    → complete in finite spaces
- Time? $O(b^m)$: terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first;
- Space? $O(bm)$, i.e., linear space!
- Optimal? No.

Planning Agent

Google :
STRIPS language, graphplan



Local search algorithms / Metaheuristics Optimisation

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution;

- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens

- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it;

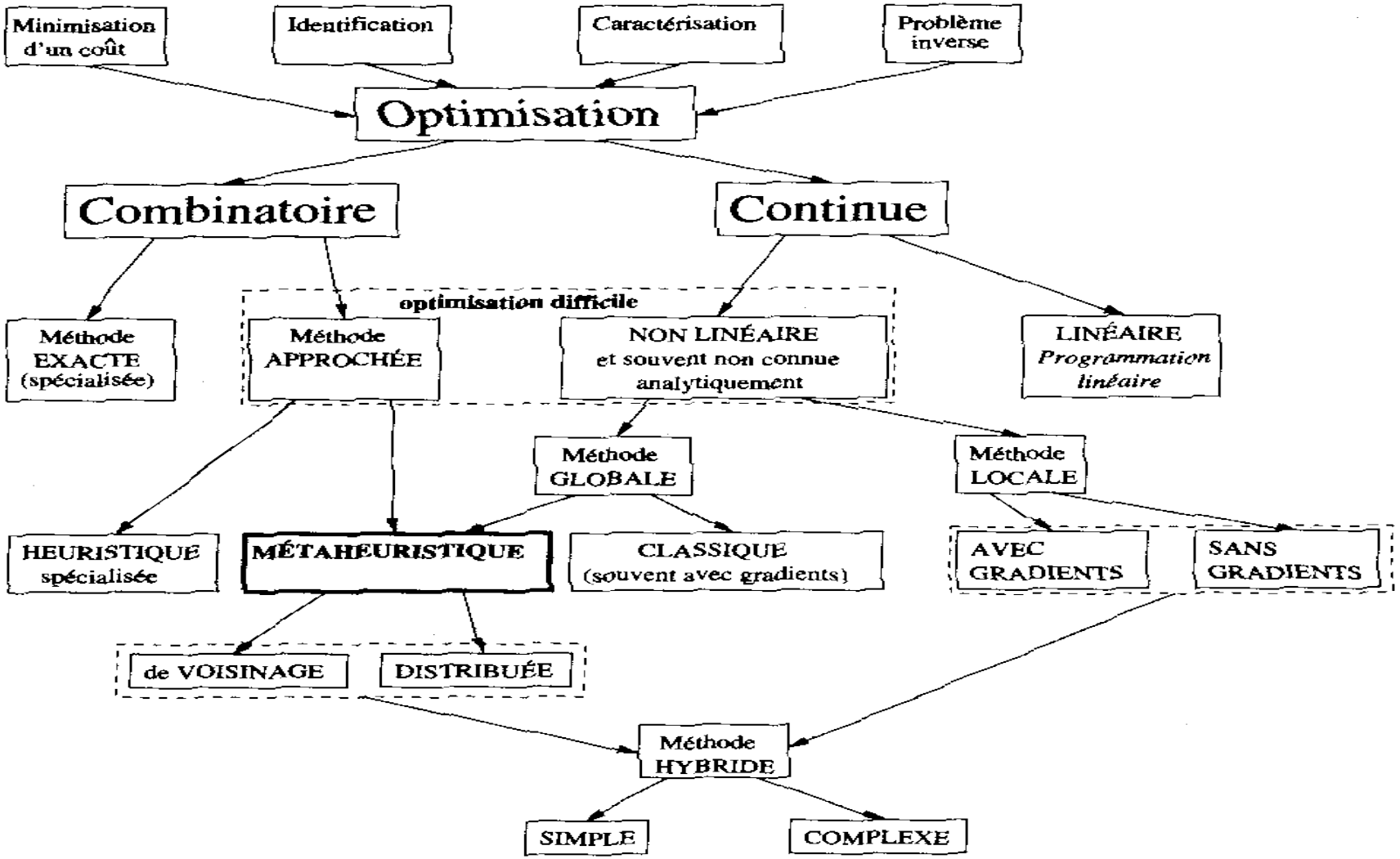
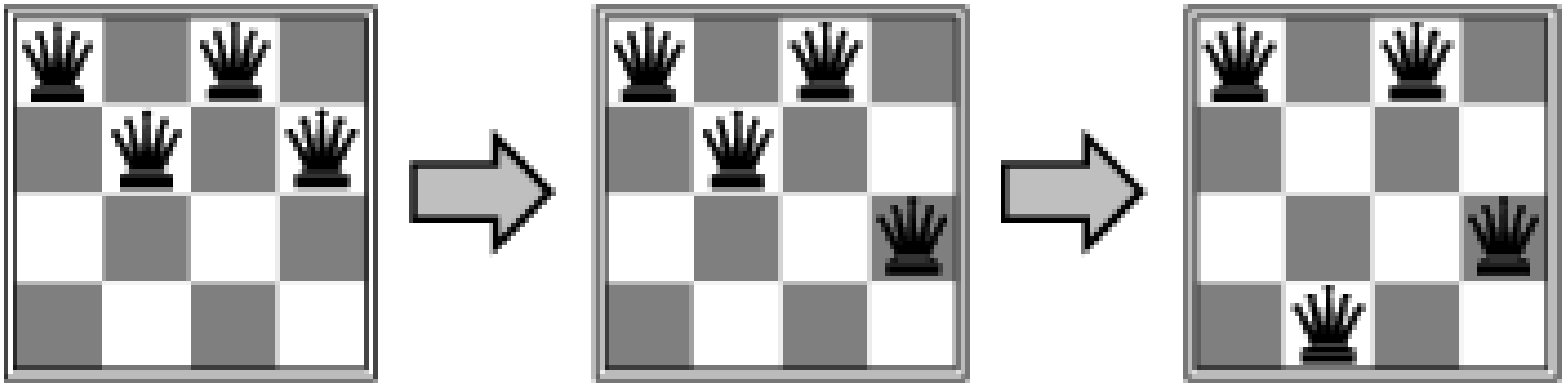


FIG. 11 – *Classification générale des méthodes d'optimisation mono-objectif.*

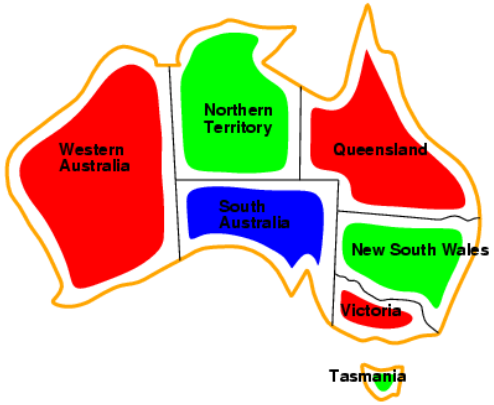
Example: *n*-queens

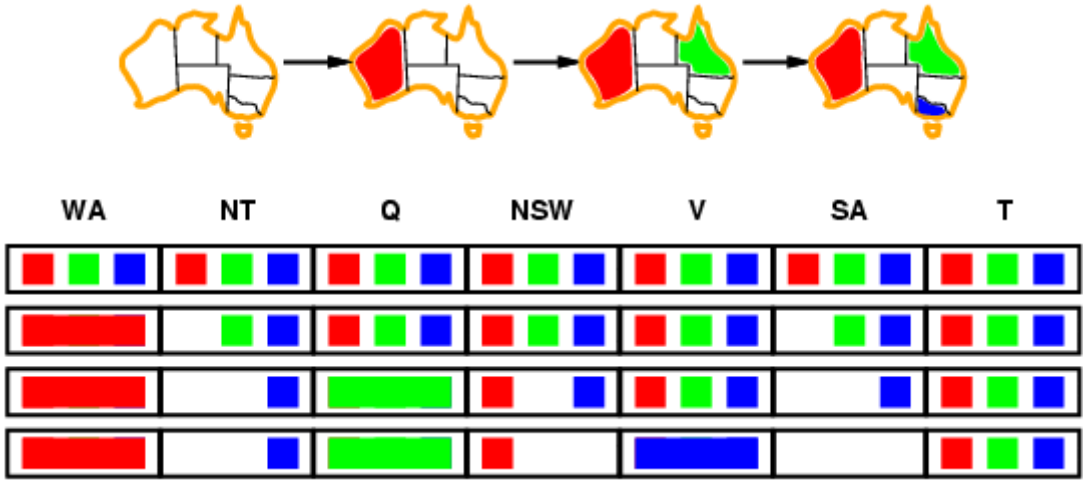
- Put *n* queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



•CSP (linear programming)

Example: Map-Coloring

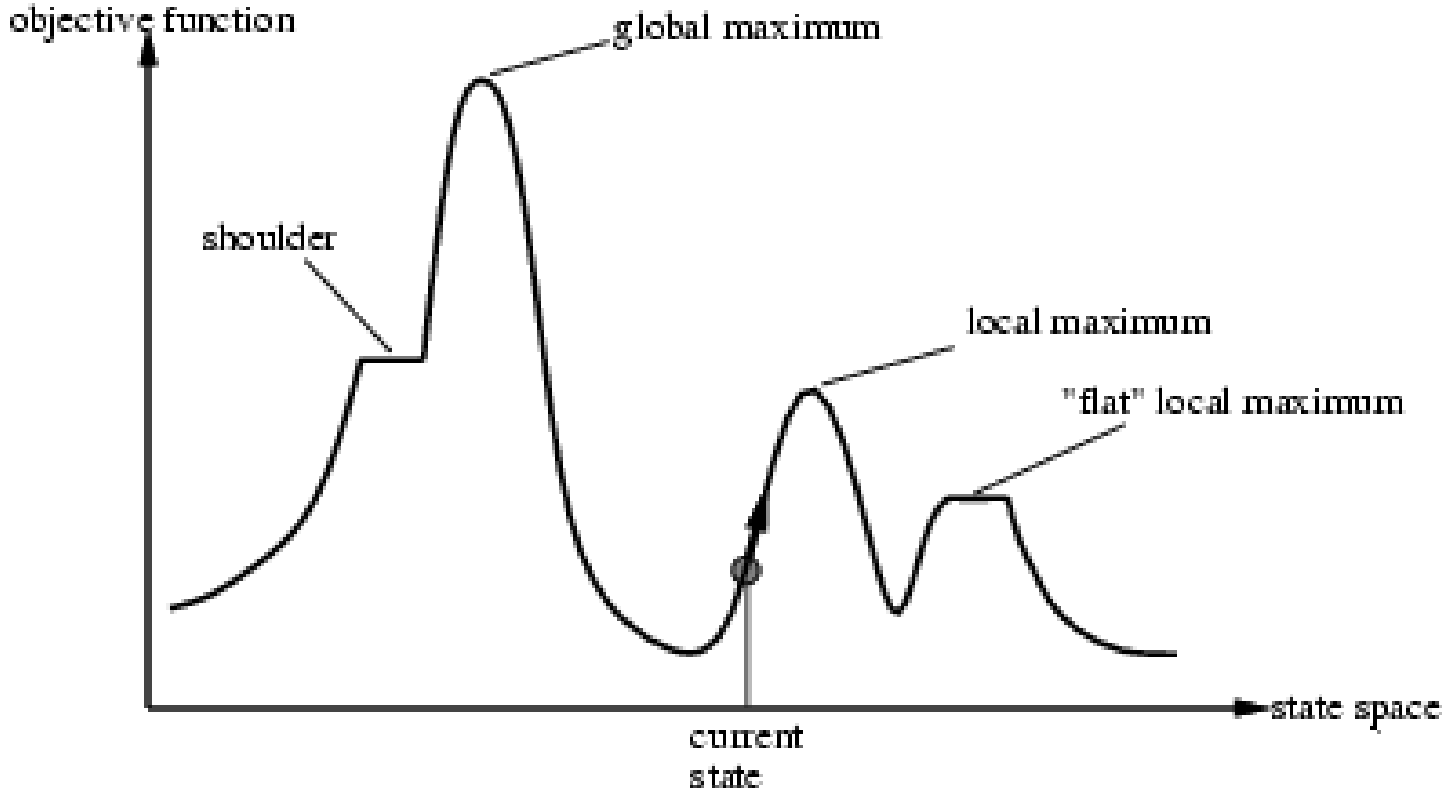




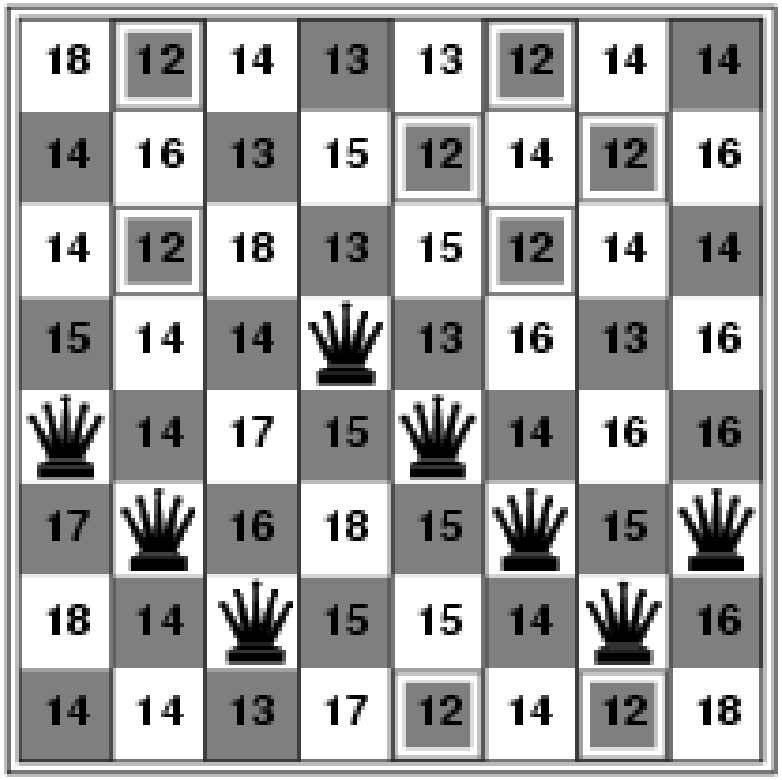
- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

•Hill-climbing search

Problem: depending on initial state, can get stuck in local maxima

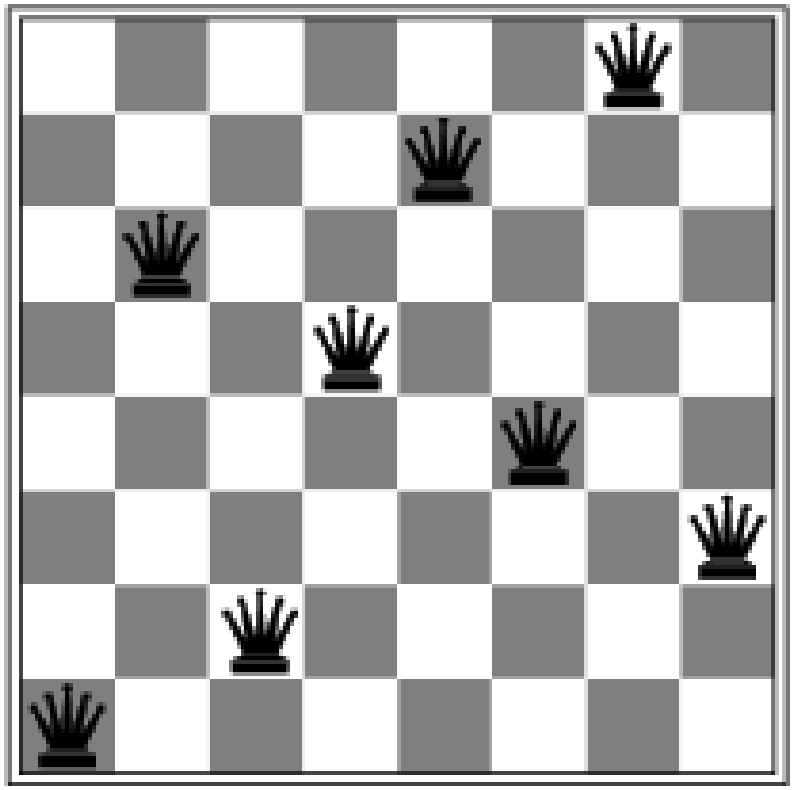


Hill-climbing search: 8-queens problem



- h = number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$ for the above state

Hill-climbing search: 8-queens problem



- A local minimum with $h = 1$

•Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

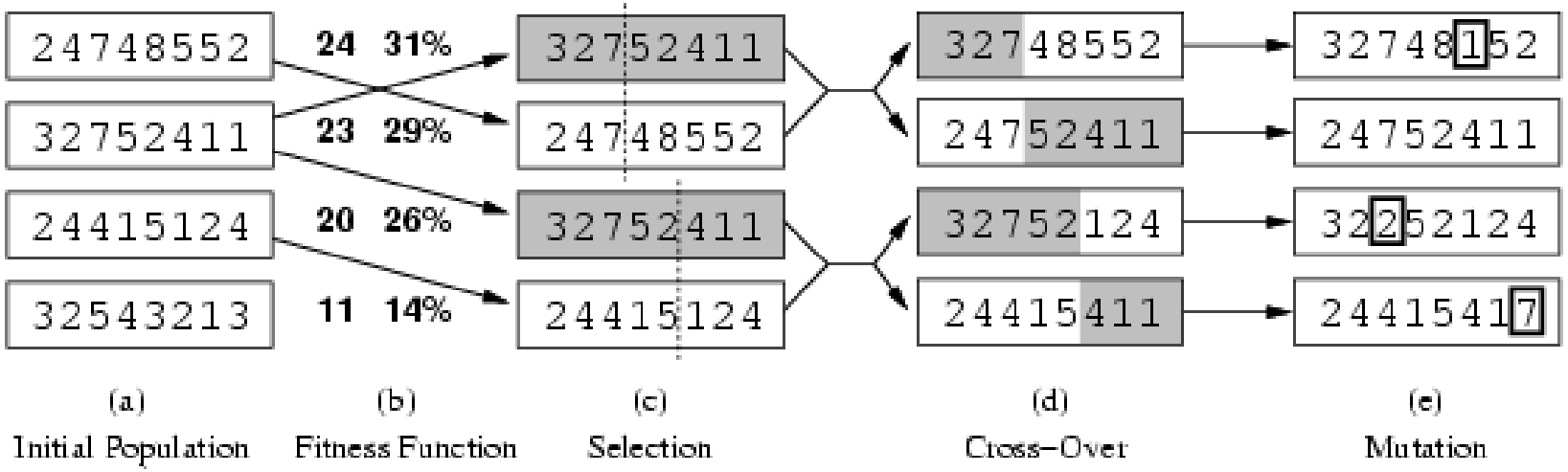
Properties of simulated annealing search

- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc

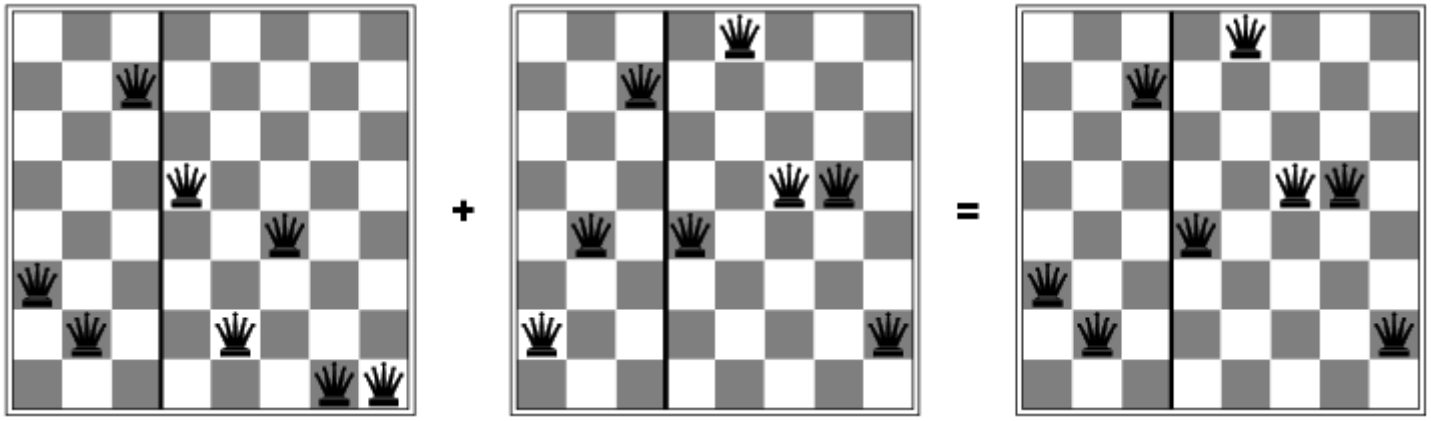
- Local beam search
- Genetic algorithms

Genetic algorithms

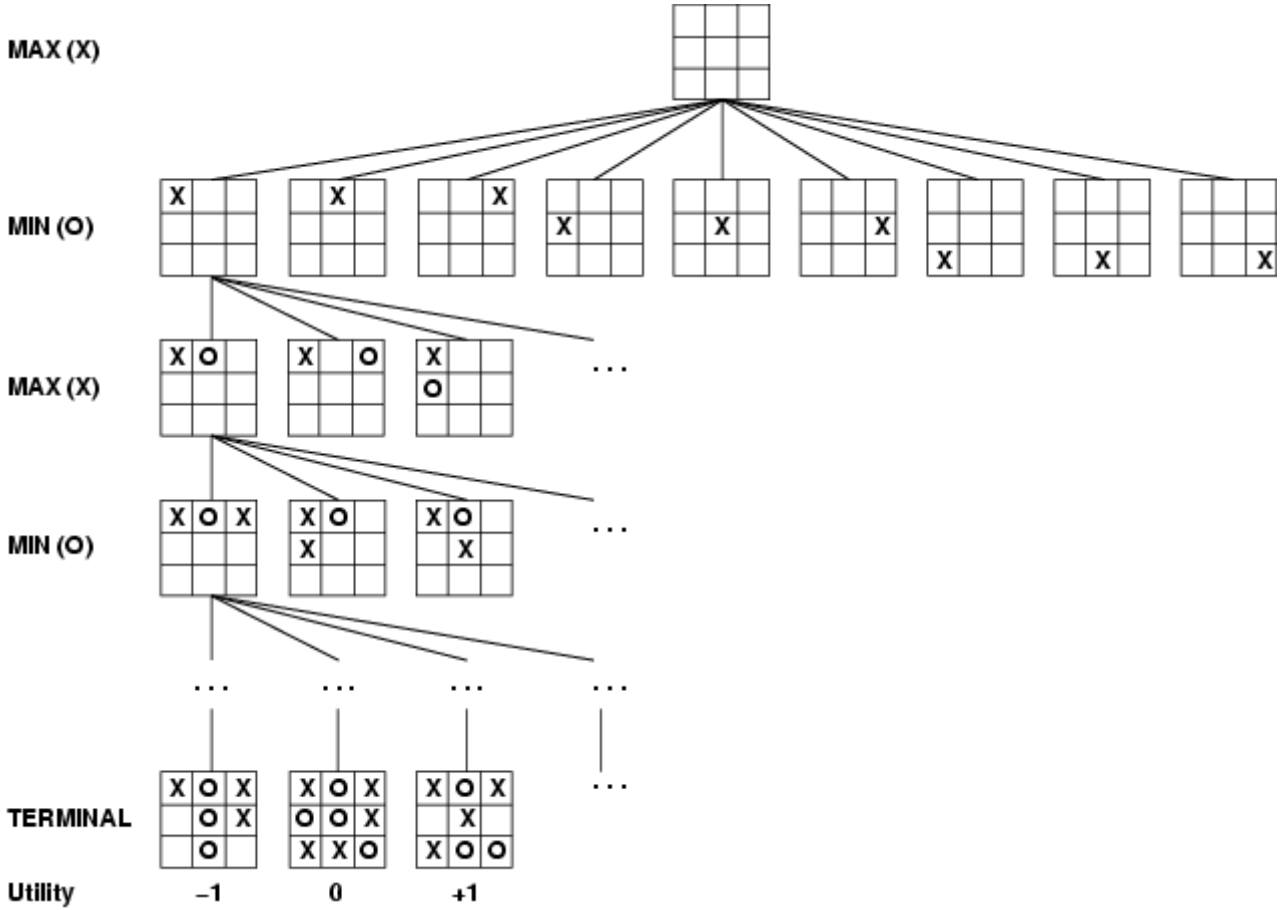


- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithms



Adversarial search Agent/ Games



Properties of minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first exploration)

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  → exact solution completely infeasible

Properties of α-β

- With "perfect ordering," time complexity = $O(b^{m/2})$

Resource limits

Suppose we have 100 secs, explore 10^4 nodes/sec

→ 10^6 nodes per move;

Standard approach:

- cutoff test:

  e.g., depth limit (perhaps add quiescence search);

- evaluation function

  = estimated desirability of position;

•For chess, typically linear weighted sum of features

$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

•e.g., $w_1 = 9$ with

$f_1(s)$ = (number of white queens) – (number of black queens), etc.

• Cutting off search

MinimaxCutoff is almost identical to *MinimaxValue*

Does it work in practice?

$b^m = 10^6$, b=35 → m=4;

4-ply lookahead is a hopeless chess player!

- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 12-ply ≈ Deep Blue, Kasparov

Deterministic games in practice

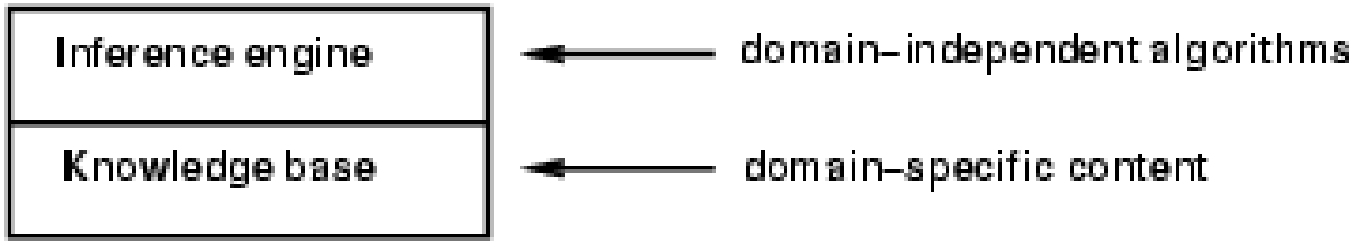
- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.

- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

- Othello: human champions refuse to compete against computers, who are too good.

- Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Logical Agent / Knowledge-based Agent



- Knowledge base = set of sentences in a formal language
- Declarative approach to build an agent (or other system):
  - `Tell` it what it needs to know
- Then it can `Ask` itself what to do - answers should follow from the KB
- Agents can be viewed at the knowledge level
  i.e., what they know, regardless of how implemented
- Or at the implementation level
  - i.e., data structures in KB and algorithms that manipulate them

- The agent must be able to:
  - Represent states, actions, etc.
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

Google :
CLASSIC, CLIPS, PROLOG

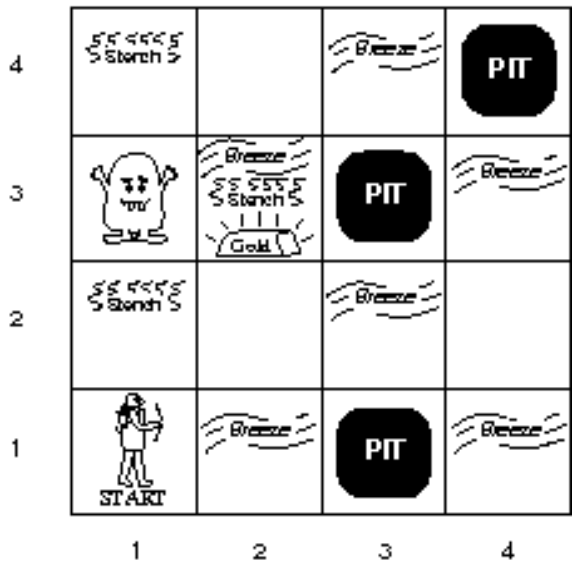
Syntaxe, Sémantique, Modèles d'interprétation du monde

« Rio est la capitale de Suisse », Vrai , Faux ?
« La Suisse est en Europe »
Rio est en Europe ?

Table de vérité

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Sémantique d'un langage : vérité de toutes phrases par rapport à tout monde possible.
1 monde possible = 1 modèle
En logique standard, tout phrase doit être ou Vraie ou Fausse
En arithmétique, les modèles de la phrase « x+y=4 » sont (x=1,y=3), (x=2,y=2)....



1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

(a)

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

(b)

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(a)

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

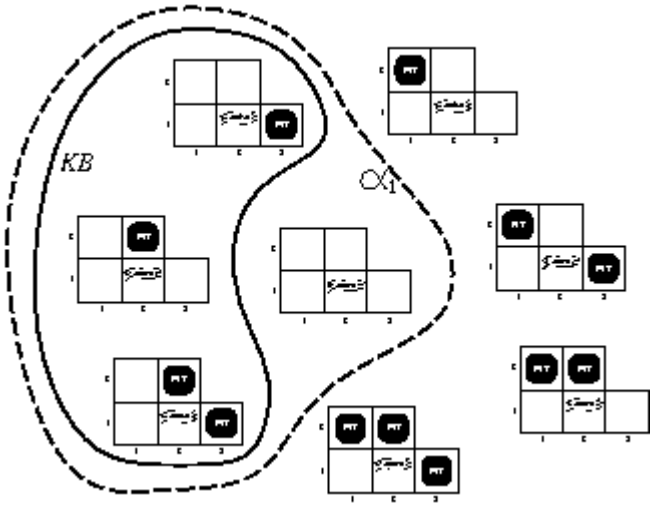
1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

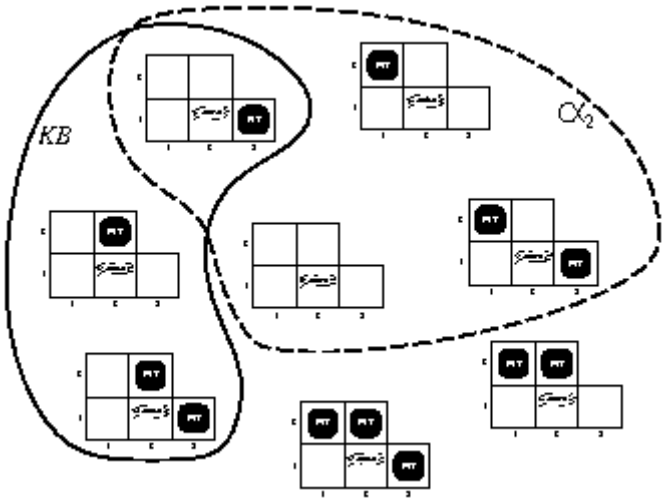
(b)

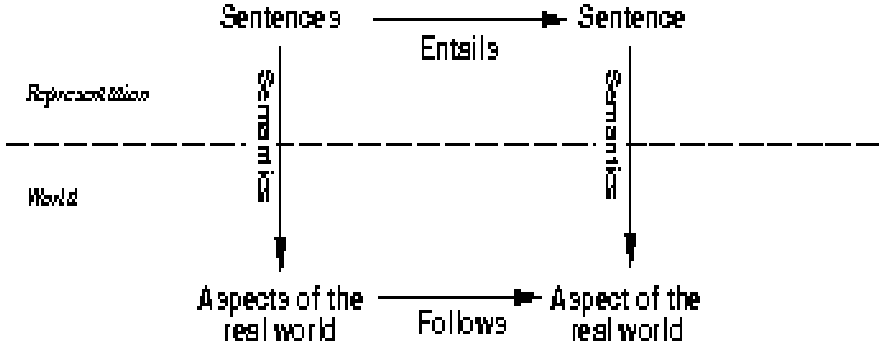
Model checking method.
KB|=alpha1 ? Conclusion logique
Par exemple, alpha1=non $P_{1,2}$

KB :
R1: non P11
R2 : B11<=> P12 ou P21
R3 : B21 <=>P11 ou P22 ou P31
R4 : non B11
R5 : B21

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
$false$	$false$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$false$	$false$	$false$	$false$	$false$	$true$	$false$	$true$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$false$	$true$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$true$	$false$	$false$	$false$	$false$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$false$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$true$	$false$	$false$	$false$	$true$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$true$	$true$	$true$	$true$	$true$	$true$	$true$	$false$	$false$







Existe-t-il un algorithme capable de dériver alpha1 à partir de KB :
KB |-- alpha1 ? Si oui, plus rapide qu'énumérer les modèles.
On est sur que si KB et non alpha1 débouche sur une contradiction alors KB |-- alpha1et KB |= alpha1 : syntaxe et sémantique s'accordent en logique des prédicats du premier ordre, et complétude par la méthode de résolution par réfutation
alors que essayer KB |-- alpha1 directement pas complet -> PROLOG

Inference-based agents in the wumpus world : order 0

A wumpus-world agent using propositional logic:

$\neg P_{1,1}$

$\neg W_{1,1}$

$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$

$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$

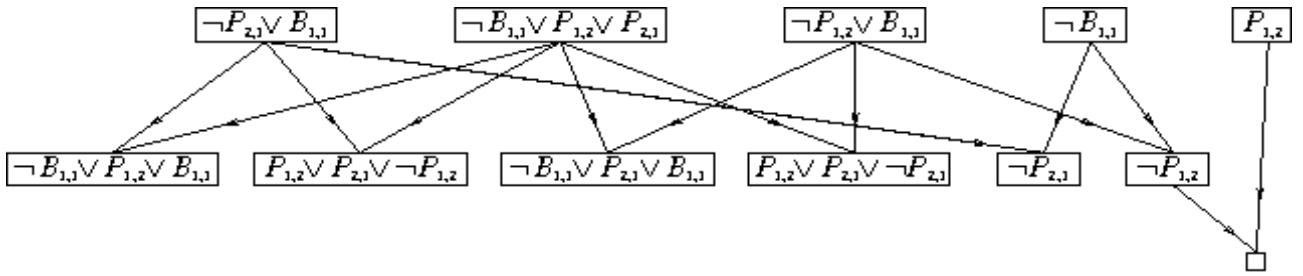
$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$

$\neg W_{1,1} \vee \neg W_{1,2}$

$\neg W_{1,1} \vee \neg W_{1,3}$

…

$\Rightarrow$ 64 distinct proposition symbols, 155 sentences



Knowledge base for the wumpus world : order 1

- Perception : $\forall t,s,b\ \text{Percept}([s,b,\text{Glitter}],t) \Rightarrow \text{Glitter}(t)$
- Reflex : $\forall t\ \text{Glitter}(t) \Rightarrow \text{BestAction}(\text{Grab},t)$

$\forall x,y,a,b\ \textit{Adjacent}([x,y],[a,b]) \Leftrightarrow$
$[a,b] \in \{[x+1,y], [x-1,y],[x,y+1],[x,y-1]\}$

Properties of squares:

$\forall s,t\ \textit{At}(\text{Agent},s,t) \wedge \text{Breeze}(t) \Rightarrow \text{Breezy}(t)$

Squares are breezy near a pit:

- Diagnostic rule---infer cause from effect

  $\forall s\ \text{Breezy}(s) \Rightarrow \exists r\ \text{Adjacent}(r,s) \wedge \text{Pit}(r)$

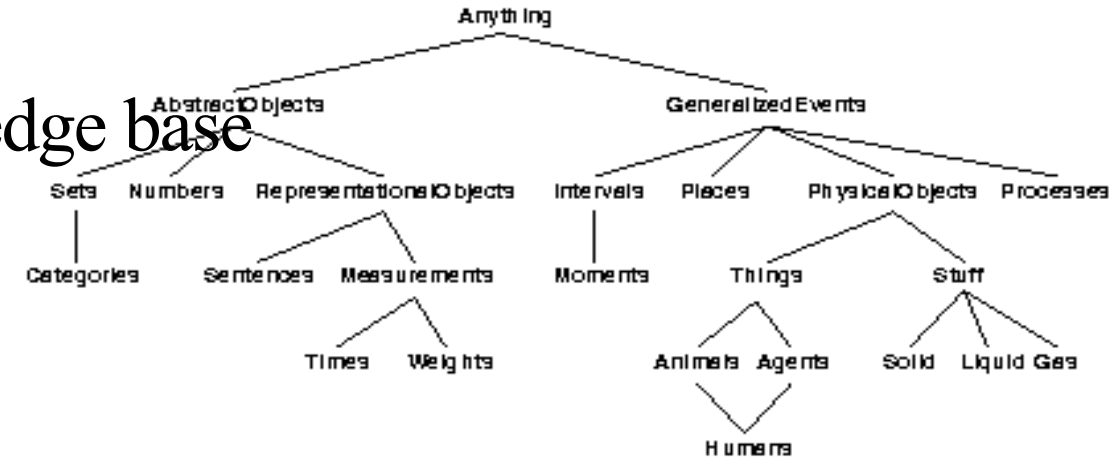
- Causal rule---infer effect from cause

  $\forall r\ \text{Pit}(r) \Rightarrow [\forall s\ \text{Adjacent}(r,s) \Rightarrow \text{Breezy}(s)]$

  ...

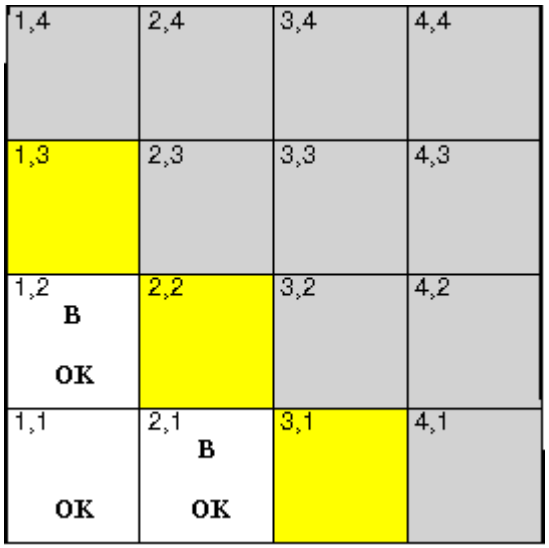
Knowledge engineering in FOL

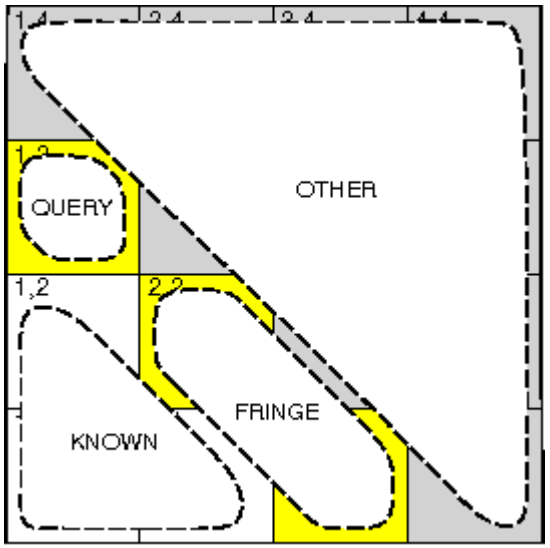
1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

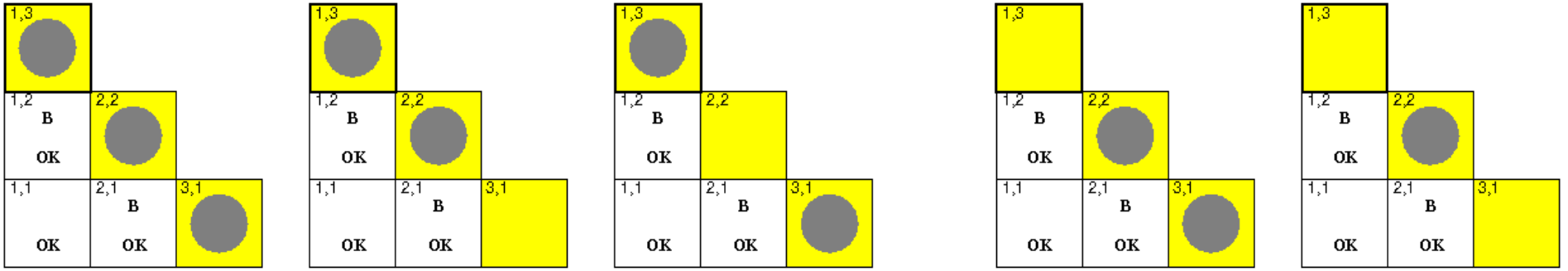


Ontology

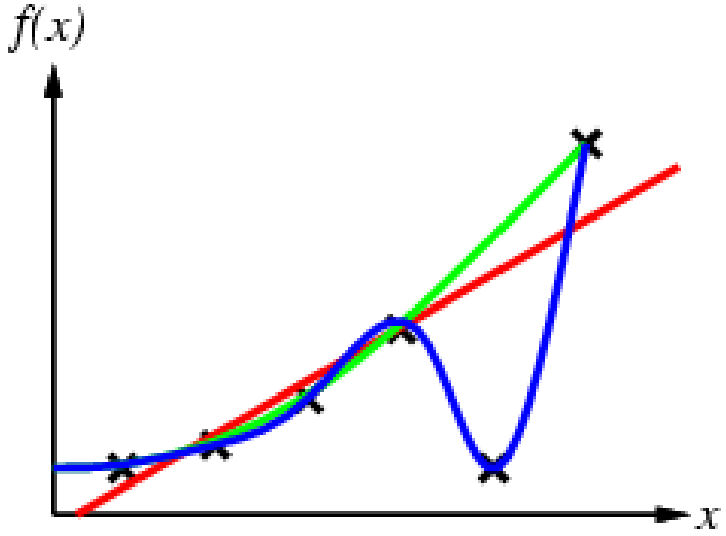
Probabilistic Agent







Learning Agent



- Inductive learning or supervised
- Pattern Recognition
- Data Mining