# Brief Article

The Author

October 14, 2014

## Contents

# 1 Lecture 1: Introduction to Artificial Intelligence

## 1.1 What is Intelligence?

Something is intelligent if it can communicate, it has internal knowledge, it has world knowledge, it has intentions and plans to fulfil these intentions and it has creativity. AI is the study of mental faculties through the use of computational models. The goals of

AI practitioners is to find out about the nature of intelligence and to build an intelligent machine, that is, to build a machine that thinks like a human (thinks rationally) and acts like a human (acts rationally).

In 1950, Turing suggested that the major components of AI are knowledge, reasoning, language understanding and learning.

### 1.2 Rational Behaviour

Rational Behaviour is doing the right thing, that which is expected to maximise goal achievement, given the available information.

### 1.3 Rational Agents

A agent is an entity that perceives and acts. In abstract form, an agent is a function that maps percept histories to actions: $f : P \rightarrow A$. For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance. There is a caveat though. Computational limitations make perfect rationality unachievable. Therefore we must design the best program for a given machine's resources.

### 1.4 Autonomous Agency

Autonomy is the ability to operate independently, while agency is a internal goal structure and external behaviour which generally serves to satisfy a goal structure. The requirements of autonomous agency are:

- Pragmatics
- Generalisation and specialisation
- Incremental learning
- Goal-driven learning
- Defeasibility - open in principle to revision, valid objection, forfeiture, or annulment.
- Uncertainty

## 2 Lecture 2: Intelligent Agents

### 2.1 Agents

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through actuators. For a human agent, sensors would

be our eyes or ears, while our actuators would be our hands, legs, mouth and other body parts. For a robot they would be cameras and motors, respectively.
An agent function maps from percept histories to actions: $f : P \rightarrow A$
The agent program runs on the physical architecture to produce this function.
The agent itself is a combination of the architecture and the program.

## 2.2 Rationality and Rational Agents

Rationality depends on a performance measure, the agent's prior knowledge of the environment, the actions that the agent can perform and the percept sequence to data. The formal definition is:

`For each possible percept sequence, a tonal agent should select an action that is expected to maximise its performance measure, given the evidence provided by the percept sequence and the agent's built-in knowledge.`

The agent is considered autonomous if its behaviour is determined by its own experience.

## 2.3 Task Environment

To design a rational argument, you have to specify a task environment. This is made by PEAS:

- Performance Measure
- Environment
- Actuators
- Sensors

### 2.3.1 Example - Automated Taxi Driver

- Performance Measure - Safe, fast, legal, comfortable trip, minimise fuel consumption, maximise profits.
- Environment - Road types, road contents, customers, operating conditions.
- Actuators - Control over the car, communication with other vehicles and passengers.
- Sensors - Cameras, sonar, speedometer, GPS, odometer, engine sensors, speech recognizer.

### 2.3.2 Example - Internet Shopping Agent

- Performance Measure - Cheap, good quality, appropriate product.
- Environment - Current WWW sites, vendors.
- Actuators - Display to user, follow URL, fill in form.
- Sensors - HTML pages (text, graphics, scripts)

## 2.4 Environment Types

The environment type largely determines the agent design.

- Fully observable vs Partially obserable- An agent's sensors give it access to the complete state of the environment at all times.
- Known vs Unknown - An agent knows the "laws" of the environment.
- Single vs Multi Agent - An agent operating by itself in an argument
- Deterministic vs Stochastic - The next state is completely determined by the current state and the action executed by the agent.
- Episodic vs Sequential - The agent's experience is divided into atomic episodes. The next episode does not depend on previous actions. In each episode the agent precepts a percept and performs a single action.
- Static vs Dynamic - The agent is unchanged while an agent is deliberating.
- Discrete vs Continuous) - Pertains to number of states, the way time is handled, and number of percepts and actions. E.G, state may be continuous, but actions may be discrete.

## 2.5 Agent Types

The agent type is based on how actions are selected.

- Simple reflex - current percept
- Model based - + internal state
- Goal based - + goal
- Utility based - + utility function
- Learning

### 2.6 How components of agent programs work?

Depends on the representations of states:

- Atomic - each state is indivisible (Search game playing, Markov Decision Processes)
- Factored - splits each state into attributes, each of which has a value (Propositional logic, Planning, Bayesian Networks, Machine Learning).
- Structured - represents how things are related to each other (First order logic, Bayesian networks, Semantic networks).

## 3 Lecture 3-7: Problem Solving As Search - ONGOING

The objectives for problem solving include:

- Problem formulation
- Control strategies
  - Tentative
    - Uninformed
      - Backtrack
      - Tree and Graph Search
    - Informed
      - Best-first (greedy) search
      - A
      - A*
  - Irrevocable
    - Informed
      - Hill climbing
      - Greedy search
      - Local beam search
      - Simulated annealing
      - Genetic algorithms
- Adversarial search
  - Optimal decisions
  - $\alpha - \beta$ pruning
  - Imperfect, real-time decisions

## 3.1 A problem-solving agent

```
Function Simple-Problem-Solving-Agent(percept)
returns seq

persistent: seq - action sequence, initially null
state - description of current world state
goal - a goal, initially null
problem - a problem formulation

state <- UpdateState(state,percept)
goal <- FormulateGoal(state)
problem <- FormulateProblem(state,goal)
seq <- Search(problem)
return seq
```

### 3.1.1 Example - Romania

The problem is: On holiday in romania; currently in Arad. Flight leaves tomorrow from Bucharest. The formualate goal is to be in Bucharest, and the problem includes the states (various cities) and actions (driving between these cities). The way to find the solution is to find the sequence of cities that you can cross between.

## 3.2 Problem Formulation

The basic constituents of problem formulation are the states, goals, actions and constraints. The state space is the set of all states reachable from the initial state by any sequence of actions. The path in the state space is any sequence of actions leading from one state to another.
To represent a problem you need

- An initial state
- Operators (actions) and a transition model
- Constraints
- A goal test
- A path cost function

For the romainian example given above, this would correspond to:

- An initial state - "at Arad"

- Operators (actions) Go(Sibiu, Go(Timisoara),.. and a transition model (Result(In(Arad),Go(Zerind))-¿In(Zerind)
- Constraints - None
- A goal test - In(Bucharest)
- A path cost function - The sum of all distances, or the number of actions executed.

For an algorithm that governs the assembly of an object using a robot, they might be

- States - Real valued coordinates of robot joint angles
- Actions - Motions by the robot
- Constraints - Arm cannot fully rotate in certain angles
- Goal test - Complete assembly
- Path cost - time to execute

## 3.3 Selecting a State Space

The real world is complex. This means that the state space must be abstracted for problem solving. An abstract state may equal a set of real states (being In(Zerind) might mean you are in one of a few different hotels in Zerind). In this way, an abstract action can be a complex combination of real actions. A trip from one city to another represents many individual actions. A abstract solution is a solution that can be expanded into a set of paths in the real world. Each abstract action should be easier to perform than solving the original problem.

## 3.4 Control Strategies

Control strategies can be separated by two defining characteristics, there tentativeness (irrevocable/tentative) and informedness (uninformed/informed).

## 3.5 Tentative Control Strategies

### 3.5.1 Backtracking

In backtracking we keep one path at a time only. If we fail, we go back to the last decision point and erase the failed path. Backtracking occurs when we generate a previously encountered state description, we pass an arbitrary number of rules without reaching our goal, or there are no more applicable rules.

### 3.5.2 Graph Search

We keep track of several paths simultaneously. This is done using a structure called a search tree/graph. A graph is a set of nodes, while arcs connect certain pairs of nodes (in a directed graph these arcs are one way). If a node $n_i$ is accessible from $n_k$, then there is a path between the two nodes. If you expand a node, you are finding all of its children. Using a search tree, the root is the initial state, while it's children are the nodes accessible from it. Leaves are states without visible successor. At each iteration, pick a leaf node and expand it.

## 3.6 Search Strategies

A search strategy is defined by picking the order of node expansion. These strategies are evaluated along several 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 any path in the state space (may be infinite)

## 3.7 Uninformed search strategies

These search strategies use only the information available in the problem definition

### 3.7.1 Breadth-first search (BFS)

Expand the shallowest unexpanded node. Uses a large amount of space.

- Implementation: FIFO queue. Put successors at the end of the queue.
- Complete: Yes (if b is finite)
- Time: $O(b^d)$
- Space: $O(b^d)$ (keeps every node in memory)
- Optimal: Yes (if all actions have the same cost)

### 3.7.2 Uniform-cost search (UFS)

Expand the least-cost unexpanded node.

- Implementation: insert in order of increasing path cost.(The same as BFS if all steps cost equal).
- Complete: Yes, if all steps cost ¿= 0
- Time: $O(b_{1+floor(C*0)})$, C is the cost of the optimal solution.
- Space: $O(b_{1+floor(C*0)})$
- Optimal: Yes - nodes expanded in increasing order of g(n) = cost of path to node n

### 3.7.3 Depth-first search

Expand deepest unexpanded node

- Implementation - LIFO - insert successors in front of queue.
- Complete - Infinite-state spaces: No, Finite-state spaces: Yes
- Time - $O(b^m)$, terrible if m is much larger than d
- Space - $O(bm)$
- Optimal - No

### 3.7.4 Iterative Deepening Search

Check all nodes of D=0, then D=1, then D=2.

- Complete: Yes
- Time: $O(b^d)$
- Space: $O(bd)$
- Optimal? Yes, if step costs are identical.

## 3.8 Informed Search Strategies: Best-first Search

Heuristic Graph search procedures use heuristic information to help reduce the search. This involves the use of an evaluation functions - a real valued function used to compute the "promise" of a node.

### 3.8.1 Definitions

- $k(n_i, n_j)$ - actual cost of minimal cost path between $n_i$ and $n_j$
- $h*(n) = min\{k(n, t_j)\}$ - Minimum of all the $k(n, t_j)$ over the entire set of nodes $\{t_j\}$
- $g*(n) = k(s, n)$ - Minimum cost from the start node $s$ to $n$
- $f*(n) = g*(n) + h*(n)$ - Cost of an optimal path constrained to go through $n$
- $f*(s) = h*(s)$ - Cost of an unconstrained optimal path from s to goal.
- $f(n)$ - estimate of the minimal cost path constrained to go through node $n$
- $g(n)$ - estimate of $g*(n)$. Usual choice is the cost of the path in the search tree from $s$ to $n \rightarrow g(n) \geq g*(n)$
- $h(n)$ - heuristic function.

## 3.9 Greedy Best-First Search

- Description: Expands the node that is closest to the goal. $f(n) = h(n)$
- Example: $h_{SLD}(n)$ = Straight-Line Distance to the goal.
- Complete: In infinite spaces, no, in finite state spaces, Yes.
- Time: $O(b^m)$
- Space: $O(b^m)$
- Optimal: No

## 3.10 A

- Description: Graphsearch using the evaluation function $f(n) = g(n) + h(n)$. It expands the next node on the fronteir with the smallest value of f(n).
- Complete: Yes
- Time: ?
- Space: $O(b^d)$
- Optimal: No

### 3.11 A*

See ProblemSolving Pg66

- Description: If $\forall n h(n) \leq h*(n)$
- Complete: Yes
- Time: $O(b^{max|h-h*|})$
- Space: $O(b^d)$
- Optimal: Yes

### 3.12 Problems

A problem with few restrictions on actions is called a relaxed problem. The cost of an optimal solution to a relaxed problem is an admissable heuristic for the original problem.

### 3.13 Irrevocable Search Algorithms

In many optimization problems, the goal state is the solution and the state space is a set of "complete configurations". To find configurations that satisfy constraints, we use local search algorithms that keep a single "current" state and then try to improve it.

### 3.14 Hill Climbing

Hill Climbing continually selects the best possible next state. This has the issue of hitting a local maxima.

### 3.15 Local Beam Search

Local Beam Search generates k randomly generated states, then at each iteration all the successors of all k states are generated. If any of these are valid goal states, return it. If not, select the top k best successors from the complete list and repeat.

### 3.16 Simulated Annealing

Simulated Annealing escapes local maxima by allowing some bad moves but gradually decreasing their frequency. Involves both a Temperature (T), and an Annealing schedule (the rate at which the temperature is lowered). See Pg 86. One can rpove that if T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1. It is widely use in VLSI layout and airline scheduling.

## 3.17 Genetic Algorithms

A successor state is generated by combining two parent states. This involves starting with a population of k randomly generated states. A state is represented as a string over a finite alphabet of genes (often a string of 0s and 1s). A fitness function is used to produce the next generation of states by selection, crossover and mutation.

## 3.18 Adversarial Search Algorithms

The following section covers two person, perfect information games. The two players are named MAX and MIN. A position favourable to Max has a value ¿ 0, while a position favourable to Min has a value ¡ 0. The goal is to find a winning strategy for Max. For all nodes representing a game situation where it is MIN's move next, show that Max can win from every position to which Min might move or show that Max can win from just one position to which Max might move.
This has problems. You must specifiy a move for every possible opponent reply. This can be difficult with an unpredictable opponent. Time limits are also an issue. Not all games can be searched to the end.

### 3.18.1 MinMax Ideas

You have to choose the move with the highest minimax value: best achievable payoff against the best play. This is complete, optimal, with a time complexity of $O(b^m)$ and a space complexity of O(bm).

For chess, b is roughly 35, and m = 1000. An exact solution is completely infeasible.

## 3.19 Resource Limits

In order to cover the most ground with given resource limits. The standard approach is a cutoff test (limit to a depth limit). We can use an evaluation function that estimates the desirability of a position (number of white queens vs black queens for chess). We can also use forward pruning (only look at the n-best moves).

## 3.20 $\alpha - \beta$ Procedure

An