# Lecture Notes:

## Intelligent Agents and their Environments

### Types of Environments:

Observable:

* Fully - Agents sensors describe environment fully (chess)
* Partial - Some parts of the environment not visible; noisy sensors (war game)

Deterministic:

* Deterministic - next state fully determined by current state and users actions (connect 4)
* Stochastic - random changes, can't be predicted exactly (tipping point)

Sequential:

* Episodic - next state does not depend on previous actions (conveyer belt)
* Sequential - actions affect future state (crossword)

Static:

* Static - environment doesn't change no matter the actions (rubiks cube)
* Dynamic - environment can change at any time or even continuously (car)

Discrete:

* Discrete - percepts, actions and episodes are discrete (chess)
* Continuous - continuous time flow (car)

No of Agents:

* Single - (rubiks cube)
* Multi-agent - (strategy game)

### Type of intelligent Agents:

Simple-reflex agent: Action depends on immediate percepts. Implemented by condition-action rules

Example:

* Agent - Mail Sorting Robot
* Environment - Conveyer belt of letter
* Rule - e.g. Edinburgh, put in a certain bag

Model-Based Reflex Agent: Action may depend on history or unperceived aspects of the world. Need to maintain internal world model

Example:

* Agent - Robot vacuum cleaner
* Environment - dirty room, furniture
* Model - map of room, which areas are already cleaned

Goal-Based Agent: Agents so far have fixed, implicit goals. We want agents with variable goals. Forming plans to achieve goals is a topic for later.

Example:

* Agent - Robot maid
* Environment - House & people
* Goals - clean clothes, tidy room, table laid, etc.

Utility-Based Agent: Agents so far have a single goal. Agents may have to juggle conflicting goals. Need to optimise over a range of goals. (Utility means measure of goodness)

Example:

* Agent - Autonomous Car
* Environment - Roads, vehicles, safety signs
* Goals - stay safe, reach destination, be quick, obey law, save fuel etc.

Learning Agent: Generate problems which will test performance. Perform activities according to rules, goals, models, utilities etc. Monitor performance and identify non-optimal activity. Identify and implement improvements.

## Problem Solving and Search

### Different problem types:

Deterministic, fully-observable -> Single state problem

* Agent knows exactly which state it will be in, solution is a sequence

Non-observable -> sensorless problem (conformant problem)

* Agent may have no idea where it is, solution is a sequence

Nondeterministic and/or partially observable -> contingency problem

* percepts provide new information about current state
* often interleave search, execution

Unknown state space -> exploration problem

### Problem-Solving Agents - Example:

On holiday in Romania; currently in Arad

Flight leaves tomorrow from Bucharest

Formulate goal:

* Be in Bucharest

Formulate problem:

* States: various cities
* actions: drive between cities

Find solution:

* sequence of cities to travel through

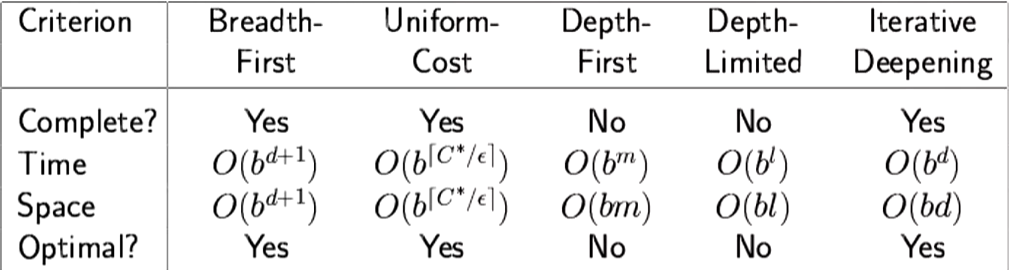
## Search and Strategies

### Evaluating search strategies:

* completeness - does it always find a solution
* time complexity - number of nodes generated
* space complexity - maximum number of nodes
* optimality - does it always find lowest cost solution

Time and space complexity are measured in:

* B - maximum branching factor
* D - depth of least cost solution
* M - maximum depth of state space



### Different search strategies

**Iterative deepening search:**

* Complete? Yes
* Time Complexity? O()
* Space Complexity? O(bd)
* Optimal? Yes (if cost = 1 per step)

**Breadth first search:**

* Complete? Yes
* Time Complexity? O() (worst-case)
* Space Complexity? O() (keeps every node in memory)
* Optimal? Yes (if cost = 1 per step)

**Depth first search:**

* Complete? No: fails in infinite-depth spaces, spaces with loops
* Time complexity? O(): terrible if m is much larger than d
* Space complexity? O(bm): i.e. linear space
* Optimal? No

## Informed Search Algorithms

### More search strategies

**Best first search:**

* An instance of general tree-search or graph-search
  + Use of an evaluation function f(n) for each node n (estimates how desireable an option is)
  + Expand most desirable unexpanded node, usually the node with the lowest evaluation

**Greedy best first search:**

* Evaluation function f(n) = h(n) (heuristic)
  + Estimated cost of cheapest path from state at node n to a goal state
* Greedy best first search expands the node that appears to be closest to the goal
* Complete? No (can get stuck in loops)
* Time complexity? O() for tree version
* Space complexity? O() keeps all nodes in memory
* Optimal? No

**A\* search:**

* Evaluation function f(n) = g(n) + h(n)
  + g(n) - cost so far
  + h(n) - estimated cost from n to goal
  + f(n) - estimated total cost of path through n to goal
* Avoid expanding paths that are already expensive
* Complete? Yes (unless infinitely many nodes)
* Time complexity? Exponential
* Space Complexity? Keeps all nodes in memory
* Optimal? Yes

### Heuristics

Any method that is believed or practically proven to be useful for the solution of a given problem.

* No guarantee that it will always work or lead to an optimal solution

We use heuristics to guide tree search

* This may not change the worst case complexity of the algorithm. But can help in the average case

We introduce conditions (admissibility, consistency) in order to identify good heuristics, i.e. those which actually lead to an improvement over uniformed search.

**Admissible Heuristics:**

A heuristic h(n) is admissible if for every node n:

h(n) <= h\*(n)

where h\*(n) is the true cost to reach the goal state from n.

An admissible heuristic never overestimates the cost to reach the goal, i.e. it is optimistic.

**Consistent Heuristics:**

A heuristic h(n) is consistent if for every node n, every successor of n' of n generated by any action a.

If a node has been reached through A\* search, there wouldn’t be any more optimal paths to that node if the heuristic is consistent.

## Smart Search Using Constraints

### CSPs in short

**Constraint Satisfaction Problems (CSPs):**

* State – A set of variables/states, , , , …
* Domain – Each variable has a non-empty domain of possible values:
* Constraints – A set of constraints to be satisfied, e.g. (unary), (binary)
* Solution – Assign a value to each variable such that none of the constraints are broken

This is a simple example of a formal representation language. Which allows useful general-purpose algorithms with more power than standard search algorithms.

**Standard Search Formulation:**

States are defined by the values assigned so far:

Initial State - the empty assignment{}

Successor Function - assign a value to an unassigned variable that does not conflict with current assignment.

Goal Test - the current assignment is complete

For a CSP with n variables, every solution appears at depth n

**Example of CSP - Map colouring:**

* Variables: {WA, NT, Q, NSW, V, SA, T}
* Domains: {red, green, blue}
* Constraints: adjacent regions must have different colours

Variable ordering and value selection heuristics help significantly

Forward checking prevents assignments that guarantee later failure

Constraint propagation, does additional work to constrain values and detect inconsistencies

**Backtracking:**

* Effectively the same as depth first search

**A hand writing on a whiteboard

Description automatically generated with medium confidence**

**The AC-3 Algorithm:**

1. Turn each constraint into two arcs. (e.g. A != B becomes A != B and B != A)
2. Add all arcs to the agenda
3. Repeat till agenda is empty:

* Take arc (Xi, Xj) off the agenda and check it
* For every possible value of Xi, there must be at least one value of Xj that satisfies the arc
* Remove any inconsistent value from Xi
* If Xi has changed after removing inconsistent values, add ALL arcs in the form (Xk, Xi) to the if they are not on the agenda.

Now, the problem is “Arc Consistent” and backtracking will be able to solve the updated problem much faster.

See example at: [Constraint Satisfaction: the AC-3 algorithm](https://www.youtube.com/watch?v=4cCS8rrYT14)

## Adversarial Search

### Different solutions and constraints to search problems

**Games Vs Search Problems:**

Unpredictable opponents -> solution is a strategy/policy

* specify a move for every possible opponent reply

Time limits -> unlikely to find goal, must approximate

|  |  |  |
| --- | --- | --- |
| Types of Games | Deterministic | Chance |
| Perfect information | chess, checkers | backgammon, monopoly |
| Imperfect information | battleships | poker, scrabble |

**Minimax Search:**

* the origin node (n=0) and all even depth nodes maximizes, all odd depth nodes minimizes.
* Complete? Yes (if tree is finite)
* Time Complexity? O()
* Space Complexity? O(bm)
* Optimal? Yes (against an optimal opponent)

**Diagram

Description automatically generated**

**Alpha - Beta pruning:**

* alpha is value of best i.e. highest choice found so far at any point for MAX
* beta is value of best i.e. lowest choice found so far at any point for MIN
* Pruning does not affect the result. Good move ordering improves effectiveness of pruning.
* Initial call: alpha = -inf, beta = +inf
* If Maximizing player’s turn:
* alpha = max(alpha, evaluated child nodes’ evals)
* if alpha <= alpha: prune the rest of the sub tree
* If Minimizing player’s turn:
* beta = min(beta, evaluated child nodes’ evals)
* if beta <= alpha: prune the rest of the sub tree

Timeline

Description automatically generated

With perfect ordering, time complexity becomes O()

* branching factor goes from b to
* doubles solvable depth of search

**Resource Limits:**

Suppose we have 100 seconds to make a move and can explore 10k nodes a second.

* 1M nodes per move
* If average branching factor
* Hence 4 ply lookahead is an average chess player

**Evaluation function:**

EVAL(s) = + + … +

where each is a weight and each is a feature of state s

Example:



## Logical Agents

**Knowledge Base:**

A knowledge base(KB) in a **Declarative approach** keeps track of things in the environment:

* TELL it facts about the environment, or
* ASK for inference, answers should follow from the KB

For example,

* TELL: Father of John is Bob
* TELL: Jane is John’s Sister
* TELL: John’s Father is the same as John’s Sister’s Father
* ASK: Who’s Jane’s Father

**Logic:**

* **Logics** are formal languages for representing information such that conclusions can be drawn
* **Syntax** defines the sentences in the language
* **Semantics** defines the *meaning* of sentences; define truth of a sentence in a world

E.g.

* Syntax: x+2 ≥ y is a sentence, x + 2 > + is not a sentence
* Semantics: x+2 ≥ y is true iff the number x+2 is no less than the number y

**Entailment:**

Entailment means that one thing follows from another:

KB ⊨ α

Knowledge base KB entails sentence α if and only if in every model where KB is true, α is true.

For example,

* (x = 0) ⊨ (xy = 0)
* (p = True) ⊨ (p OR q)

**Models:**

m is a model of α if α is true in m.

For example, if we have sentence α: xy = 0, there can be models as follows:

* x = 0, y = 1
* x = 0, y = 12345
* x = 12345, y = 0

M(α) is the set of all models of α.

KB ⊨ α iff M(KB) ⊆ M(α)

Effectively, all models of KB are more specific than all models of the sentence, α.

The stronger an assertion, the fewer models it has.

**Inference:**

* = sentence α can be derived from KB by inference procedure
* Soundness: is sound if whenever , it is also true that .
* Completeness: is complete if whenever , it is also true that .

Graphical user interface, text, application, chat or text message

Description automatically generated

## Effective Propositional Inference

**Conversion to CNF:**

A picture containing graphical user interface

Description automatically generated

**DPLL:**

**Early termination:**

* A clause is true if one of its literals is true
* A sentence is false if any of its clauses is false

**Pure symbol heuristic:**

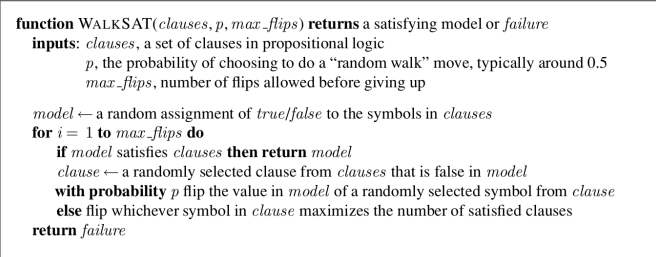
* Pure symbol: always appears with the same “sign” or polarity in all clauses
* e.g., In the three clauses (A ∨ ¬B), (¬B ∨ ¬C), (C ∨ A): A and B are pure, C is impure
* Make literal containing a pure symbol true

**Unit clause heuristic:**

* Unit clause: only one literal in the clause. E.g. (A)
* The only literal in a unit clause must be true, also includes clauses where all but one literal is false.

**WalkSAT:**

**Incomplete, local** search algorithm. EVAL function is the Min-conflict heuristic: Minimizing the number of unsatisfied clauses.



Often there will be a Cut-off, in this case, *max\_flips,* so algorithm would not infinitely run when it is stuck in local space. Uses randomness of flipping values in a randomly selected symbol from clause to escape confining local space.

## First Order Logic

Atemporal Predicates: Objects e.g. water(x), teabag(x)

Fluent Predicates: hot(x), In(x, y), empty(x)

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Diagram

Description automatically generated

A picture containing shape

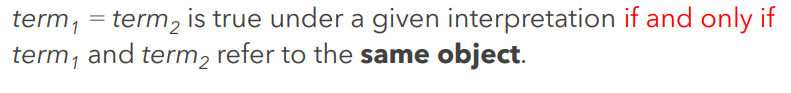
Description automatically generated

Diagram

Description automatically generated

A screenshot of a computer

Description automatically generated



## Unification and Generalised Modus Ponens

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

For method of finding the MGU check lecture 12 PDF

Table

Description automatically generated

## Resolution

## Situation Calculus

## Introduction to Planning

## Algorithms in Symbolic Planning

## Planning and Acting in the Real World

## Monitoring, Planning and Hierarchical Plans

## Uncertainty, Rationality and Probability

## Probabilities and Bayes’ Rule

## Introduction to Bayesian Networks

## Exact Inference in Bayesian Networks

## Approximate Inference in Bayesian Networks

## Time and Uncertainty

## Dynamic Bayesian Network

## Decision Making Under Uncertainty

## Markov Decision Process

# Definitions

Agent: Something that makes decisions for itself

Environment: Where the agent is based

Sensors: How the agent perceives things (microphone etc.)

Goal: What an agent tries to do

Actuators: Actions agents perform to achieve its goals

Simple-reflex agent: Agent which actions depend on immediate percepts.

Model-based reflex agent: Agent that acts upon a world model it has been given.

Goal based agent: Agent that acts to achieve a specific goal.

Utility based agent: Agent that optimises between multiple goals.

Backtracking: Depth first search with one variable assigned per node

B: maximum branching factor

D: depth of least cost solution

M: maximum depth of state space

# Past Paper Solutions

## 2022 Non-Resit