Notes for "Introduction to AI"

Simon Holm

AI501 - Instruction to AI

May 12, 2025

Contents

1 Introduction

Notes and exercises for lectures and TA-sessions.

2 Notes

2.1 Introduction and definitions

Different schools of though

- Systems that act like humans
- Systems that think like humans
- Systems that think rationally
- Systems that act rationally

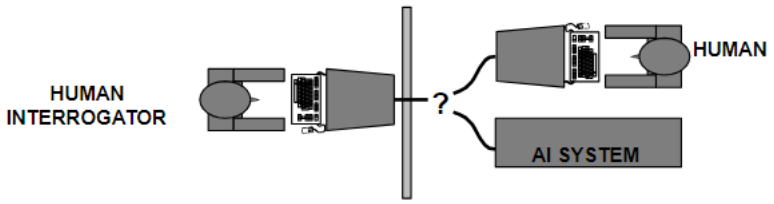


Figure 1:

The Turing Test: "The Imitation Game": Alan Turing said, that any machine that can pass the test qualifies as AI.

2.1.1 Thinking rationally: Laws of Thought

Several Greek schools developed various forms of logic including

- Notation and rules of derivation for thoughts (using logic based on rules)

  This results in problems of cause such as...

- Not all intelligent behavior is mediated by logical deliberation
  (Example: Reactive agents ⇒ remove hand from stove)

2.1.2 Acting rationally

Rational behavior: doing the right thing

The right thing: that which is expected to maximize goal achievement, given the available information

2.1.3 Rational agents

Note: This course is about designing rational agents

An agent is an entity that perceives and acts for any given class of environments and tasks, we seek the agent (or class of agents) with the best performance

Difficulty: computational limitations make perfect rationality unachievable Therfore we must design programs to find best results with this in mind.

2.1.4 Agents and environments

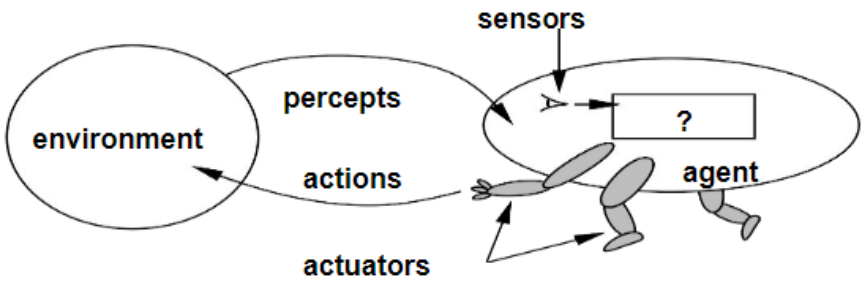


Figure 2:

Agents include humans, robots, softbots, thermostats, etc.

An agent can be anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators

The agent function (f) maps from perception to action:

$$f : P \Rightarrow A$$

Vacuum-cleaner world

2.1.5 Informed searched strategies

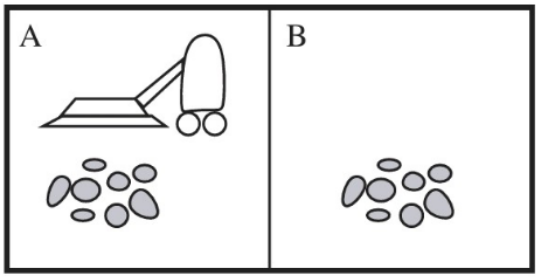


Figure 3: Vacuum-cleaner AI example

2.1.6 Performance measure

How to measure the environment sequence/performance with vacuum cleaner example?

- one point per square cleaned up in time T?
- one point per clean square per time step, minus one per move?
- penalize for $>$ k dirty squares?

Note: rational $\neq$ successful

PEAS

2.1.7 Informed searched strategies

- Performance measure
- Environment
- Actuators
- Sensors

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments	Touchscreen/voice entry of symptoms and findings
Satellite image analysis system	Correct categorization of objects, terrain	Orbiting satellite, downlink, weather	Display of scene categorization	High-resolution digital camera
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, tactile and joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, raw materials, operators	Valves, pumps, heaters, stirrers, displays	Temperature, pressure, flow, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, feedback, speech	Keyboard entry, voice

Figure 4: PEAS with agent type included

2.1.8 Key Concepts in AI Environments

- Fully/Partially Observable: agent has access to all/part of relevant information.
- Deterministic/Non deterministic or Stochastic: next state is entirely predictable/non predictable from the current state and action.
- Episodic/Sequential: actions depends on previous/sequence of actions.
- Static/Dynamic: environment does not change/change while the agent is deciding on an action.
- Discrete/Continuous: The state and action spaces consist of distinct, countable options/non finite options
- Single/Multi-agent: Only one/More than one agent is acting in the environment (competition or collaboration)

The environment type largely determines the agent design
The real world is (of course) partially observable, non deterministic, sequential, dynamic, continuous, multi-agent **Note: for episodic environment, no knowledge before perception is needed in any way**

2.1.9 Agent types

- simple reflex agents

  Selects actions based solely on the current percept, ignoring the history or future consequences

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Figure 5: A simple reflex agent pseudo code

```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
  if status = Dirty then return Suck
  else if location = A then return Right
  else if location = B then return Left
```

Figure 6: Example using vacuum agent

  - Use only the current percept to make decisions and this often is not enough
  - Requires full observability
  - Possible infinite loops (e.g., vacuum cleaner move left and right even if clean)

- model based agents

  Use an internal model of the environment to keep track of unseen aspects and predict the effects of actions. Model represent the best-guess for the of the current environment

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              transition_model, a description of how the next state depends on
                    the current state and action
              sensor_model, a description of how the current world state is reflected
                    in the agent's percepts
              rules, a set of condition–action rules
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, transition_model, sensor_model)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Figure 7: A model based agent pseudo code

  - Knowing the state ≠ knowing the goal
  - E.g., robot that can explore a maze but does not know its goal

- goal-based agents

  adjusts its actions to achieve specific goal/goals.

- utility-based agents

  an agent that acts based not only on what the goal is, but the best way to reach that goal (usefullness)

- Leaning agents

  Leaning agents improve their performance over time by acquiring knowledge and adapting to the environment.

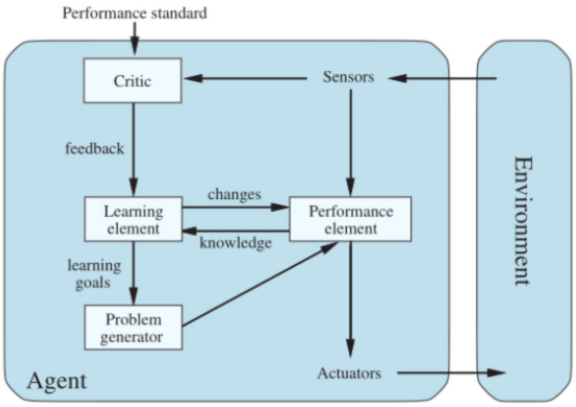


Figure 8: Enter Caption

Key Components:

- Performance Element: Select actions using the current knowledge
- Learning Element: Responsible for making improvements based on feedback.
- Critic: Provides feedback on the agent's actions to guide learning. Suggest how performance element needs to be modified to do better in the future
- Problem Generator: Suggests exploratory actions to discover new knowledge.

Pros:

- Adapt to changing environments.
- Operate effectively with incomplete or evolving knowledge.

Cons:

- Balancing exploration (learning) and exploitation (acting optimally).
- Managing computational complexity in large environments.

2.1.10 AGI

Artificial General Intelligence (AGI) is one of tree types of artificial intelligence.

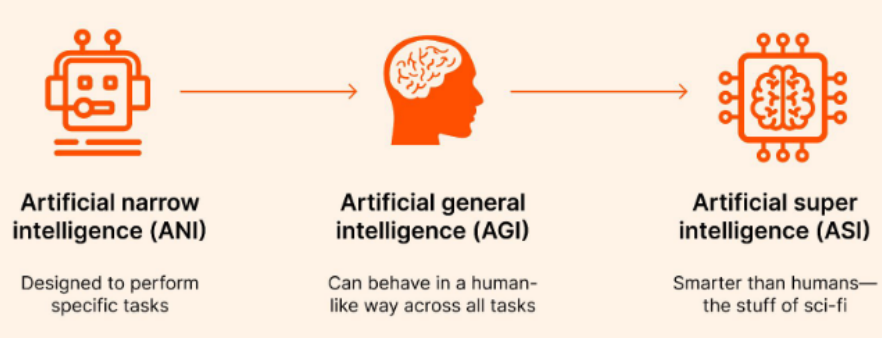


Figure 9:

Key Characteristics of AGI:

- Generalization: Can transfer knowledge from one domain to another, adapting to new tasks without needing extensive retraining.
- Autonomy: Functions independently, making decisions and solving problems without predefined rules or heavy supervision.
- Human-Like Cognition: Capable of reasoning, problem-solving, and understanding language, emotions, and context at a human level.
- Learning Efficiency: Learns new skills or knowledge with minimal input, akin to how humans learn.

For an AI to be an AGI it must be able to pass something like the Turing test 100% of the time. This makes the virtually impossible to define something as an AGI with certainty.

2.1.11 AI & Philosophy

Philosopher John Searle (1980):

- **weak AI**: idea that machines could act as if they were intelligent
- **strong AI**: assertion that machines that do so are actually consciously thinking (not just simulating thinking)

Some philosophers claim that a machine that acts intelligently would not be actually thinking, but would be only a simulation of thinking. Types of AI like ChatGPT falls under this category.

2.1.12 Problem-solving agents

(Problem-solving) agents need to find sequences of actions that achieve a desired goal.

Ideally an agent should be Goal-Oriented: Have to know what goal needs to be achieved. Search-Based: Explore one ore more options to achieve goal. Knowledge-Independent: Can function in environments with little prior knowledge We need to know the following to define a task for an agent

- State space: All the possible states the system can be
- Initial State: The starting point of the problem.
- Actions and Transition Model: Possible steps the agent can take and their consequences.
- Goal: Criteria to determine if the goal is reached.
- Action cost function: A measure to evaluate the efficiency of solutions. You want a good agent instead of a bad one.

2.1.13 Problem types

- When fully observable $\implies$ single state problem

  solutions must be a sequence because the agent knows what move does exactly what
- Non-observable $\implies$ conformant problem

  agent doesn't know where it is, solution (if any) is a sequence
- Nondeterministic and/or partially observable $\implies$ contingency problem. When agent percepts, new knowledge is gained about the current state

  solution is a contingent plan or a policy often interleave search, execution
- Unknown state space $\implies$ explorations problem (offline)

2.2 Search strategies

- Completeness: does it always find a solution if one exists?
- Time complexity: number of nodes generated/expanded
- Space complexity: maximum number of nodes in memory
- Optimality: does it always find a least-cost solution?

2.2.1 Uninformed Search Strategies

- Breadth-first search

  Expand worst note, and then continue.

  Properties of breadth-first search.

  - Complete if finite
  - Time is exponential
  - O(b^d) Keeps every node in memory, therefore can become big problem

    Where b is branch factor (splits), and d is depth (generation)
  - Optimal? only if costs are all the same

- Depth-first search

  Brute force expansion

  Properties of Depth-first search.

  - Complete: No, will fail in infinite depth spaces or spaces with loops

    Modify to avoid repeats.
  - Time: Possibly very bad
  - Space Maximal depth of the tree
  - Optimal: No

- Depth-limited search

  Depth-first search, but limit number of layers in the tree.

- Iterative deepening search

  Limit memory, but still use breadth-first search

  Properties of Iterative deepening search.

  - Complete: Yes
  - Time: max $O(b^d)$
  - Space: max $O(bd)$
  - Yes, if step cost = 1

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value no
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Figure 10: Pseudo code for search

2.2.2 Directional searching

- Forward search
  - Start from initial state
  - Explore from start
- Backwards search
  - Start from goal
  - Work backwards to find path
  - Good when easier to determine predecessors of the state than successors.
- Bidirectional search
  - Searches simultaneously from the initial state and the goal.
  - Meets in the middle, reducing search depth.
  - Each search (forward and backward) could explores approximately roughly half the depth of the solution space $\implies$ reduce the number of nodes visited

2.2.3 Informed (Heuristic) searched strategies

Using problem-specific knowledge to guide exploration. (Find solutions more efficiently, when using domain-specific hints about the location of goals)

- Heuristic Function ($h(n)$): estimated cost of the cheapest path from the state at node n to a goal state.
- Evaluation Function ($f(n)$): Combines heuristic and path cost for decision-making (f(n) = g(n) + h(n)).
- Greedy search
  - Complete: No. Can get stuck in loops
    Complete in finite space with repeated-state checking
  - Time: similar to deep search but a good heuristic can give dramatic improvement
  - Space: similar to deep search
  - Optimal: No

2.2.4 A* search

A* search is a tool to use when using an actual searching strategy Idea: avoid expanding paths that are already expensive
Evaluations function $f(n) = g(n) + h(n)$

- gn cost so far to reach n
- estimated cost to reach goal from n
- Complete: Yes
- Time: exponential in [relative error in h · length of soln.]
- Space: keeps all nodes in memory
- Optimal: yes

  It does not expand any nodes that exceed the optimal cost

A* search uses and **admissible heuristic**

- $h(n) \geqslant h^*(n)$ where $h^*(n)$ is the true cost from n (i.e., never overestimates)
- $h(n) \geqslant 0 + h(g) = 0$ for any goal G

Theorem: A$\star$ search is optimal

2.2.5 Dominance

When one heuristic function is clearly better than another $(h_2(n) \geqslant h_1(n)$

2.2.6 Search in complex environments: Local Search

When searching in a complex environment, trees and paths are not always preferred. *Example TSP problem, where a traveler has to visit key places with the shortest path for the trip.*

- Keep a single current state.
- Try to improve it

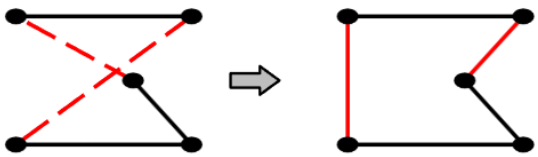


Figure 11: Local search

2.2.7 Hill Climbing

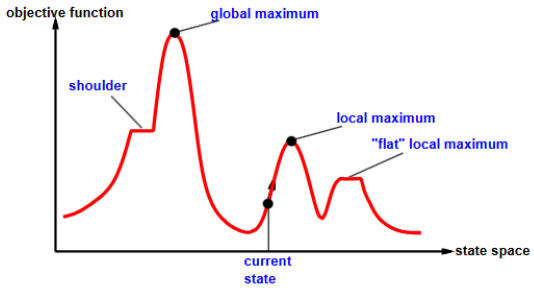


Figure 12: Hill Climbing

Problems with hill climbing

- You could possibly be stuck in a local maxima or flat maxima

Eliminate problems

- Simulated annealing

  Escape local maxima by allowing som "bad" moves

  + gradually decrease their size and frequency(AKA decrease the temperature).

  $S' \leftarrow$ randomly from S successors.

  $\Delta E = V(S') - V(S)$ where V is value

  If $\Delta E > 0$, then select S'

  Else select S' with prop $e^{\frac{\Delta E}{T}}$

Properties of simulated annealing

- High temperature: Encourages exploration by accepting worse solutions.
- Low temperature: Focuses on exploitation by refining towards the best solution.
- Cooling Schedule: rate of temperature decrease
  - Too fast: May miss the global optimum.
  - Too slow: Computationally expensive.
  - Common schedules: $T \propto \frac{1}{log(t)}$, where t is the iteration step.
  - Global Optimality Guarantee: if the temperature is decreased at an appropriate rate simulated annealing will always reach the global best state (based on properties of the Boltzmann distribution $\rightarrow$ temperature must decrease asymptotically towards 0)10

2.2.8 Generic algorithms

Simulate natural selection. (Stochastic local beam search and generate successors from pairs of states)

- Initial population
- Fitness

  How good/optimal is this individual
- Selection

  Choosing the best/fittest individuals
- Crossover

  Combines creatures from two parents to create offspring
  - Cross-over rate

    Typically set 0.6-0.9 (60-90% of the population undergoes crossover)

    Higher rates encourage more exploration and combining of good solutions.
  - Mutation rate

    Usually kept low (0.001–0.01) to avoid excessive randomness.

    Helps maintain diversity and escape local optima.14
- Mutation

  Introduce random/new traits or changes to maintain diversity.
- Selection Method strategies
  - Roulette Wheel: Probabilistic, favors fitter individuals.
  - Tournament Selection: Picks the best among (random) subsets.
  - Elitism: Directly carries over top solutions to the next generation. Used to preserve the best solutions.
- Number of Generations
  - Stop condition: Based on either fixed generations or convergence (no improvement over time).

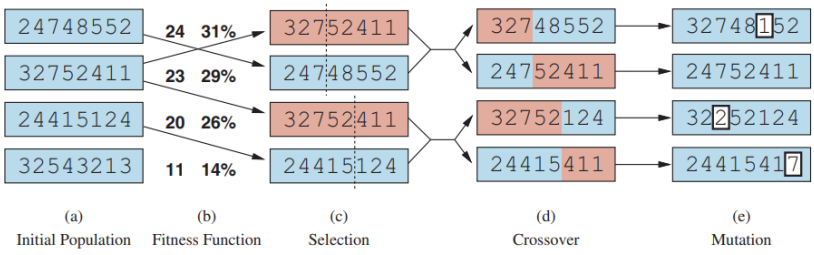


Figure 13: A Genetic Algorithm

2.2.9 Continuous state spaces

Discretization methods: turn continuous space into discrete space

- Gradiant Descent: moving in the direction of the steepest descent.
  - Initialize
  - Compute gradient
  - Update rule
  - Learning rate
  - Repeat
- Newtons method: faster convergence using second derivatives
- Linear programming.

2.2.10 Search With Nondeterministic Actions

An agent might does not know what state is ends up in after an action.
Idear:
When agent in a complex enviroment, you can use AND or OR actions

- `OR` Choose to take one action
- `AND` Have a plan for all states after action

```
function AND-OR-SEARCH(problem) returns a conditional plan, or failure
  return OR-SEARCH(problem, problem.INITIAL, [])

function OR-SEARCH(problem, state, path) returns a conditional plan, or failure
  if problem.IS-GOAL(state) then return the empty plan
  if IS-CYCLE(state, path) then return failure
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(problem, RESULTS(state, action), [state] + path)
      if plan ≠ failure then return [action] + plan
  return failure

function AND-SEARCH(problem, states, path) returns a conditional plan, or failure
  for each s_i in states do
      plan_i ← OR-SEARCH(problem, s_i, path)
      if plan_i = failure then return failure
  return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n-1} then plan_{n-1} else plan_n]
```

Figure 14: AND/OR ACTIONS

Note: failure → means that there is no non-cyclical solution. For loop notes with different possible paths, agents will eventually choose the right path

2.2.11 Search In Partially Observable Environments

The agent has limited or noise information about the environment

2.2.12 NO Observations Allowed

Sensor-less solutions:

- Robots orienting parts using a sequence of actions
- Doctor prescribing generic antibacterial medicine

Conformant Planning: generates a sequence of actions that guarantees success for all states in the initial belief state.24

2.3 Online Search

- Offline search: complete solution before taking any action
- Online search: first it takes an action, then it observes the environment and computes the next action
  - Random Walk:
    - Select path at random
    - Will eventually find a goal (if space in finite and safe.)
    - Can be very slow
  - Learning real-time A*
    - Build a map of the environment and chooses the "apparent best" move
    - Assume that a non-explored state will immediately lead to goal (prioritizes exploration)

2.4 Adversarial Search And Games

2.4.1 Types of Games

- Perfect information
  - Deterministic
    - Chess, Checkers, Go, Othello
  - Chance
    - Backgammon, monopoly
- Imperfect Information
  - Deterministic
    - Battleships, Cluedo
  - Chance
    - Bridge, Poker, Scrabble nuclear war

2.4.2 Games vs search strategies

,

Unpredictable oppenent, therefore solution is a strategy, where planning a move fore every possible opponent reply.
Also time limits makes it unlikely to find a goal assuming that opponent play optimally

2.4.3 Two-player Zero-sum Games

Setting

- Two players taking turns in a fully observable enviroment
- A terminal test, which is true when the game is over and false Terminal test otherwise
- A utility function (also called an objective function or payoff function), which defines the final numeric value to player p when the game ends in terminal state s. Depending on the game these values can differentiate. Commonly $(1, 0, 1/2)$ or $(1, 0, -1)$

For zero-sum: Always good for one player, bad for the other (no win-win)

2.4.4 Minimax

Always make the move that is best against the most optimal move for opponent

- In a nonterminal state, MAX prefers to move to a state of maximum value when it is MAX's turn to and MIN prefers a state of minimum value (that is, minimum value for MAX and thus maximum value for MIN). it is shown as:

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s, \textsc{Max}) & \text{if } \textsc{Is-Terminal}(s) \\ \max_{a \in \textsc{Actions}(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{To-Move}(s) = \textsc{Max} \\ \min_{a \in \textsc{Actions}(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{To-Move}(s) = \textsc{Min} \end{cases}$$

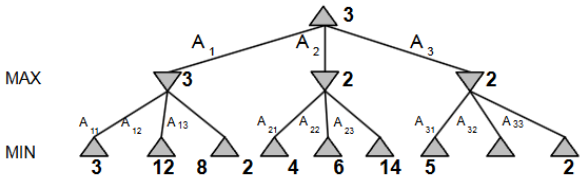


Figure 15: Minimax example

```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v, move ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v, move ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

Figure 16: Minimax pseudoscope

2.4.5 $\alpha - \beta$ Pruning

α Best value that the maximizing player can guarantee at any point-

β branch is pruned if the minimizing player can provide a value less than α (maximizing player would never choose it)

2.4.6 Limited computation time

Idear: cut branch at a certain depth

- You would require a heuristic that provides an estimate of a position's desirability

How it works.

- Define a Depth Limit + Use the Evaluation Function

- Approximate which player has the best position
- Now use minimax method, to choose best path
- **Problem**: fails to see beyond limit, therefore short-sighted moves :((*Horizon effect*).

2.4.7 Monte Carlo Tree Search

For each game, the actions that leads to wins are favored
Steps:

- Selection: navigate the tree using a balance of exploitation (best average score) and exploration (least visited nodes).
- Expantion: Add new child nodes when unexplored moves are encountered.
- Simulation: Simulate random playouts from the new node to estimate outcomes.
- Backpropagation: Update the scores of all nodes along the path to the root based on the simulation result.

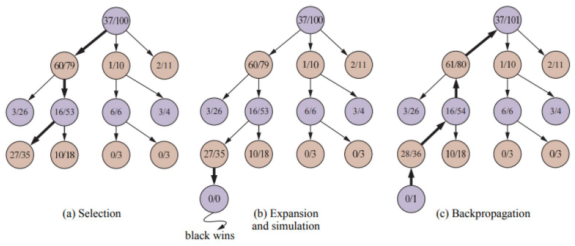


Figure 17: Monte Carlo Tree Search example(black vs pink)

Selection Policy: Effective selection policy is called "Upper Confidence bounds applied to Trees" (UCT).
UCT ranks each possible move based on an upper confidence bound formula

$$UCB_l(n) = \frac{U(n)}{N(n)} + C \cdot \sqrt{\frac{log(PARENT(n)}{N(n)}}$$

Where:

- $U(n)$ is the total utility of all playouts that went through node n
- $N(n)$ is the number of playouts through node n
- $PARENT(n)$ is the parent node of n in the tree
- C is a constant that makes $\sqrt{\frac{log(PARENT(n)}{N(n)}}$ bigger, thus favors exploration (theoretical argument that C should be $\sqrt{2}$ but in practice try and choose the one that performs best) (Ideally $C > 1$)

This function returns the child node (state) that has the highest estimated win rate after exploring and simulating games.

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ← NODE(state)
  while IS-TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

Figure 18: Monte Carlo psudocode

Note: we might reach resource limit.
For this we do Cutoff-Test rather than Terminal-Test.

- Terminal-Test

  Full search to terminal states

- Cutoff-Test

  Stops the search at a predefined depth or when a time limit is reached.

Eval Function Instead of Utility

- Utility Function: Returns the exact value of a terminal state (e.g., win/loss/draw in chess).
- Evaluation Function (Eval): Estimates the "desirability" of a non-terminal position based on features like material advantage, board control, etc.
- Allows assessing states without reaching the end of the game.

2.4.8 Nondeterministic/Stochastic Games

In nondeterministic games, chance is introduced (dice or card-shuffling)
expect minimax value = compute the expected value for change nodes

- value over all outcomes, weighted by probability of each chance action

For this, search tree becomes way worse.

2.4.9 Games of imperfect information

E.g., card games where you only know your own cards, but not the opponants cards.
Typically we can calculate a probability for each possible deal
Seems just like having one big dice roll at the beginning of the game
Idear:

- compute the minimax value of each action in each deal
- choose the action with highest expected value over all deals
- special case: if an action is optimal for all deals, it's optimal

2.4.10 Limitations of game search algorithms

- $\alpha - \beta$ search is vulnerable to errors in the heuristic function.
- You can easily wast computational time picking a move that is obvious.
- Algorithm will reason on individual moves, while humans reason on a more abstract level.
- Possibility to incorporate Machine Learning into game search process :).

2.5 Constraint Satisfactions Problems

Imperative vs declarative approaches.

- Imperative:
  - Step by step algorithms
- Declaration
  - What am i looking for
  - specify the constraints for a solution
  - Not using by steps, but a machine finds answer for you

2.5.1 Defining Constraint Satisfaction Problems

Tree components

- X is a set of variables $\{X_1 \dots X_n\}$
- D is a set of domains $\{D_1 \dots D_n\}$, one domain for earch variable
- C is a set of constraints that specify the legal combination of variables

A CSP will combine all variables with a domain within the constraints.

2.5.2 Standard search formulation

- Initial state
- Assign a value to an unassigned variable that does not conflict with current assignment
- If we assigned all variables → problem solved
- If there is a variable we can not assign → backtrack and change value
- If we backtrack to the first variable → problem is not satisfiable

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, assignment)
          if inferences ≠ failure then
              add inferences to csp
              result ← BACKTRACK(csp, assignment)
              if result ≠ failure then return result
              remove inferences from csp
          remove {var = value} from assignment
  return failure
```

Figure 19: Backtracking in CSP

2.5.3 Improving backtracking effiviency

General-purpose methods can give huge gains in speed:

- Can we detect inevitable failure early?
- Which variable should be assigned next?
- In what order should its values be tried?

2.5.4 Arc consistency

Simplest from of propagation makes each arc consistent

- $X \rightarrow Y$ is consistent iff for every value x of X there is some allowed y for Y

2.5.5 Local consistency

- Node Consistency (NC)
  - Ensures each variable satisfies its unary constraints (i.e., constraints involving only itself)
  - Example: "must start after 10 AM" removes invalid times from its domain
- Arc Consistency (AC)
  - Ensures that for every variable-value pair, there exists at least one valid value in every connected variable.
- Path Consistency
  - Ensures that if two variables satisfy a constraint, a third variable can still take a consistent value.
  - Example: In a meeting room allocation, if Alice can meet in Room A or B, and Bob can only meet in Room B, then Carol must be able to meet in Room B to ensure feasibility

  (PC)k-Consistency **(Generalized Consistency)**
  - Extends path consistency to k variables.
  - Example: In graph coloring, 3-consistency ensures that given two assigned colors, the third connected node can still be colored properly.
  - Very powerfull but costly (n^k)

2.5.6 Variable Heuristic

- Minimum remaining values
  - Pick the value with the smallest domain
  - Quickly reduces the search space
- Degree Heuristic
  - Pick the variable affecting the nost constraints
  - Maximizes propagation
- Adaptive strategy: Dom/Wdeg (Domain over Weighted Degree)
  - Weighted Degree = sum of the constraints weight, based on past failures
  - Choose the variable that minimizes Domain Size / Weighted Degree

2.5.7 Value Heuristic

- Least Constraining Value (LCV)
  - Choose the value that eliminates the fewest options for other variables
  - Makes for more flexibility in the search space
- Most Constraining Value (MCV)
  - Pick the value that will rule out more values using the inference
  - Helps detect failure earlier
- Min-Value / Max-Value
  - Strategy: Pick the smallest (or largest) value first
  - Useful when values have a natural order (e.g., cost minimization)

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
          max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
      if current is a solution for csp then return current
      var ← a randomly chosen conflicted variable from csp.VARIABLES
      value ← the value v for var that minimizes CONFLICTS(csp, var, v, current)
      set var = value in current
  return failure
```

Figure 20: Local Search of CSP

2.5.8 Backjumping

Jump back further than just backtracking.
In some cases backjumping will avoid exploring useless assignments

2.5.9 Local Search of CSP

2.5.10 Symmetries

Try to remove symmetries to reduce search space and avoids exploring equivalent solutions
Add constraints to allow one solution per group of symmetric solutions

2.6 Knowlege based agents

A Knowledge-Based Agent is an intelligent system that uses stored knowledge and logical reasoning to make decisions. It perceives the environment, updates its knowledge, and takes actions based on reasoning.

2.6.1 Entailment

$KB \models \alpha$
This means that KB "entails" α, if and only if α is true in all worlds where KB is true.
e.g. $x = 0 \vDash x \cdot y = 0$

Model-checking, assume that KB is the knowledge of the agent and α is the objective knowledge without KB

2.6.2 Inference

$KB \vdash_i \alpha$ i is sound if

$$whenever KB \vDash$$

2.6.3 Horn form

When at most one component in a clause is positive ($A = positive, \neg A = negative$)

$$(A \vee \neg B) \wedge (B \vee C)$$

2.6.4 Forward chaining

$\alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta$

- does the known facts imply more knowledge?
- If it does add this to known facts
- repeat with new knowledge until completeness

2.6.5 Backwards chaining

Start at the end and go back until you find a known fact. This way you can avoid loops.

2.6.6 Some DPLL tricks

-

If reading is necessary 7.1 to 7.6.1

2.7 Quantifying Uncertainty

Real world problems contain uncertainties due to:

- Partial observation
- Nondeterminism
- Adversaries

Algorithm 1 DT-AGENT(*percept*) **returns** an action

1: **Input:** *belief state*: ▷ Beliefs about the current state
2: **Input:** *action*: ▷ the agent's action
3: Update **belief state** based on *action* and *percept*
4: Calculate outcome probabilities for actions given **belief state**
5: Select *action* with highest expected utility, given probabilities of outcomes and utility information
6: **return** *action*

2.7.1 Terms

- sample space (Ω) = all posiple outcomes

  Note: $\sum(P(\omega) = 1$, where atomic events (ω) = 1 event in sample space

- The sum rule: $P(A \vee B) = P(A) + P(A) - P(A \wedge B)$ or $P(A \cup B) = P(A) + P(A) - P(A \cap B)$

2.7.2 Conditional probability

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$

$$P(B \mid A) = \frac{P(B \wedge A)}{P(A)}$$

So...

$$P(A \wedge B) = P(A \mid B) \cdot P(B) = P(B \mid A) \cdot P(A)$$

3 Exercises

3.1 Exercise 2.1

For each of the following assertions, say whether it is true or false and support your answer with examples or counter-examples where appropriate.

a) An agent that senses only partial information about the state cannot be rational.

   False, an agent can take rational steps based on the information.

b) There exist task environments in which no pure reflex agent can behave rationally.

   True, a reflex to a environment cant always be rational.

c) There exists a task environment in which every agent is rational.

   True. Using a very simple task (like not moving, where agent cant move) the agent can only be rational

d) The input to an agent program is the same as the input to the agent function.

False, agent program runs a function.

e) Every agent function is implementable by some program/machine combination.

No, not all problems can be solved using an agent.

f) Suppose an agent selects its action uniformly at random from the set of possible actions. There exists a deterministic task environment in which this agent is rational.

Yes for some random actions there is always some environment where the action will be rational

g) It is possible for a given agent to be perfectly rational in two distinct task environments.

True

h) Every agent is rational in an unobservable environment.

With performance measure an agent, the agent can still be rations

i) A perfectly rational poker-playing agent never loses.

False, poker involves luck. (If it would play it self, one of them would still win)

3.2 Exercise 2.2

For each of the following activities, give a PEAS description of the task environment and charac-terize it.

a) Playing soccer.

b) Exploring the subsurface oceans of Titan.

c) Shopping for used AI books on the Internet.

d) Playing a tennis match.

e) Practicing tennis against a wall.

f) Performing a high jump.

g) Knitting a sweater.

h) Bidding on an item at an auction.

a-h	Observable	Known/Unknown	Agents	Deterministic	Episodic	Static	Discrete
a	Partially	Known	Multi	Non	Sequential	Dynamic	Continuous
b	Partially	Known-ish	Single (hopefully)	Non	Sequential	Dynamic	Continuous
c	Partially	Known	Multi	Non	Sequential	Dynamic	Discrete
d	Partially	Known	Multi	Non	Sequential	Dynamic	Continuous
e	Fully	Known	Single	Deterministic	Sequential	Dynamic	Continuous
f	Fully	Known	Single	Deterministic	Episodic	Static	Continuous
g	Fully	Known	Single	Deterministic/	Episodic	Static	Continuous
h	Partially	Known	Single	Deterministic	Sequential	Dynamic	Discrete

3.3 Exercise 2.3

Let us examine the rationality of various vacuum-cleaner agent functions.

a) Show that the simple vacuum-cleaner agent function described in Figure 12 is indeed rational.

It's action takes us closer to a clean enviroment

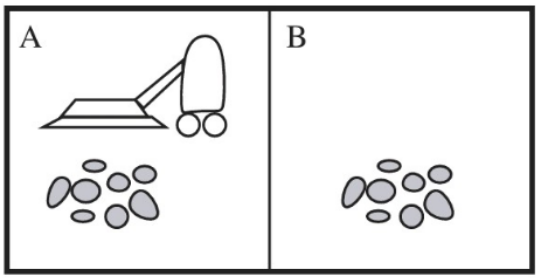


Figure 21: A vaccum-clearner world with just two locations

Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[B, Clean]$	*Left*
$[B, Dirty]$	*Suck*
$[A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Dirty]$	*Suck*
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Clean], [A, Dirty]$	*Suck*
⋮	⋮

Figure 22: Partial tabulation of a simple agent function for the vaccum-clearner world shown in figure 11

b) Describe a rational agent function for the case in which each movement costs one point. Does the corresponding agent program require internal state?

Check if state is clean, if not: clean and move. if clean move. Save in memory which state is clean and repeat untill all states are clean.

c) Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense for the agent to learn from itsexperience in these cases? If so, what should it learn? If not, why not?

Learn enviroment

3.4 Exercise 2.4

Define in your own words the following terms: agent, agent function, agent program, rationality,autonomy, reflex agent, model-based agent, goal-based agent, utility-based agent, learning agent.'

Agent.

Agent function.

Agent program.

Rationality.

Autonomy.

Reflex agent.

Model-based agent.

Goal-based agent.

Utility-based agent.

Learning agent.

3.5 Exercise 2.5

3.6 3 part 1 Basic search. 1

Give a complete problem formulation for each of the following problems. Choose a formulation that is precise enough to be implemented.

(a) There are six glass boxes in a row, each with a lock. Each of the first five boxes holds a key unlocking the next box in line; the last box holds a banana. You have the key to the first box,and you want the banana.

- State-space:
  [Have key, locked box, unlock box, find key] → [Next box]
  [Have key, locked box, unlock box, find banana] → [Stop]
- Initial state:[Have key, locked box, unlock box, find key]
- Goal: Find banana
- Actions: [Unlock box], [Grab key], [Go to next box]
- Transition model:
  [Locked box + Unlocking box] → [Unlocked box]
  [Unlock box + Grab key] → [Move to next box]
  [Unlock box] → [Stop]
- Action cost: 1

(b) You start with the sequence ABABAECCEC, or in general any sequence made from A, B, C,and E. You can transform this sequence using the following equalities: AC = E, AB = BC, BB = E, and Ex = x for any x. For example, ABBC can be transformed into AEC, and then AC, and then E. Your goal is to produce the sequence E.

- State-space: (A—B—C—E)*
- Initial state: [A sequence]
- Goal: E with lowest cost posible
- Actions: [AC = E], [AB = BC], [BB = E], [Ex = x]
- Transition model:
  [AC → E] and [E → AC]
  [AB → BC] and [BC → AB]
  [BB → E] and [E → BB]
  [Ex → x] and [x → Ex]
- Action cost: 1

(c) There is an $n \cdot n$ grid of squares, each square initially being either unpainted floor or a bottomless pit. You start standing on an unpainted floor square, and can either paint the square under you or move onto an adjacent unpainted floor square. You want the whole floor painted.

- State-space: $n \cdot n \cdot 3$ states. either painted, unpainted or your mother
- Initial state: Unpainted floor
- Goal: all floors painted
- Actions: [Paint], [Check next floor state], [Move], [Stop]
- Transition model:
  [Paint] → [Check next state]
  [Check next state + Next floor is a pit] → [Check another floor]
  [Check next state + Next floor is unpainted] → [Move]
  [Check next state + No next state is unpainted] → [Move]
  [Paint + Check if finished] → [Stop]
- Action cost: 1

3.7 3 part 1 Basic search. 2

A robot has the task of navigating out of a maze. The robot starts in the center of the maze facing north. It can turn to face north, east, south, or west. The robot can also move forward a certain distance, although it will stop before hitting a wall.

(a) Formulate this problem. How large is the state space?

* State-space: [Position], [Face north], [Face east], [Face west], [Face south]
* Initial state: [The middle], [Facing north]
* Goal: Escape
* Actions: [Move], [Turn], [Stop]
* Transition model:
  [Move] → [Turn]
  [Turn] → [Move]
* Action cost: 1

(b) In navigating a maze, the only place we need to turn is at the intersection of two or morecorridors. Reformulate this problem using this observation. How large is the state space now?

For this there will be no dead ends, so smaller than (a)

(c) From each point in the maze, we can move in any of the four directions until we reach aturning point, and this is the only action we need to do. Reformulate the problem using these actions. Do we need to keep track of the robot's orientation now?

No we do not

(d) In our initial description of the problem we already abstracted from the real world, restrictingactions and removing details. List three such simplifications we made.

Observable enviroment

Locked to 90° turns

2D enviroment

3.8 3 part 1 Basic search. 3

Consider a 9×9 grid of squares, each of which can be colored red or blue. The grid is initiallycolored all blue, but you can change the color of any square any number of times. Imaginingthe grid divided into nine 3×3 sub-squares, you want each sub-square to be all one color butneighboring sub-squares to be different colors.

- State-space: Some regions have a certain color.
- Initial state: No squares are colored
- Goal: All squares are colored but no adjacent regions are the same color
- Actions: [Paint], [Check next floor state], [Move], [Stop]
- Transition model:

  [Paint] → [Check next state]

  [Check next state + Next floor is a pit] → [Check another floor]

  [Check next state + Next floor is unpainted] → [Move]

  [Check next state + No next state is unpainted] → [Move]

  [Paint + Check if finished] → [Stop]
- Action cost: 1

3.9 3 part 1 Basic search. 6

Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.

a) Formulate the problem precisely, making only those distinctions necessary to ensure a validsolution. Draw a diagram of the complete state space.

- State: $(M_1, C_1, B_1, M_2, C_2)$
- Initial: $3, 3, \leftarrow, 0, 0$
- Goasl: $0, 0, \rightarrow, 3, 3$
- Actions: 1M, 2M, 1M+1C, 1C, 2C
- Cost: 1

b)

c)

3.10 3 part 1 Basic search. 7

- Suppose that the pieces fit together exactly with no slack. Give a precise formulation of thetask as a search problem.
  - State: Any fitting combination of pieces with no overlap
  - Initial: No pieces
  - Goal: No loose ends and all pieces used.
  - Actions: Place a connection
  - Cost: 1
- Identify a suitable uninformed search algorithm for this task and explain your choice.

  Depth-first
- Explain why removing any one of the "fork" pieces makes the problem unsolvable.

  if you remove one, you would end up with one un-connectable lose end.

d) Give an upper bound on the total size of the state space defined by your formulation. (Hint: think about the maximum branching factor for the construction process and the maximum depth, ignoring the problem of overlapping pieces and loose ends. Begin by pretending that every piece is unique.)

$(7 \cdot 4)^{32} \geqslant |S|$ because of the 4 switches

3.11 3 part 2 Informed Search. 1

Trace the operation of A*search applied to the problem of getting to Bucharest from Lugoj usingthe straight-line distance heuristic. That is, show the sequence of nodes that the algorithm will consider and the f, g and h score for each node

3.12 Exam june 2021

An orchard owner wants to automate the process of har-vesting fruit from the trees in the orchard. For this pur-pose, an AI system is being designed to control a robotto go from tree to tree, harvest all the fruit in each tree,and return to its shelter. The system should find a routethat is as short as possible, to minimize fuel consumptionand the total time spent harvesting. From the point of view of this system, collecting thefruit from a tree is done implicitly whenever the harvestingrobot is near to it. The robot has enough capacity to carryall the fruit collected, so that one round trip is guaranteedto suffice (however, the optimal route may include passingthrough the shelter several times). The layout of the trees and the available roads to go from onetree to the other is shown on the picture on the right, where the road lengths are given in meters.

(a) Classify this problem according to the following dimensions: fully/partially observable; deterministic/stochastic; discrete/continuous.

Fully observable

Deterministic

Continuous

(b) Formulate this problem precisely as a search problem: describe the state space, the initial state,the actions, the transition model, the goal test, and the cost function. Make sure to abstractas much as possible from details that are not relevant for solving the problem.

State space: [Position], [Tree with friuts]

Initial [Shelter], [All trees]

Goal: [Shelter], [No trees]

Actions: [Go to connected node]

Transition: Update position, if tree is in list, remove it

Cost: Distance between nodes

(c) Which of the dimensions identified earlier change in your abstraction?

It is now descrete

(d) Assume that the successors of each node are generated in alphabetical order of the robot'sposition: if the robot is atC, then the possible moves are to B,D,E or Shelter, in that order.Assume also that duplicate states are skipped.

i Write the partial routes corresponding to the ten nodes first visited by breadth-first search.

ii

iii Write the solution found by depth-first search.(Note: There can be different states where the robot is in the same position.)

vi Propose a (simple) admissible heuristic for this problem.

Cost for amount of unvisited trees left

v Write the complete route found by greedy best-first search using your heuristic.

3.13 4 part 1 search in complex environments

Give the name of the algorithm that results from each of the following special cases.

a Local beam search with $k = 1$

Since k is the amount of states it keeps, this is equal to hill climbing

b Local beam search with one initial state and no limit on the number of states retained

c Simulated annealing with $T \to 0$ at all times (and omitting the termination test)

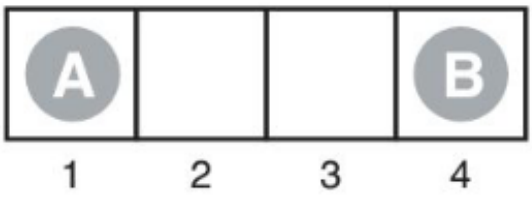
For this $e^{\frac{\Delta E}{T}} \to 0$ This results to the algorithm picking the a random of the best paths. this is kinda like Hill Climbing

d Simulated annealing with $T \to \infty$ at all times.

For this $e^{\frac{\Delta E}{T}} \to 1$ This results to the algorithm picking the first of the random paths. This is random walk.

e Genetic algorithm with population size $N = 1$.

We can only mutate at random, which is not very smart. Random walk



h!

Figure 23: The starting position of a simple game. Player A moves first. The two players take turns moving, and each player must move his token to an open adjacent space in either direction. If the opponent occupies an adjacent space, then a player may jump over the opponent to the next open space if any. (For example, if A is on 3 and B is on 2, then A may move back to 1.) The game ends when one player reaches the opposite end of the board. If player A reaches space 4 first, then the value of the game to A is +1; if player B reaches space 1 first, then the value of the game to A is -1.

3.14 4 part 6 search in complex environments

Consider the sensor-less version of the erratic vacuum world shown in figure 4.14.

a Draw the belief-state space reachable from the initial belief state $1, 2, 3, 4, 5, 6, 7, 8$, and explain why the problem is unsolvable.

b Draw the belief-state space reachable from the initial belief state $1, 3, 5, 7$, and explain why the problem is unsolvable.

3.15 4 part 9 search in complex environments

Consider the following problem: there are two coins in a table, both covered (so it is unknown which side is facing up). The only possible action at each stage is: choose one coin, toss it, check the result, and cover it again. The goal is to have both coins facing the same way (heads or tails). Draw the complete And-Or search tree for this problem and use it to make a contingency plan that will solve it.

3.16 4 part 10 search in complex environments

Suppose that an agent is in a 3x3 maze environment like the one shown in Figure 4.19. The agent knows that its initial location is (1,1), that the goal is at (3,3), and that the actions Up, Down, Left and Right have their usual effects unless blocked by a wall. The agent does not know where the internal walls are. In any given state, the agent perceives the set of legal actions; it can also tell whether the state is one it has visited before.

(a) Explain how this online search problem can be viewed as an offline search in belief-state space, where the initial belief state includes all possible environment configurations. How large is the initial belief state? How large is the space of belief states?

belief state is 2^12 since there are 12 places for a wall to be. they can either be there or not.

(b) How many distinct percepts are possible in the initial state?

there are two possible walls adjacent the initial state, therefore 2^2

3.17 6 part 4 Adversarial search and games

Consider the two-player game described in Figure 5.17.

(a) Draw the complete game tree, using the following conventions:

- Write each state as (sA,sB), wheresAandsBdenote the token locations.
- Put each terminal state in a square box and write its game value in a circle.
- Put loop states (states that already appear on the path to the root) in double squareboxes. Since their value is unclear, annotate each with a "?" in a circle.

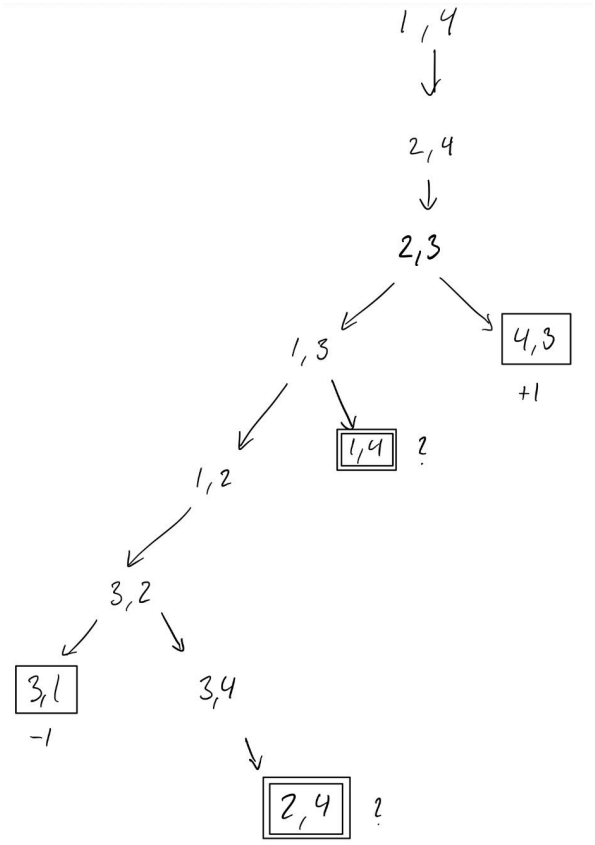


Figure 24: This is the drawn tree

[h!]

(b) Now mark each node with its backed-up minimax value (also in a circle). Explain how you handled the "?" values and why.

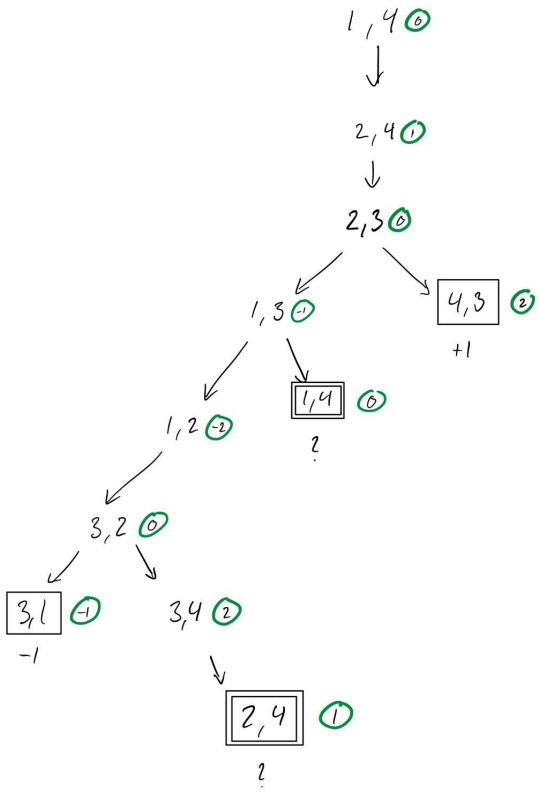


Figure 25: Now with minimax values(in green circles)

(c) Explain why the standard minimax algorithm would fail on this game tree and briefly sketch how you might fix it, drawing on your answer to (b). Does your modified algorithm give optimal decisions for all games with loops?

(d) This 4-square game can be generalized to n squares for any $n > 2$. Prove that A wins if n is even and loses if n is odd.

They will move until adjacent. Depending on n being even or odd one one player will jump over the other to vicory

3.18 6 part 5 Adversarial search and games

Consider the following game, played on the map in Figure 5.16 (a). There are two players, a pursuer (P) and an evader (E); the pursuer wants to catch the evader, and the evader wants to avoid being caught. We assume now that the players take turns moving. The game ends only when the players are on the same node; the terminal payoff to the pursuer is minus the total time taken.(The evader "wins" by never losing.) An example is shown in Figure 5-16 (b).

(a) Copy the game tree and mark the values of the terminal nodes.

(b) Next to each internal node, write the strongest fact you can infer about its value (a number,one or more inequalities such as "$\geqslant 14$", or a "?").

(c) Beneath each question mark, write the name of the node reached by that branch.

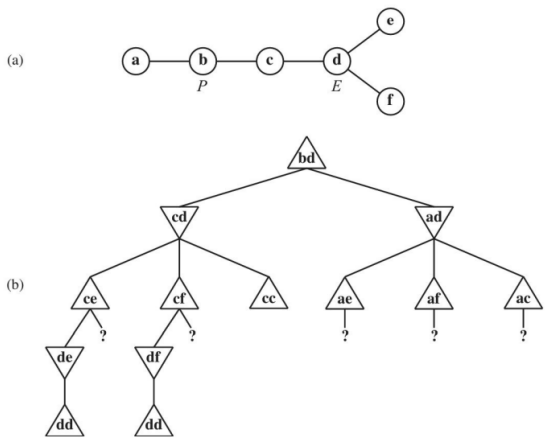


Figure 26: (a) A map where the cost of every edge is 1. Initially the pursuer P is at node b and the evader E is at node d. (b) A partial game tree for this map. Each node is labeled with the P, E positions. P moves first. Branches marked "?" have yet to be explored.

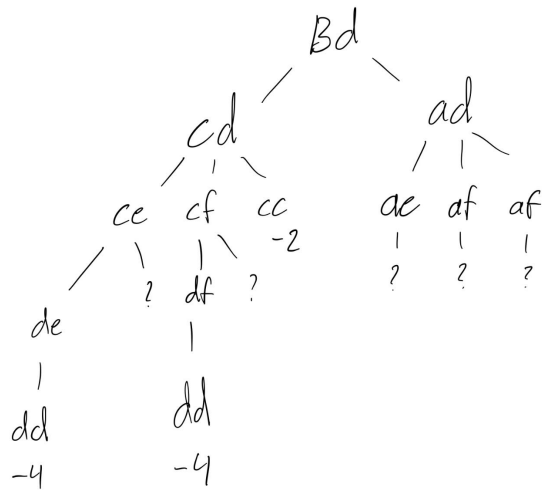


Figure 27: Game tree with value of terminal nodes

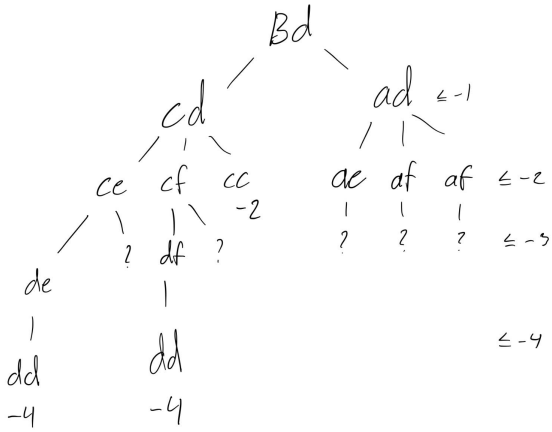


Figure 28: Game tree with new values for all internal nodes

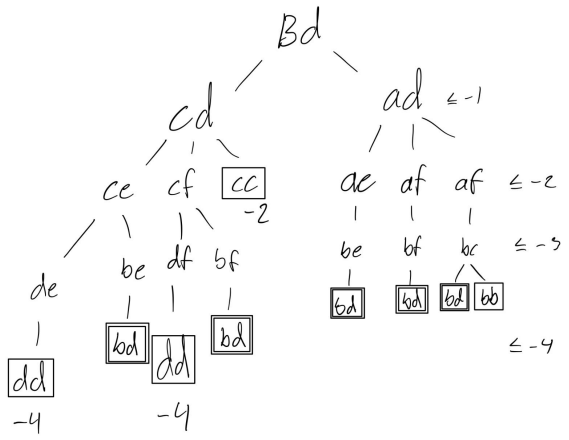


Figure 29: Game tree with explored nodes

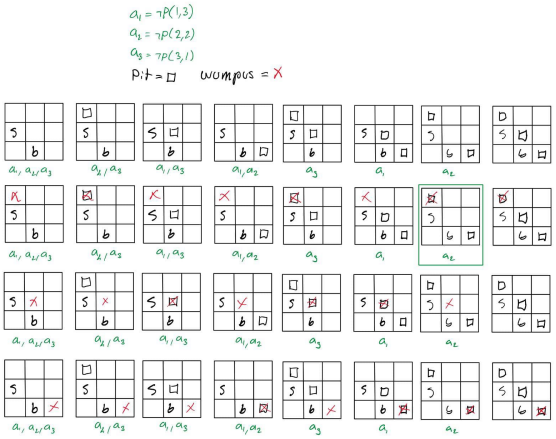


Figure 30: Wumpus world exercise

(d) Explain how a bound on the value of the nodes in (c) can be derived from consideration of shortest-path lengths on the map, and derive such bounds for these nodes. Remember the cost to get to each leaf as well as the cost to solve it.

3.19 7 part 1 Logical Agents

Suppose the agent in the Wumpus world has explored three squares, having perceived nothing in [1,1], a breeze in [2,1], and a stench in [1,2], and is now concerned with the contents of [1,3], [2,2], and [3,1]. Each of these can contain a pit, and at most one can contain a wumpus. Following the example in the book, construct the set of possible worlds. (You should find 32 of them.) Mark the worlds in which the KB is true and those in which each of the following sentences is true:

α_1 = "There is no pit in [1,3]."

α_2 = "There is no pit in [2,2]."

α_3 = "There is no pit in [3,1]."

Use this information to show that $KB \vDash \alpha_1$, $KB \vDash \alpha_2$ and $KB \vDash \alpha_3$.

3.20 7 part 3 Logical Agents

Which of the following are correct? **Note:**

(a) $(A \vee B) \wedge \neg(A \Rightarrow B)$ is satisfiable.

$(A \vee B) \wedge \neg(\neg A \wedge B)$

$(A \vee B) \wedge (A \vee \neg B)$

Not satisfiable

(b) $(A \Leftrightarrow B) \wedge (\neg A \vee B)$ is satisfiable.

Satisfiable

(c) $A \wedge B \vDash A \Leftrightarrow B$

True

(d) $A \Leftrightarrow B \vDash A \vee B$

False

(e) $A \Leftrightarrow B \vDash \neg A \vee B$

True

(f) $(A \wedge B) \Rightarrow C \vDash (A \Rightarrow C) \vee (B \Rightarrow C)$

$\neg(A \wedge B) \vee C \vDash (\neg A \vee C) \vee \neg B$

True

(g) $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \vDash A \vee B$

True

(h) $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \vDash (A \vee B) \wedge (\neg D \vee E)$

False

(i) $False \vDash true$

True

(j) $True \vDash False$

False

(k) $(C \vee (\neg A \wedge \neg B))$ is equivalent to $((A \Rightarrow C) \wedge (B \Rightarrow C))$

$(C \vee (\neg A \wedge \neg B)) \equiv ((\neg A \vee C) \wedge (\neg B \vee C))$

$(C \vee \neg A) \wedge (C \vee \neg B)) \equiv (\neg A \vee C) \wedge (\neg B \vee C)$

These are equivalent

(l) $(A \Leftrightarrow B) \Leftrightarrow C$ has the same number of models as $A \Leftrightarrow B$ for any fixed set of proposition symbols that include A, B and C.

$(A, B, C), (\neg A, \neg B, C), (\neg A, B, \neg C), (A, \neg B, \neg C)) = 4$

$(A, B, C), (A, B, \neg C), (\neg A, \neg B, C), (\neg A, \neg B, \neg C) = 4$

This si true

3.21 7 part 8 Logical Agents

According to some political pundits, a person who is radical (R) is electable (E) if they are conservative (C), but otherwise is not electable.

(a) Which of the following are correct representations of this assertion?

i $(R \wedge E) \Leftrightarrow C$

False

ii $R \Rightarrow (E \Leftrightarrow C)$

True

iii $R \Rightarrow ((C \Rightarrow E) \vee \neg E$
$\neg R \vee \neg C \vee E \vee \neg E$

(b) Which of the sentences in (a) can be expressed in Horn form?

i $(R \wedge E) \Leftrightarrow C$
$C \Rightarrow (R \wedge E) \wedge ((C \wedge E) \Rightarrow C)$
$\neg C \vee (R \wedge E) \wedge (\neg(C \wedge E) \vee C)$
$(\neg C \vee R) \wedge (\neg C \vee E) \wedge (\neg R \vee E \vee C)$
True

ii $R \Rightarrow (E \Leftrightarrow C)$
True

iii $R \Rightarrow ((C \Rightarrow E) \vee \neg E$

3.22 12 part 6 probalitity theory

	toothache		*¬toothache*	
	catch	*¬catch*	*catch*	*¬catch*
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

Figure 31: Tooth ache probability

Given the full joint distribution in Figure 3, calculate the following?

(a) $P(toothache)$
$0.108 + 0.012 + 0.016 + 0.064 = 0.2$

(b) $P(\neg catch)$
$0.012 + 0.064 + 0.008 + 0.576 = 0.66$

(c) $P(\neg cavity \mid catch)$
$\frac{0.016+0.144}{((0.108+0.016)+0.072)+0.144} = 0.47$

(d) $P(cavity \mid toothache \cup \neg catch)$
$\frac{0.012}{0.108+0.012+0.016+0.064+0.008+0.576} = 0.01530612245$

3.23 12 part 8 probability theory

Deciding to put probability theory to good use, we encounter a slot machine with three independent wheels, each producing one of the four symbols bar, bell, lemon, or cherry with equal probability. The slot machine has the following payout scheme for a bet of 1 coin (where "?" denotes that we don't care what comes up for that wheel):

- bar/bar/bar pays 20 coins
$20 \cdot (\frac{1}{4})^3 = 0.31$
- bell/bell/bell pays 15 coins
$15 \cdot (\frac{1}{4})^3 = 0.23$
- lemon/lemon/lemon pays 5 coins
$5 \cdot (\frac{1}{4})^3 = 0.08$
- cherry/cherry/cherry pays 3 coins
$3 \cdot (\frac{1}{4})^3 = 0.05$

- cherry/cherry/? pays 2 coins

  $2 \cdot (\frac{1}{4})^2 - (\frac{1}{4})^3 = 0.11$

- cherry/?/? pays 1 coin

  $2 \cdot (\frac{1}{4})^2 - (\frac{1}{4})^3 = 0.1875$

(a) Compute the expected "payback" percentage of the machine. In other words, for each coin played, what is the expected coin return?

$\mathbf{E}[x] = 0.31 + 0.23 + 0.08 + 0.05 + 0.11 + 0.1875 = 0.9675$

3.24 12 part probability theory

Consider a coin with a heads probability of 2/3 being tossed three times. Denote by the random variable C the number of times the coin toss results in a heads. Evaluate the following statements related to the random variable C.

(a) $P(C = 1) > P(C = 3) \Leftrightarrow \frac{2}{3} > (\frac{2}{3})^3 = 0.66 > 0.296$

(b) $\mathbb{E}[C] > 2$

$\mathbb{E}[C] = 3 \cdot 1 \cdot \left(\frac{2}{3} \cdot \frac{1}{3} \cdot \frac{1}{3}\right) + 3 \cdot 2 \cdot \left(\frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3}\right) + 3 \cdot \left(\frac{2}{3} \cdot \frac{2}{3} \cdot \frac{2}{3}\right) = 2$

(c) $\mathbb{E}[C^2] = 3 \cdot 1^2 \cdot \left(\frac{2}{3} \cdot \frac{1}{3} \cdot \frac{1}{3}\right) + 3 \cdot 2^2 \cdot \left(\frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3}\right) + 3^2 \cdot \left(\frac{2}{3} \cdot \frac{2}{3} \cdot \frac{2}{3}\right) = 4.66$

And we know that $\mathbb{E}[C]^2 = 2^2 = 4$

So $4.66 > 4$ so *true*

(d) $\frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{3} < \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{2}{3}$ this is *true*

(e) since $\mathbb{V}[C] = \mathbb{E}[C^2] - \mathbb{E}[C]^2$

$\mathbb{V}[C]$ must be smaller than $\mathbb{E}[C^2]$ so *true*

3.25 12 part probability theory

After your yearly checkup, the doctor has bad news and good news. The bad news is that you tested positive for a serious disease and that the test is 99positive when you do have the disease is 0.99, as is the probability of testing negative when you don't have the disease). The good news is that this is a rare disease, striking only 1 in 10,000 people of your age. Why is it good news that the disease is rare? What are the chances that you actually have the disease?

The probability of $P(positive \mid sick) = 0.99$
The probability of $P(negative \mid \neg sick) = 0.99$
The probability for the statistic is $P(sick) = 0.0001$
Using Bayes' rule says that $P(sick \mid positive) = \frac{P(positive|sick) \cdot P(sick)}{P(positive)}$
We know that $P(\neg sick) = \frac{9999}{10000}$ and that $P(positive \mid \neg sick) = \frac{1}{100}$
So $P(positive) = P(positive \mid sick) \cdot P(sick) + P(positive \mid \neg sick) \cdot P(\neg sick) = \frac{5049}{50000}$