ARTIFICIAL INTELLIGENCE

P23CAE22 | L:3 T:0 P:0 C:3 | 45 Periods

Complete Unit-wise Study Notes with Tables and Examples

Unit	Title	Periods
Unit I	Introduction to AI	9
Unit II	Solving Problem by Searching	9
Unit III	Local Search and Optimization	9
Unit IV	Constraint Satisfaction Problems	9
Unit V	Knowledge Representation and Reasoning	9

UNIT I
INTRODUCTION TO ARTIFICIAL INTELLIGENCE

1.1 What is Artificial Intelligence?

Artificial Intelligence (AI) is the branch of computer science that aims to create machines capable of performing tasks that normally require human intelligence such as reasoning, learning, problem-solving, perception, and language understanding.

Definition	Source
Science and engineering of making intelligent machines	John McCarthy (1956)
Study of agents that perceive and act on their environment	Russell & Norvig
Making machines do things requiring intelligence if done by humans	Marvin Minsky

1.2 Foundations of AI

Discipline	Contribution to AI
Mathematics	Logic, probability, algorithms, optimization
Computer Science	Data structures, algorithms, complexity
Psychology	Cognitive models, human learning and perception
Linguistics	Natural language processing, grammar, semantics
Neuroscience	Biological neural networks inspirations
Philosophy	Logic, rationality, mind-body problem
Economics	Decision theory, utility, game theory

1.3 History of Artificial Intelligence

Year/Era	Key Event
1943	McCulloch & Pitts — first mathematical model of a neuron
1950	Alan Turing — Turing Test ('Computing Machinery and Intelligence')
1956	Dartmouth Conference — term 'Artificial Intelligence' coined by John McCarthy
1957-1970	Early enthusiasm: GPS, LISP, checkers-playing programs
1970s	First AI Winter — funding cuts due to failed promises
1980s	Expert Systems boom: MYCIN, XCON; Japan 5th Generation project
1987-1993	Second AI Winter — expert systems too costly; Japan project failed
1993-2010	ML resurgence; Deep Blue beats Kasparov (1997); Statistical methods dominate
2010s-now	Deep Learning, AlphaGo, GPT-4, LLMs, Generative AI

1.4 The State of the Art in AI

- Game Playing: AlphaGo, AlphaZero — defeated world champions in Go and Chess

- Natural Language Processing: GPT-4, BERT for language understanding and generation
- Computer Vision: Object detection (YOLO), face recognition, medical imaging AI
- Robotics: Autonomous vehicles (Tesla, Waymo), Boston Dynamics robots
- Healthcare: AI-powered cancer detection in radiology, drug discovery
- Recommendation Systems: Netflix, Spotify, YouTube personalization engines

1.5 Risks and Benefits of AI

Benefits	Risks
Automates repetitive tasks	Job displacement and unemployment
Better medical diagnosis accuracy	Bias and discrimination in AI decisions
24/7 availability (chatbots, systems)	Privacy violations through surveillance
Faster scientific discovery	Autonomous weapons (lethal AI systems)
Improved accessibility (speech-to-text)	Deep fakes and AI-generated misinformation

1.6 Intelligent Agents

An intelligent agent perceives its environment through sensors and acts upon it through actuators. The PEAS framework defines any agent:

- **Performance measure:** Criteria for evaluating agent's success
- **Environment:** External context the agent operates in
- **Actuators:** How agent acts on the environment
- **Sensors:** How agent perceives the environment

Note: PEAS Example — Self-driving car: P=safe arrival; E=roads, traffic; A=steering, brakes; S=cameras, GPS, lidar

1.7 The Concept of Rationality

A rational agent selects actions that maximize expected performance given its percept history and built-in knowledge. Four factors determine rationality:

1. The performance measure defining success criteria
2. The agent's prior knowledge about the environment
3. The actions available to the agent
4. The agent's percept sequence to date

1.8 Nature of Environments

Property	Type A	Type B	Example
Observability	Fully observable	Partially observable	Chess vs Poker
Outcomes	Deterministic	Stochastic	Chess vs Backgammon
Episodes	Episodic	Sequential	Image classification vs Chess
Timing	Static	Dynamic	Crossword vs Taxi driving
States	Discrete	Continuous	Chess vs Self-driving
Agents	Single-agent	Multi-agent	Crossword vs Chess

1.9 Structure of Agents

Agent Type	Description	Knowledge
Simple Reflex	Acts on current percept only	Condition-action rules
Model-Based Reflex	Maintains internal world state	World state + transition model
Goal-Based	Acts to achieve specified goals	Goal information + planning
Utility-Based	Maximizes utility function	Utility function + probability
Learning	Improves performance through experience	Performance element + critic + learner

UNIT II
SOLVING PROBLEM BY SEARCHING

2.1 Problem-Solving Agents

A problem-solving agent finds sequences of actions to achieve goal states. Problem components:

5. Initial State: Starting configuration of the agent
6. Actions: Set of possible moves from any state
7. Transition Model: New state resulting from action in a state
8. Goal Test: Check if current state satisfies the goal
9. Path Cost: Numerical cost of a sequence of actions

2.2 Example Problems

Problem	States	Goal
8-Puzzle	Tile arrangements	Goal tile arrangement
8-Queens	Queen positions on board	No queen attacks another
Romania Map	Current city	Reach Bucharest
Vacuum World	Location + dirt	No dirt anywhere

2.3 Search Data Structures

- **Node:** State, parent node, action, path cost, depth
- **Frontier:** Open list of nodes to be explored (queue/stack/priority queue)
- **Explored set:** Closed list of already expanded states — prevents revisiting

2.4 Uninformed Search Strategies

BFS — Breadth-First Search

- Expands shallowest unexpanded node; uses FIFO queue
- Complete: Yes | Optimal: Yes (uniform cost) | Time: O(b^d) | Space: O(b^d)

DFS — Depth-First Search

- Expands deepest unexpanded node; uses LIFO stack
- Complete: No (infinite loops) | Optimal: No | Time: O(b^m) | Space: O(bm)

Uniform Cost Search (UCS)

- Expands node with lowest path cost g(n); uses priority queue
- Complete: Yes | Optimal: Yes | Handles varying step costs

Algorithm	Complete	Optimal	Time	Space
BFS	Yes	Yes (uniform)	O(b^d)	O(b^d)
DFS	No	No	O(b^m)	O(bm)
UCS	Yes	Yes	O(b^C/e)	O(b^C/e)

IDDFS	Yes	Yes	O(b^d)	O(bd)
Bidirectional	Yes	Yes	O(b^(d/2))	O(b^(d/2))

2.5 Iterative Deepening DFS (IDDFS)

IDDFS runs depth-limited search with limits 0,1,2,... until solution found. Combines BFS completeness/optimality with DFS linear space advantage O(bd).

Note: IDDFS is preferred for uninformed search when solution depth is unknown. Node revisits add only ~11% overhead.

2.6 Informed (Heuristic) Search

Heuristic h(n) estimates cost from node n to goal. Guides search toward promising states.

Greedy Best-First Search

- Expands node with smallest h(n); not optimal; may follow wrong paths

A* Search

Evaluates f(n) = g(n) + h(n) where g(n)=actual cost so far, h(n)=heuristic estimate:

- Optimal if h(n) is admissible: h(n) <= h*(n) (never overestimates)
- Consistent heuristic: h(n) <= c(n,a,n') + h(n') — ensures non-decreasing f values
- Complete and optimally efficient with admissible consistent heuristic

2.7 Heuristic Functions for 8-Puzzle

Heuristic	Formula	Admissible?	Consistent?
Misplaced tiles	Count tiles not in goal position	Yes	No
Manhattan distance	Sum of |row_curr-row_goal| + |col_curr-col_goal| for each tile	Yes	Yes
Pattern database	Exact cost for subsets of tiles	Yes	Yes

Note: Manhattan distance dominates misplaced tiles: h_manhattan(n) >= h_misplaced(n) for all n. More informed = fewer nodes expanded.

UNIT III
LOCAL SEARCH AND OPTIMIZATION PROBLEMS

3.1 Local Search Overview

Local search operates on a single current state without tracking paths. Useful for optimization where the goal state itself is the solution (not the path to it). Uses O(1) memory.

3.2 Hill-Climbing Search

Moves to the neighbor with the highest value. Stops at local maximum where no neighbor is better.

Problem	Cause	Solution
Local maximum	Peak better than neighbors, not global	Random restart hill climbing
Plateau	All neighbors have equal value	Random walk, sideways moves
Ridge	Sequence of local maxima	Diagonal moves, SA

- **Steepest-Ascent HC:** Pick the neighbor with highest improvement
- **Stochastic HC:** Pick randomly from uphill moves
- **Random-Restart HC:** Run multiple times from random initial states — practically complete

3.3 Simulated Annealing (SA)

SA allows occasional downhill moves with probability P = e^(delta_E / T) where T = temperature:

- High T: frequently accepts bad moves (exploration)
- Low T: rarely accepts bad moves (exploitation)
- Cooling schedule: T decreases from T_max to 0 over time
- Theoretical guarantee: finds global optimum if T decreases logarithmically

Note: SA is used in VLSI chip layout, airline scheduling, and protein structure prediction.

3.4 Local Beam Search

Maintains k states simultaneously. Generates all successors of all k states and selects the best k. Stochastic beam search selects k proportional to fitness — like natural selection.

3.5 Local Search in Continuous Spaces

- **Gradient Descent:** x = x - alpha * grad_f(x) — minimize loss (used in deep learning)
- **Gradient Ascent:** x = x + alpha * grad_f(x) — maximize objective
- **Newton-Raphson:** Uses second derivative (Hessian) for faster quadratic convergence

3.6 Nondeterministic Actions: AND-OR Trees

When actions have uncertain outcomes, solutions must be contingency plans:

- OR nodes: agent chooses an action (branching by choice)
- AND nodes: environment determines outcome (branching by uncertainty)
- Solution = conditional plan: if-then-else tree covering all outcomes
- Example: erratic vacuum world where Suck may dirt both squares or clean one

3.7 Minimax Algorithm

Minimax computes optimal decisions in two-player zero-sum games:

10. MAX player: selects action with maximum minimax value
11. MIN player: selects action with minimum minimax value
12. Leaf nodes: evaluate with utility function (+1 win, -1 loss, 0 draw)
13. Back-propagate values up the tree

Property	Value
Complete	Yes (finite game tree)
Optimal	Yes (vs. optimal opponent)
Time Complexity	O(b^m)
Space Complexity	O(bm)

3.8 Alpha-Beta Pruning

Cuts branches that cannot affect the final minimax decision:

- alpha: best value MAX can guarantee (starts at -inf)
- beta: best value MIN can guarantee (starts at +inf)
- Prune when alpha >= beta
- Best case with perfect ordering: O(b^(m/2)) — doubles searchable depth

Note: Alpha-beta gives the SAME result as minimax but much faster. Used in chess engines (Deep Blue).

3.9 Monte Carlo Tree Search (MCTS)

MCTS builds a tree based on random simulations. Four phases:

14. Selection: use UCB1 = w/n + C*sqrt(ln(N)/n) to select promising node
15. Expansion: add one or more child nodes to the selected node
16. Simulation: play out random moves until terminal state (rollout)
17. Backpropagation: update win/visit counts up the tree

Note: MCTS powers AlphaGo. No hand-crafted evaluation function needed. UCB1 balances exploitation vs. exploration.

3.10 Stochastic Games and Expectiminimax

Stochastic games add chance nodes (dice, card draws) to the game tree:

- MAX nodes: choose action to maximize expected utility
- MIN nodes: choose action to minimize expected utility
- CHANCE nodes: compute weighted average: E[U] = sum P(outcome) * U(outcome)
- Example: Backgammon — dice rolls are chance nodes

UNIT IV
CONSTRAINT SATISFACTION PROBLEMS

4.1 Defining CSPs

A Constraint Satisfaction Problem is defined by: Variables X={X1,...,Xn}, Domains D={D1,...,Dn}, Constraints C. Goal: complete assignment satisfying all constraints.

Component	Map Coloring Example
Variables	States: WA, NT, Q, NSW, V, SA, T
Domains	Colors: {Red, Green, Blue} for each state
Constraints	Adjacent states must have different colors: WA!=NT, WA!=SA, ...
Solution	WA=Red, NT=Green, Q=Red, NSW=Green, V=Red, SA=Blue, T=Red

Types of Constraints

- **Unary:** Restricts one variable: SA != Green
- **Binary:** Restricts two variables: WA != NT
- **Global:** Involves 3+ variables: AllDiff(Q1,Q2,Q3,Q4) in Sudoku

4.2 Constraint Propagation: AC-3 Algorithm

AC-3 enforces arc consistency by pruning domain values that violate constraints:

18. Initialize queue with all arcs (Xi, Xj)
19. Dequeue arc (Xi, Xj); remove values from Di with no compatible value in Dj
20. If Di changed, enqueue all arcs (Xk, Xi) for all neighbors Xk
21. Repeat until queue empty. If any domain becomes empty: no solution exists

Note: AC-3 Time complexity: O(cd^3) where c = number of constraints, d = domain size.

4.3 Backtracking Search with Heuristics

- **MRV (Minimum Remaining Values):** Choose variable with fewest legal values — most constrained variable first
- **LCV (Least Constraining Value):** Choose value that eliminates fewest choices for neighbors
- **Forward Checking:** After assignment, prune inconsistent values from unassigned neighbors
- **MAC:** Maintain Arc Consistency after each assignment — stronger than FC

4.4 Local Search for CSPs: Min-Conflicts

Start with a complete assignment (may violate constraints). Repeat: pick a conflicted variable; reassign to value minimizing violations:

- Solves million-queens problem in about 50 steps on average
- Used in NASA Space Shuttle mission scheduling

4.5 Propositional Logic

Symbol	Connective	Truth
NOT P (neg P)	Negation	True when P is False

P AND Q	Conjunction	True when both P,Q are True
P OR Q	Disjunction	True when at least one is True
P IMPLIES Q	Implication	False only when P=T, Q=F
P IFF Q	Biconditional	True when P and Q have same value

Key Inference Rules

- **Modus Ponens:** From (P -> Q) and P, conclude Q
- **Modus Tollens:** From (P -> Q) and NOT Q, conclude NOT P
- **Resolution:** From (P OR Q) and (NOT P OR R), conclude (Q OR R)

4.6 First-Order Logic (FOL)

Component	Description	Example
Constants	Specific objects	John, KingJohn, 5
Predicates	Relations/properties	King(x), Brother(x,y)
Functions	Maps objects to objects	Father(John), LeftLeg(x)
Universal (for all)	True for every object	forall x: Man(x) -> Mortal(x)
Existential	True for some object	exists x: Crown(x) AND OnHead(x,John)

4.7 Inference in FOL

- **Unification:** Find substitution theta making two sentences identical: UNIFY(Knows(John,x), Knows(John,Jane)) = {x/Jane}
- **Forward Chaining:** Start from known facts; apply rules to derive new facts; continue until goal reached
- **Backward Chaining:** Start from goal; find rules proving it; recursively prove subgoals (Prolog-style)
- **Resolution Refutation:** Add NOT(goal) to KB; derive empty clause to prove goal by contradiction

Method	Direction	Complete	Used In
Forward Chaining	Facts -> Goal	Yes (Horn clauses)	Production systems, databases
Backward Chaining	Goal -> Facts	Yes (Horn clauses)	Prolog, expert systems
Resolution	Refutation	Yes (general FOL)	Automated theorem proving

UNIT V
KNOWLEDGE REPRESENTATION AND REASONING

5.1 Ontological Engineering

Ontological engineering builds formal, structured representations of domain knowledge:

- Upper ontology: general concepts valid across all domains (time, space, events, objects)
- Domain ontology: specific to a field (medical, legal, financial, engineering)
- Languages: OWL (Web Ontology Language), RDF, DAML
- Applications: Google Knowledge Graph, Semantic Web, medical expert systems

5.2 Categories and Objects

- **Category:** Class of objects sharing properties: Animal, Vehicle, Person
- **Subtype:** Specialization: Dog is a subtype of Animal (Dog subset Animal)
- **Member:** Individual instance: Lassie is a member of Dog (Lassie in Dog)
- **Natural kinds:** Categories defined by intrinsic essence: Gold, Water, Species

5.3 Events and Situation Calculus

Represents dynamic worlds where actions change state over time:

- Situation: Complete description of the world at a moment in time
- Action: What an agent does to change the situation
- Fluent: Property that can change: On(Block,Table,s), Holding(robot,obj,s)
- Result(a,s): The situation that results from performing action a in situation s
- Frame Problem: How to represent all things that do NOT change — solved by successor state axioms

5.4 Mental Objects and Modal Logic

- **Necessity (box P):** P is true in all possible worlds
- **Possibility (diamond P):** P is true in at least one possible world
- **Knowledge K(agent,P):** Agent knows that P is true
- **Belief B(agent,P):** Agent believes P (may be false)

5.5 Reasoning Systems: Semantic Networks

Semantic networks represent knowledge as graphs where nodes are objects/concepts and edges are relations. Support inheritance reasoning:

- If All Birds have Wings and Tweety is a Bird, then Tweety has Wings
- Description Logics formalize semantic networks with TBox (terminological) and ABox (assertion) components
- Default inheritance: properties hold unless explicitly overridden (penguins can't fly)

5.6 Default and Non-Monotonic Reasoning

- **Default Logic:** Assumptions hold unless contradicted: 'Typically birds fly' — overridden for penguins
- **Non-monotonic reasoning:** Adding new facts can invalidate previous conclusions (unlike classical logic)

- **Open World Assumption (OWA):** Unknown facts may be true or false — used in FOL and OWL
- **Closed World Assumption (CWA):** Unknown facts are assumed false — used in Prolog and databases

5.7 Classical Planning: STRIPS

STRIPS (Stanford Research Institute Problem Solver) represents planning problems:

Component	Description	Example: Move(b,x,y)
Preconditions	Conditions that must be true	On(b,x) AND Clear(b) AND Clear(y)
Add List	Facts made true by action	On(b,y) AND Clear(x)
Delete List	Facts made false by action	On(b,x) AND Clear(y)

5.8 Planning Algorithms

Algorithm	Approach	Strength
Forward State Search	Apply actions from init state toward goal	Heuristics like FF, LAMA work well
Backward Search (Regression)	From goal, find actions achieving it	Less branching in some domains
GRAPHPLAN	Mutual exclusion layers + backward extraction	Polynomial for easy problems
SAT Planning	Encode as SAT, use solver	Optimal, handles large problems

5.9 Probability and Uncertainty

- **Prior P(A):** Unconditional probability of A without evidence
- **Conditional P(A|B):** P(A AND B) / P(B) — probability of A given B is known
- **Joint distribution:** P(X1,...,Xn) — complete probability over all variable combinations
- **Marginalization:** P(Y) = sum over all z of P(Y, Z=z) — summing out variables

5.10 Bayes' Rule

Bayes' Rule: P(H|E) = P(E|H) * P(H) / P(E)

- P(H|E): Posterior — probability of H after seeing evidence E
- P(E|H): Likelihood — how probable is E if H is true
- P(H): Prior — initial belief in H before evidence
- P(E): Marginal likelihood — normalizing constant

Note: Bayes' Rule is fundamental to all probabilistic AI. Used in spam filters, medical diagnosis, robot localization.

5.11 Naive Bayes Classifier

Assumes conditional independence of features given class: P(C|x1..xn) proportional to P(C) * product P(xi|C)

Aspect	Detail

Independence assumption	Features are conditionally independent given class (rarely true but works well)
Training	Estimate P(C) and P(xi|C) from labeled data (count frequencies)
Prediction	Argmax_C of P(C) * product P(xi|C)
Smoothing	Laplace smoothing prevents zero probabilities for unseen features
Applications	Spam detection, text classification, sentiment analysis, medical diagnosis

5.12 Bayesian Networks

Bayesian Networks are DAGs where nodes are random variables, edges are dependencies, each node has a CPT:

- Joint probability: P(X1,...,Xn) = product of P(Xi | Parents(Xi))
- Compact: O(n * 2^k) parameters vs O(2^n) for full joint (k = max parents)
- Inference: compute P(query | evidence) using variable elimination or MCMC
- Classic example: Burglary-Alarm-Earthquake network (Russell & Norvig)

-- End of Study Material | P23CAE22 Artificial Intelligence | 45 Periods --