

CS3243 Cheatsheet by Luke Aidan Tan

Lect 2: Uninformed Search

Problem-Solving Agents: Goal-based agents using atomic representations. They assume environments are observable, deterministic, and discrete.

Search Problem Formulation

State Space (s_i): Abstract data type describing the environment.

Initial State (s₀): Starting point.

Actions(s): Returns possible actions {a₁,...,a_k}.

Transition Model T(s,a): Returns state s' from taking action a.

Goal Test: Returns 1 if state is a goal.

Path Cost: Sum of action costs; usually assumes costs ≥0.

Tree Search vs. Graph Search

Tree Search: Explores all paths; may fail in cycles (infinite loops).

Graph Search: Maintains a reached (visited) set to avoid redundant paths.

V1: Adds to frontier only if state was never reached.

V2: Adds if never reached OR if new path is cheaper.

Uninformed Search Algorithms

Definition: No domain knowledge beyond the problem formulation.

Breadth-First Search (BFS)

Frontier: FIFO Queue.

Strategy: Explores shallowest nodes first.

Early Goal Test: Check goal when pushing to frontier.

Properties: Complete (if b is finite); Optimal only if action costs are uniform.

Uniform-Cost Search (UCS)

Frontier: Priority Queue (by path cost g(n)).

Strategy: Explores cheapest paths first.

Late Goal Test: Check goal when popping (ensures optimality).

Properties: Complete and Optimal if costs > ε > 0.

Depth-First Search (DFS)

Frontier: LIFO Stack.

Strategy: Explores deepest nodes first.

Properties: Not complete (infinite paths); Not optimal. Space efficient: O(bm) or O(m) with backtracking.

Iterative Deepening (IDS)

Strategy: Run DLS with limit l=0,1,2,...

Properties: completeness of BFS with space complexity of DFS.

Comparison Table (Tree Search)

Criterion	BFS	UCS	DFS	DLS	IDS
Complete?	Yes ¹	Yes ^{1,2}	No ³	No	Yes ¹
Optimal?	No ⁴	Yes	No	No	No ⁴
Time	$O(b^d)$	$O\left(b^{1+\lceil \frac{c^*}{\epsilon} \rceil}\right)$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^d)$	$O\left(b^{1+\lceil \frac{c^*}{\epsilon} \rceil}\right)$	$O(bm)$	$O(bl)$	$O(bd)$

Notes:

- Finite *b* and solution exists.
- Action costs > ε > 0.
- Space must be finite.
- Optimal if action costs are identical.

Graph Search Summary

Definition: Unlike Tree Search, Graph Search maintains a reached set (visited nodes) to avoid redundant paths and cycles.

Time and space complexities are now bounded by the size of the search space: |V| + |E| [nodes + edges].

Comparison Table (Graph Search)

Criterion	BFS	UCS	DFS	DLS	IDS
Complete?	Yes ¹	Yes ^{1,2}	No ³	No	Yes ¹
Optimal?	No ⁴	Yes	No	No	No ⁴
Time			$O(V + E)$		
Space			$O(V + E)$		

Notes:

- Complete if branching factor *b* is finite AND either has a solution or max depth *m* is finite.
- Complete if all action costs are > ε > 0.

- DFS is incomplete unless the search space is finite (*b* and *m* are both finite).
- Cost optimal if action costs are identical (and several other specific cases).

Implementation Rules for CS3243

- General Default:** Unless otherwise mentioned, assume **Tree Search**.
- Graph Search Assumption:**
 - Use **V2** for UCS (allows updating reached costs).
 - Use **V1** for all other uninformed search algorithms.
- Redundancy:** We do not check for “cheaper paths” in Graph Search for BFS or DFS because costs do not affect their traversal and they do not guarantee optimality anyway.

Default Search Space Assumptions

- Branching factor *b* is finite.
- Goal depth *d* is finite (a solution exists).
- Maximum depth *m* is infinite.
- All action costs are > ε > 0.

Algorithm Strategies

- BFS:** Frontier is a **FIFO Queue**. Uses **Early Goal Test** (checks goal when pushing to frontier).
- UCS:** Frontier is a **Priority Queue** (by *g(n)*). Uses **Late Goal Test** (checks when popping).
- DFS:** Frontier is a **LIFO Stack**. Space can be *O(m)* with backtracking.
- IDS:** Iterative DLS (*l* = 0, 1, 2...). Combines BFS completeness with DFS space efficiency.

Lect 3: Informed Search (Heuristics)

Informed Search: Uses domain-specific knowledge to find solutions more efficiently than uninformed search. It utilizes a **heuristic function** h(n) to estimate the cost from node **n** to the nearest goal.

Heuristic Functions

- h(n):** Estimated cost of the cheapest path from the state at node **n** to a goal state.
- Consistency:** h(n) is consistent if for every node **n** and every successor **n'** generated by action **a**, h(n) <= cost(n, a, n') + h(n').
- Admissibility:** h(n) is admissible if it never overestimates the cost to reach the goal (i.e., h(n) <= h(**n**), where **h(n)** is the true cost).
- Note:** All consistent heuristics are admissible, but not all admissible heuristics are consistent.

Greedy Best-First Search

- Frontier:** Priority Queue ordered by f(n) = h(n).
- Strategy:** Expands the node that appears to be closest to the goal.
- Properties:** Neither optimal nor complete (can get stuck in loops).
- Complexity:** Time and Space are *O(b^m)* in the worst case.

A* Search

- Frontier:** Priority Queue ordered by f(n) = g(n) + h(n).
- g(n):** Cost to reach node **n** from the start.
- f(n):** Estimated total cost of path through **n** to goal.

A* Optimality & Completeness

- Tree Search:** A* is optimal if h(n) is admissible.
- Graph Search:** A* is optimal if h(n) is consistent.
- Completeness:** A* is complete if **b** is finite and all action costs > ε > 0.

Heuristic Design

- Dominance:** If h2(n) >= h1(n) for all **n**, then h2 dominates h1 and is usually more efficient.
- Relaxed Problems:** Admissible heuristics can be derived by solving a “relaxed” version of the problem with fewer constraints.
- Combining Heuristics:** If you have multiple admissible heuristics, h(n) = max(h1(n), h2(n), ..., hk(n)) is also admissible and dominant.

Algorithm	Complete	Optimal	Time/Space
Greedy BFS	No	No	$O(b^m)$
A* Search	Yes	Yes ¹	$O(b^d)$

Notes: 1. Optimal under admissibility (Tree) or consistency (Graph).

Lect 4: Heuristics

Lect 5: Local Search

Lect 6: Constraint Satisfaction Problems (CSP)

Definition: A state representation using:

- Variables:** $X = \{x_1, ..., x_n\}$
- Domains:** $D = \{d_1, ..., d_k\}$ for each x_i
- Constraints:** $C = \{c_1, ..., c_m\}$ specifying legal combinations of values

Goal: A **complete** (all variables assigned) and **consistent** (all constraints satisfied) assignment.

Constraint Types

- Unary:** Scope = 1 variable.
- Binary:** Scope = 2 variables (represented as edges in a graph).
- Global:** Scope > 2 (represented in hypergraphs).

Backtracking Search (BT)

Basic DFS for CSPs. One variable assigned per level. **Algorithm Efficiency:** Improved by choosing (A) which variable to assign, (B) which value to try, and (C) detecting early failure via inference.

(A) Variable-Order Heuristics

- Goal:** Fail-first (prune large subtrees early).
- Minimum-Remaining-Values (MRV):** Choose variable with fewest legal values. Also called “Most Constrained Variable.”
- Degree Heuristic:** Tie-breaker for MRV. Choose variable involved in the most constraints with other **unassigned** variables.

(B) Value-Order Heuristics

Goal: Fail-last (try to find a solution immediately).

- Least-Constraining-Value (LCV):** Choose value that rules out the **fewest** choices for neighboring unassigned variables. Maximizes flexibility.

(C) Inference & Consistency

Ensure local consistency to prune the search space.

Forward Checking (FC)

When *x* is assigned, delete inconsistent values from unassigned neighbors.

Limitation: Does not look ahead far enough to detect all failures (e.g., doesn't detect if two unassigned neighbors become incompatible with each other).

Node Consistency

Variable *x_i* is node-consistent if all values in *D_i* satisfy *x_i*'s unary constraints.

Arc Consistency (AC)

X_i is arc-consistent wrt *X_j* iff for **every** value *x* ∈ *D_i*, there exists some *y* ∈ *D_j* that satisfies the binary constraint (*X_i*, *X_j*).

- Arcs are **directed** (asymmetric).
- Binary constraints = 2 arcs.

AC-3 Algorithm

- Init:** Queue contains all arcs in CSP.
- Process:** Pop (*X_i*, *X_j*). Use REVISE to prune *D_j*.
- Propagation:** If *D_j* changes, add all neighbors *X_k* of *X_j* (except *j*) back to queue as (*X_k*, *X_j*).
- Complexity:** *O(n²d³)* for *n* variables, domain size *d*.

Strategy	Heuristic / Algorithm
Var Order	MRV (Min Remaining Values)
Tie-break	Degree Heuristic
Value Order	LCV (Least Constraining Value)
Inference	FC, AC-3 (Arc Consistency)

Lect 7: Adversarial Search

Definition: Searching in environments where your opponent's actions can spoil your plans.

Zero-Sum: *U*(MAX, *s*) + *U*(MIN, *s*) = 0. Winner gets paid, loser pays.

MAX: Player 1; wants to maximize value (our agent).

MIN: Player 2; wants to minimize value (opponent).

Minimax Algorithm

Simulates play against a utility-maximizing opponent to find an optimal strategy.

- Process:** Uses Backwards Induction to propagate utility values from terminal states up the tree.

- Minimax(s):**
 - Utility(*s*, MAX) if Is-Terminal(*s*)
 - $\max_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a))$ if To-Move(*s*) = MAX

- $\min_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a))$ if To-Move(*s*) = MIN
- Properties:** Complete (if finite); Optimal (against optimal opponent).
- Complexity:** Time *O(b^m)*; Space *O(bm)* (DFS-based).

alpha-beta Pruning

Prunes subtrees that won't affect the final decision by maintaining bounds.

- alpha (a):** MAX's best (highest) option seen so far on path.
- beta (b):** MIN's best (lowest) option seen so far on path.
- Pruning Rules:**
 - In a MIN node *n*, stop if ∃ MAX ancestor *m* where $\alpha(m) \geq \beta(n)$.
 - In a MAX node *n*, stop if ∃ MIN ancestor *n* where $\beta(n) \leq \alpha(m)$.
- Effectiveness:** “Perfect” ordering reduces complexity to *O(b^{2/3})*, allowing search twice as deep.

Heuristic Minimax

Used when trees are too massive (e.g., Chess) to reach terminal states.

- Cutoff-Test:** Replaces terminal test with a depth limit or resource limit.
- Evaluation Function (Eval):** Estimates a state's utility using features.
- Linear Weighted Sum:** $f(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$.
- Features:** E.g., in Chess: no. of queens, controlled squares, threatened pieces.

Algorithm	Efficiency / Best Case
Minimax	Time $O(b^m)$, Space $O(bm)$
a-b Pruning	Perfect order: $O(b^{2/3})$
Chess Tree	$\approx 10^{123}$ nodes (requires heuristics)
Stochastic	Adds “chance” nodes; uses Expected Value

Lect 8: Logical Agents

Knowledge-Based Agents: Maintain a **Knowledge Base (KB)** of sentences in a formal language.

- Tell:** Add new info to KB.
- Ask:** Query what follows from KB.

Entailment (α ⊨ β): β follows logically from α if in every model where α is true, β is also true.

Inference (α ⊢ β): Sentence β can be derived from α by procedure *i*.

- Soundness:** *i* derives only entailed sentences ($\models \subseteq \vdash$).
- Completeness:** *i* can derive all entailed sentences ($\models \subseteq \vdash$).

Propositional Logic

Syntax: Atomic propositions (*P*, *Q*), operators (¬, ∧, ∨, →, ⇔).

Truth Tables:

<i>P</i>	<i>Q</i>	¬ <i>P</i>	<i>P</i> ∧ <i>Q</i>	<i>P</i> → <i>Q</i>
T	T	F	T	T
T	F	F	F	F
F	T	T	F	T
F	F	T	F	T

Note: *P* → *Q* is false **only** if *P* is T and *Q* is F.

Equivalences:

- $P \rightarrow Q \equiv \neg P \vee Q$
- $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ (De Morgan)
- $P \rightarrow Q \equiv \neg Q \rightarrow \neg P$ (Contrapositive)

Validity: True in **all** models (Tautology).

Satisfiability: True in **some** model.

Unsatisfiability: True in **no** models.

CS1231 Stuff

Syntax: Objects, Relations (Predicates), Functions, Quantifiers (∀, ∃).

Quantifiers:

- $\forall x P(x)$: “For all *x*, *P* is true.” (Usually pairs with →).
- $\exists x P(x)$: “There exists an *x* such that *P* is true.” (Usually pairs with ∧).
- $\neg \forall x P(x) \equiv \exists x \neg P(x)$
- $\neg \exists x P(x) \equiv \forall x \neg P(x)$

Higher-Order Logic: FOL allows quantifying over objects, but NOT over predicates/relations.

Sentence to Proposition Mapping

- If **A then B:** *A* → *B* (Implies)
- A only if B:** *A* → *B*
- B if A:** *A* → *B*
- A if and only if B:** *A* ⇔ *B* (Biconditional)
- A unless B:** $\neg B \rightarrow A$ (or *A* ∨ *B*)
- A but B / A although B:** *A* ∧ *B*
- Neither A nor B:** $\neg A \wedge \neg B$ (or $\neg(A \vee B)$)

Law Name	Disjunction Version	Conjunction Version
----------	---------------------	---------------------

De Morgan's	$\neg(p \vee q) \equiv \neg p \wedge \neg q$	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
Idempotent	$p \vee p \equiv p$	$p \wedge p \equiv p$
Associative	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Commutative	$p \vee q \equiv q \vee p$	$p \wedge q \equiv q \wedge p$
Distributive	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Identity	$p \vee \text{False} \equiv p$	$p \wedge \text{True} \equiv p$
Domination	$p \vee \text{True} \equiv \text{True}$	$p \wedge \text{False} \equiv \text{False}$
Complement	$p \vee \neg p \equiv \text{True}$	$p \wedge \neg p \equiv \text{False}$
Absorption	$p \vee (p \wedge q) \equiv p$	$p \wedge (p \vee q) \equiv p$
Double Negation	$\neg(\neg p) \equiv p$	
Conditional ID	$(p \rightarrow q) \equiv (\neg p \vee q)$	$(p \Leftrightarrow q) \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Lect 9a: Logic Agents

Inference Properties

- Soundness:** (“KB” tack.r.short_A alpha) arrow (“KB” models alpha). A only infers valid sentences.
- Completeness:** (“KB” models alpha) arrow (“KB” tack.r.short_A alpha). A can infer all valid sentences.
- Deduction Theorem:** (“KB” models alpha) iff (“KB” arrow alpha) is valid.
- Proof by Contradiction:** (“KB” models alpha) iff (“KB” and not alpha) is unsatisfiable.

Resolution Algorithm

Sound and complete for Propositional Logic.

- Convert (“KB” and not alpha) to **Conjunctive Normal Form (CNF)**.
- Repeatedly apply resolution rule to pairs of clauses.
- Rule:** (ell_1 or dots or x) and (m_1 or dots or not x) derives (ell_1 or dots or m_1 or dots).
- If empty clause (square) found, alpha is entailed. If no more resolutions possible, alpha is not entailed.

CNF Conversion Rules

- Eliminate $\Leftrightarrow$: $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$
- Eliminate $\rightarrow$: $\neg \alpha \vee \beta$
- De Morgan: $\neg(\alpha \vee \beta) \rightarrow (\neg \alpha \wedge \neg \beta)$; $\neg(\alpha \wedge \beta) \rightarrow (\neg \alpha \vee \neg \beta)$
- Double Negation: $\neg(\neg \alpha) \rightarrow \alpha$
- Distribute $\vee / \wedge$: $(\alpha \vee (\beta \wedge \gamma)) \rightarrow (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Lect 9b: Uncertainty I

Stochastic Environments

Intermediate states cannot be determined purely by actions.

- Sources:** Partial observability, noisy sensors, outcome uncertainty, modeling complexity.

Probability Basics

- Random Variable (X):** Quantifies a random outcome (Boolean or Discrete).
- Axioms:**

$$\sum P(v) = 1; P(A) + P(B) = P(A \cap B) + P(A \cup B)$$
- Joint Probability:** Likelihood of atomic events (x, y) occurring together:

$$p(x, y) = \Pr[X = x \wedge Y = y]$$

- Marginalization:**

$$p(x) = \sum_y p(x, y)$$

- Conditional Probability:**

$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

- Bayes’ Rule:**

$$\Pr[A|B] = \frac{\Pr[B|A]\Pr[A]}{\Pr[B]}$$

Term	Logic	Probability
Knowledge	Certainty (“KB”)	Likelihood (Pr)
Method	Resolution	Inference/Bayes
Action	Entailment (models)	Max Expected Utility

Resolution

A sound and complete inference rule for PL and FOL. Rule:

$$\frac{(\ell_1 \vee \dots \vee \ell_k), (m_1 \vee \dots \vee m_n)}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

where ℓ_i and m_j are complementary literals.

Proof by Contradiction: To prove $\text{KB} \models \alpha$, show $\text{KB} \wedge \neg \alpha$ is unsatisfiable (derives empty clause $\square$).

Lect 10: Uncertainty II

Probability Fundamentals

Bayes Rule:

$$\Pr[A|B] = \frac{\Pr[B|A]\Pr[A]}{\Pr[B]}$$

Chain Rule:

$$\Pr[R_1 \wedge \dots \wedge R_k] = \prod_{j=1}^k \Pr[R_j \mid R_1 \wedge \dots \wedge R_{j-1}]$$

Independence: A, B indep. iff

$$\Pr[A \wedge B] = \Pr[A]\Pr[B] \text{ or } \Pr[A|B] = \Pr[A]$$

Marginalization:

$$\Pr[X] = \sum_{y \in X} \Pr[X = y]$$

Bayesian Networks (BN)

Represent joint distributions via a Directed Acyclic Graph (DAG).

- Nodes:** Random variables.
- Edges:** $X \text{ to } Y$ means X directly influences Y .
- CPT:** Distribution over Z for each combination of parent values: $\Pr[Z \mid \text{Parents}(Z)]$.
- Complexity:** Requires $O(n2^k)$ values for k parents, vs $O(2^n)$ for full joint.

Conditional Independence

A, B are conditionally independent given S if:

$$\Pr[A \wedge B \mid S] = \Pr[A|S]\Pr[B|S]$$

Markov Blanket: A node is conditionally independent of all other nodes given its **parents**, **children**, and **children’s parents**.

Naive Bayes Model

Assumes effects E_i are conditionally independent given a Cause.

$\Pr[\text{Cause} \mid E_1, \dots, E_n] = \alpha \Pr[\text{Cause}] \prod_i \Pr[E_i \mid \text{Cause}]$

- α is a normalization constant.
- Highly efficient: Size of joint table becomes linear: $2(n - 1) + 1$.

BN Relationships

- Independent Causes (V-structure):**

$$\Pr[A, B, C] = \Pr[C|A, B] \Pr[A] \Pr[B]$$

- Cond. Indep. Effects:**

$$\Pr[A, B, C] = \Pr[B|A] \Pr[C|A] \Pr[A]$$

- Causal Chain:**

$$\Pr[A, B, C] = \Pr[C|B] \Pr[B|A] \Pr[A]$$

Decision-Theoretic Agents

Decision Theory = Probability Theory + Utility Theory.

Principle: Maximum Expected Utility (MEU).

- Agent is rational iff it chooses actions that maximize utility.
- Expected Utility:** $\sum_{s \in \mathcal{T}} \text{utility}(s) \times \Pr(s \mid a_i, s')$.
- Accounts for uncertainty in stochastic environments where actions don’t always lead to the same state.

Scenario	Network Size (Boolean)
Full Joint	$2^n - 1$ entries
Absolute Indep.	n entries
Naive Bayes	$2(n - 1) + 1$ entries
Bayesian Net	$O(n2^k)$ entries