6.4110 Cheat Sheet (AI Algorithms)

Gatlen Culp (gculp@mit.edu) · May 2025

1 Probability & Bayes Filters

Prediction: $\text{bel}^-(x_t) = \int p(x_t|x_{t-1}, u_{t-1}) \cdot \text{bel}(x_{t-1})\, dx_{t-1}$

- State estimation over time using dynamics model
- Incorporates control inputs and prior belief

Update: $\text{bel}(x_t) = \text{eta} \cdot p(z_t|x_t) \cdot \text{bel}^-(x_t)$

- Corrects prediction using measurement model
- eta is normalization constant: $\frac{1}{P(z_t)}$

Log Odds: $l_t = l_{t-1} + \log\left(\frac{p(z_t|x_t)}{p(z_t|\neg(x_t))}\right)$

- Numerically stable for binary states
- Recover probability: $P(x) = 1 - \frac{1}{1+\exp(l)}$

Filters:

- **Kalman Filter (KF)**: Optimal for linear systems with Gaussian noise
- **Extended Kalman Filter (EKF)**: Linearizes non-linear systems with Jacobians
- **Unscented Kalman Filter (UKF)**: Propagates carefully chosen sample points
- **Particle Filter (PF)**: Represents belief with particles, handles multimodal distributions
- **Histogram Filter**: Discretizes state space, exact but scales poorly

Convergence: PF var $\propto \frac{1}{N}$; resample when $N_{\text{eff}} < \frac{N}{2}$

- **Effective Sample Size (ESS)**: $N_{\text{eff}} = \frac{1}{\sum_i}(w_i^2)$
- Resampling trades variance for bias

1.1 Gaussian Formulas

$X \sim \mathcal{N}(\mu_x, \sigma_x^2), Y \sim \mathcal{N}(\mu_y, \sigma_y^2)$:

Sum of Gaussians

$$X + Y \sim \mathcal{N}(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$$

Multivariate Gaussian

$$(X, Y) \sim \mathcal{N}\left((\mu_x, \mu_y), \begin{pmatrix} \sigma_x^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_y^2 \end{pmatrix}\right)$$

Conditional Gaussian

$$P(X|Y=y) = \mathcal{N}\left(\mu_x + \frac{\sigma_{xy}^2}{\sigma_y^2}(y - \mu_y), \sigma_x^2 - \frac{(\sigma_{xy}^2)^2}{\sigma_y^2}\right)$$

2 Graphical Models

Bayes Net: Directed acyclic graph representing

$$\Pr(X_1, ...X_n) = \prod_i \Pr(X_i \mid \text{Parents}(X_i))$$

Markov Network: Undirected graph with potential functions over cliques

2.1 D-separation

Determines if $X \perp\!\!\!\perp Y \mid (Z \coloneqq \{Z_1, Z_2, ...\})$

1. Treat edges as undirected, identify paths from $X \to Y$
2. Check if path blocked by Z
3. If all paths blocked, then conditionally independent

2.2 Markov Blanket (MB)

Shield that makes X independent from all other variables:

$$\text{MB(X)} \coloneqq X.\text{parents} \cup X.\text{children} \cup X.\text{spouses}$$

- X.spouses = parents of X's children
- $P(X \mid \text{MB}(X), Y) = P(X \mid \text{MB}(X))$

2.3 Variable Elimination (Type of Sum-Product on Markov Chains)

In MMs, factor out independent vars by grouping as $\gamma_X(Y = y)$ to show providing y isolates the effect of X

- You select the ordering
- Requires there be no loops
- Performs exact inference for:
  - Bayes Nets (Directed GMs)
  - MMs **Hidden Markov Models (HMMs)** (Undirected GMs)
  - Factor Graphs
  - Conditional Random Fields
- Complexity: $O\left(d^{\{w+1\}}\right)$ where w = tree-width
- Ordering affects efficiency (min-degree, min-fill heuristics)
- Works for Bayes Nets, Markov models, factor graphs

$$\sum_{d\in\{0,1\}} \left[P_{C|D}(c|d) \cdot P_D(d)\right] = P_C(c) = \gamma_D(c)$$

2.4 Sum-Product (Belief Propagation)

Message-passing algorithm for exact inference on trees:

2.4.1 Variable to Factor

$$\mu_{x\to f}(x) = \prod_{\forall g\in\{\text{ne}(x)\setminus f\}} \mu_{g\to x}(x)$$

$\text{ne}(x) \coloneqq$ Neighbors of x
$f =$ factor x is passing to

2.4.2 Factor to Variable

$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f\setminus x}\left[\phi_f(\mathcal{X}_f) \cdot \prod_{y\in(\text{ne}(f)\setminus x)} \mu_{y\to f}(y)\right]$$

$y =$ Incoming vars
$\mathcal{X}_f \coloneqq$ All vars connected to factor f (very confusingly, inside the sum it is referring to the assignments though.)
$\sum_{\mathcal{X}_f\setminus x} =$ Summation over all assignments of vars $\mathcal{X}_f \setminus x$

ie: Marginalize out all other vars

2.4.3 Marginal

$$\mu_{f\to x}(x) \propto \prod_{f\in\text{ ne}(x)} \mu_{f\to x}(x)$$

- Converges to exact solution on trees
- May still work well on loopy graphs (loopy BP)
- Max-product variant finds most likely assignment

3 Sampling & Approx Inference

3.1 Bayesian Network Sampling

3.1.1 Ancestral (Forward) Sampling:

Draw exact i.i.d. samples from joint distribution

```
order ← topological_sort(G)
for i = 1...M:
    for v in order:
        x_{i[v]} ← Sample(P(v | pa(v) = x_{i[pa(v)]}))
return {x_i}_{i=1}^M
```

- **Estimator**: $\hat{E}[f(V)] = \frac{1}{M}\sum_{i=1}^M f(x^{(i)})$
- **Pros**: Simple, exact, O(MN) complexity, parallelizable
- **Cons**: Cannot condition on evidence efficiently, needs full CPT access

3.1.2 Rejection Sampling: Sample from $P(V \mid E = e)$ by discarding mismatches

```
accepted ← []
while |accepted| < M:
    x ← AncestralSample(G)
    if x[E] = e:
        accepted.append(x)
return accepted
```

- **Acceptance rate**: $\hat{A} = P(E = e)$
- **Expected cost**: O(MN / $\hat{A}$)
- **Pros**: Unbiased, exact draws
- **Cons**: Exponential slowdown for rare evidence, wastes computation

3.1.3 Importance Sampling & Likelihood Weighting: Re-weight samples from easier proposal

- **Generic Importance Sampling**:

$$E_{P[f(X)]} = \frac{\sum_{i=1}^M w_i f(x^{(i)})}{\sum_{i=1}^M w_i}$$

- **Weights**: $w_i = \frac{P(x^{(i)})}{Q(x^{(i)})}$ where we draw $x^{(i)}$ from proposal Q
- **Likelihood Weighting** (evidence nodes fixed, others sampled forward):

```
order ← topological_sort(G)
for i = 1...M:
    weight ← 1
    for v in order:
        if v ∈ E:
            weight *= P(v = e_v | pa(v) = x_{i[pa(v)]})
            x_{i[v]} ← e_v
        else:
            x_{i[v]} ← Sample(P(v | pa(v) = x_{i[pa(v)]}))
    store (x_i, weight)
return {(x_i, weight)}_{i=1}^M
```

- **Weight**: $w(z) = \frac{P(z,e)}{Q(z)} = \prod_{e_j\in E} P(e_j \mid \text{pa}(e_j))$
- **Pros**: Uses all draws, works with tiny $P(E = e)$, parallelizable
- **Cons**: High variance if weights spread widely, requires stable numerical handling

Practical Tips:

- Pre-compute topological ordering for static networks
- Normalize weights periodically to avoid underflow
- Use log-space accumulation for numerical stability
- Consider adaptive proposals or MCMC when evidence is deep in the graph
- For sequential models, combine LW with systematic resampling (particle filtering)

Mnemonic: A-R-I

- Ancestral – **All** nodes forward
- Rejection – **Reject** bad evidence
- Importance – **Importance-weight** good evidence

3.2 Rejection Sampling

```
function REJECTION-SAMPLING(X, e, bn, N)
inputs: X, the query variable
    e, observed values for variables E
    bn, a Bayesian network
    N, the total number of samples to be generated
local variables: C, a vector of counts for each value of X, initially zero
for j = 1 to N do
    x ← PRIOR-SAMPLE(bn)
    if x is consistent with e then
        C[j] ← C[j] + 1 where x_j is the value of X in x
return NORMALIZE(C)
```

sample from easier distribution, reject if doesn't match evidence

- Simple but inefficient for unlikely evidence

3.3 Gibbs Sampling

1. Initialize all vars $X = (X_1, ..., X_n)$
2. Repeatedly sample X_i from $\Pr[X_i \mid X_{\neg i}]$ ($X_{\neg i} \coloneqq X - \{X_i\}$)
3. After enough steps, burn-in

- Markov blanket simplifies conditional computation: $X_i \sim \Pr(X_i|\text{MB}(X_i))$

3.4 Importance Sampling

1. Choose proposal dist. $Q(X)$ that is easy to sample from
2. Draw samples $x^{(1)}, ...x^{(s)}$ from $Q(x)$
3. Compute importance weights $W^{(s)} = \frac{P(x^{(s)})}{Q(x^{(s)})}$
4. Estimate $E[f(X)] \approx \sum_{i=1}^s \left(\frac{W^{(i)}}{\sum_j} W^{(j)}\right) f(x^{(i)})$

Sample $x \sim Q$, weight $w = \frac{P}{Q}$; var high if $Q \not\approx P$

- Adaptive importance sampling adjusts proposal during sampling
- Annealed importance sampling bridges from easy to target distribution

3.5 Markov Chain Monte Carlo (MCMC)

- Framework for sampling from complex distributions
- Constructs Markov chain with target as stationary distribution
- **Metropolis-Hastings**: Accept new state with probability min(1, P(new)Q(old|new)/P(old)Q(new|old))
- **Gibbs sampling**: Special case updating one variable at a time
- Convergence diagnostics: ESS (effective sample size), PSRF < 1.1

4 Temporal Models

4.1 Hidden Markov Models (HMM)

- Joint:

$$\Pr(X_{0:T}, Z_{1:T}) = \Pr(X_0) \prod_{t=1}^{T} [\Pr(Z_t|X_t) \cdot \Pr(X_t|X_{t-1})]$$

- Forward-backward algorithm for smoothing
- Viterbi algorithm for most likely state sequence
- **Smoothing**: Estimating $P\left(x_t|z_{\{1:T\}}\right)$ for some past time $t < T$
  - Uses both past observations (forward pass) and future observations (backward pass)
  - Provides more accurate state estimates than filtering by using all available data
  - Implemented using the forward-backward algorithm combining α_t and β_t

4.1.1 Viterbi Algorithm

Find most likely state sequence $x^*_{1:T}$ given observations $z_{1:T}$

$\delta_1(i) \leftarrow P(x_1 = i) \cdot P(z_1|x_1 = i)$ for all i
$\psi_1(i) \leftarrow 0$
For $t = 2$ to T:
  $\delta_{t(j)} \leftarrow \max_i[\delta_{t-1}(i) \cdot P(x_t = j|x_{t-1} = i)] \cdot P(z_t|x_t = j)$
  $\psi_{t(j)} \leftarrow \text{argmax}_i[\delta_{t-1}(i) \cdot P(x_t = j|x_{t-1} = i)]$
$x^*_T \leftarrow \text{argmax}_i \delta_{T(i)}$
For $t = T - 1$ down to 1:
  $x^*_t \leftarrow \psi_{t+1}(x^*_{t+1})$

- $\delta_{t(i)}$: Probability of most likely path ending in state i at time t
- $\psi_{t(i)}$: Backpointer to previous state in most likely path
- Uses log space for numerical stability in practice

4.2 HMM forward

$$\alpha_t = (A^T \cdot \alpha_{t-1}) \cdot B(z_t)$$

- $\alpha_{t(i)} = \Pr\left(z_{\{1:t\}}, x_t = i\right)$
- Backward: $\beta_{t(i)} = \sum_j A_{ij} B_{j(z_{\{t+1\}})} \beta_{\{t+1\}}(j)$
- Smoothing: $P\left(x_t|z_{\{1:T\}}\right) \propto \alpha_{t(i)}\beta_{t(i)}$

4.3 Kalman

$$\text{Predition } \hat{x}^- = F\hat{}x$$
$$P^- = F \cdot P \cdot F^T + Q$$
$$\text{update } K = P^- H^{T(H \cdot P^- \cdot H^T + R)^{\{-1\}}}$$

- State update: $\hat{x} = \hat{x}^- + K(z - H\hat{x}^-)$
- Covariance update: $P = (I - \text{KH})P^-$
- Information form efficient for high-dimensional measurements

5 Graph Search

Equivalences:

- $h = 0 \Rightarrow$ A* ≡ UCS
- $g = 0 \Rightarrow$ A* ≡ Greedy
- cost=1 $\Rightarrow$ BFS ≡ UCS
- **Admissible**: $h(n) \leq$ true cost
- **Consistent**: $h(n) \leq \text{cost}(n, n') + h(n')$
- A* with consistent h expands nodes in order of increasing f-value
- Admissible but inconsistent h may require reopening closed nodes

6 Classical Planning Heuristics

7 Logic & FOL Resolution

Propositional logic: SAT solvers (DPLL, WalkSAT)

- **Conjunctive Normal Form (CNF)**: $(A \vee B) \wedge (\neg C \vee D)$
- Clause learning improves backtracking efficiency

Resolution: refutation complete for CNF

- Resolve clauses: $(A \vee B)$ and $(\neg A \vee C) \rightarrow (B \vee C)$
- Complete for refutation (proving unsatisfiability)
- Forward/backward chaining for Horn clauses

7.1 First-Order Logic (FOL)

7.1.1 Vocabulary & Symbols

- **Constant**: Names a single object (A, B, ...) - Denotes $\mathcal{I}(c) \in \mathcal{U}$ in model
- **Predicate**: k-ary relation (P, Q, R) - Denotes $\mathcal{I}(P) \subset \mathcal{U}^k$
- **Function**: k-ary mapping (f, g, h) - Denotes $\mathcal{I}(f) : \mathcal{U}^k \rightarrow \mathcal{U}$
- **Variable**: Placeholder (x, y, z) - Assignment $\sigma(x) \in \mathcal{U}$
- **Connective**: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$ - Boolean operations
- **Quantifier**: $\forall$ ("for all"), $\exists$ ("there exists") - Range: $\mathcal{U}$

7.1.2 Syntax

Terms (denote objects):

- Constant or variable: A, x
- Function application: f(t_1,...,t_k)

Atomic formula: P(t_1,...,t_k) or t_1 = t_2

Sentences (closed formulas):

- Built with connectives on formulas
- Quantifiers on variables: $\forall x\alpha$, $\exists y\beta$

7.1.3 Semantics

Model $M = (\mathcal{U}, \mathcal{I})$:

- **Universe** $\mathcal{U}$: non-empty set of objects
- **Interpretation** $\mathcal{I}$: maps symbols to denotations

Term evaluation:

$$[[c]]^m_\sigma = \mathcal{I}(c),$$
$$[[x]]^m_\sigma = \sigma(x),$$
$$[[f(t_1, ..., t_k)]]^m_\sigma = \mathcal{I}(f)\left([[t_1]]^m_\sigma, ...\right)$$

Satisfaction/Truth (for assignment σ and model m):

$$m, \sigma \vDash P(t_1, ..., t_k) \quad \text{iff} \quad \left([[t_1]]^m_\sigma, ...\right) \in \mathcal{I}(P)$$

$$m, \sigma \vDash (t_1 = t_2) \quad \text{iff} \quad [[t_1]]^m_\sigma = [[t_2]]^m_\sigma$$

A sentence α is true in m ($m \vDash \alpha$) iff $m, \sigma \vDash \alpha$ for every σ

7.1.4 Semantic Notions

- **Satisfiable**: $\exists m : m \vDash \alpha$ – has at least one model
- **Unsatisfiable/Contradiction**: Not satisfiable
- **Valid/Tautology**: $\forall m : m \vDash \alpha$ – write $\vDash \alpha$
- **Entailment**: $\Gamma \vDash \beta$ if every model of Γ is also a model of β
- **Logical equivalence**: $\alpha \equiv \beta \Leftrightarrow \vDash (\alpha \Leftrightarrow \beta)$

7.1.5 Laws & Tips

- **De Morgan**: $\neg\forall x\alpha \equiv \exists x\neg\alpha$ and $\neg\exists x\alpha \equiv \forall x\neg\alpha$
- **Quantifier Shift**: $\forall x(\alpha \wedge \beta) \equiv (\forall x\alpha) \wedge \beta$ (when x not in β)
- **Universal Instantiation**: From $\forall x\alpha$ infer $\alpha\left[\frac{t}{x}\right]$
- **Existential Generalization**: From $\alpha\left[\frac{t}{x}\right]$ infer $\exists x\alpha$
- **Skolemization**: $\exists x \forall y P(x, y) \rightarrow \forall y P(f(y), y)$

7.1.6 Applications

- Knowledge representation & rule-based AI
- Relational database queries (SQL is subset of FOL with finite domains)
- Formal verification/program logics
- Type systems
- Natural-language semantics

7.2 Horn Clauses

Horn Clause: A clause (disjunction of literals) with exactly one positive literal

- Implication form: $\alpha \wedge \beta \wedge \gamma \Rightarrow \delta$
- Basis for many logic programming languages

Datalog: Horn clauses with no function symbols

- More efficient inference
- Decidable (guaranteed to terminate)

Prolog: Horn clauses with depth-first backward chaining

- Foundation of logic programming
- Adds extra features for handling negation, equality, and side-effects

7.3 Skolemization

Transforms statements in FOL to statements in predicate logic

Rename vars to be unique
Convert $\alpha \Rightarrow \beta$ to $\neg\alpha \vee \beta$
Push in negations (not $\forall x.\alpha$ to $\exists x.\neg\alpha$)
Prenex normal form (quantifiers at beginning) (same order)
Replace all $\exists$ with new function of enclosing var
Drop universal quantifiers
Convert to CNF
  Move ¬ inward
  Distribute ∨ over ∧

7.3.1 Others

Clause: Disjunction of literals (atom ∨ ¬atom)

8 MDP & Reinforcement Learning

Q-V Relationship:

$$V(s) = \max_a Q(s, a)$$

- V(s) represents the value of following the optimal policy from state s
- Q(s,a) represents the value of taking action a in state s, then following the optimal policy
- Policy extraction from Q-values:

$$\pi(s) = \text{argmax}_a Q(s, a)$$

Bellman Equation:

$$V(s) = \max_a \sum_{\{s'\}} T(s, a, s')[R(s, a, s') + \gamma V(s')]$$

- Expected future discounted reward starting from state s
- Policy extraction:

$$\pi(s) = \text{argmax}_a \sum_{\{s'\}} T(s, a, s')[R(s, a, s') + \gamma V(s')]$$

Value Iteration: $V_{\{k+1\}} = B[V_k]$ (contraction γ)

Initialize $V(s) = 0$ for all $s \in S$
repeat
  $\Delta = 0$
  for each $s \in S$:
    $v = V(s)$
    $V(s) = \max_a \sum_{\{s'\}} T(s, a, s')[R(s, a, s') + \gamma V(s')]$
    $\Delta = \max(\Delta, |v - V(s)|)$
until $\Delta < \varepsilon$
return V

Policy Iteration: eval π, then improve:

$$\pi'(s) = \text{argmax}_a Q^{\pi(s,a)}$$

Initialize $\pi(s)$ arbitrarily for all $s \in S$
repeat
  Solve $V^{\pi(s)} = \sum_{s'} T(s, \pi(s), s')\left[R(s, \pi(s), s') + \gamma V^{\pi(s')}\right]$
  policy_stable = true
  for each $s \in S$:
    old_action = $\pi(s)$
    $\pi(s) = \text{argmax}_a \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma V^{\pi(s')}\right]$
    if old_action $\neq \pi(s)$ then policy_stable = false

9 until policy_stable
10 return π

Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_{\{a'\}} Q\left(s_{\{t+1\}}, a'\right) - Q(s_t, a_t)\right)$$

- Model-free: learns directly from experience
- Off-policy: learns about optimal policy while following exploratory policy
- Guaranteed convergence with sufficient exploration

SARSA: $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$
- On-policy: learns about the policy being followed
- Safer in dangerous environments with exploration

Temporal Difference (TD)(λ):
- Balances immediate rewards with long-term returns
- $\lambda = 0$: Pure TD learning; $\lambda = 1$: Monte Carlo learning
- Eligibility traces provide credit assignment through time

9 POMDP Approximations

Most Likely State (MLS): Plan in most-likely state $s^* = \text{argmax}_s b(s)$
- **When works**: Unimodal, sharply peaked beliefs
- **When fails**: Diffuse/multimodal beliefs; information-gathering critical
- Approximates belief as point mass: $V(b) \approx V_{\text{MDP}(s^*)}$
- Policy: $\pi(b) \approx \pi_{\text{MDP}(s^*)}$

Most Likely Observation (MLO): Assume most-likely obs $z^* = \text{argmax}_z P(z \mid b, a)$
- **When works**: Highly peaked observation models
- **When fails**: Noisy observations; diverse outcomes important
- Prunes observation branching in planning
- Next belief:

$$\begin{aligned} b_{t+1}(s_{t+1}) &= \Pr(s_{t+1} \mid b_t, a_t, z_t) \\ &\propto \Pr(s_{t+1} | b_t, a_t) \Pr(z_t | s_{t+1}) \\ &= \sum_{s_t} \underbrace{\Pr(s_{t+1} \mid s_t, a_t) b_t(s_t)}_{\text{Transition Update}} \cdot \underbrace{\Pr(o_t \mid s_{t+1}, a_T)}_{\text{Obs. Update}} \end{aligned}$$

Quick-MDP Approximation (QMDP): Use MDP $Q's \Rightarrow a^* = \text{argmax}_a \sum_s b(s) Q_{\text{MDP}(s,a)}$
- Assumes perfect state knowledge after one step
- Ignores observation process in planning
- No value for information-gathering actions

Other Approaches:
- **Receding Horizon Control (RHC)**: Finite horizon H, execute first action, repeat
- **Point-based Value Iteration (PBVI)**: Sample belief points, backup over samples
- **POMCP**: Monte-Carlo tree search for POMDPs

10 Motion Planning

10.1 RT (Random Tree) Algorithms

- Regular RT: selects random (not nearest) node to extend

10.1.1 Rapidly-exploring Random Tree (RRT)

- Probabilistically complete (finds path, not necessarily optimal)
- Biases search toward unexplored regions
- Effectively handles high-dimensional spaces
- Typically leads to jagged paths since tree is never overwritten

1 Initial config (starting tree or node q_{init})
2 Repeat until goal within range or max-iters K
3 Sample point randomly q_{rand}
4 Find nearest node in tree q_{near}
Extend nearest node with a new node q_{new} distance
5 Δq along path to sample (if there is a collision, place the new node before the collision)

10.1.2 RRT*

- Optimal version of RRT that rewires tree for better paths
- Guaranteed asymptotic optimality

10.2 Probabilistic Roadmap Method (PRM)

PRM is a way of BUILDING a graph from a known C-Space (and then a search algorithm can be run on top of it).
- Probabilistically complete

1 Sample n collision-free points
2 Connect nearby points with collision-free paths
3 Search resulting roadmap with A* or similar

- **Multi-query**: Reuse roadmap for multiple planning problems
- **PRM***: Asymptotically optimal variant with connection radius
- **Variants**: LazyPRM, EST, BIT, **FMT**

11 CSP Algorithms

11.1 CSP Definitions

$\mathcal{X}$:= set of variables
$= \{X_1, ..., X_n\}$
$\mathcal{D}$:= sets of domains
$= \{D_1, ..., D_n\}$(ex: $D_i = \{r, g, b\}$)
$\mathcal{C}$:= set of constraints
$= \{C_1, ..., C_n\}$

Where C_i contains a scope (tuple of vars ex: (X_1, X_4, X_7)) and a relation (legal values, ex: $X_1 \neq X_4 \neq X_7$)

Arc: Directed binary constraint between two variables, denoted $C_{\{XY\}}$
- Represents constraint from variable X to variable Y
- Consistent if for every value of X, there exists a compatible value for Y

Consistent Assignment: For variables X, Y and constraint $C_{\{XY\}}$, an assignment to X is consistent if: $\forall x \in D(X), \exists y \in D(Y) : (x, y)$ satisfies $C_{\{XY\}}$

Arc Consistency: A directed constraint $C_{\{XY\}}$ is arc consistent if all values in $D(X)$ have at least one consistent value in $D(Y)$

11.2 Arc Consistency-3 (AC-3)

1 queue := {All arcs C_{XY}}
2 while queue:
3 $(X, Y) \leftarrow$ queue.pop()
4 if revise(X,Y):
5 if D(X) empty:
6 return failure
7 Add all arcs (Z, X) to queue where $Z \neq Y$
8 return success

11.3 Others

Heuristics: MRV (min remaining values) + Degree; Min-conflicts good for N-Queens
- **Minimum Remaining Values (MRV)**: choose variable with fewest legal values
- **Degree**: tie-breaker, choose variable with most constraints
- **Least Constraining Value (LCV)**: assign value that rules out fewest choices
- **Min-conflicts**: local search, flip to minimize violations

Global constraints: specialized algorithms for common patterns
- AllDiff: efficient for distinct value constraints
- Symmetry breaking reduces search space
- Constraint propagation algorithms vary by constraint type

12 Monte Carlo Tree Search (MCTS)

Goal/Theory
- Always have a "best route/sequence of actions" estimation even if stopped prematurely
- Starting from root, start expanding the tree, giving each node an estimated value V_i by simulating a random path ("rolling out") from that node and updating it if a child's random rollout is determined to be better than the parent's
- Good when branching factor is high.

Algorithm

1. Selection – Use max UCB1$(S_i) := \overline{V_i} + C\sqrt{\frac{\ln(N)}{n_i}}$ from root downward to determine next node to expand
   $i :=$ State or node index
   $\overline{V_i} :=$ Avg Value $= \frac{t_i}{n_i}$,
   $t_i :=$ Total value (of child nodes)
   $C :=$ Exploration const. higher = more explore,
   $N :=$ Nodes Visited,
   $n_i :=$ times i visited
2. Expansion – If i visited ($n_i > 0$), and thus has an estimated value, expand to i's children and evaluate
3. Simulation – From a starting node, take random actions (or some default policy) until reaching terminal node with value.
4. Backpropogation – Update statistics based on value or terminal node up to root (when using UCB1, this means updating the counts and total values)

13 PDDL

Structure: Domain file (rules/actions) + Problem file (instance)
- **Domain**: Types, predicates, functions, actions, constants
- **Problem**: Objects, initial state, goal state

Components:
- **Types**: Hierarchical categorization (e.g., `truck airplane - vehicle`)
- **Predicates**: Boolean state relations (e.g., `(at ?obj ?loc)`)
- **Actions**: State transformations with:
  - Parameters: Objects acted upon
  - Preconditions: Required state for execution
  - Effects: State changes after execution

Syntax Example:

```
(define (domain logistics)
    :effect (and (not (at ?v ?from)) (at ?v ?to)))
    :precondition (at ?v ?from)
    :parameters (?v - vehicle ?from ?to -
location)
  (:action move
  (:predicates (at ?obj ?loc))
  (:types truck airplane - vehicle)
```

Advanced Features:
- **Numeric Fluents**: Continuous quantities (`:functions`)
- **Durative Actions**: Actions with time (`?duration`, `:at start/end/over`)
- **Derived Predicates**: High-level properties (`:derived`)
- **Conditional Effects**: Context-dependent outcomes (`when`)

Advantages:
- Declarative: Specify what, not how
- Domain-problem separation enables reuse
- Formal semantics for automated analysis
- Standard interface to planning algorithms

Admissible: Doesn't ever give an overestimation to the goal.

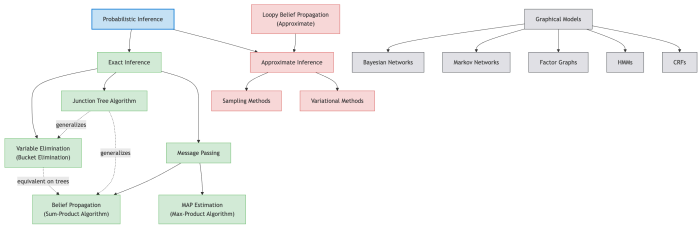
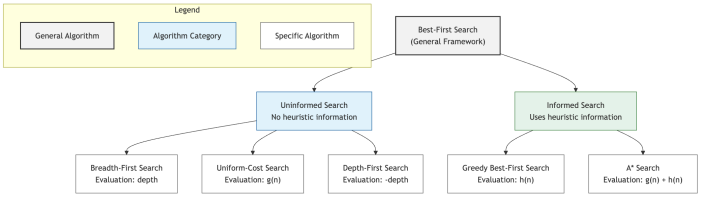


Figure 1: Inference Algorithms

07.02 Taxonomy of Search Algorithms



07.03 Evaluation Functions

The key to understanding Best-First Search variants is their evaluation function:

Algorithm	Evaluation Function	Description
Breadth-First Search	depth	Expands shallowest unexpanded node
Uniform-Cost Search	(g(n))	Expands node with lowest path cost from start
Depth-First Search	-depth	Expands deepest unexpanded node
Greedy Best-First Search	(h(n))	Expands node that appears closest to goal
A* Search	(f(n) = g(n) + h(n))	Expands node with lowest estimated total cost

Figure 2: Search Algorithms

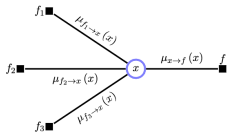
Procedure 5.2 (Max-Product messages on Factor Graphs)**.**

Given a distribution defined as a product on subsets of the variables, $p(\mathcal{X}) = \frac{1}{Z}\prod_f \phi_f(\mathcal{X}_f)$, provided the factor graph is singly connected we can carry out maximisation over the variables efficiently.

Initialisation Messages from leaf node factors are initialised to the factor. Messages from leaf variable nodes are set to unity.

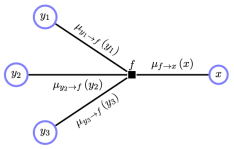
Variable to Factor message

$$\mu_{x\to f}(x) = \prod_{g\in\{\mathrm{ne}(x)\setminus f\}} \mu_{g\to x}(x)$$



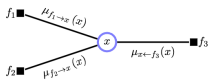
Factor to Variable message

$$\mu_{f\to x}(x) = \max_{\mathcal{X}_f\setminus x} \phi_f(\mathcal{X}_f) \prod_{y\in\{\mathrm{ne}(f)\setminus x\}} \mu_{y\to f}(y)$$



Maximal State

$$x^* = \operatorname*{argmax}_x \prod_{f\in\mathrm{ne}(x)} \mu_{f\to x}(x)$$



This algorithm is also called *belief revision*.

Figure 3: Inference Algorithms

— End of Cheat Sheet —