

# Introduction to Artificial Intelligence

Daniel Deng

## 1 Agents

**Definition 1.1** (Reflex Agent). A reflex agent chooses actions based on its current perception of the world.

**Definition 1.2** (Planning Agent). A planning agent chooses actions based on hypothesized consequences of actions.

## 2 General Search Problems

### 2.1 Heuristics

**Definition 2.1** (Heuristic). A heuristic  $h(n)$  is a function that estimates the distance from state  $n$  to the goal state for a particular search problem. It is often solutions of relaxed problems.

1. A heuristic is admissible if  $0 \leq h(n) \leq h^*(n)$  where  $h^*$  is the true cost to goal state;
2. A heuristic is consistent if  $h(n) - h(n+1) \leq c(n, n+1)$  where  $c$  is the cost between states  $n$  and  $n+1$ .

*Remark.* Consistency necessarily implies admissibility.

### 2.2 Search Algorithms

Table 1: Search algorithms.

	Fringe	Complete	Optimal	Time	Space
Depth-First Search	Stack	<i>iff</i> no cycle	No	$O(b^m)$	$O(bm)$
Breadth-First Search	Queue	Yes	<i>iff</i> uniform cost	$O(b^s)^1$	$O(b^s)^1$
Uniform Cost Search	PQ $(g(n))^2$	<i>iff</i> positive cost	Yes	$O(b^{c^*/\epsilon})^3$	$O(b^{c^*/\epsilon})^3$
Greedy Search	PQ $(h(n))$	-	No	-	-
$A^*$ Tree Search	PQ $(h(n) + g(n))$	-	<i>iff</i> $h(n)$ admissible	-	-
$A^*$ Graph Search <sup>4</sup>	PQ $(h(n) + g(n))$	-	<i>iff</i> $h(n)$ consistent	-	-

<sup>1</sup>  $s$  = depth of solution.

<sup>2</sup>  $g(n)$  = cumulative path cost.

<sup>3</sup>  $c^*/\epsilon$  = effective solution depth ( $c^*$  = cost of the cheapest solution;  $\epsilon$  = minimum cost of cost-contour arcs).

<sup>4</sup> Compared to tree search, graph search keeps a closed set of expanded states to check against to prevent duplicate expansions.

*Remark.* Implementation of search algorithms differ only in fringe strategies.

### 3 Constrained Satisfaction Problems

**Definition 3.1** (Constrained Satisfaction Problems). Constrained Satisfaction Problems (CSPs) are a type of **identification problem** defined by variable  $X_0, \dots, X_n$  with values from a domain  $D$  that satisfies a set of constraints.

---

**Algorithm 1:** Backtracking search.

---

**Input:** A constraint satisfaction problem  $P$ .

**Output:** A complete assignment  $A$ .

```

1 Function BS( $P$ ):
2   return EXPLORE ( $P$ , {})
3 Function EXPLORE( $P$ ,  $A$ ):
4   if  $A$  is complete then
5     return  $A$ 
6    $unassigned \leftarrow$  an unassigned VARIABLES( $P$ )
7   foreach  $value \in$  DOMAIN( $unassigned$ ) do
8     if  $value$  is consistent with all CONSTRAINTS( $P$ ) then
9       add { $unassigned \leftarrow value$ } to  $A$ 
10       $attempt \leftarrow$  EXPLORE( $P$ ,  $A$ )
11      if  $attempt$  failed then
12        remove { $unassigned \leftarrow value$ } from  $A$ 
13      else
14        return  $attempt$ 
15   return failed

```

---

### 3.1 Filtering

**Definition 3.2** (Arc Consistency).

Arc  $X \rightarrow Y$  is consistent  $\Leftrightarrow$   
 $(\forall x \in D_x)(\exists y \in D_y)(y \text{ can be assigned to } Y \text{ without violating a constraint.})$

---

**Algorithm 2:** Arc consistency filtering.

---

**Input:** A constraint satisfaction problem  $P$ .  
 /\*  $O(|S||A|^3)$  runtime \*/

```

1 Function AC-3( $P$ ):
2    $Q \leftarrow$  empty Queue
3   enqueue all arcs  $\in P$ 
4   while  $Q$  is not empty do
5      $(X_i, X_j) \leftarrow$  dequeue from  $Q$ 
6     if FILTER( $X_i, X_j$ ) is successful then
7       foreach  $X_k \in \text{NEIGHBORS}(X_i)$  do
8         enqueue  $(X_k, X_i)$ 
9 Function FILTER ( $tail, head$ ):
10   $result \leftarrow$  false
11  foreach  $value \in \text{DOMAIN}(tail)$  do
12    if  $value$  violates some constraint with all values in  $\text{DOMAIN}(head)$  then
13      delete  $value$  from  $\text{DOMAIN}(tail)$ 
14       $result \leftarrow$  true
15  return  $result$ 

```

---

### 3.2 Ordering

**Definition 3.3** (Minimum Remaining Values). The MRV policy chooses an unassigned variable that has the fewest valid remaining values in order to induce backtracking earlier and reduce potential node expansions.

**Definition 3.4** (Least Constraining Value). The LCV policy chooses a value assignment that violates the least amount of constraints, which requires additional computation such as running arc consistency test on each value.

### 3.3 Structure

Given a tree-structured CSP, represent it as a directed acyclic graph. Enforcing arc consistency in reverse topological order then assigning in topological order ensures a runtime of  $O(nd^2)$  (as opposed to  $O(d^n)$  in the general case).

TODO: nearly tree-like CSPs and tree decomposition.

## 4 Local Search

**Definition 4.1** (Local Search). A search strategy that improve a single option until no further improvements can be made. Typically, local search is faster and more memory efficient than other search algorithms but is generally neither complete nor optimal.

**Definition 4.2** (Hill Climbing). A CSP strategy that randomly selects a conflicting variable and reassign values using min-conflicts heuristics.

*Remark.* Efficiency of the algorithm depends on  $R = \frac{\text{number of constraints}}{\text{number of variables}}$ ; computation time is approximately constant time except when  $R$  approaches the *critical ratio*.

### 4.1 Simulated Annealing

---

**Algorithm 3:** Simulated annealing.

---

**Input:** A problem  $P$  and a schedule/mapping from time to "temperature"  $T$ .

**Output:** A solution state.

/\* Escape local maxima by allowing downhill movement based on a "temperature"-dependent probabilistic function. \*/

```

1 current  $\leftarrow$  initial state of  $P$ 
2 for  $t \leftarrow 1$  to  $\infty$  do
3    $temp \leftarrow T[t]$ 
4   if  $temp = 0$  then
5     return current
6   else
7      $next \leftarrow$  a randomly selected successor of current
8      $\Delta \leftarrow \text{VALUE}[next] - \text{VALUE}[current]$ 
9     if  $\Delta > 0$  then
10       $current \leftarrow next$ 
11     else
12       $currnet \leftarrow next$  with probability  $e^{\frac{\Delta}{temp}}$ 
```

---

### 4.2 Genetic Algorithm

**Definition 4.3** (Genetic Algorithm). A strategy that keeps the best  $N$  hypothesis at each step based on a fitness function and generate next generation using pairwise cross-over operations (and, optionally, mutation operations).

## 5 Games

**Definition 5.1** (Games). Games are multi-agent search problems that could be zero-sum (adversarial) or general sum.

### 5.1 Minimax

**Definition 5.2** (Minimax). A zero-sum-game algorithm that assumes the opponent is an optimal adversary.

---

**Algorithm 4:** Minimax with alpha-beta pruning.

---

**Input:** A game state  $S$   
**Output:** The root minimax value.  
 /\* initialize  $\alpha \leftarrow -\infty$  and  $\beta \leftarrow \infty$  \*/

```

1 Function VALUE( $S, \alpha, \beta$ ):
2   if  $S$  is a terminal state then
3     return known terminal value
4   if the agent is maximizing then
5     return MAX-VALUE( $S, \alpha, \beta$ )
6   if the agent is minimizing then
7     return MIN-VALUE( $S, \alpha, \beta$ )

8 Function MAX-VALUE( $S, \alpha, \beta$ ):
9    $v \leftarrow -\infty$ 
10  foreach successor  $S'$  of  $S$  do
11     $v \leftarrow \text{MAX}(v, \text{VALUE}(S'))$ 
12    if  $v \geq \beta$  then
13      return  $v$ 
14     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
15  return  $v$ 

16 Function MIN-VALUE( $S, \alpha, \beta$ ):
17    $v \leftarrow \infty$ 
18  foreach successor  $S'$  of  $S$  do
19     $v \leftarrow \text{MIN}(v, \text{VALUE}(S'))$ 
20    if  $v \leq \alpha$  then
21      return  $v$ 
22     $\beta \leftarrow \text{MIN}(\beta, v)$ 
23  return  $v$ 

```

---

## 5.2 Expectimax

**Definition 5.3** (Expectimax). A zero-sum-game algorithm that assumes the opponent acts based on some probability distribution.

---

**Algorithm 5:** Expectimax.

---

**Input:** A game state  $S$

**Output:** The root minimax value.

```

1 Function VALUE( $S$ ):
2   if  $S$  is a terminal state then
3     return known terminal value
4   if the agent is maximizing then
5     return MAX-VALUE( $S$ )
6   if the agent is randomizing then
7     return EXP-VALUE( $S$ )
8 Function MAX-VALUE( $S$ ):
9    $v \leftarrow -\infty$ 
10  foreach successor  $S'$  of  $S$  do
11     $v \leftarrow \text{MAX}(v, \text{VALUE}(S'))$ 
12  return  $v$ 
13 Function EXP-VALUE( $S$ ):
14   $v \leftarrow 0$ 
15  foreach successor  $S'$  of  $S$  do
16     $v \leftarrow \mathbb{E}[\text{VALUE}(S')] \text{ // expected value}$ 
17  return  $v$ 

```

---



## 6 Markov Decision Processes

*Remark.* Finite horizons (finite timestep before an agent terminates) and/or discount factors ( $\gamma$ ) ensure an agent terminates in MDP.

**Definition 6.1** (Transition Function).  $T(s, a, s') = P(s' \mid s, a)$

### 6.1 Value Iteration

---

**Algorithm 6:** Value Iteration

---

**Input:** A MDP  $(S, A, R, T, \gamma)$ .

**Output:** The optimal policy  $\pi^*(s)$  for all  $s \in S$ .

```

1 Function POLICY-EXTRACTION():
2    $V^* \leftarrow \text{VALUE-ITERATION()} \text{ // optimal value}$ 
3   foreach  $s \in S$  do
4      $\pi^*(s) \leftarrow \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$ 
5   return  $\pi^*(s) \mid \forall s \in S$ 
6 Function VALUE-ITERATION():
7   //  $O(|S|^2|A|)$  runtime
8   Initialize  $V_0(s) \leftarrow 0$  for all  $s \in S$ 
9   while  $V_{k+1}(s) \neq V_k(s) \mid \forall s \in S$  do
10    // repeat until values converge
11     $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$ 
12  return  $V^*(s) \leftarrow V_{k+1}(s) \mid \forall s \in S$ 

```

---

### 6.2 Policy Iteration

**Definition 6.2** (Policy Iteration). Define an initial policy  $\pi_0$  (can be arbitrary, but ideality close to the optimal policy). Then, iteratively solve  $\forall s \in S$

$$V^{\pi_i}(s) = \sum_{s'} T(s, \pi_i(s), s') [R(s, \pi_i(s), s') + \gamma V^{\pi_i}(s')] \quad (\text{policy evaluation})$$

$$\pi_{i+1}(s) \leftarrow \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')] \quad (\text{policy improvement})$$

until  $\pi(s)$  converges for all  $s \in S$  (yields  $\pi^*$ ).

*Remark.* Policy evaluation solves a system of  $|S|$  linear equations.

*Remark.* Policy iteration converges faster than value iteration.

## 7 Reinforcement Learning

*Remark.* Reinforcement learning operates on MDP problems where the  $T$  and  $R$  functions are unknown.

### 7.1 Model-Based Learning

**Definition 7.1** (Model-Based RL). An algorithm that counts and normalizes sample outcomes  $s'$  for each  $s, a$  to construct  $\hat{T}(s, a, s')$  and discovers each  $\hat{R}(s, a, s')$  through exploration. The approximated MDP is then solved by value or policy iteration.

### 7.2 Model-Free Learning

**Definition 7.2** (Direct Evaluation). An algorithm that fixes some policy  $\pi$  and empirically computes  $V^\pi(s)$  for all  $s \in S$  by averaging total sample utility for a given state.

*Remark.* Direct evaluation wastes information about state connections and will take a long time to learn.

**Definition 7.3** (Temporal Difference Learning).

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

*Remark.* Can't extract policy.

**Definition 7.4** (Q-Learning).

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[R(s, a, s') + \gamma \max_{a'} Q(s', a')]$$

*Remark.* Q-learning is an example of off-policy learning, in which the algorithm can learn the optimal policy even by taking sub-optimal or random actions.

**Definition 7.5** (Approximate Q-Learning). Represent q-values as weighted sums of features  $Q(s, a) := \vec{w} \cdot \vec{f}(s, a)$ . The update rule for Q-Learning then becomes

$$\begin{aligned} \Delta &\leftarrow [R'(s, a, s') + \gamma \max_{a'} Q(s', a')] - Q(s, a) \\ w_i &\leftarrow w_i + \alpha \cdot \Delta \cdot f_i(s, a) \end{aligned}$$

**Definition 7.6** ( $\epsilon$ -Greedy Policies). Define some probability  $0 \leq \epsilon \leq 1$  to act randomly and explore.  $\epsilon$  should be lowered over time to favor more exploitation as the learning becomes complete.

**Definition 7.7** (Exploration Function).

$$\begin{aligned} Q(s, a) &\leftarrow (1 - \alpha)Q(s, a) + \alpha[R(s, a, s') + \gamma \max_{a'} f(s', a')] \\ f(s, a) &= Q(s, a) + \frac{\text{const.}}{\text{count}(Q(s, a))} \end{aligned}$$

*Remark.*  $f(s, a)$  shown here is only a common example where the “bonus” for exploration diminishes as q-states are explored more and more.

## 8 Bayesian Network

**Definition 8.1** (Conditional Independence).  $\forall x, y, z : P(x|z, y) = P(x|z)$

**Definition 8.2** (Bayes' Net). Bayes' Net is a graphic model that describes complex joint distributions using simple, local distributions (conditional probabilities)

- Nodes—variables (with domains) [assigned = observed, unassigned = unobserved]
- Arcs—interactions (i.e., encode conditional independence when lacking arrows)  $\implies P(x_i|x_1, \dots, x_{i-1}) = P(x_i|\text{parents}(X_i))$
- A conditional distribution for each node

*Remark.* Bayes' Net does not imply causation, only the lack thereof

**Definition 8.3** (Inference by Enumeration). General procedure:

1. Join Factors:  $\forall r, t : P(r, t) = P(r)P(t|r)$
2. Eliminate:  $P(T) = \sum_r P(R, T)$
3. Normalize:  $P(Q|e_1, \dots, e_k) = \frac{P(Q, e_1, \dots, e_k)}{\sum_q P(Q, e_1, \dots, e_k)}$

**Definition 8.4** (Variable Elimination). Faster than inference by enumeration via interleaving joining and marginalizing