Introduction to Artificial Intelligence

Daniel Deng

1 Search Problems

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.

Definition 1.3 (Search Problem)**.** A search problem consists of a state space, a successor function, a start state, and a goal test.

2 Search Algorithms

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.

Definition 2.2 (Admissibility)**.** A heuristic is admissible, or optimistic, if $0 \leq h(n) \leq h^*(n)$ where h^* is the true cost to goal state.

Definition 2.3 (Consistency)**.** 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.

Table 1: Search algorithms.

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

[1] s = depth of solution.
[2] $g(n)$ = cumulative path cost.
[3] c^*/ϵ = effective solution depth (c^* = cost of the cheapest solution; ϵ = minimum cost of cost-contour arcs).
[4] 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 constrains.

Algorithm 1: Backtracking search.

```
Input: A constraint satisfaction problem P.
Output: A complete assignment A.
Function BS(P):
    return EXPLORE (P, {})
Function EXPLORE(P, A):
    if A is complete then
        return A
    unassigned ← an unassigned VARIABLES(P)
    foreach value ∈ DOMAIN(unassigned) do
        if value is consistent with all CONSTRAINTS(P) then
            add {unassigned ← value} to A
            attempt ← EXPLORE(P, A)
            if attemp failed then
                remove {unassigned ← value} from A
            else
                return attempt
    return failed
```

3.1 Filtering

Definition 3.2 (Arc Consistency)**.**

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

Algorithm 2: Arc consistency filtering.

```
Input: A constraint satisfaction problem P.
Function AC-3(P):
    Q ← empty Queue
    enqueue all arcs ∈ P
    while Q is not empty do
        (X_i, X_j) ← dequeue from Q
        if FILTER(X_i, X_j) is successful then
            foreach X_k ∈ NEIGHBORS(X_i) do
                enqueue (X_k, X_i)

Function FILTER (tail, head):
    result ← false
    foreach value ∈ DOMAIN(tail) do
        if value violates some constraint with all values in DOMAIN(head) then
            delete value from DOMAIN(tail)
            result ← true
    return reuslt
```

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

Improve a single option until no further improvements can be made.

Remark. Generally, local search is faster and more memory efficient at the expense of completeness and optimality.

4.1 Iterative Algorithm for CSP/Hill Climbing

Starting with a "complete" state, randomly select any conflicted variables 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*.

4.2 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.                  */
```

$current \leftarrow$ initial state of P
for $t \leftarrow 1$ *to* ∞ **do**
    $temp \leftarrow T[t]$
    if $temp = 0$ **then**
        return *current*
    else
        $next \leftarrow$ a randomly selected successor of *current*
        $\Delta \leftarrow$ VALUE[*next*] - VALUE[*current*]
        if $\Delta > 0$ **then**
            $current \leftarrow next$
        else
            $currnet \leftarrow next$ with probability $e^{\frac{\Delta}{temp}}$

5 Genetic Algorithms

Keep best N hypotheses at each step (selection) based on a fitness function and have pairwise crossover operations (and, optionally, mutation operations) to generate a new set of hypotheses.

6 Games

Algorithm 4: Minimax with alpha-beta pruning.

Input: A game state S

Output: The root minimax value.

```
/* initialize α ← −∞ and β ← ∞                                   */
Function VALUE(S, α, β):
    if S is a terminal state then
        return knwon temrinal value
    if the agent is maximizing then
        return MAX-VALUE(S, α, β)
    if the agent is minimizing then
        return MIN-VALUE (S, α, β)
Function MAX-VALUE(S, α, β):
    v ← −∞
    foreach successor S′ of S do
        v ← MAX(v, VALUE(S′))
        if v ≥ β then
            return v
        α ← MAX(α, v)
    return v
Function MIN-VALUE(S, α, β):
    v ← ∞
    foreach successor S′ of S do
        v ← MIN(v, VALUE(S′))
        if v ≤ α then
            return v
        β ← MIN(β, v)
    return v
```

Algorithm 5: Expectimax.

Input: A game state S
Output: The root minimax value.

Function `VALUE`(S)**:**
  if *S is a terminal state* **then**
    return *knwon temrinal value*
  if *the agent is maximizing* **then**
    return `MAX-VALUE`(S)
  if *the agent is randomizing* **then**
    return `EXP-VALUE`(S)

Function `MAX-VALUE`(S)**:**
  $v \leftarrow -\infty$
  foreach *successor S' of S* **do**
    $v \leftarrow$ `MAX`(v, `VALUE`(S'))
  return v

Function `EXP-VALUE`(S)**:**
  $v \leftarrow 0$
  foreach *successor S' of S* **do**
    $v \leftarrow \mathbb{E}\left[\texttt{VALUE}(S')\right]$ `// expected value`
  return v

Remark. Typically, performing Minimax and Expectimax to terminal states are too costly, which can be solved by terminating early and estimating the state utility with evaluation functions.

7 Markov Decision Processes

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

Definition 7.1 (Transition Function)**.** $T(s, a, s') = P(s' \mid s, a)$

Definition 7.2 (Value Iteration)**.** Initialize $V_0(s) \leftarrow 0$ for all $s \in S$. Compute $\forall s \in S$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')] \qquad \text{(value iteration)}$$

until $V_i(s)$ convereges for all $s \in S$ (yields V^*). Then, compute $\forall s \in S$

$$\pi^*(s) \leftarrow \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')] \qquad \text{(policy extraction)}$$

Definition 7.3 (Policy Iteration)**.** Define an initial policy π_0 (can be arbitrary, but ideality close to the optimal policy). Then, iteratively solve $\forall s \in S$

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

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

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

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

Remark. Policy iteration converges faster than value iteration.