

Jonathan Yu (jsyu)

Let m be the max depth of the search tree, b be the average branching factor, s be the depth of the shallowest solution, ϵ to be the lowest cost step, and C be the lowest costing solution. This also assumes DFS has path checking and that actions are not arbitrarily cheap in UCS.

Algorithm	Complete	Optimal	Time	Space
DFS	Y	N	$O(b^{m+1})$	$O(bm)$
BFS	Y	N^*	$O(b^{s+1})$	$O(b^s)$
ID	Y	N^*	$O(b^{s+1})$	$O(bs)$
UCS	Y^*	Y	$O(b^{C/\epsilon})$	$O(b^{C/\epsilon})$

Uniform-cost orders by path cost or backward cost $g(n)$. Best-first orders by goal proximity or forward cost $h(n)$. A^* search orders by the sum $f(n) = g(n) + h(n)$. A^* terminates when we dequeue a goal.

```

Pop state  $s$  with lowest  $f(s)$  in queue
Add  $s$  to expanded list
if  $s = GOAL$  then
    return Success
else
    for all  $s'$  in successors( $s$ ) do
         $f' = g(s') + h(s') = g(s) + cost(s, s') + h(s')$ 
        if  $s'$  not expanded and (not in PQ or ( $s' \in PQ$  and  $f(s') > f'$ )) then
            Promote/insert  $s'$  with new value  $f'$  in PQ
            previous( $s'$ ) =  $s$  (update parent node of  $s'$ )
        end if
    end for
end if

```

Uniform-cost expands in all directions, whereas A^* expands mainly towards the goal based on its heuristic. Limitations of A^* : In the worst case, we may need to explore all states, and we may have to keep all nodes in memory. A^* is optimally efficient (given h , no other optimal algorithm will expand fewer nodes).

Consistent heuristic: $h(s) \leq cost(s, s') + h(s')$. Consistent \implies optimal since all later nodes will be at least as expensive.

Admissible heuristic: $h(s) \leq cost(s)$. $h_a \geq h_c$ (h_a dominates h_c if $\forall n : h_a(n) \geq h_c(n)$).

Iterative deepening A^* : Complete, optimal, more expensive computation cost than A^* but memory order L as in DFS!

CSP: backtracking, ordering (minimum remaining values, least constrained value), arc consistency ($x \rightarrow y$ consistent iff for every x in the tail there is some y in the head could be assigned without violating constraints), forward checking (enforce arc consistency with each assignment)

```

Initialize queue to all arcs (binary constraints in csp)
while queue not empty do
    ( $X_i, X_j$ ) = poll from queue
    Revise(csp,  $X_i, X_j$ )
    if any changes to domain( $X_i$ ) then
        if domain( $X_i$ ) empty return inconsistent
        for all  $X_k \in$  Neighbors( $X_i$ ) except  $X_j$  do
            Add ( $X_k, X_i$ ) to queue
        end for
    end if
end while

```

With arc consistency, we will never have to backtrack while solving a CSP. Local search for CSP starts with invalid full assignment and make local adjustments to try to satisfy all constraints. Min conflicts sets v_i to the value that minimizes the number of constraints violated.

Search/Optimization: Hill-climbing (greedy local search), stochastic search (randomized hill climbing: with probability p randomly choose, with probability $1 - p$ greedily choose). Simulated annealing sets $p = e^{-(E-E')/T}$ simulates physics. Genetic Algorithm: reproduce using crossover, then mutate and pick best offspring. The representation is the most important part! Gradient descent: Pick a point, determine a descent direction, choose a step, update, repeat until stopping criterion is satisfied.

Reasoning over time: hidden Markov models and dynamic Bayes' nets. $P(X_{t+1}|e_{1:t}) = \sum_{x_t} P(X_{t+1}|x_t)P(x_t|e_{1:t})$, $B'(X_{t+1}) = \sum_{x_t} P(X'|x)B(x_t)$, $P(X_{t+1}|e_{1:t+1}) \propto P(e_{t+1}|X_{t+1})P(X_{t+1}|e_{1:t})$, $B(X_{t+1}) \propto P(e|X)B'(X_{t+1})$

Entropy(S) = $\sum_{i=1}^c -p_i \log_2 p_i$, Gain(S,A) = Entropy(S) - $\sum_{i=1}^{\text{values}(A)} \frac{|S_{v_i}|}{|S|} \text{Entropy}(S_{v_i})$

Linear Regression (least squares): $\text{argmin}_w \sum_i (y_i - wx_i)^2$, $w = \frac{\sum_i x_i y_i}{\sum_i x_i^2}$

Multivariate Regression: model is $y = Xw$, solution is $w = (X^T X)^{-1} X^T y$

k -fold Cross Validation: Randomly break the dataset into k partitions. For each partition, train on all the points not in that partition and find the test-set-sum of errors on the points in that partition, and then report the mean error.

Test-set	Downside	Upside
Leave-one-out	Unreliable due to variance	Cheap
10-fold	Experience	Doesn't waste data
3-fold	Wastes data, 10 times more expensive than test-set	Only wastes 10%
	Wastier	Slightly better than test-set

Use CV to choose model/algo/classification/regression/densities. CV can overfit: dataset with 50 records and 1000 attributes, choose linear regression model with best attributes, but it would have looked good even if the output had been purely random. To account for this, hold out an additional testset before doing any model selection or use randomization testing.

Halving Algorithm: Output the majority vote of all experts who have been always right until now, whenever there is a mistake we throw away at least half of the experts: At most $\log_2(n)$ mistakes

Weighted Majority Algorithm: Start with all experts having weight 1, predict based on weighted majority, penalize mistakes by cutting weight in half. Let there be n experts so that W (total weight) starts at n . After each mistake W drops by at least 25%, after M mistakes, W is at most $n(3/4)^M$. Weight of best expert is $(1/2)^M$ and $M \leq 2.4(m + \log_2 n)$

Multiplicative Weights Algorithm: Fix an $\eta \leq 1/2$, for each decision i , $w_i^{(t+1)} = w_i^{(t)}(1 - \eta m_i^{(t)})$

Limitations of Heuristic Search: Assumes actions are deterministic, but what about actions with multiple possible outcomes?

Value iteration using Bellman's Equation:

$$V^h(s_i) = \max_a \left(R(s_i) + \gamma \sum_{s_j \in S} p(s_j | s_i, a) V^{h-1}(s_j) \right)$$

Value iteration converges if discount factor < 1 because Bellman is a contraction and converges to a unique solution which is the optimal function

Policy iteration: Take a policy, estimate its value

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} p(s' | s, \pi(s)) V^\pi(s') \text{ and improve the policy } \pi'(s_i) = \text{argmax}_a \left(R(s_i) + \gamma \sum_{s_j \in S} p(s_j | s_i, a) V^\pi(s_j) \right)$$

Repeat previous 2 steps until policy converges

Policy Iteration: lots of actions / good initial policy, Value iteration: few actions, acyclic

Bandits: Bayesian formulation: Initial state: prior for each arm $(f(p_1), f(p_2), \dots, f(p_n))$. When we pull arm j , we receive reward r , and then use Bayes Rule to update the distribution: $f(p_j | r) = \frac{p(r | p_j) f(p_j)}{\int_{p_j} p(r | p_j) f(p_j)}$

Non-Bayesian formulation: Regret is $R^\pi(T) = \max_i T p_i - E[\sum_{t=1}^T p_{\pi(t)}]$

MLE: \hat{p}_j , confidence interval on p_j is $\left[\hat{p}_j - 1.96 \sqrt{\frac{\hat{p}_j(1-\hat{p}_j)}{\text{num pulls}}}, \hat{p}_j + 1.96 \sqrt{\frac{\hat{p}_j(1+\hat{p}_j)}{\text{num pulls}}} \right]$

Direct Estimation (Monte Carlo): $V(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V(s')$

Temporal Distance learning: Initialize $V(s)$ arbitrarily, and then for each trial: Initialize s

For each step in trial, observe reward $R(s)$, take action $a = \pi(s)$, observe next state s' ,

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t + \gamma V(s_{t+1}) - V(s_t)]$$

, $s \leftarrow s'$ The TD update can be viewed as an approximation that simply chooses an update of the observed successor s' rather than in the direct method that weights all possible successors according to their probability; for large numbers of transitions, the two will be equal since the successor states will occur in proportion to their transition probability. For $V(s)$ to converge to the correct value, α_t must satisfy $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

$$Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)) \text{ and } \pi(s) = \text{argmax}_a Q(s, a)$$