2 - Search Algorithms

知识点 & 题目

Basic state model S(P): Classical Planning

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function $s' = f(a, s)$ for $a \in A(s)$
- positive action costs $c(a, s)$

- A solution is a sequence of applicable actions that maps s0 into SG, and it is optimal if it **minimizes sum of action costs** (e.g., # of steps)
- Different models and controllers obtained by relaxing assumptions in blue

Blind search vs. heuristic (informed) search:

- Blind search algorithms: Only use the basic ingredients for general search algorithms.
  - e.g., Depth First Search (DFS), Breadth-first search (BrFS), Uniform Cost (Dijkstra), Iterative Deepening (ID)
- Heuristic search algorithms: Additionally use heuristic functions which estimate the distance (or remaining cost) to the goal.
  - e.g., A, *IDA*, Hill Climbing, Best First, WA, *DFS B&B, LRTA*, . . .
- For satisficing planning, heuristic search vastly outperforms blind algorithms pretty much everywhere.
- For optimal planning, heuristic search also is better (but the difference is less pronounced).
- Blind search does not require any input beyond the problem.
  - Pro: No additional work for the programmer.
  - Con: It's not called "blind" for nothing . . . same expansion order regardless what the problem actually is. Rarely effective in practice.
- Informed search requires as additional input a heuristic function h that maps states to estimates of their goal distance.
  - Pro: Typically more effective in practice.
  - Con: Implement h.
  - Note: In planning, h is generated automatically from the declarative problem description.

Systematic search vs. local search

- Systematic search algorithms: Consider a large number of search nodes simultaneously.
- Local search algorithms: Work with one (or a few) candidate solutions (search nodes) at a time.
- This is not a black-and-white distinction; there are crossbreeds (e.g., enforced hill-climbing).
- For satisficing planning, there are successful instances of each.
- For optimal planning, systematic algorithms are required.

Search Terminology

Search node n: Contains a *state* reached by the search, plus information about how it was reached.

Path cost $g(n)$: The cost of the path reaching n.

Optimal cost g^*: The cost of an optimal solution path. For a state s, $g^*(s)$ is the cost of a cheapest path reaching s.

Node expansion: Generating all successors of a node, by applying all actions applicable to the node's state s. Afterwards, the *state* s itself is also said to be expanded.

Search strategy: Method for deciding which node is expanded next.

Open list: Set of all *nodes* that currently are candidates for expansion. Also called frontier.

Closed list: Set of all *states* that were already expanded. Used only in graph search, not in tree search (up next). Also called explored set.

Search States vs. Search Nodes

- **Search states** s: States (vertices) of the search space.
- **Search nodes** σ: Search states, plus information on where/when/how they are encountered during search.

What is in a search node?

Different search algorithms store different information in a search node σ, but typical information includes:

- *state*(σ): Associated search state.
- *parent*(σ): Pointer to search node from which σ is reached.
- *action*(σ): An action leading from *state*(*parent*(σ)) to *state*(σ).
- $g(\sigma)$: Cost of σ (cost of path from the root node to σ).

For the root node, *parent*(σ) and *action*(σ) are undefined.

Criteria for evaluating search strategies

Guarantees:

Completeness: Is the strategy guaranteed to find a solution when there is one?

Optimality: Are the returned solutions guaranteed to be optimal?

Complexity:

Time Complexity: How long does it take to find a solution? (Measured in generated states.)

Space Complexity: How much memory does the search require? (Measured in states.)

Typical state space features governing complexity:

Branching factor b: How many successors does each state have?

Goal depth d: The number of actions required to reach the shallowest goal state.

Blind Search

- Breadth-First Search
- Depth-First Search
- Iterative Deepening Search

Heuristic Search: Systematic

- Greedy best-first search
- A*
- Weighted A*
- Iterative deepening A* (IDA*)
- Bidirectional A* Enhanced (BAE*)

Heuristic Search: Local

- Hill-climbing
- Enforced hill-climbing
- Beam search, tabu search, genetic algorithms, simulated annealing, . . .

Heuristic Function L2 P25 T3

- Heuristic function h estimates the cost of an optimal path to the goal.
  - Search gives a preference to explore states with small h.
- Remaining cost:
  - The perfect heuristic h*, assigns every state its remaining cost as the heuristic value.
- Search performance depends crucially on the **informedness of h** and on the **computational overhead** of computing h.
- Extreme cases:

- $h = h^*$: Perfectly informed; computing it = solving the planning task in the first place.
- $h = 0$: No information at all; can be “computed” in constant time.

- Successful heuristic search requires a good trade-off between h’s informedness and the computational overhead of computing it.
- Devise methods that yield good estimates at reasonable computational costs.
- **Definition (Safe/Goal-Aware/Admissible/Consistent).** *Let Π be a planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$, and let h be a heuristic for Π. The heuristic is called:*
  - *safe if $h^*(s) = \infty$ for all $s \in S$ with $h(s) = \infty$;*
  - *goal-aware if $h(s) = 0$ for all goal states $s \in S^G$;*
  - *admissible if $h(s) \le h^*(s)$ for all $s \in s$;*
  - *consistent if $h(s) \le h(s') + c(a)$ for all transitions $s \xrightarrow{a} s'$.*
- If h is consistent and goal-aware, then h is admissible.
- If h is admissible, then h is goal-aware.
- If h is admissible, then h is safe.
- No other implications of this form hold.

Breadth-First Search L2 P14 T1

- Strategy: Expand nodes in the order they were produced (FIFO frontier).
- Completeness: yes
- Optimality:
  - Yes, for uniform action costs. Breadth-first search always finds a shallowest goal state.
  - If costs are not uniform, this is not necessarily optimal.
- Complexity:

Time Complexity: Say that b is the maximal branching factor, and d is the goal depth (depth of shallowest goal state).

- Upper bound on the number of generated nodes? $b + b^2 + b^3 + \cdots + b^d$: In the worst case, the algorithm generates all nodes in the first d layers.
- So the time complexity is $O(b^d)$.
- And if we were to apply the goal test at node-expansion time, rather than node-generation time? $O(b^{d+1})$ because then we’d generate the first $d + 1$ layers in the worst case.

Space Complexity: Same as time complexity since all generated nodes are kept in memory.

- Which is the worse problem, time or memory?
  - Typically exhausts RAM memory within a few minutes.
- Breadth-first search is optimal but uses exponential space.

Depth-First Search L2 P17 T1

- Strategy: Expand the most recent nodes in (LIFO frontier).
- Completeness:
  - No, because search branches may be infinitely long: No check for cycles along a branch!
  - Depth-first search is complete in case the state space is acyclic, e.g., Constraint Satisfaction Problems. If we do add a cycle check, it becomes complete for finite state spaces.
- Optimality: No. After all, the algorithm just "chooses some direction and hopes for the best".
  - Depth-first search is a way of "hoping to get lucky".
- Complexity:
  - Space: Stores nodes and applicable actions on the path to the current node. So if m is the maximal depth reached, the complexity is **O(bm)**.
  - Time: If there are paths of length m in the state space, **O(b^m)** nodes can be generated. Even if there are solutions of depth 1!
    - If we happen to choose "the right direction" then we can find a length-l solution in time O(bl) regardless how big the state space is.
- Depth-first search uses linear space but is not optimal.

Iterative Deepening Search L2 P19 T1

- Completeness: yes
- Optimality: yes
- Complexity:
  - Space: O(bd)
  - Time:

Breadth-First-Search	$b + b^2 + \cdots + b^{d-1} + b^d \in O(b^d)$
Iterative Deepening Search	$(d)b + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + 1b^d \in O(b^d)$

- IDS combines the advantages of breadth-first and depth-first search.
- It is the preferred blind search method in large state spaces with unknown solution depth.

Greedy best-first search T2

Greedy Best-First Search (with duplicate detection)

open $:=$ **new** priority queue ordered by ascending $h(\textit{state}(\sigma))$
open.insert(make-root-node(init()))
closed $:= \emptyset$
while not *open*.empty():
    $\sigma :=$ *open*.pop-min() /* get best state */
    if $\textit{state}(\sigma) \notin \textit{closed}$: /* check duplicates */
        closed $:= \textit{closed} \cup \{\textit{state}(\sigma)\}$ /* close state */
        if is-goal(state(σ)): **return** extract-solution(σ)
        for each $(a, s') \in$ succ(*state*(σ)): /* expand state */
            $\sigma' :=$ make-node(σ, a, s')
            if $h(\textit{state}(\sigma')) < \infty$: *open*.insert($\sigma'$)
return unsolvable

- Completeness: Yes, for safe heuristics. (and duplicate detection to avoid cycles)
- Optimality: No
- Invariant under all strictly monotonic transformations of h
    - e.g., scaling with a positive constant or adding a constant.
- Priority queue: e.g., a min heap.
- “Check Duplicates”: Could already do in “expand state”; done here after “get best state” only to more clearly point out relation to A.

A* L2 P33 T2

A* (with duplicate detection and re-opening)

open $:=$ **new** priority queue ordered by ascending $g(\textit{state}(\sigma)) + h(\textit{state}(\sigma))$
open.insert(make-root-node(init()))
closed $:= \emptyset$
best-g $:= \emptyset$/* maps states to numbers */
while not *open*.empty():
    $\sigma :=$ *open*.pop-min()
    if $\textit{state}(\sigma) \notin \textit{closed}$ **or** $g(\sigma) <$ *best-g*(*state*(σ)):
        /* re-open if better g; note that all σ' with same state but worse g
          are *behind* σ in *open*, and will be skipped when their turn comes */
        closed $:= \textit{closed} \cup \{\textit{state}(\sigma)\}$
        best-g(*state*(σ)) $:= g(\sigma)$
        if is-goal(state(σ)): **return** extract-solution(σ)
        for each $(a, s') \in$ succ(*state*(σ)):
            $\sigma' :=$ make-node(σ, a, s')
            if $h(\textit{state}(\sigma')) < \infty$: *open*.insert($\sigma'$)
return unsolvable

- f -value of a state: defined by f (s) := g(s) + h(s).
- Generated nodes: Nodes inserted into open at some point.
- Expanded nodes: Nodes popped from open for which the test against closed and distance succeeds.

- Re-expanded nodes: Expanded nodes for which state in closed upon expansion (also called re-opened nodes).
- Completeness: Yes, for safe heuristics. (Even without duplicate detection.)
- Optimality: Yes, for admissible heuristics. (Even without duplicate detection.)
- Popular method: break ties (f (s) = f (s')) by smaller h-value.
- If h is admissible and consistent, then A never re-opens a state. So if we know that this is the case, then we can simplify the algorithm.
- Common, hard to spot bug: check duplicates at the wrong point.

Weighted A* T2

Weighted A* (with duplicate detection and re-opening)

```
open := new priority queue ordered by ascending g(state(σ)) + W * h(state(σ))
open.insert(make-root-node(init()))
closed := ∅
best-g := ∅
while not open.empty():
        σ := open.pop-min()
        if state(σ) ∉ closed or g(σ) < best-g(state(σ)):
           closed := closed ∪ {state(σ)}
           best-g(state(σ)) := g(σ)
           if is-goal(state(σ)): return extract-solution(σ)
           for each(a, s′) ∈ succ(state(σ)):
                σ′ := make-node(σ, a, s′)
                if h(state(σ′)) < ∞: open.insert(σ′)
return unsolvable
```

The weight $W \in \mathbb{R}_0^+$ is an **algorithm parameter:**

- For $W = 0$, weighted A^* behaves like uniform-cost search.
- For $W = 1$, weighted A^* behaves like A^*.
- For $W \to \infty$, weighted A^* behaves like greedy best-first search.

Properties:

- For $W > 1$, weighted A^* is bounded suboptimal: if h is admissible, then the solutions returned are at most a factor W more costly than the optimal ones.

Hill-climbing

Hill-Climbing

```
σ := make-root-node(init())
forever:
   if is-goal(state(σ)):
     return extract-solution(σ)
   Σ' := { make-node(σ, a, s') | (a, s') ∈ succ(state(σ)) }
   σ := an element of Σ' minimizing h /* (random tie breaking) */
```

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this complete or optimal? No.
- Can easily get stuck in local minima where immediate improvements of $h(\sigma)$ are not possible.
- Many variations: tie-breaking strategies, restarts, ...

Enforced hill-climbing A1

Enforced Hill-Climbing: Procedure *improve*

```
def improve(σ0):
    queue := new fifo queue
    queue.push-back(σ0)
    closed := ∅
    while not queue.empty():
        σ = queue.pop-front()
        if state(σ) ∉ closed:
           closed := closed ∪ {state(σ)}
           if h(state(σ)) < h(state(σ0)): return σ
           for each (a, s') ∈ succ(state(σ)):
               σ' := make-node(σ, a, s')
               queue.push-back(σ')
    fail
```

$\rightsquigarrow$ Breadth-first search for state with strictly smaller h-value.

Enforced Hill-Climbing

```
σ := make-root-node(init())
while not is-goal(state(σ)):
        σ := improve(σ)
return extract-solution(σ)
```

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this optimal? No.
- Is this complete? In general, no. Under particular circumstances, yes. Assume that h is goal-aware.
  $\rightarrow$ Procedure *improve* fails: no state with strictly smaller h-value reachable from s, thus (with assumption) goal not reachable from s.
  $\rightarrow$ This can, for example, not happen if the state space is undirected, i.e., if for all transitions $s \rightarrow s'$ in Θ_Π there is a transition $s' \rightarrow s$.

Properties of search algorithms

	DFS	BrFS	ID	A*	HC	IDA*
Complete	No	Yes	Yes	Yes	No	Yes
Optimal	No	Yes*	Yes	Yes	No	Yes
Time	∞	b^d	b^d	b^d	∞	b^d
Space	$b \cdot d$	b^d	$b \cdot d$	b^d	b	$b \cdot d$

- Parameters: d is solution depth; b is branching factor
- Breadth First Search (BrFS) optimal when costs are uniform
- A*/IDA* optimal when h is **admissible**; $h \leq h^*$

Summary

Distinguish: World states, search states, search nodes.

- World state: Situation in the world modelled by the planning task.
- Search state: Subproblem remaining to be solved.
  - In progression, world states and search states are identical.
  - In regression, search states are sub-goals describing sets of world states.
- Search node: Search state + info on "how we got there".

Search algorithms mainly differ in order of node expansion:

- Blind vs. heuristic (or informed) search.
- Systematic vs. local search.

题目

Quiz

Question!

If we set $h(n) := 0$ for all n, what does A^* become?

(A): Breadth-first search. (B): Depth-first search.
(C): Uniform-cost search. (D): Depth-limited search.

→ (C): Same expansion order. (Details in book-keeping of open/closed states may differ.)

Question!

If we set $h(n) := 0$ for all n, what can greedy best-first search become?

(A): Breadth-first search. (B): Depth-first search.
(C): Uniform-cost search. (D): A), B) and C)

→ h implies no ordering of nodes at all, so this fully depends on how we break ties in the open list. (A): FIFO, (B): LIFO, (C): Order on g. (Details in book-keeping of open/closed states may differ.)

Question!

Is informed search always better than blind search?

(A): Yes. (B): No.

→ In greedy best-first search, the heuristic may yield larger search spaces than uniform-cost search. E.g., in path planning, say you want to go from Melbourne to Sydney, but $h(\text{Perth}) < h(\text{Canberra})$.

→ In A^* with an admissible heuristic and duplicate checking, we cannot do worse than uniform-cost search: $h(s) > 0$ can only reduce the number of states we must consider to prove optimality.

→ Also, in the above example, A^* doesn't expand Perth with *any* admissible heuristic, because $g(\text{Perth}) > g(\text{Sydney})$!

→ "Trusting the heuristic" has its dangers! Sometimes g helps to reduce search.

I would encourage you to draw a small graph containing 3 nodes: Melbourne, Sydney and Perth. Start the search in Melbourne, and set Sydney as your goal. Draw the costs of travelling from any pair of cities in hours: Melb-Pert = 30h, Perth-Syd=45h, and Melb-Syd=10h. Test the statements above to check your intuitions.

Question 4

0 / 1 pts

Consider general search problems, which of the following are true?

Correct!

- [x] Code that implements A* tree search can be used to run Uniform-cost search.

Yes, you just need to set f(n)=g(n).

- [ ] A* tree search is optimal with any heuristic function.

You Answered

- [x] A* tree search is complete with any heuristic function.

Only if the heuristic is safe!

If the heuristic is not safe, then it can assign $h(n) := \infty$ to all nodes *n* that lead to a solution

- [ ] A* graph search is guaranteed to expand no more nodes than DFS.

Correct!

- [x] The max of two admissible heuristics is always admissible.

Correct! if h1 $\leq$ h* and h2 $\leq$ h*, then it follows that max(h1,h2) $\leq$ h*

You Answered

- [x] A heuristic that always evaluates to h(s) = 1 for non-goal search nodes s is always admissible.

A simple example where this is not true: If all action costs are 0, then this heuristic is not admissible. Check the video about the properties and relationships that can be proved

Definition (Safe/Goal-Aware/Admissible/Consistent). *Let Π be a planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$, and let h be a heuristic for Π. The heuristic is called:*

- *safe if $h^*(s) = \infty$ for all $s \in S$ with $h(s) = \infty$;*
- *goal-aware if $h(s) = 0$ for all goal states $s \in S^G$;*
- *admissible if $h(s) \leq h^*(s)$ for all $s \in s$;*
- *consistent if $h(s) \leq h(s') + c(a)$ for all transitions $s \xrightarrow{a} s'$.*

→ **Relationships?**

Proposition. *Let Π be a planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$, and let h be a heuristic for Π. If h is consistent and goal-aware, then h is admissible. If h is admissible, then h is goal-aware. If h is admissible, then h is safe. No other implications of this form hold.*

Question 5

1 / 1 pts

Heuristic functions estimate the distance from a state to:

- the closest neighbour
- the initial state
- Correct! the closest goal state

  Correct! heuristic functions estimate the cost of solving the problem from the current state, which is the cost to the nearest goal

- the furthest goal state

Question 6

1 / 1 pts

What is a SAFE heuristic function?

- If a solution does not exist from state s, then h(s) = infinity
- Correct! If a solution exists from state s, then h(s) < infinity

  Correct! Safe heuristics ensure that no state will have a heuristic value of infinity if a solution can be found from that state. That way states with a heuristic value of infinity can safely be ignored when searching for a solution.

- If a solution does not exist from state s, the h(s) > infinity
- If a solution exists from state s, then h(s) = infinity

Question 7

1 / 1 pts

A goal aware heuristic assigns h=0 to all goal states

- Correct! True

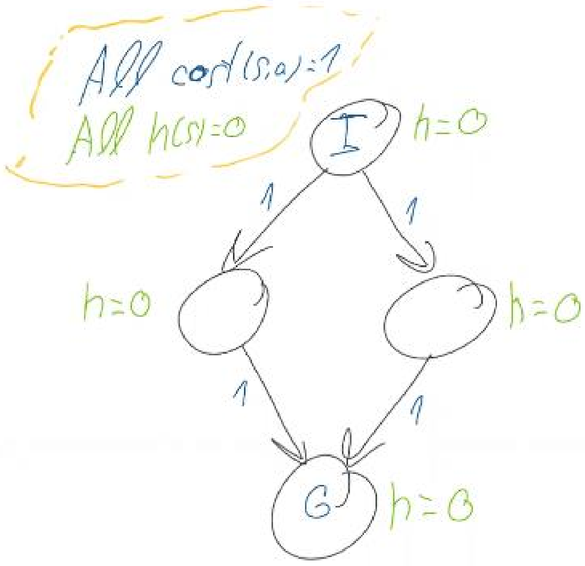
  Correct!

- False

Question 8

1 / 1 pts

Given the graph below, is the heuristic:



- Consistent, Goal Aware and Admissible
- Goal Aware and Admissible
- Goal Aware, Admissible and Safe

Correct!

- Consistent, Goal Aware, Admissible and Safe

Consistent: Yes because h(s) - h(s') = 0 for all transitions (as h(s)=0 for all states). c(a)=1 for all actions. Therefore h(s)-h(s') = 0 < c(a) = 1
Goal Aware: Yes because h=0 for state G
Admissible: Yes as h(s)=0 <= h*(s) for all states
Safe: Yes as no states with h*(s) = infinity

Question 9

1 / 1 pts

Heuristic search performance depends on:

Correct!

- the informedness of the heuristic and computation overhead

Correct, heuristic performance is always a balance between how well it directs the search (informedness) and how long it takes to compute (computation overhead)

- Only on how close our heuristic is to h*
- the day of the month

- only on how close our heuristic is to h=0

Question 1

1 / 1 pts

WA* uses f(n) = g(n) + w*h(n). Then, with w=0 it becomes:

- A*
- Correct! Uniform-cost search (Dijkstra)
- Breadth-first search
- Greedy Best First Search

Question 2

1 / 1 pts

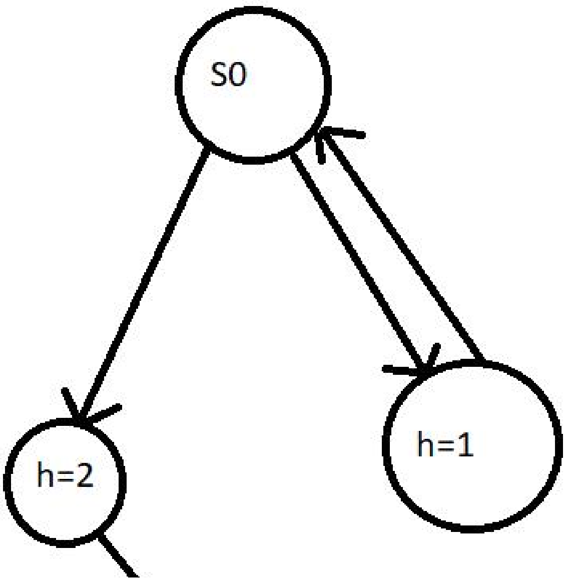
WA* with w = infinity becomes

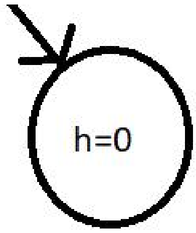
- A*
- Uniform-cost search (Dijkstra)
- Breadth-first search
- Correct! Greedy Best First Search

Question 3

0 / 1 pts

For the following graph, assuming the h=0 node is the goal, Enforced Hill Climbing is guaranteed to find a solution





Correct Answer

Yes

You Answered

No

We can guarantee the search will never get stuck in a state that doesn't lead to the goal, as a path can be found from every state to the goal.