CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

INFORMED SEARCH

nicholas.teh@u.nus.edu

CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

CONTENT SUMMARY

nicholas.teh@u.nus.edu

NOTATIONS (ALSO TUTORIAL 3 QUESTION 1)

- **g(n):** actual/min path cost from initial state *s* to state *n*
- **h(n):** heuristic, the approximated path cost from state *n* to the goal state *g*   [Properties: admissibility, consistency]
- **f(n):** evaluation function used by the algorithm to *search smartly*
  - Greedy best-first search: $f(n) = h(n)$
  - A* search: $f(n) = g(n) + h(n)$

KEY CONCEPTS

- **Heuristic**
  - Admissibility:
    - Be able to come up with an admissible heuristic given a problem, and prove it is so.
    - Be able to reason the dominance relationship between heuristics
  - Consistency:
    - Be able to prove a heuristic is consistent

KEY CONCEPTS

- **Informed Search Algorithms**
  - Greedy Best-First Search
  - A* Search
  - Tracing, properties, analysis using <u>completeness</u> and <u>optimality</u>.

HEURISTIC

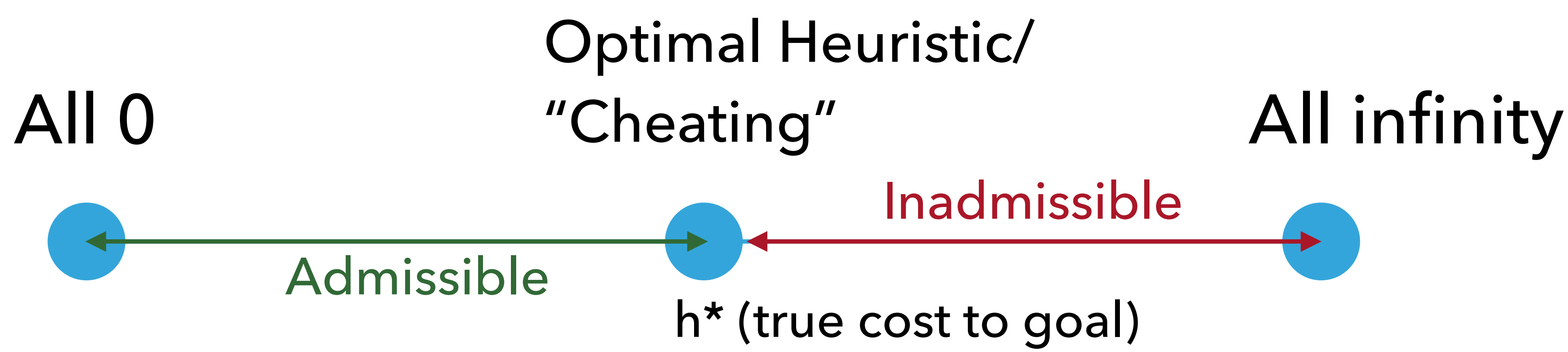
- Informed search makes use of heuristics to make search faster by exploiting problem-specific knowledge. Order of node expansion still matters: which one more promising?
- [Definition] Heuristic: **guess of how far I am from the goal** and heuristic at every goal node should be 0.
  - *Trivial heuristics*: 0 for all nodes, infinity everywhere with 0 at the goal node
  - Actual distance/*Optimal heuristic* (seen problem before) is also a heuristic, but is unrealistic

These two heuristics are prohibited in the quizzes/exams.

ADMISSIBILITY

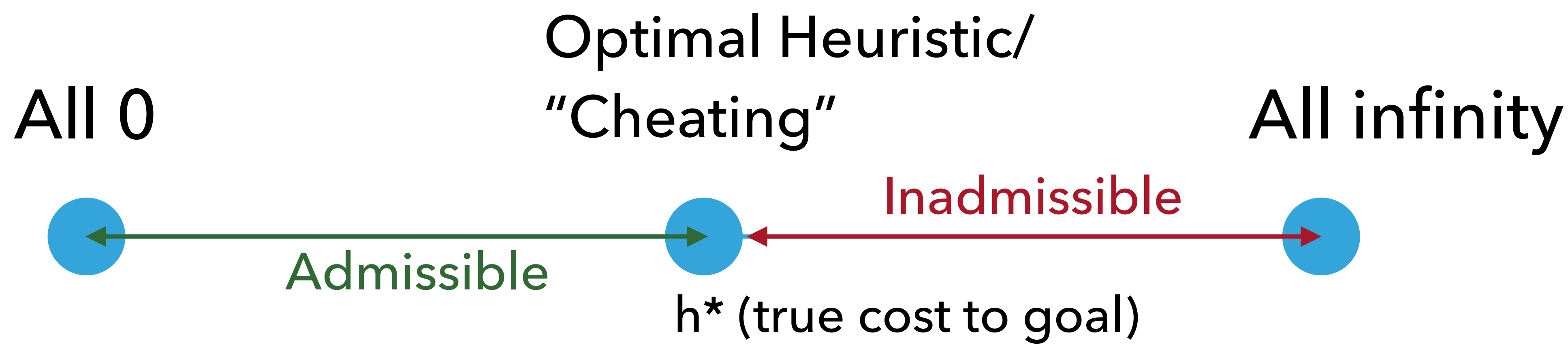
- h(n) is admissible if **for all *n***, $h(n) \leq h^*(n)$
  - $h^*(n)$ is the true/optimal cost to reach goal state from *n*
  - Never overestimates the cost to reach goal state
  - Inadmissible means there exists at least one node (it can be just one node) that violates the above.

ADMISSIBILITY



- The max/min of 2 admissible heuristic is __(1)__
- The max of 2 inadmissible heuristic is __(2)__
- The min of 2 inadmissible heuristic may be either… depends **Why?**
- The max of 1 admissible and 1 inadmissible heuristic is __(3)__
- The min of 1 admissible and 1 inadmissible heuristic is __(4)__

ADMISSIBILITY



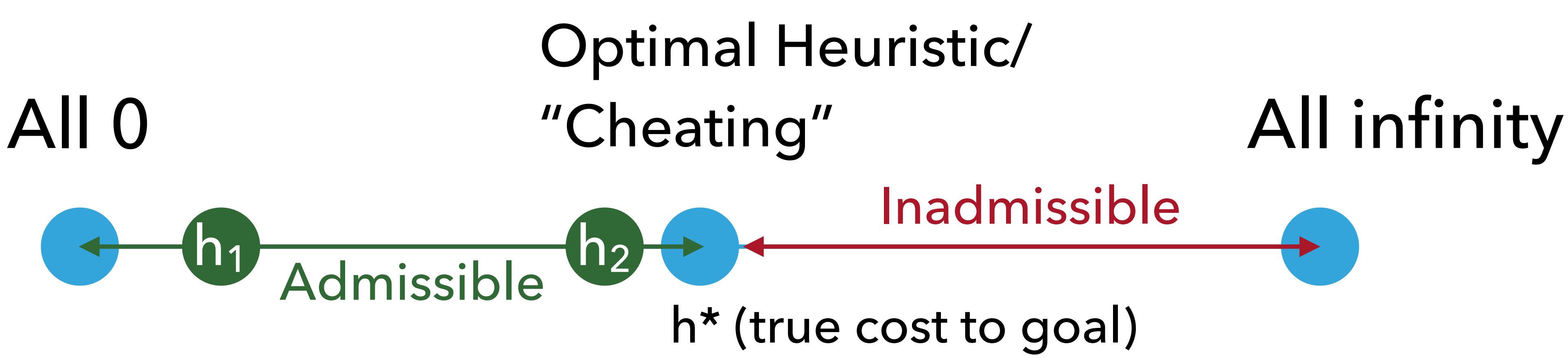
- The max/min of 2 admissible heuristic is <u>admissible</u>
- The max of 2 inadmissible heuristic is <u>inadmissible</u>
- The min of 2 inadmissible heuristic may be either… depends **Why?**
- The max of 1 admissible and 1 inadmissible heuristic is <u>inadmissible</u>
- The min of 1 admissible and 1 inadmissible heuristic is <u>admissible</u>

ADMISSIBILITY, MORE FORMALLY…

- Given 2 **admissible** heuristics h_i and h_j:
  - MAX(h_i, h_j) is also admissible. MAX(h_i, h_j) dominates both h_i and h_j.
  - MIN(h_i, h_j) is also admissible.
- Given 2 **inadmissible** heuristics h_i and h_j:
  - MAX(h_i, h_j) is inadmissible.
  - MIN(h_i, h_j) may be admissible or inadmissible.
- Given 1 **admissible** heuristic h_i and 1 **inadmissible** h_j:
  - MAX(h_i, h_j) is inadmissible, MIN(h_i, h_j) is admissible

DOMINANCE

- Usually defined (rather, more meaningful) for 2 admissible heuristics. But the same definition applies even if inadmissible heuristics are involved.
- If $h_2(n) \geq h_1(n)$ for all n, then h_2 dominates h_1. h_2 incurs lower search cost (underestimate less) than h_1, if h_1 and h_2 are admissible.
- Recall the line: if both are admissible, then h_2 is closer to the optimal heuristic than h_1.



DOMINANCE

- **In A* search, a dominant admissible heuristic leads to lower search costs.**
- *The more you underestimate, the more uncertainty there is, the more you have to "search" to actually get to the goal.*
- A* expands nodes that have a lower

  f(n) = g(n) + h(n) value first
- The higher h(n) is, the less nodes A* expands, making it faster, and so lower search cost.

CREATING ADMISSIBLE HEURISTICS (PROBLEM RELAXATION)

- A problem with fewer restrictions on actions is called a *relaxed problem* (easier to calculate). Think of it as bending the rules (e.g. instead of walking one step at a time, you can fly!).
- The more you bend, the more it underestimates the cost, because if you follow the rules, you have more restrictions, cost should be higher.

CREATING ADMISSIBLE HEURISTICS (PROBLEM RELAXATION)

- **An example: 8-puzzle (instantiation of k-puzzle!)**
- **Rules**: A tile can move from square A to square B if
  - (1) A is horizontally or vertically adjacent to B, and
  - (2) B is blank
- From this, we can generate three relaxed problems: Bend/ignore rules!
  - a tile can move from A to B if A is adjacent to B (Manhattan distance) (ignore rule (2))
  - a tile can move from A to B if B is blank (ignore rule (1))
  - a tile can move from A to B (# misplaced tiles) (ignore rule (1) & (2))

CREATING ADMISSIBLE HEURISTICS (PROBLEM RELAXATION)

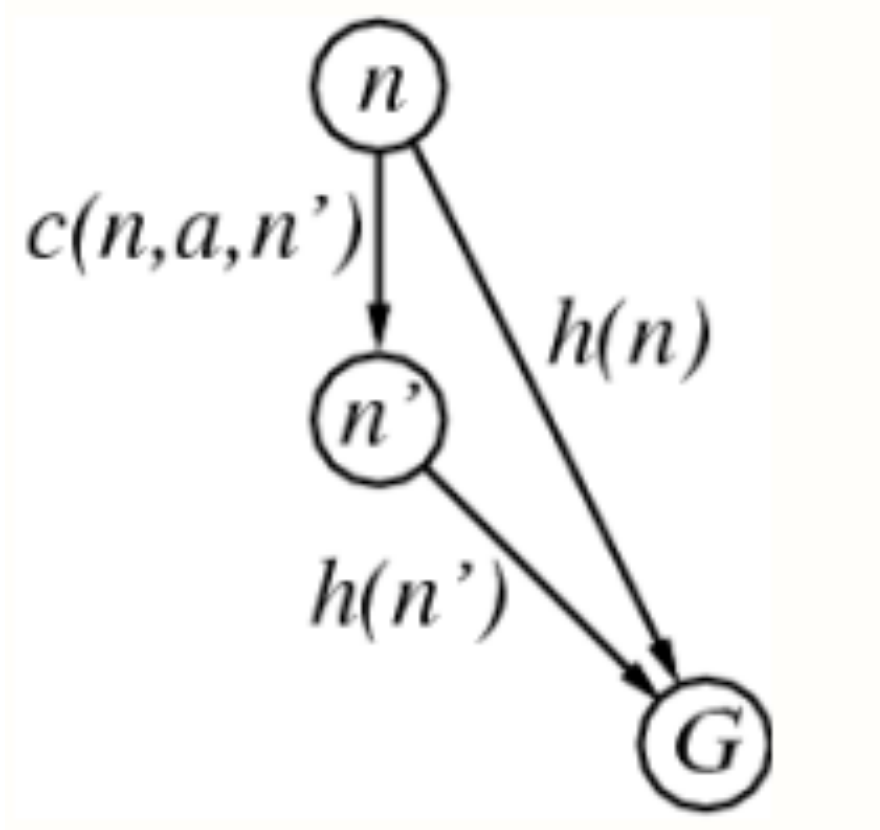
Properties:

- Any optimal solution in the original problem is also a solution in the relaxed problem.
- The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.
- Relax less is better (admissible with higher cost means closer to optimal!)

CONSISTENCY

- h(n) is consistent if for every node *n*, and every successor *n′* of *n* generated by action *a*,

  $h(n) \leq d(n, n') + h(n')$



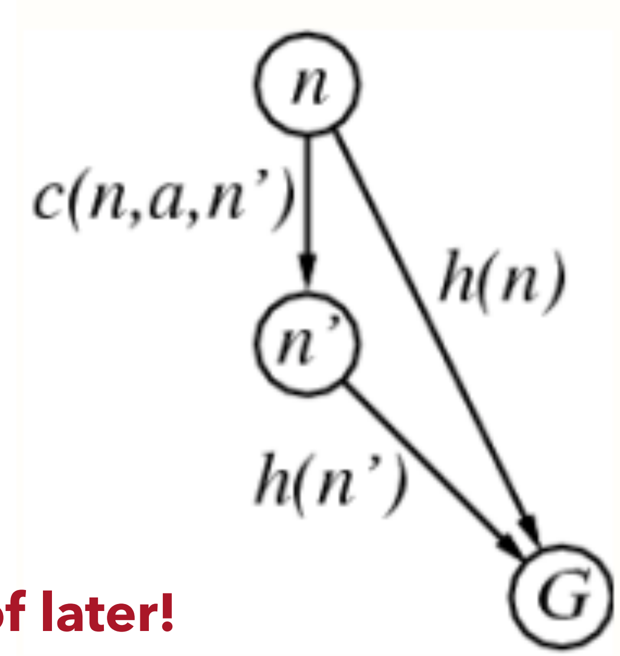
Basically the triangle inequality

THE CONNECTION BETWEEN ADMISSIBILITY AND CONSISTENCY?

- h(n) is **admissible** if for all *n*, h(n) ≤ h*(n)
  - h*(n) is the true/optimal cost to reach goal state from *n*
  - Never overestimates the cost to reach goal state
  - Inadmissible means there exists at least one node (it can be just one node) that violates the above.

- h(n) is **consistent** if for every node *n*, and every successor *n′* of *n* generated by action *a*,

h(n) ≤ d(n, n′) + h(n′)



We'll see the proof later!

INFORMED SEARCH ALGORITHMS

- **Best-First Search**
  - Use an evaluation function f(n) for each node n, where f is left open to define.
  - Cost estimate: expand node with lowest f first.
  - Note special cases (different choices of f: greedy, A*, etc.)
- **Greedy Best-First Search**
  - Evaluation function $f(n) = h(n)$ (heuristic function)
    = estimate cost of cheapest path from n to goal
  - At each stage, expands node that <u>appears</u> to be closest to goal.
  - Note special cases (difference choices of h may yield similar algorithms to what we know)

GREEDY BEST-FIRST SEARCH

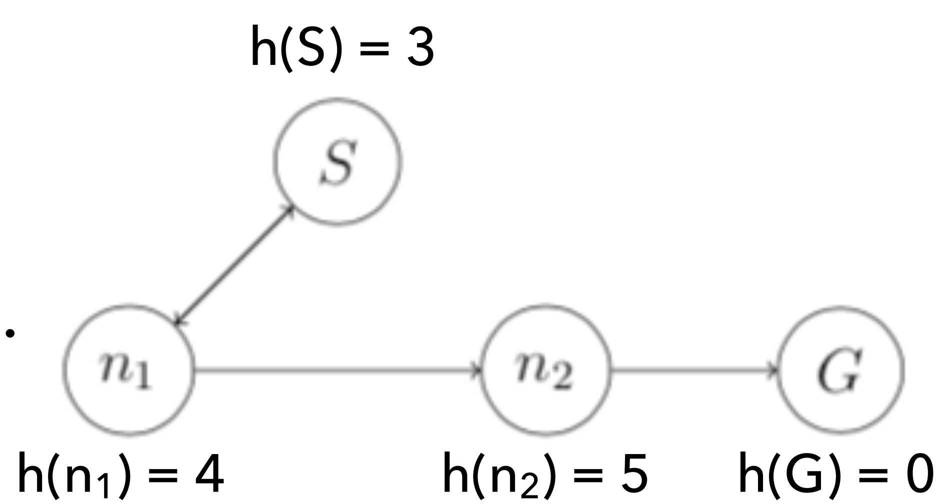
- **Is there a problem with solely relying on heuristics?**

- Let's take a look… we will analyse in terms of **completeness** and **optimality**.

GREEDY BEST-FIRST SEARCH

f(n) = h(n)

- **Tree-based Greedy Best-First Search**
  - Recall tree-based: I can repeat nodes.
- Tree-based GBFS is not complete.
  - Recall complete: can always reach goal (if exists).
- Stuck in an infinite loop because of short-sightedness.



- Each time S is explored, we add n_1 to the front of frontier (it's the only option)
- Each time n_1 is explored, we add S to the front of frontier (h(S) = 3 < 5 = h(n_2)).
- n_2 is never at the front of the frontier. This causes the greedy best-first search algorithm to continuously loop over S and n_1. So it's **not complete**.

GREEDY BEST-FIRST SEARCH

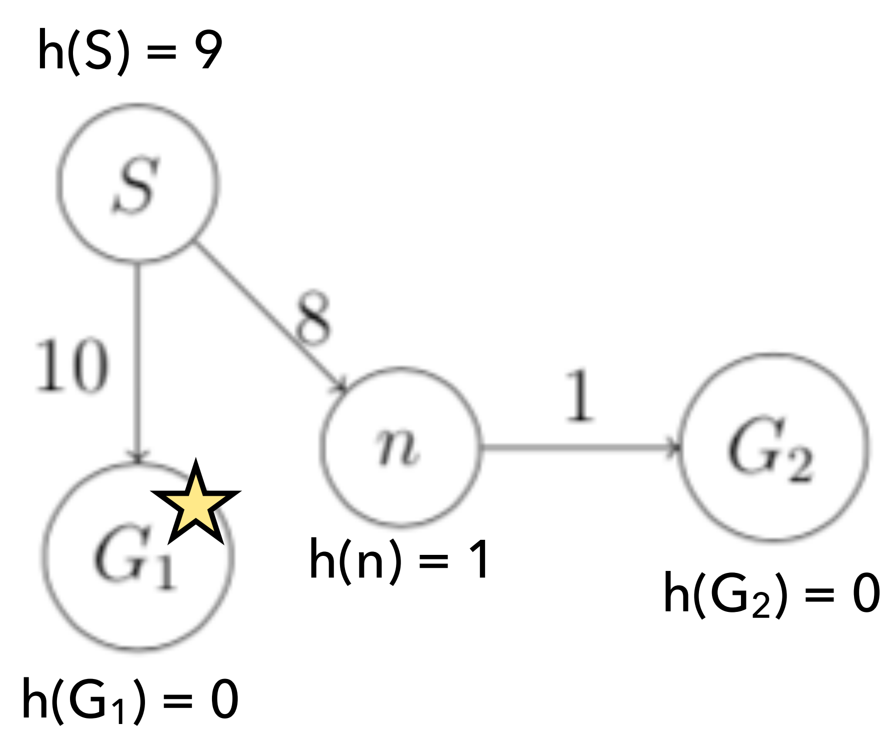
f(n) = h(n)

- **Graph-based Greedy Best-First Search**
  - Recall graph-based: I don't repeat nodes.
- Graph-based GBFS is complete.
- Assuming a finite branching factor, b, the graph-based variant of the greedy best-first search algorithm will **eventually visit all states within the search space** and thus find a goal state
- *(We always assume finite number of states in state space/nodes in search graph - not the same as finite depth* in a search tree)

GREEDY BEST-FIRST SEARCH

f(n) = h(n)

- **<u>Both Greedy Best-First Search</u>**
- Both Tree-based and Graph-based GBFS is <u>not optimal</u>.
- With either variant of the greedy best-first search algorithm, when S is explored, G_1 would be added to the front of the Frontier and then explored next, resulting in the algorithm returning the non-optimal $S \rightarrow G_1$ path.



I basically set an expensive "trap suboptimal goal state" which GBFS immediately falls for.

INFORMED SEARCH ALGORITHMS

- **A* Search**
  - Use an evaluation function $f(n) = g(n) + h(n)$ for each node n.
  - Cost estimate: expand node with lowest *f* first.
  - ***Question: When is A* equivalent to UCS*** *(uninformed search!)****?***

A* SEARCH

- **Tree-based A* Search**
- If I use an **admissible heuristic**, guaranteed to be optimal.
  - h(n) is admissible if for all n, $\mathbf{h(n) \leq h^*(n)}$
- **Graph-based A* Search**
- If I use a **consistent heuristic**, guaranteed to be optimal.
  - h(n) is consistent if for every node n, and every successor n′ of n using action a, $\mathbf{h(n) \leq d(n, n') + h(n')}$

CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

INFORMED SEARCH ALGORITHMS

nicholas.teh@u.nus.edu

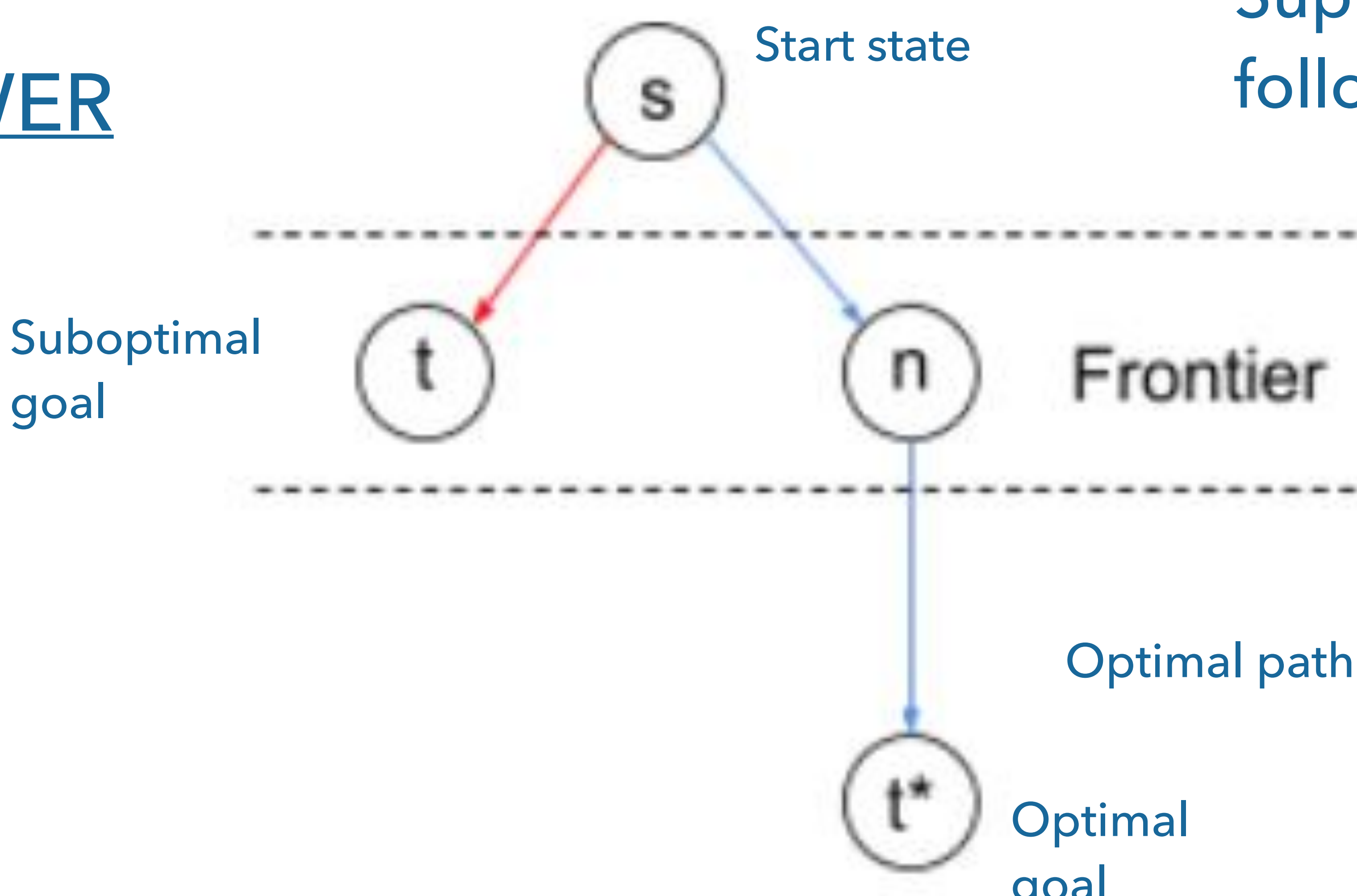
TUTORIAL 3 QUESTION 1 (TREE-BASED A* SEARCH)

- **Prove that the tree-based variant of the A* search algorithm is optimal when an admissible heuristic is utilised.**

Suppose we have the following search tree.

ANSWER

-



An optimal solution implies that **any intermediate node on the optimal path** *n* must be expanded before suboptimal goal *t*.

TUTORIAL 3 QUESTION 1 (TREE-BASED A* SEARCH)

ANSWER (Continued)

- We're going to prove it by contradiction.
- Assume, for a contradiction, that suboptimal goal *t* is expanded before any optimal path intermediate node *n*
- *Then* $f(t) \leq f(n)$, since A* uses f to determine expansion
- However, since *t* is not on the optimal path, and *t** is optimal, we have $f(t) > f(t^*) = g(t^*) + h(t^*)$.
- Since t* is a goal node, $h(t^*) = 0$, so we get $f(t) > g(t^*)$.
- $f(t) > g(t^*) = g(n) + d(n, t^*)$ where $d(n, t^*)$ is actual cost from n to t*

TUTORIAL 3 QUESTION 1 (TREE-BASED A* SEARCH)

ANSWER (Continued)

- $f(t) > g(t^*) = g(n) + d(n, t^*) = g(n) + h^*(n)$
  where $d(n, t^*)$ is actual cost from n to t*
- $f(t) > g(n) + \underline{h^*(n)} \geq g(n) + \underline{h(n)}$ because $h(n)$ is admissible (question says an admissible heuristic is used)
- $\mathbf{f(t) >} g(n) + h(n) = \mathbf{f(n)}$
- Which contradicts $f(t) \leq f(n)$.
- *Note: we do not consider f(t) = f(n) since that will mean f(t) is equally optimal - we defined optimal goal t* and suboptimal goal t*

TUTORIAL 3 QUESTION 2 (GRAPH-BASED A* SEARCH)

- **Prove that the graph-based variant of the A* search algorithm is optimal when a consistent heuristic is utilised.**

TUTORIAL 3 QUESTION 2 (GRAPH-BASED A* SEARCH)

- **Prove that the graph-based variant of the A* search algorithm is optimal when a consistent heuristic is utilised.**

ANSWER Let n′ be a successor node of n by taking some action *a*.

- A heuristic h(n) is consistent if for all *n*, $h(n) \leq d(n, n') + h(n')$
- LEMMA: $f(n') = \underline{g(n')} + h(n') = \underline{g(n) + d(n, n')} + h(n')$

$$\geq g(n) + h(n) \qquad \text{by consistency}$$
$$= f(n)$$

- So we get $f(n') \geq f(n)$. The evaluation function at a later node is always ≥ evaluation function at earlier node. Let's prove by contradiction.

TUTORIAL 3 QUESTION 2

ANSWER (Continued)

- What that also means is that A* search explores nodes in a non-decreasing order of *f* value;
  - Essentially, with each exploration, we may add a new contour (similar to how UCS explores nodes in a non-decreasing order of g value)
  - When A* expands n, the optimal path to *n* has been found (again, similar to UCS)

TUTORIAL 3 QUESTION 2

ANSWER (Continued)

- Proof by contradiction:
  - Assume *n* is explored, but the path to *n* is NOT optimal.
  - This means there exists some node on the optimal path to *n* that was NOT explored, but IS on the frontier. Let this node be *m*. (this has to be true, because, well, it has to be considered)
  - But since A* explores nodes in a non-decreasing order of *f* value, *m* have to be explored before *n*, since we're on the non-optimal path to *n*.

TUTORIAL 3 QUESTION 3 (TRACING)

- Trace yourself and verify with tutorial solutions. You should be familiar with tracing all known algorithms.
- Read the proof of admissibility (directly makes use of the definition)

DIAGNOSTIC QUIZ (PROVE ADMISSIBLE)

- $h_1(n)$ = number of misplaced tiles
  $h_2(n)$ = manhattan distance
- **For 2 admissible heuristics h_1 and h_2, and where h_2 dominates h_1, we define:**

  $$h_3 = (h_1 + h_2)/2 \qquad\qquad h_4 = h_1 + h_2$$

  Are h_3 and h_4 admissible? If they are, compare their dominance with respect to h_1 and h_2.

DIAGNOSTIC QUIZ (PROVE ADMISSIBLE)

ANSWER

- If I can show the heuristic is dominated by an admissible heuristic, then I can prove it's admissible.
- Since h_2 dominates h_1, $h_1(s) \leq h_2(s)$ for all *n*,

$$h_3(n) = \frac{h_1(n) + h_2(n)}{2} \leq \frac{h_2(n) + h_2(n)}{2} = h_2(n) \leq h^*(n)$$

By the definition of h_3

Since h_2 dominates h_1

Simple arithmetic

Because h_2 is admissible

So h_3 is admissible

DIAGNOSTIC QUIZ (PROVE ADMISSIBLE)

ANSWER (Continued)

- h_4 is not admissible (in general), and in this specific scenario if we were to talk about 8-puzzle.
  - h_1 is Number of misplaced tiles
  - h_2 is Manhattan distance

Pick h_1 and h_2 such that both are "at the border" of the admissible region. Then taking the sum will push them into the inadmissible range.

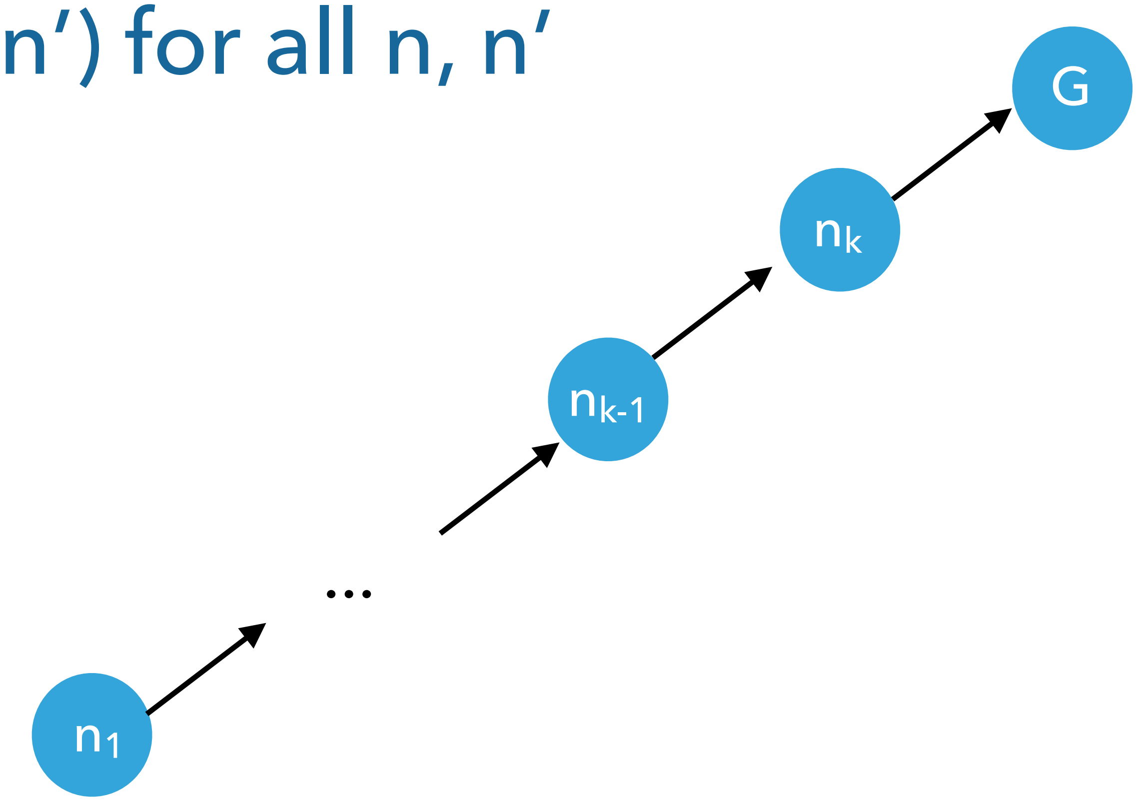
Consider a board/state *n* where moving one tile will reach the goal. Then both heuristics will give 1, and h_4(s) will give 2, not admissible.

TUTORIAL 3 QUESTION 4 (CONSISTENT → ADMISSIBLE)

- **If a heuristic is consistent, it is also admissible. Prove it.**

- Consistency: $h(n) \leq d(n, n') + h(n')$ for all n, n' (n' is successor of n)

- Also recall that $\mathbf{d(n, G) = h^*(n)}$



TUTORIAL 3 QUESTION 4 (CONSISTENT → ADMISSIBLE)

- **If a heuristic is consistent, it is also admissible. Prove it.**

ANSWER

- Consistency: $h(n) \leq d(n, n') + h(n')$ for all n, n' (n' is successor of n)
- So do induction/ start from the end
  $h(n_k) \leq d(n_k, G) + h(G) = h^*(n_k)$
  $h(n_{k-1}) \leq d(n_{k-1}, n_k) + h(n_k) \leq d(n_{k-1}, n_k) + d(n_k, G) + h(G) = h^*(n_{k-1})$
  $h(n_{k-2}) \leq d(n_{k-2}, n_{k-1}) + h(n_{k-1}) \leq d(n_{k-2}, n_{k-1}) + d(n_{k-1}, G) + h(G) = h^*(n_{k-2})$
  …
  $h(n_1) \leq \ldots.. = h^*(n_1)$
- So the heuristic is admissible for all nodes *n*!

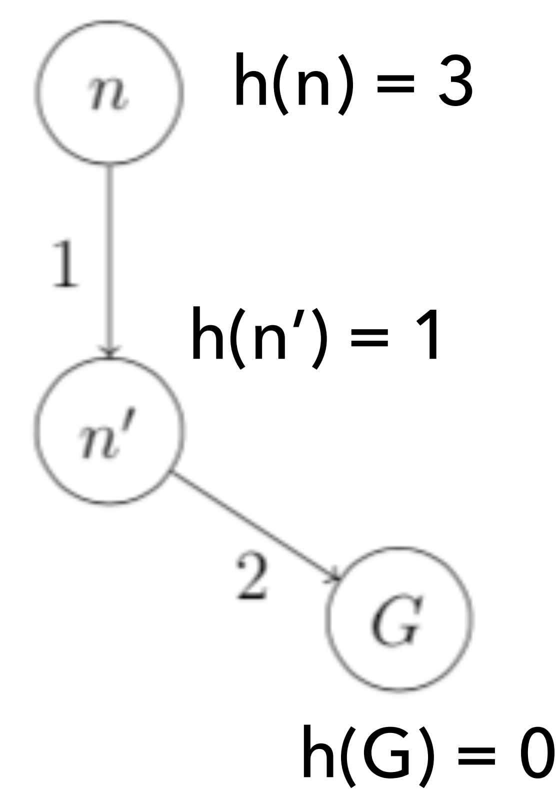
TUTORIAL 3 QUESTION 4 (ADMISSIBLE DOESN'T → CONSISTENT)

- **Give an example of an admissible heuristic function that is not consistent.**

TUTORIAL 3 QUESTION 4 (ADMISSIBLE DOESN'T → CONSISTENT)

- **Give an example of an admissible heuristic function that is not consistent.**

ANSWER



- Then, *h* is admissible, since
  $h(n) = 3 \leq h^*(n) = 1 + 2 = 3$
  $h(n') = 1 \leq h^*(n) = 2$
- But *h* is not consistent because
  $3 = h(n) > d(n, n') + h(n') = 2$

TUTORIAL 3 QUESTION 5

- **You have learned before that A* using graph search is optimal if h(n) is consistent. Does this optimality still hold if h(n) is admissible but inconsistent?**

TUTORIAL 3 QUESTION 5

- **You have learned before that A* using graph search is optimal if h(n) is consistent. Does this optimality still hold if h(n) is admissible but inconsistent?**

ANSWER

- **Yes.** We can construct an example.

DIAGNOSTIC QUIZ (ALSO AY19/20 SEM 2 MIDTERM EXAM)

- **PROVE/DISPROVE: Suppose that the A* search algorithm utilises $f(n) = w \times g(n) + (1 - w) \times h(n)$, where $0 \leq w \leq 1$ (instead of $f(n) = g(n) + h(n)$). For any value of w, an optimal solution will be found whenever h is a consistent heuristic.**

DIAGNOSTIC QUIZ (ALSO AY19/20 SEM 2 MIDTERM EXAM)

- **PROVE/DISPROVE: Suppose that the A* search algorithm utilises $f(n) = w \times g(n) + (1 - w) \times h(n)$, where $0 \leq w \leq 1$ (instead of $f(n) = g(n) + h(n)$). For any value of w, an optimal solution will be found whenever h is a consistent heuristic.**

ANSWER

- False. When $w = 0$, we get greedy best-first search, which is suboptimal (as proven in Question 2c)
- You do not need to prove greedy best first search if you quote the property that it is suboptimal as proven during in tutorials.
- Because this question does not explicitly ask you to prove it.