Heuristic Search

CMPUT 261: Introduction to Artificial Intelligence

P&M §3.6

Logistics

- TA office hours begin this week
  - Including a quick Python refresher
  - See eClass page for times and meeting links
  - *TIME CHANGE:* Mahdi's office hours are now Wednesday 3:00-4:00pm
- Assignment #1 released later today
  - Download from (and submit on) eClass
  - Due: **Tuesday, January 31 at 11:59pm**

Lecture Outline

1. Logistics & Recap
2. Heuristics
3. A* Search

After this lecture, you should be able to:

- Implement and demonstrate the operation of A* search on a graph
- Identify whether a heuristic is admissible
- Construct an admissible heuristic for an arbitrary search problem
- Identify whether one heuristic dominates another
- Construct a dominating heuristic for a set of given heuristics
- Explain when a heuristic will allow more efficient exploration

Recap: Uninformed Search

Different **search strategies** have different properties and behaviour

- **Depth first search** is space-efficient but not always complete or time-efficient
- **Breadth first search** is complete and always finds the shortest path to a goal, but is not space-efficient
- **Iterative deepening search** can provide the benefits of both, at the expense of some time-efficiency
- All three strategies must potentially expand **every node**, and are not guaranteed to return an **optimal solution**
- **Least cost first search** is **optimal** (under some conditions), but *still* must potentially expand every node

Recap: Iterative Deepening Search

Input: a *graph*; a set of *start nodes*; a $goal$ function

```
for max_depth from 1 to ∞:
    more_nodes := False
    frontier := {⟨s⟩ | s is a start node}
    while frontier is not empty:
        select the newest path ⟨n_0, …, n_k⟩ from frontier
        remove ⟨n_0, …, n_k⟩ from frontier
        if goal(n_k):
            return ⟨n_0, …, n_k⟩
        if k < max_depth:
            for each neighbour n of n_k:
                add ⟨n_0, …, n_k, n⟩ to frontier
        else if n_k has neighbours:
            more_nodes := True
    end-while
    if more_nodes = False:
        return None
```

Bonus: Time Complexity of Iterated Deepening Search

- **Breadth-first search** requires $O(b^m)$ time, because in the worst case it visits **every path once**
- **Iterative deepening search** has **worse** time complexity, because it visits every path at least once, and many paths multiple times.
- But **how much** worse?

Claim: Iterated deepening search has time complexity no worse than $O(mb^m)$ (i.e., m **times worse** than breadth first search)

1. Paths of length 1 are visited m times; paths of length 2 are visited $m - 1$ times; ... ; paths of length m are visited 1 time.
2. In other words, every path is visited m **times or fewer**

Note: This is a very **loose bound**. See the text for a much tighter bound.

Recap: Optimality

Definition:
An algorithm is **optimal** if it is guaranteed to return an optimal (i.e., **minimal-cost**) solution **first**.

- Depth-first search, breadth-first search, iterative deepening search are **not** optimal
- Least-cost first search **is** optimal (*if* there is a positive lower bound on arc costs)

Recap: Search Strategies

	Depth First	Breadth First	Iterative Deepening	Least Cost First
Selection	Newest	Oldest	Newest, multiple	Cheapest
Data structure	Stack	Queue	Stack, counter	Priority queue
Complete?	Finite graphs only	Complete	Complete	Complete if cost(p) $> \varepsilon$
Space complexity	$O(mb)$	$O(b^m)$	$O(mb)$	$O(b^m)$
Time complexity	$O(b^m)$	$O(b^m)$	$O(mb^m)$ **	$O(b^m)$
Optimal?	No	No	No	Optimal

Domain Knowledge

- Domain-specific knowledge can help speed up search by identifying **promising directions** to explore
- We will encode this knowledge in a function called a **heuristic function** which **estimates** the cost to get from a node to a goal node
- The search algorithms in this lecture take account of this heuristic knowledge when **selecting** a path from the frontier

Heuristic Function

Definition:
A **heuristic function** is a function $h(n)$ that returns a non-negative estimate of the cost of the **cheapest** path from node n to **some** goal node.

- For paths: $h(\langle n_0, \ldots, n_k \rangle) = h(n_k)$
- Uses only **readily-available** information about a node (i.e., easy to compute)
- **Problem-specific**

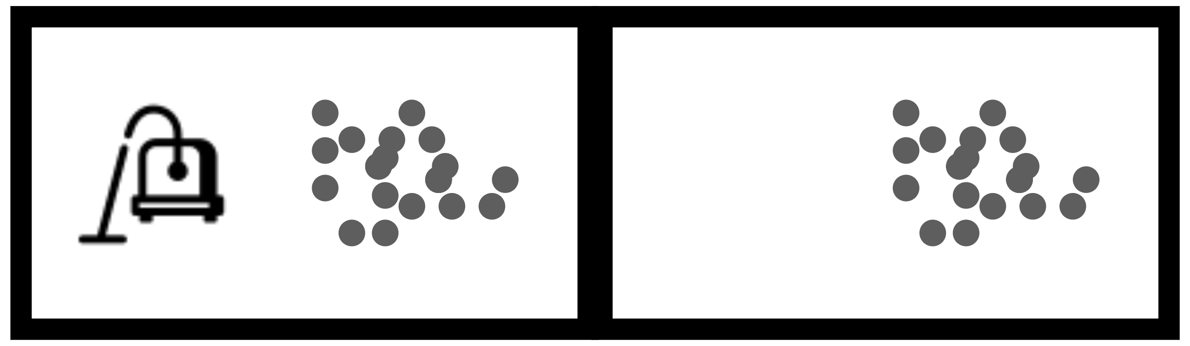
Admissible Heuristic

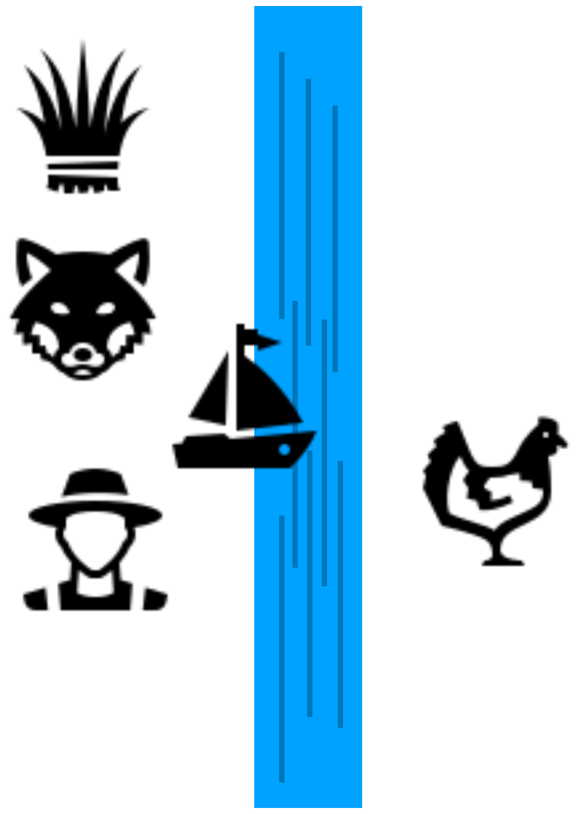
Definition:
A heuristic function is **admissible** if $h(n)$ is **always less than or equal** to the **actual cost** of the cheapest path from n to any goal node.

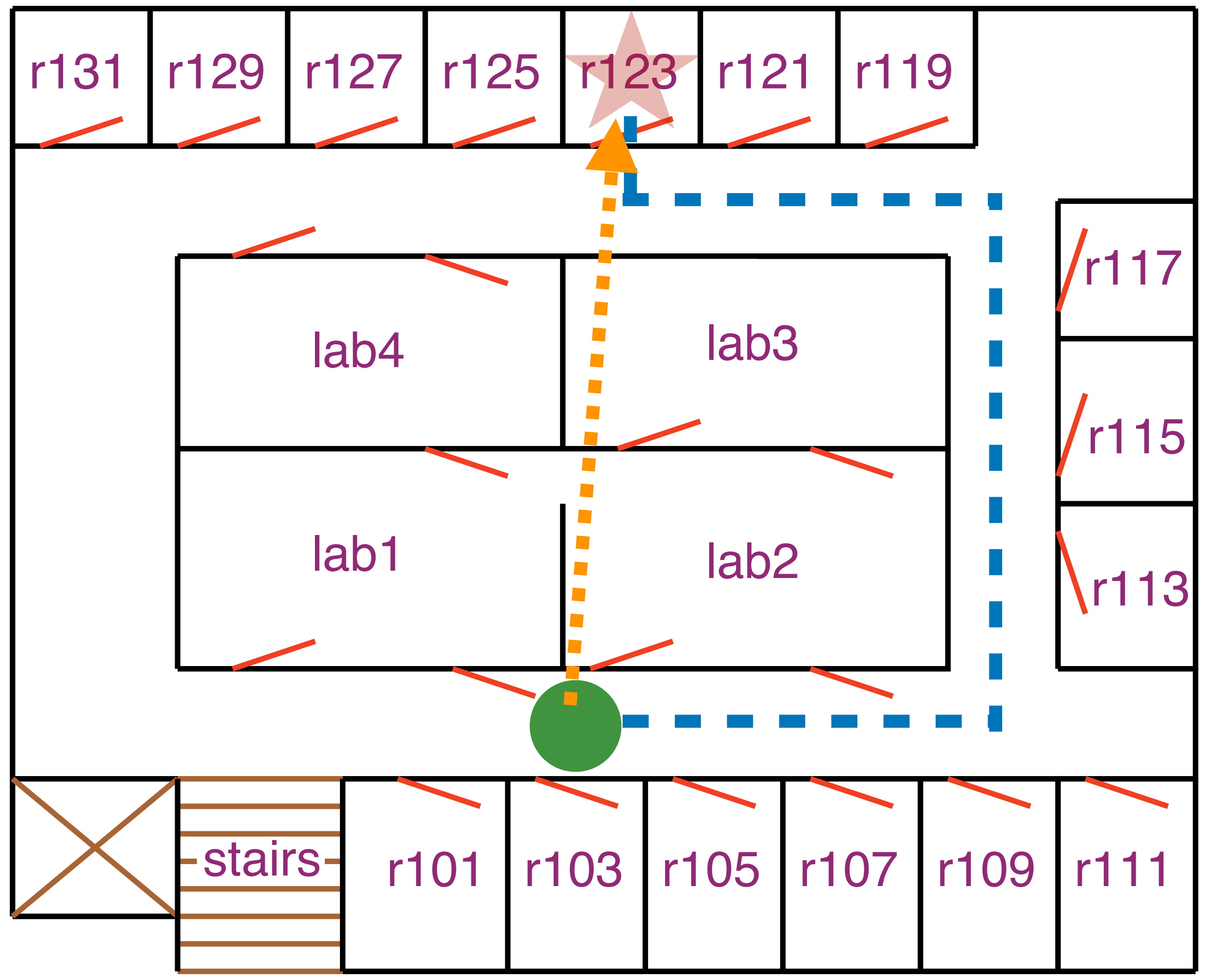
- i.e., $h(n)$ is a **lower bound** on cost$(\langle n, \ldots, g \rangle)$ for any **goal node** g

Example Heuristics

- **Number of dirty rooms** for **VacuumBot** (ignores the need to move between rooms)
- **Euclidean distance** for **DeliveryBot** (ignores that it can't go through walls)
- **Points** for chess pieces (ignores positional strength)
- **Farmer** problem?







Question: Which of these heuristics are **admissible**? *Why?*

Constructing Admissible Heuristics

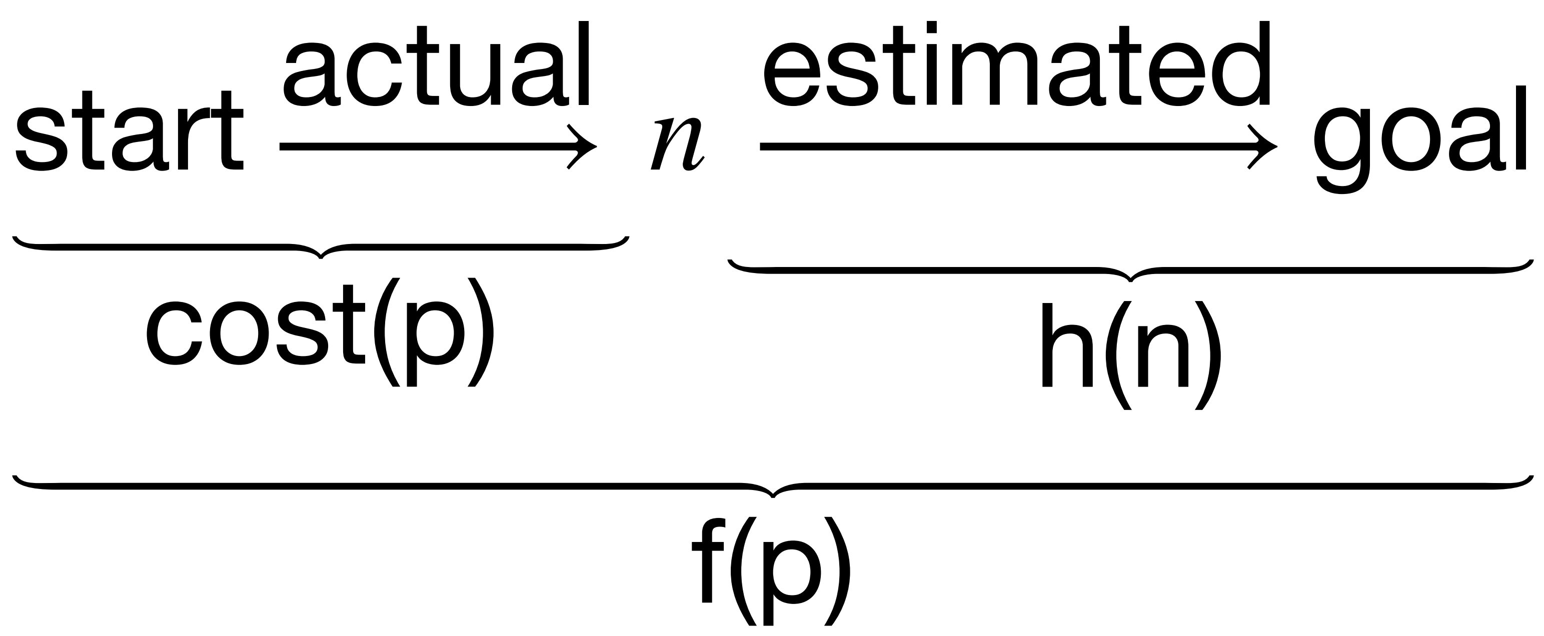
- Search problems try to find a cost-minimizing path, subject to **constraints** encoded in the search graph
- How to construct an easier problem? **Drop** some constraints.
  - This is called a **relaxation** of the original problem
- The cost of the optimal solution to the relaxation will always be an **admissible heuristic** for the original problem (**Why?**)
- **Neat trick**: If you have two admissible heuristics h_1 and h_2, then $h_3(n) = \max\{h_1(n), h_2(n)\}$ is admissible too! (**Why?**)

Simple Uses of Heuristics

- **Heuristic depth first search:** Add neighbours to the frontier in **decreasing order** of their heuristic values, then run depth first search as usual
  - Will explore most promising successors first, but
  - Still explores **all paths** through a successor before considering other successors
  - Not complete, not optimal
- **Greedy best first search:** Select path from the frontier with the **lowest heuristic** value
  - Not guaranteed to work any better than breadth first search (**why?**)

A* Search

- A* search uses **both** path cost information and heuristic information to select paths from the frontier
- Let $f(p) = \text{cost}(p) + h(p)$
  - $f(p)$ **estimates** the total cost to the nearest goal node **starting from** p
- A* removes paths from the frontier with **smallest** $f(p)$
- When h is **admissible**,
  $p^* = \langle s, \ldots, n, \ldots, g \rangle$ is a **solution**, and
  $p' = \langle s, \ldots, n \rangle$ is a **prefix** of p^*:
  - $f(p') \leq \text{cost}(p^*)$ (**why?**)



A* Search Algorithm

Input: a *graph*; a set of *start nodes*; a $goal$ function

$frontier$:= $\{\langle s \rangle \mid s$ *is a start node*$\}$
while $frontier$ is not empty:
 select f-minimizing path $\langle n_0, \ldots, n_k \rangle$ from *frontier*
 remove $\langle n_0, \ldots, n_k \rangle$ from $frontier$
 if $goal(n_k)$:
  return $\langle n_0, \ldots, n_k \rangle$
 for each neighbour n of n_k:
  add $\langle n_0, \ldots, n_k, n \rangle$ to *frontier*
end while

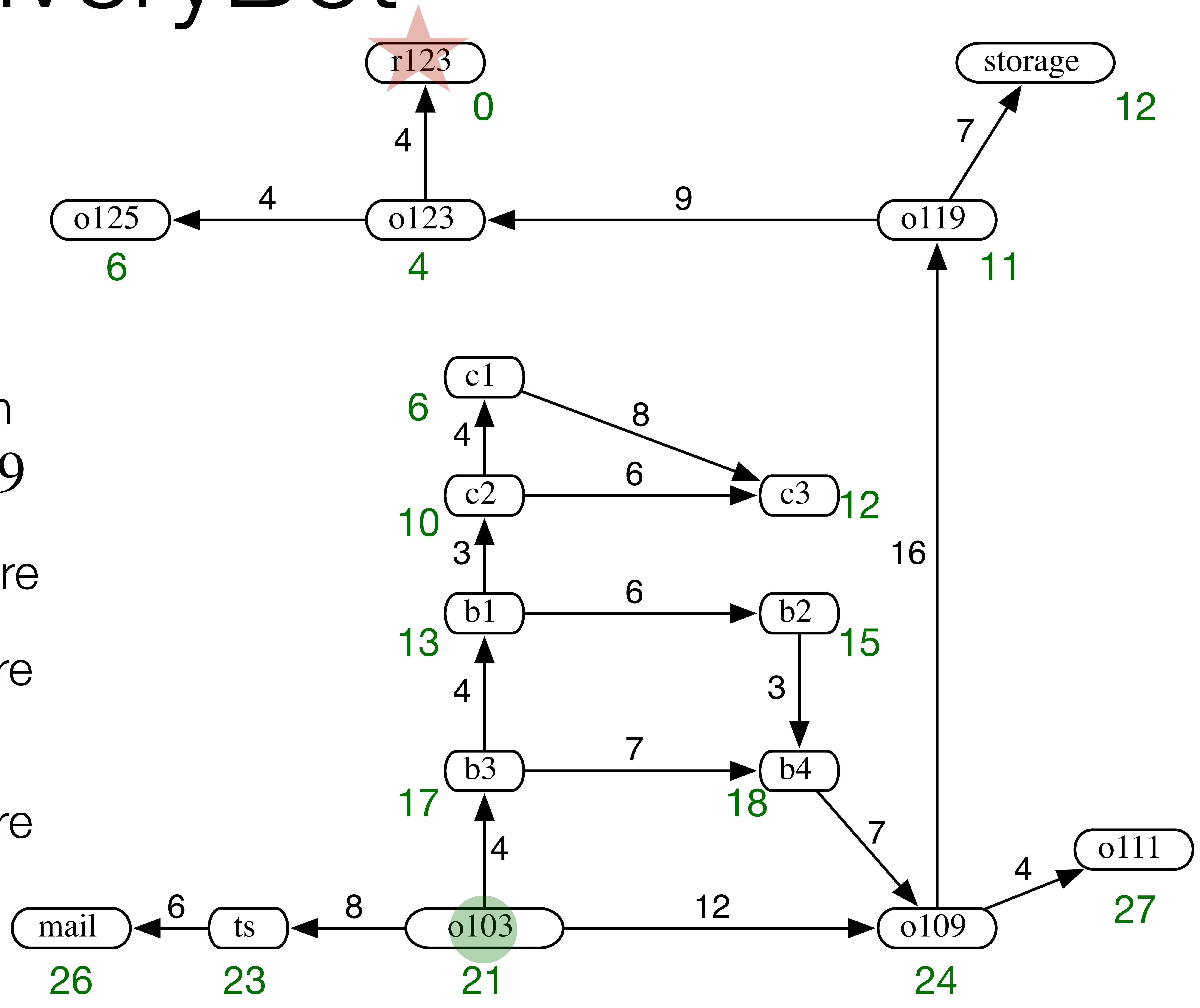
i.e., $f(\langle n_0, \ldots, n_k \rangle) \leq f(p)$ for all other paths $p \in frontier$

Question:

What **data structure** for the frontier implements this search strategy?

A* Search Example: DeliveryBot

- Heuristic: **Euclidean distance**
- **Question:** What is $f(\langle o103, b3 \rangle)$? $f(\langle o103, o109 \rangle)$?
- A* will spend a bit of time exploring paths in the labs before trying to go around via $o109$
- At that point the heuristic starts helping more
- **Question:** Does breadth-first search explore paths in the lab too?
- **Question:** Does breadth-first search explore any paths that A* does not?



A* Optimality

Theorem:

If there is a solution of finite cost, A* using heuristic function h always returns an **optimal** solution (in **finite time**), if

1. The branching factor is **finite**, and
2. All **arc costs** are greater than some $\epsilon > 0$, and
3. h is an **admissible** heuristic.

Proof:

1. **No suboptimal solution** will be removed from the frontier whenever the frontier contains a **prefix of the optimal solution**
2. The **optimal solution** is guaranteed to be **removed from the frontier** eventually

A* Optimality Proofs: A Lexicon

An **admissible heuristic**: $h(n)$

$f(\langle n_0, \ldots, n_k \rangle) = \text{cost}(\langle n_0, \ldots, n_k \rangle) + h(n_k)$

A **start node**: s

A **goal node**: z (i.e., goal$(z) = 1$)

The **optimal solution**: $p* = \langle s, \ldots, a, b, \ldots z \rangle$

A **prefix** of the optimal solution: $p' = \langle s, \ldots, a \rangle$

A **suboptimal solution**: $g = \langle s, q, \ldots, z \rangle$

A* Optimality

An **admissible heuristic**: $h(n)$
$f(\langle n_0, \ldots, n_k \rangle) = \text{cost}(\langle n_0, \ldots, n_k \rangle) + h(n_k)$
A **start node**: s
A **goal node**: z (i.e., $\text{goal}(z) = 1$)
The **optimal solution**: $p^* = \langle s, \ldots, a, b, \ldots z \rangle$
A **prefix** of the optimal solution: $p' = \langle s, \ldots, a \rangle$
A **suboptimal solution**: $g = \langle s, q, \ldots, z \rangle$

Proof part 1: Optimality (no g is removed before p^*)

1. $f(g) = \text{cost}(g)$ and $f(p^*) = \text{cost}(p^*)$
   (i) $f(\langle n_0, \ldots, n_k \rangle) = \text{cost}(\langle n_0, \ldots, n_k \rangle) + h(n_k)$, and $h(z) = 0$
2. $f(p') < f(g)$
   (i) $f(\langle s, \ldots, a \rangle) = \text{cost}(\langle s, \ldots, a \rangle) + h(a)$
   (ii) $f(\langle s, \ldots, a, b, \ldots, z \rangle) = \text{cost}(\langle s, \ldots, a, b, \ldots, z \rangle) + h(z) = \text{cost}(\langle s, \ldots, a \rangle) + \text{cost}(a, b, \ldots, z \rangle)$
   (iii) $h(a) \leq \text{cost}(\langle a, b, \ldots, z \rangle)$
   (iv) $f(p') \leq f(p^*) < f(g)$ ∎

A* Completeness

An **admissible heuristic**: $h(n)$

$f(\langle n_0, \ldots, n_k \rangle) = \text{cost}(\langle n_0, \ldots, n_k \rangle) + h(n_k)$

A **start node**: s

A **goal node**: z (i.e., goal$(z) = 1$)

The **optimal solution**: $p^* = \langle s, \ldots, a, b, \ldots z \rangle$

A **prefix** of the optimal solution: $p' = \langle s, \ldots, a \rangle$

A **suboptimal solution**: $g = \langle s, q, \ldots, z \rangle$

Proof part 2: A* is **complete**

- Every path that is removed from the frontier is only replaced by more-costly paths (**why?**)
- Since individual arc costs are larger than ϵ, every path in the frontier will eventually have cost larger than k, for any finite k
  - Every path with at least $\dfrac{k}{\epsilon}$ arcs will have cost larger than k
- So every path in the frontier will eventually have cost larger than the cost of the optimal solution
- So the optimal solution will eventually be removed from the frontier
- **Question:** Why are we talking about **costs** and not **f-values**?

Comparing Heuristics

- Suppose that we have two **admissible** heuristics, h_1 and h_2
- Suppose that for every node n, $h_2(n) \geq h_1(n)$

Question: Which heuristic is better for search (with A*)?

Dominating Heuristics

Definition:
A heuristic h_2 **dominates** a heuristic h_1 if

1. $\forall n : h_2(n) \geq h_1(n)$, and
2. $\exists n : h_2(n) > h_1(n)$.

Theorem:
If h_2 dominates h_1, and both heuristics are admissible, then A* using h_2 will never remove more paths from the frontier than A* using h_1.

- i.e., better heuristics remove weakly fewer paths

Question:
Which admissible heuristic dominates **all other** admissible heuristics?

A* Analysis

For a search graph with *finite* maximum branch factor b and *finite* maximum path length m...

1. What is the worst-case **space complexity** of A*?
   [A: $O(m)$] [B: $O(mb)$] [C: $O(b^m)$] [D: it depends]

2. What is the worst-case **time complexity** of A*?
   [A: $O(m)$] [B: $O(mb)$] [C: $O(b^m)$] [D: it depends]

Question: If A* has the same space and time complexity as least cost first search, then what is its advantage?

Summary

- **Domain knowledge** can help speed up graph search
- Domain knowledge can be expressed by a **heuristic function**, which **estimates** the cost of a path to the goal from a node
- **Admissible** heuristics can be built from **relaxations** of the original problem
- *Simple* uses of heuristics do not guarantee improved performance
- **A* algorithm** for use of admissible heuristics