Heuristic Search

CMPUT 366: Intelligent Systems

P&M §3.6

Lecture Outline

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

Sanction Policy

I have been instructed to announce the new sanction policy:

For first time offenders in plagiarism, the sanction will be zero in the assignment with a discipline code of 8 annotated in the transcript and Conduct Probation, which starts immediately and ends at the end of the degree program. Under Conduct Probation if a student is found to have violated the Code of Student Behaviour a second time, the student will be recommended for suspension.

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

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 is **worse**, 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.

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 n to a goal node.

- For paths: $h(<n_1, n_2, ..., n_k>) = h(n_k)$
- 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 cost of the cheapest path from n to a goal node.

- i.e., $h(n)$ is a **lower bound** on cost(<n, ..., g>) for any **goal node** g

Example Heuristics

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

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 fringe 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

A* Search

$$\underbrace{\underbrace{\text{start} \xrightarrow{\text{actual}} n}_{\text{cost(p)}} \; \underbrace{\xrightarrow{\text{estimated}} \text{goal}}_{\text{h(n)}}}_{\text{f(p)}}$$

- A* search uses **both** path cost information and heuristic information to select paths from the frontier

- Let $f(p) = \text{cost}(p) + h(p)$

- A* removes paths from the frontier with **smallest** $f(p)$

- When h is **admissible**,
  $p^* = <s, \ldots, n, \ldots, g>$ is a **solution**, and
  $p = <s, \ldots, n>$ 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

i.e., $f(<n_1, n_2, ..., n_k>) \leq f(p)$
for all other paths $p \in$ *frontier*

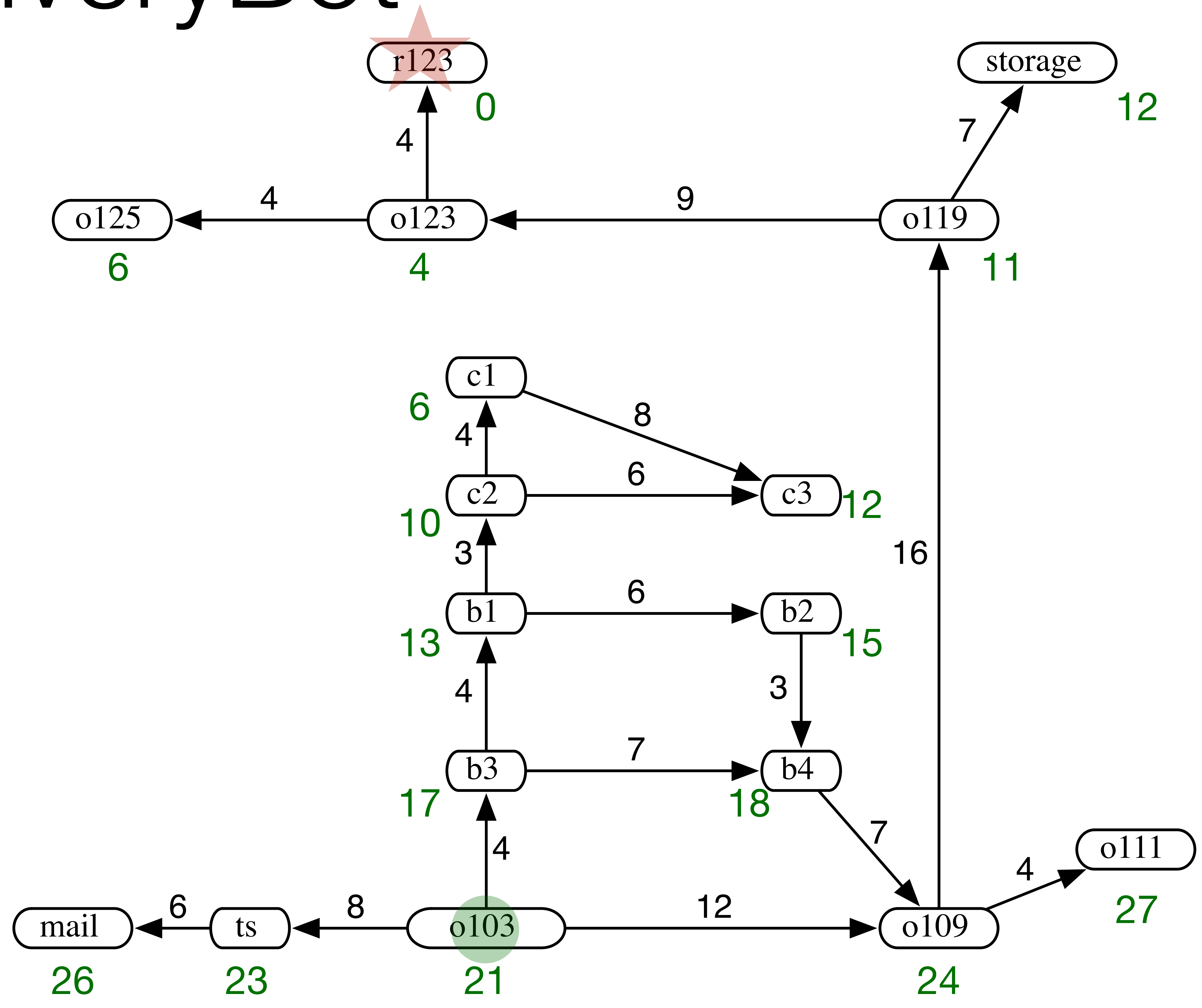
frontier := { $<s>$ | s is a start node}
while *frontier* is not empty:
    select heuristic minimizing path $<n_1, n_2, ..., n_k>$ from *frontier*
    remove $<n_1, n_2, ..., n_k>$ from *frontier*
    if *goal*(n_k):
        return $<n_1, n_2, ..., n_k>$
    for each neighbour n of n_k:
        add $<n_1, n_2, ..., n_k, n>$ to *frontier*
end while

Question:

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

A* Search Example: DeliveryBot

- Heuristic: **Euclidean distance**
- **Question:** What is f(b3)? f(109)?
- 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* Theorem

Theorem:
If there is a solution, A* using heuristic function *h* always returns an **optimal** solution, if

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

A* Theorem: Completeness

Proof part 1: A* is complete

- Since arc costs are larger than ε, every path in the frontier will eventually have cost larger than *k*, for any finite *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

A* Theorem: Optimality

Proof part 2: Optimality

- If path g is a **solution**, then $f(g)$ is equal to cost(g) **(Why?)**

- If a path p **leads to an optimal solution**, and path g is **any solution**, then $f(p) \leq f(g)$ (**Why?**)

i.e., p=<s, n_1, …, n_k>,
p^* = <s, n_1, …, n_k, n_{k+1}, …, z>,
and p^* is **optimal**

- So no **sub-optimal solution** will be removed from the frontier while a **path that leads to an optimal solution** is on the frontier.

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?

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.

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
- A* considers both **path cost** and **heuristic cost** when selecting paths: $f(p) = \mathrm{cost}(p) + h(p)$
- **Admissible** heuristics guarantee that A* will be **optimal**
- Admissible heuristics can be built from **relaxations** of the original problem
- The more **accurate** the heuristic is, the **fewer** the paths A* will explore