Branch & Bound

or, How I Learned to Stop Worrying and Love Depth First Search

CMPUT 366: Intelligent Systems

P&M §3.7-3.8

Logistics

- **Assignment #1 released**
  - Available on eClass
  - Due: Monday **February 8, 2021**
- TA office hours discussion
  - Some students have expressed preference for later times
  - Depends on TA availability

Recap: Heuristics

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.

- e.g., Euclidean distance instead of travelled distance

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$(\langle n, \ldots, g\rangle)$ for any **goal node** g

Recap: 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^*:
  - $f(p) \leq \text{cost}(p^*)$

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

Recap: A* Search Algorithm

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

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

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

Recap: A* Theorem

Theorem:

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

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

Proof:

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

Lecture Outline

1. Recap & Logistics
2. Cycle Pruning
3. Branch & Bound
4. Exploiting Search Direction

Cycle Pruning

- Even on **finite graphs**, depth-first search may not be complete, because it can get trapped in a **cycle**.
- A search algorithm can **prune** any path that ends in a node already on the path **without missing an optimal solution** (**Why?**)

Questions:

1. Is depth-first search on with cycle pruning **complete** for finite graphs?
2. What is the **time complexity** for cycle checking in **depth-first search**?
3. What is the **time complexity** for cycle checking in **breadth-first search**?

Cycle Pruning
Depth First Search

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

frontier := { $<s>$ | s is a start node}
while *frontier* is not empty:
select the newest **path** $<n_1, n_2, \ldots, n_k>$ **from** *frontier*
remove $<n_1, n_2, \ldots, n_k>$ from *frontier*
if $n_k \neq n_j$ for all $1 \leq j < k$:
if *goal*(n_k):
return $<n_1, n_2, \ldots, n_k>$
for each neighbour n of n_k:
add $<n_1, n_2, \ldots, n_k, n>$ to *frontier*
end while

Heuristic Depth First Search

	Heuristic Depth First	A*	Branch & Bound
Space complexity	$O(mb)$	$O(b^m)$	$O(mb)$
Heuristic Usage	Limited	Optimal	Optimal (if bound low enough)
Optimal?	No	Yes	Yes (if bound high enough)

Branch & Bound

- The $f(p)$ function provides a **path-specific lower bound** on solution cost starting from p
- **Idea:** Maintain a **global upper bound** on solution cost also
    - Then prune any path whose lower bound **exceeds** the upper bound
- **Question:** Where does the upper bound come from?
    - **Cheapest** solution found so far
    - Before solutions found, specified on entry
    - Can increase the global upper bound **iteratively**
(as in iterative deepening search)

Branch & Bound Algorithm

Input: a *graph*; a set of *start nodes*; a *goal* function; heuristic $h(n)$; $bound_0$

$frontier$:= { $\langle s \rangle$ | s is a start node}
$bound := bound_0$
$best := \varnothing$
while *frontier* is not empty:
    select the newest path $\langle n_1, n_2, \ldots, n_k \rangle$ from *frontier*
    remove $\langle n_1, n_2, \ldots, n_k \rangle$ from *frontier*
    if $\text{cost}(\langle n_1, n_2, \ldots, n_k \rangle) + h(n_k) \leq bound$:
        if $goal(n_k)$:
            $bound := \text{cost}(\langle n_1, n_2, \ldots, n_k \rangle)$
            $best := \langle n_1, n_2, \ldots, n_k \rangle$
        else:
            for each neighbour n of n_k:
                add $\langle n_1, n_2, \ldots, n_k, n \rangle$ to *frontier*
end while
return *best*

Branch & Bound Analysis

- If $bound_0$ is set to just above the optimal cost, branch & bound will explore no more paths than A*
(**Why?**)
- With **iterative increasing** of $bound_0$, will re-explore some lower-cost paths, but still similar time-complexity to A*

Question: How *much* should the bound get increased by?

- Iteratively increase bound to the **lowest-f-value** node that was **pruned**
- Worse than A* by no more than a **linear** factor of *m,*
by the same argument as for iterative deepening search
- Choosing next f-limit is an active area of research

Exploiting Search Direction

- When we care about finding the path to a known goal node, we can search forward, but we can often search **backward**
- Given a search graph $G=(N,A)$, **known** goal node g, and set of start nodes S, can construct a **reverse search problem** $G=(N, A^r)$:

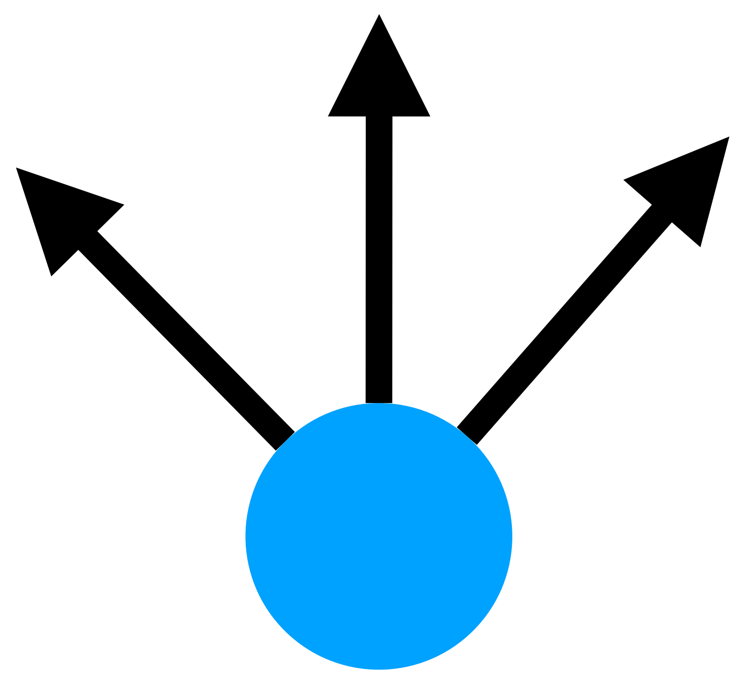
    1. Designate g as the start node
    2. $A^r = \{ <n_2,n_1> \mid <n_1,n_2> \in A \}$
    3. $goal^r(n)$ = True if $n \in S$
       (i.e., if n is a start node of the original problem)

Questions:

1. When is this **useful**?
2. When is this **infeasible**?

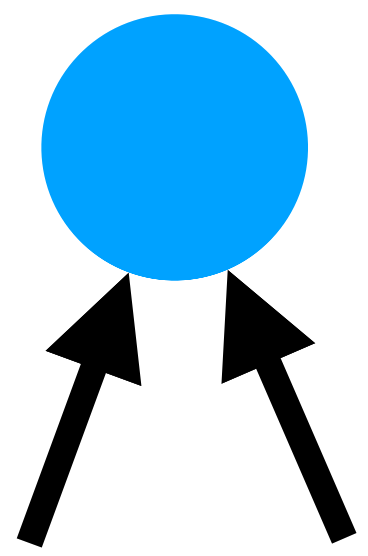
Reverse Search

Definitions:



1. **Forward branch factor**: Maximum number of **outgoing** neighbours
   Notation: *b*

   - Time complexity of forward search: $O(b^m)$



2. **Reverse branch factor**: Maximum number of **incoming** neighbours
   Notation: r

   - Time complexity of reverse search: $O(r^m)$

When the reverse branch factor is **smaller** than the forward branch factor, reverse search is more **time-efficient**.

Bidirectional Search

- **Idea:** Search backward from from goal and forward from start **simultaneously**
- Time complexity is **exponential in path length**, so exploring half the path length is an exponential improvement
    - Even though must explore half the path length **twice**
- Main problems:
    - **Ensuring** that the frontiers meet
    - **Checking** that the frontiers have met

Questions:

What bidirectional **combinations** of search algorithm make sense?

- Breadth first + Breadth first?
- Depth first + Depth first?
- Breadth first + Depth first?

Summary

- A* considers both **path cost** and **heuristic cost** when selecting paths:

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

- **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
- **Branch & bound** combines the **optimality** guarantee and **heuristic efficiency** of A* with the space efficiency of depth-first search
- Tweaking the **direction of search** can yield efficiency gains