Chapter 4: Searching

- Lecture 1 Searching. Graphs. Generic search engine.
- Lecture 2 Blind search strategies.
- Lecture 3 Heuristic search, including A^*.
- Lecture 4 Pruning the search space, direction of search, iterative deepening, dynamic programming.
- Lecture 5 Constraint satisfaction problems.
- Lecture 6 Consistency algorithms, hill climbing, randomized algorithms.

Heuristic Search

Previous methods do not take into account the goal until they are at a goal node.

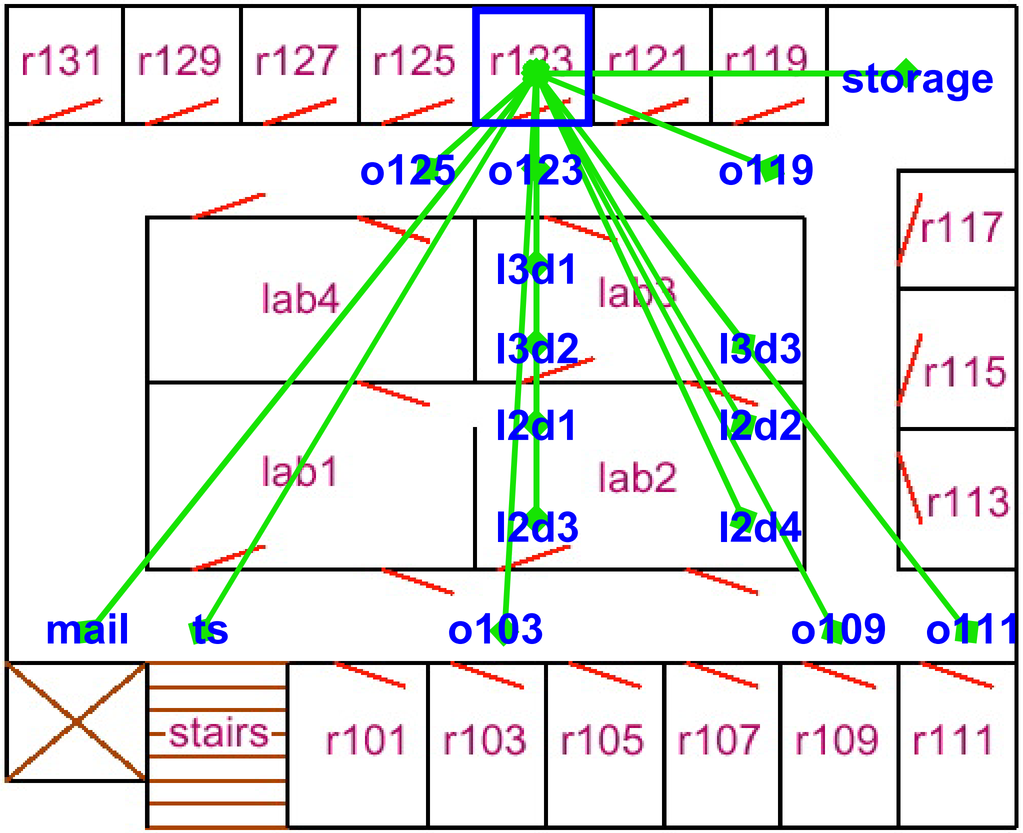
Often there is extra knowledge that can be used to guide the search: heuristics.

We use $h(n)$ as an estimate of the distance from node n to a goal node.

$h(n)$ is an underestimate if it is less than or equal to the actual cost of the shortest path from node n to a goal.

$h(n)$ uses only readily obtainable information about a node.

Robot domain - Heuristic function

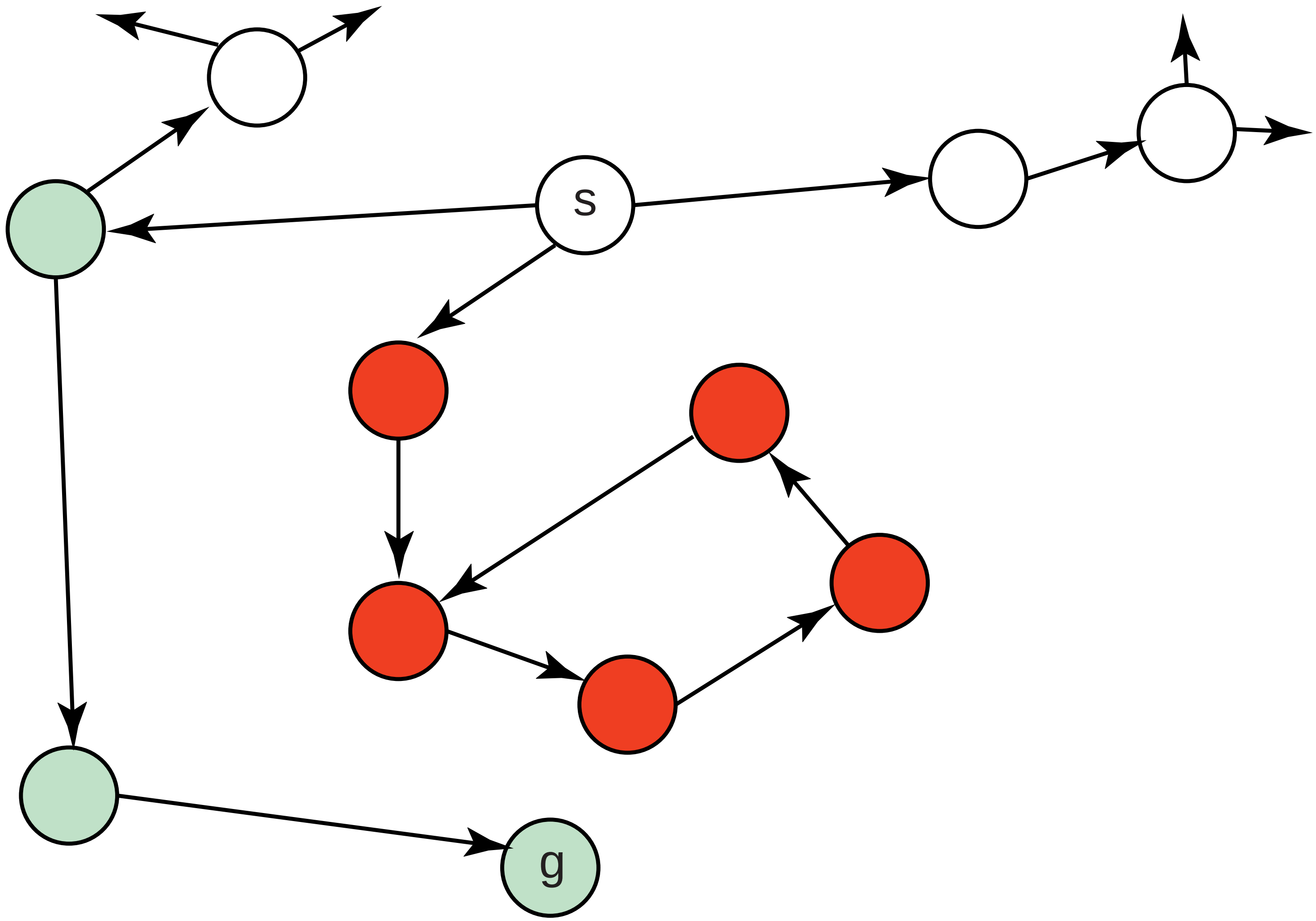


n	h(n)	n	h(n)
mail	35	ts	29
o103	21	o109	29
o111	33	o119	13
o123	4	o125	8
r123	0	l2d1	13
l2d2	19	l2d3	17
l2d4	22	l3d1	8
l3d2	10	l3d3	16
stor	12		

Best-first Search

- Idea: always choose the node on the frontier with the smallest *h*-value.
- It treats the frontier as a priority queue ordered by *h*.
- It uses space exponential in path length.
- It isn't guaranteed to find a solution, even if one exists. It doesn't always find the shortest path.

Illustrative Graph — Best-first Search



Best-first search

The – naive – way to implement Best-first search

- *Select(Node, [Node | Frontier], Frontier).*
- *add_to_frontier (Neighbors, $Frontier_1$, $Frontier_3$)* ←
  append ($Frontier_1$, Neighbors, $Frontier_2$) ∧
  sort_by_h ($Frontier_2$, $Frontier_3$)

A better way:

implement the frontier as a heap, instead as implementing it as a queue.

→ time complexity for *add_to_frontier* is *log(n)* instead of *n.*

Best first search (Robot domain)

Frontier	Node	Neighbors
o103	o103	ts, l2d3, o109
l2d3 *[17]*, ts *[29]*, o109 *[29]*	l2d3	l2d1, l2d4
l2d1 *[13]*, l2d4 *[22]*, ts *[29]*, o109 *[29]*	l2d1	l3d2, l2d2
l3d2 *[10]*, l2d2 *[19]*, l2d4 *[22]*, ts *[29]*, o109 *[29]*	l3d2	l3d1, l3d3
l3d1 *[8]*, l3d3 *[16]*, l2d2 *[19]*, l2d4 *[22]*, …	l3d1	l3d3
l3d3 *[16]*, l3d3 *[16]*, l2d2 *[19]*, l2d4 *[22]*, …	l3d3	–
l2d2 *[19]*, l2d4 *[22]*, ts *[29]*, o109 *[29]*	l2d2	l2d4
l2d4 *[22]*, l2d4 *[22]*, ts *[29]*, o109 *[29]*	l2d4	o109
o109 *[29]* ts *[29]*, o109 *[29]*	o109	o111, o119
o119 *[13]*, ts *[29]*, o109 *[29]*, o111 *[33]*	o119	storage, o123

Best first search (Robot domain, cont'd)

Frontier	Node	Neighbors
o119 *[13],* ts *[29],* o109 *[29],* o111 *[33]*	o119	storage, o123
o123 *[4],* st. *[12],* ts *[29],*o109 *[29],*o111 *[33]*	o123	r123, o125
r123 *[0]*, o125 *[8]*, st. *[12],*ts *[29],*…	r123	*is goal*

Heuristic Depth-first Search

It's a way to use heuristic knowledge in depth-first search.

Idea: order the neighbors of a node (by h) before adding them to the front of the frontier.

Locally chooses which subtree to develop, but still does depth-first search. It explores all paths from the node at the head of the frontier before exploring paths from the next node.

Space is linear in path length. It isn't guaranteed to find a solution. It can get led up the garden path.

Heuristic depth-first search

- *Select(Node, [Node | Frontier], Frontier).*
- *add_to_frontier (Neighbors, $Frontier_1$, $Frontier_2$) ←*
  sort_by_h (Neighbors, Sorted_Neighbors) ∧
  append ($Frontier_1$, Sorted_Neighbors, $Frontier_2$)

Heuristic Depth first search (Robot domain)

Frontier	Node	Neighbors
o103	o103	l2d3, o109, ts
l2d3, o109, ts	l2d3	l2d1, l2d4
l2d1, l2d4, o109, ts	l2d1	l3d2, l2d2
l3d2, l2d2, l2d4, o109, ts	l3d2	l3d1, l3d3
l3d1, l3d3, l2d2, l2d4, o109, ts	l3d1	l3d3
l3d3, l3d3, l2d2, l2d4, o109, ts	l3d3	–
l3d3, l2d2, l2d4, o109, ts	l3d3	–
l2d2, l2d4, o109, ts	l2d2	l2d4
l2d4, l2d4, o109, ts	l2d4	o119, o111

Depth first search (Robot domain / cont’d)

Frontier	Node	Neighbors
l2d4, l2d4, o109, ts	l2d4	o119, o111
o119, o111, l2d4, o109, ts	o119	o123, storage
o123, st., o111, l2d4, o109, ts	o123	r123, o125
r123, o125, st., o111, …	r123	*is goal*

Exercise A : To be discussed in Lecture 4.4

1. Discuss the differences and similarities between
   - Best-first search & Heuristic depth-first search
   - Compare their behavior with the non-heuristic variants.
2. Discuss heuristic search for SLD derivation graphs.
   - What are good heuristic functions for SLD search graphs?

A^* Search

A^* search takes the path to a node and heuristic value into account.

Let $g(n)$ be the cost of the path found to node n.

Let $h(n)$ be the estimate of the cost from n to a goal.

Let $f(n) = g(n) + h(n)$. It is an estimate of a path from the start to a goal via n.

$$\underbrace{\underbrace{start \xrightarrow{\text{actual}} n}_{g(n)} \; \underbrace{\xrightarrow{\text{estimate}} goal}_{h(n)}}_{f(n)}$$

A^* Search Algorithm

- A^* is a mix of lowest-cost-first and best-first search.
- It treats the frontier as a priority queue ordered by $f(n)$.
- It always chooses the node on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that node.

A* Search Algorithm

- *Select(Node, [Node | Frontier], Frontier).*
- *add_to_frontier (Neighbors, $Frontier_1$, $Frontier_3$) ←*
  append ($Frontier_1$, Neighbors, $Frontier_2$) ∧
  sort_by_f ($Frontier_2$, $Frontier_3$)

A* Search (Robot domain)

Frontier	g(n)	h(n)	f(n)	node	neighbors
o113				o113	l2d3, o109, ts
l2d3	*4*	*17*	21	l2d3	l2d1 [5], l2d4 [10]
ts	*12*	*29*	41		
o109	*15*	*29*	44		
l2d1	*9*	*13*	22	l2d1	l3d2 [4], l2d2 [10]
l2d4	*14*	*22*	36		
ts	*12*	*29*	41		
o109	*15*	*29*	44		
l3d2	*13*	*10*	23	l3d2	l3d1 [4], l3d3 [10]
l2d4	*14*	*22*	36		
l2d2	*19*	*19*	38		
ts	*12*	*29*	41		
o109	*15*	*29*	44		

A* Search (Robot domain / cont'd 1)

Frontier	g(n)	h(n)	f(n)	node	neighbors
l3d2	*13*	*10*	23	l3d2	l3d1 [4], l3d3 [10]
l3d1	*17*	*8*	25	l3d1	l3d3 [11]
l2d4	*14*	*22*	36		
l2d2	*19*	*19*	38		
l3d3	*23*	*16*	39		
ts	*12*	*29*	41		
o109	*15*	*29*	44		
l2d4	*14*	*22*	36	l2d4	o109 [6]
l2d2	*19*	*19*	38		
l3d3	*23*	*16*	39		
ts	*12*	*29*	41		
l3d3	*28*	*16*	44		
o109	*15*	*29*	44		

A* Search (Robot domain / cont'd 2)

Frontier	g(n)	h(n)	f(n)	node	neighbors
l2d4	*14*	*22*	36	l2d4	o109 [6]
l2d2	*19*	*19*	38	l2d2	l2d4 [5]
l3d3	*23*	*16*	39		
ts	*12*	*29*	41		
l3d3	*28*	*16*	44		
o109	*15*	*29*	44		
o109	*20*	*29*	49		
l3d3	*23*	*16*	39	l3d3	l2d2 [4]
ts	*12*	*29*	41		
l3d3	*28*	*16*	44		
o109	*15*	*29*	44		
l2d4	*24*	*22*	46		
o109	*20*	*29*	49		

A* Search (Robot domain / cont'd 3)

Frontier	g(n)	h(n)	f(n)	node	neighbors
l3d3	23	16	39	l3d3	l2d2 [4]
ts	12	29	41	ts	mail [5]
l3d3	28	16	44		
o109	15	29	44		
l2d2	27	19	46		
l2d4	24	22	46		
o109	20	29	49		
l3d3	28	16	44	l3d3	l2d2 [4]
o109	15	29	44		
l2d2	27	19	46		
l2d4	24	22	46		
o109	20	29	49		
mail	17	35	52		

A* Search (Robot domain / cont'd 4)

Frontier	g(n)	h(n)	f(n)	node	neighbors
l3d3	*28*	*16*	44	l3d3	l2d2 [4]
o109	*15*	*29*	44	o109	o119 [21], o111[5]
l2d2	*27*	*19*	46		
l2d4	*24*	*22*	46		
o109	*20*	*29*	49		
l2d2	*32*	*19*	51		
mail	*17*	*35*	52		
l2d2	*27*	*19*	46	l2d2	l2d4 [5]
l2d4	*24*	*22*	46		
o119	*36*	*13*	49		
o109	*20*	*29*	49		
l2d2	*32*	*19*	51		
mail	*17*	*35*	52		
o111	*20*	*33*	53		

A* Search (Robot domain / cont'd 5)

Frontier	g(n)	h(n)	f(n)	node	neighbors
l2d2	*27*	*19*	46	l2d2	l2d4 [5]
l2d4	*24*	*22*	46	l2d4	o109 [6]
o119	*36*	*13*	49		
o109	*20*	*29*	49		
l2d2	*32*	*19*	51		
mail	*17*	*35*	52		
o111	*20*	*33*	53		
l2d4	*32*	*22*	54		
o119	*36*	*13*	49	o119	o123 [10] ,stor [8]
o109	*20*	*29*	49		
l2d2	*32*	*19*	51		
mail	*17*	*35*	52		
l2d4	*30*	*22*	52		
o111	*20*	*33*	53		
l2d4	*32*	*22*	54		

A* Search (Robot domain / cont'd 6)

Frontier	g(n)	h(n)	f(n)	node	neighbors
o119	*36*	*13*	49	o119	o123 [10] ,stor [8]
o109	*20*	*29*	49	o109	o119 [21], o111[5]
o123	*46*	*4*	50		
l2d2	*32*	*19*	51		
mail	*17*	*35*	52		
l2d4	*30*	*22*	52		
o111	*20*	*33*	53		
l2d4	*32*	*22*	54		
stor	*44*	*12*	56		

A* Search (Robot domain / cont'd 7)

Frontier	g(n)	h(n)	f(n)	node	neighbors
o109	*20*	*29*	49	o109	o119 [21], o111[5]
o123	*46*	*4*	50	o123	r123 [4], o125 [5]
l2d2	*32*	*19*	51		
mail	*17*	*35*	52		
l2d4	*30*	*22*	52		
o111	*20*	*33*	53		
o119	*41*	*13*	54		
l2d4	*32*	*22*	54		
stor	*44*	*12*	56		
o111	*25*	*33*	58		
r123	*50*	*0*	50		
l2d2	*32*	*19*	51		
mail	*17*	*35*	52		
……					

Admissibility of A^*

If there is a solution, A^* always finds an optimal solution —the first path to a goal selected— if

- the branching factor is finite
- arc costs are bounded above zero (there is some $\epsilon > 0$ such that all of the arc costs are greater than ϵ), and
- $h(n)$ is an underestimate of the length of the shortest path from n to a goal node.

Why is A^* admissible?

- The f-value for any node on an optimal solution path is less than or equal to the f-value of an optimal solution. (As h is an underestimate).
- The search never selects a node with a higher f-value than the f-value of an optimal solution. A non-optimal solution has a higher f value — so it will never be selected.
- It halts, as the minimum g-value on the frontier keeps increasing, and will eventually exceed any finite number.

Exercise B : To be discussed in Lecture 4.4

1. Why is the underestimation property of *h(n)* necessary for admissibility?
   What unwanted behavior could happen, if *h(n)* overestimates the cost to reach a goal?
2. Proof the admissibility property of A*.