Solving problems by searching
Informed search algorithms

LECTURE 2 - part I

Heuristics • Best-first search • A* search
Greedy best-first search • Iterative Deepening Search
Iterative Deepening A* Search

Informed (Heuristics) Searches

- Uninformed search algorithms—algorithms that are given no information about the problem other than its definition.
    - some of these algorithms can solve any solvable problem, but none of them can do it efficiently

- Informed search strategy
    - one that uses problem-specific knowledge beyond the definition of the problem itself
    - can find solutions more efficiently than uninformed strategy

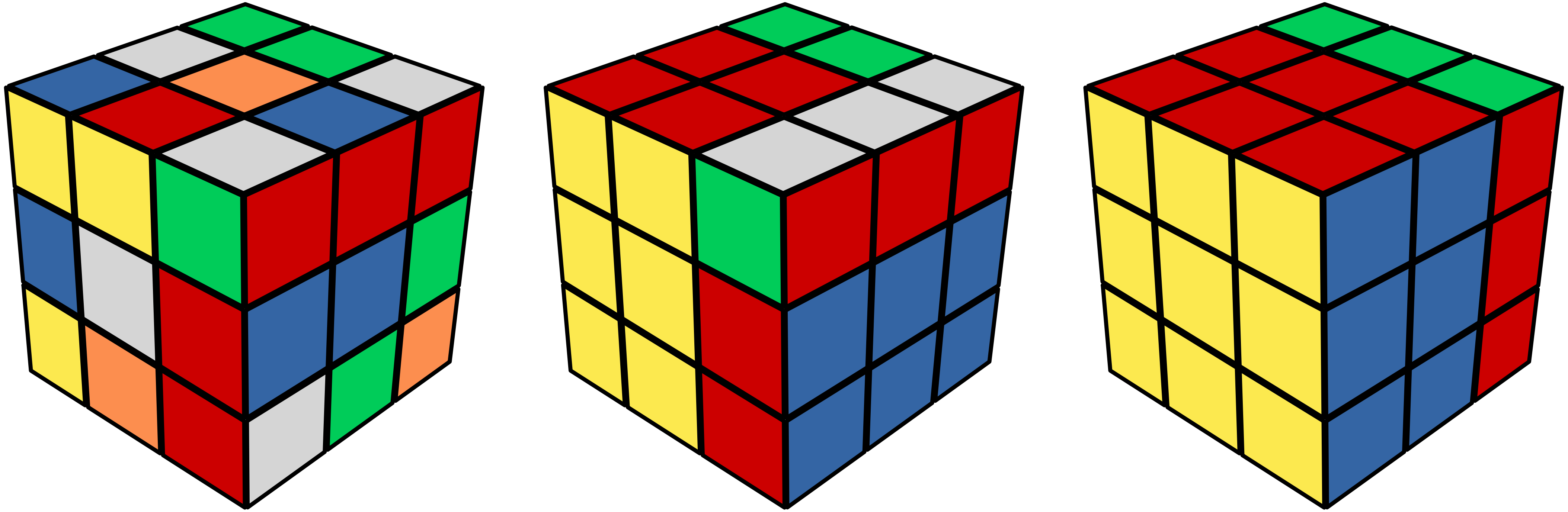
- Informed search algorithms, can do quite well given some guidance on where to look for solutions.

- All implemented using a priority queue to store frontier nodes

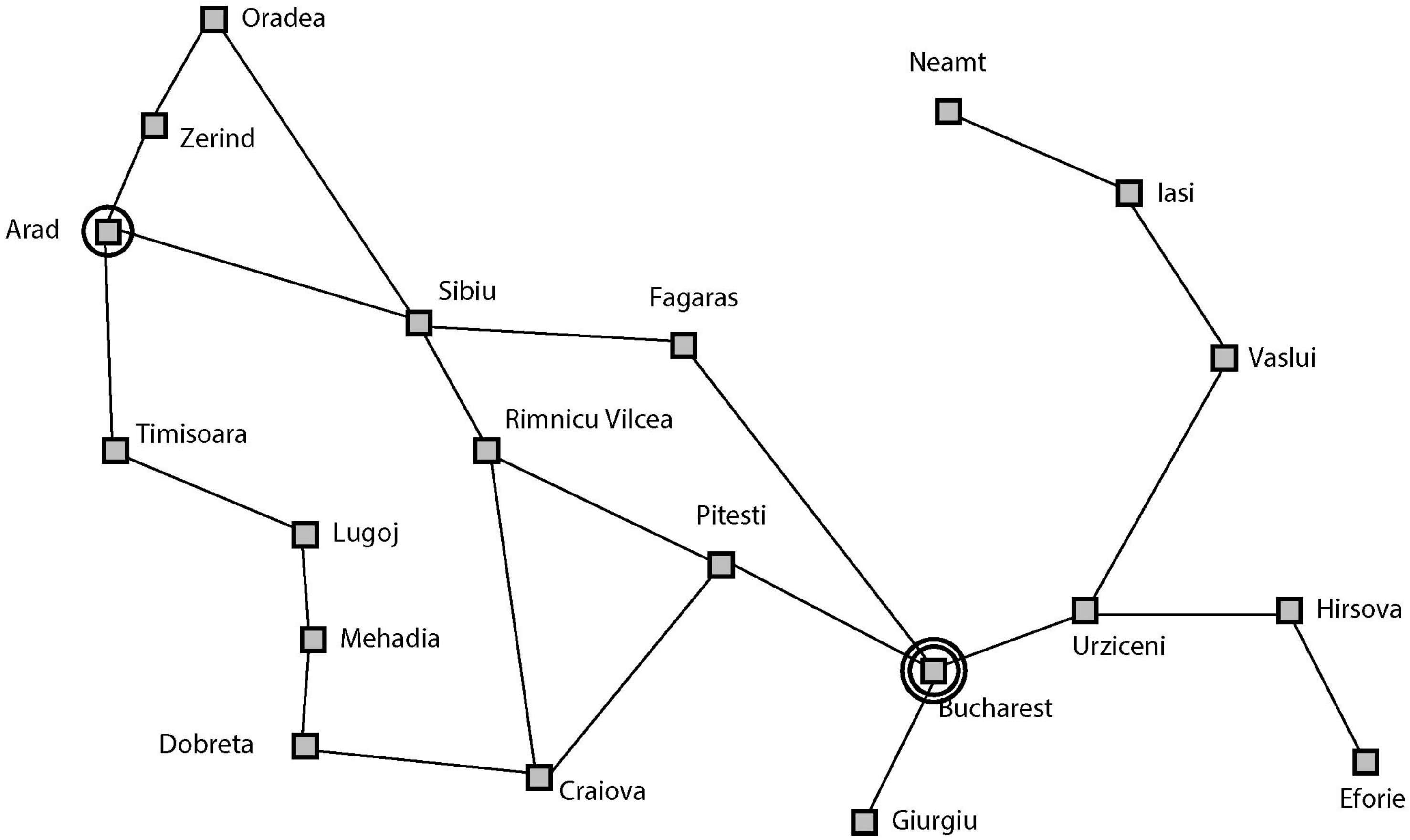
Heuristics

- Heuristics are “rules of thumb”
- Heuristics are criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal. “Heuristics” (Pearl 1984)
- Can make use of heuristics in deciding which is the most “promising” path to take during search
- In search, heuristic should be an underestimate of actual cost to get from current node to any goal — an admissible heuristic
- Denoted $h(n)$; $\underline{h(n)=0}$ when ever n is a goal node

Heuristic - intuition



Heuristics — Example



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Heuristic Search

- Idea: don’t ignore the goal when selecting paths.
- Often there is extra knowledge that can be used to guide the search: heuristics.
- h(n) is an estimate of the cost of the shortest path from node n to a goal node.
- h(n) needs to be efficient to compute.
- h can be extended to paths: $h(\langle n_0,...,n_k \rangle) = h(n_k)$.
- h(n) is an underestimate if there is no path from n to a goal with cost less than h(n).
- An admissible heuristic is a nonnegative heuristic function that is an underestimate of the actual cost of a path to a goal.

Example Heuristic Functions

- If the nodes are points on a Euclidean plane and the cost is the distance, h(n) can be the straight-line distance from n to the closest goal.

- If the nodes are locations and cost is time, we can use the distance to a goal divided by the maximum speed.

- If the goal is to collect all of the coins and not run out of fuel, the cost is an estimate of how many steps it will take to collect the rest of the coins, refuel when necessary, and return to goal position.

- A heuristic function can be found by solving a simpler (less constrained) version of the problem.

Search Strategies

General Search algorithm:

- add initial state to queue
- repeat:
  - take node from front of queue
  - test if it is a goal state; if so, terminate
  - "expand" it, i.e. generate successor nodes and add them to the queue

Search strategies are distinguished by the order in which new nodes are added to the queue of nodes awaiting expansion.

Search Strategies

- BFS and DFS treat all new nodes the same way:
    - BFS add all new nodes to the back of the queue
    - DFS add all new nodes to the front of the queue

- Best First Search uses an evaluation function f to order the nodes in the queue;
    - Similar to UCS $f(n)$ = cost $g(n)$ of path from root to node n

- Informed or Heuristic search strategies incorporate into f () an estimate of distance to goal
    - Greedy Search $f(n)$ = estimate $h(n)$ of cost from node n to goal
    - A* Search $f(n) = g(n) + h(n)$

Heuristic Function

There is a whole family of Best First Search algorithms with different evaluation functions ***f***. A key component of these algorithms is a heuristic function:

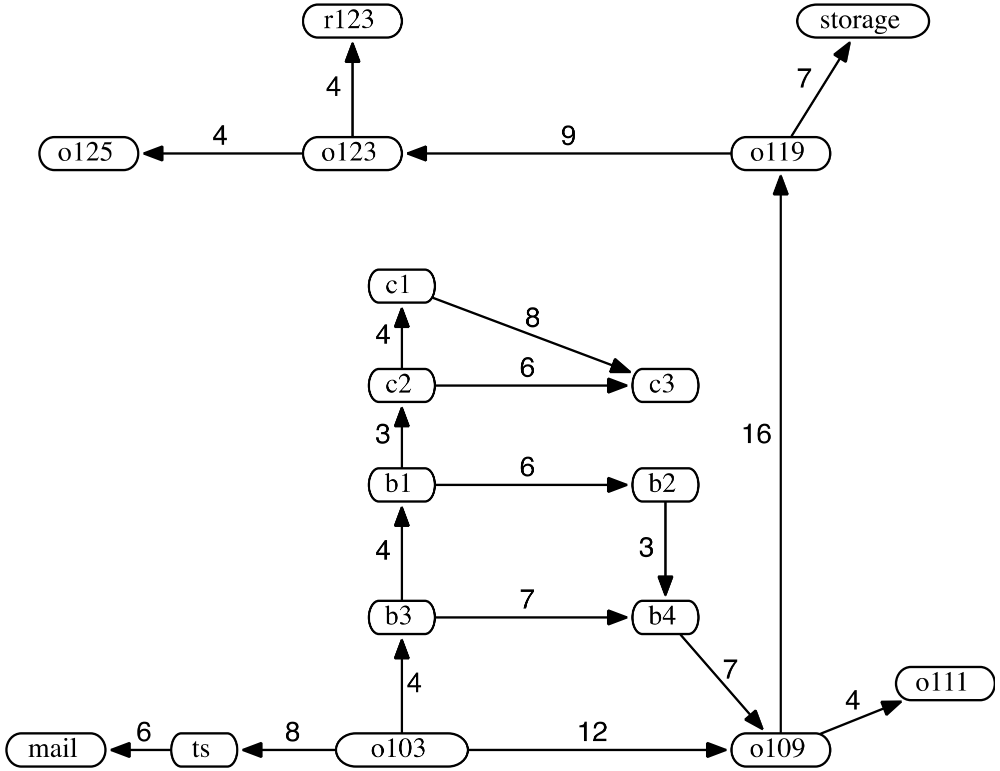
- Heuristic function h: {Set of nodes} $\rightarrow$ **R** :
    - $h(n)$ = estimated cost of the cheapest path from current node n to *goal* node.
        - in the area of search, heuristic functions are problem specific functions that provide an estimate of solution cost.
        - nonnegative, with one constraint: if n is a goal node, then $h(n) = 0$.

Delivery Robot - Heuristic Function

The ***h*** function can be extended to be applicable to paths by making the heuristic value of a path equal to the heuristic value of the node at the end of the path.

$$h(\langle n_o, \ldots, n_k \rangle) = h(n_k)$$

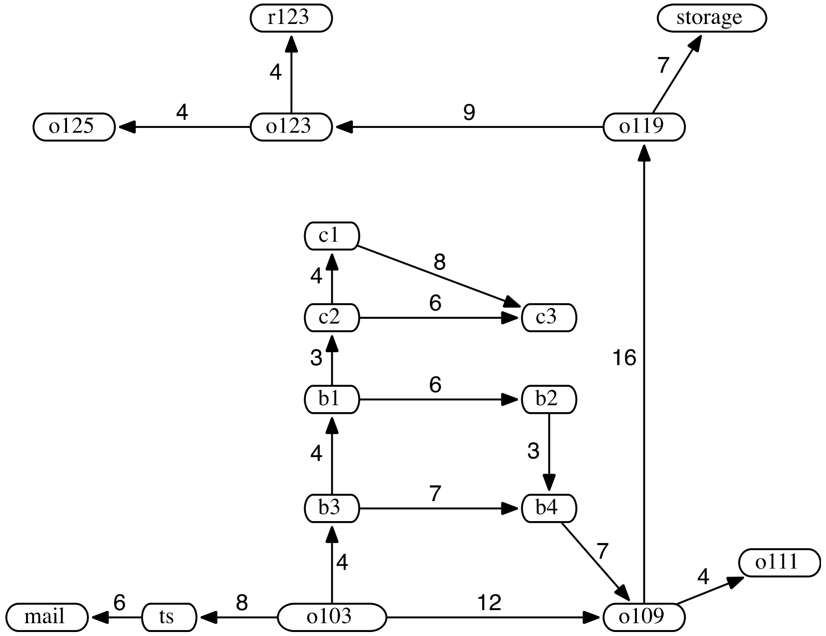
$$\begin{array}{rclrclrcl} h(mail) & = & 26 & h(ts) & = & 23 & h(o103) & = & 21 \\ h(o109) & = & 24 & h(o111) & = & 27 & h(o119) & = & 11 \\ h(o123) & = & 4 & h(o125) & = & 6 & h(r123) & = & 0 \\ h(b1) & = & 13 & h(b2) & = & 15 & h(b3) & = & 17 \\ h(b4) & = & 18 & h(c1) & = & 6 & h(c2) & = & 10 \\ h(c3) & = & 12 & h(storage) & = & 12 & & & \end{array}$$



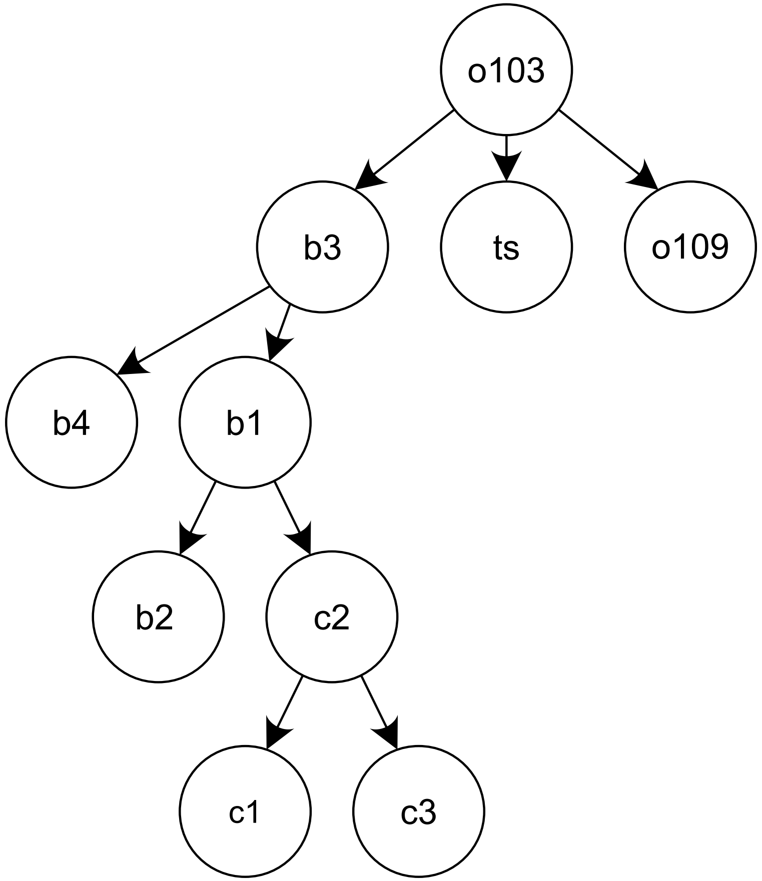
Greedy best-first search

- Idea: select the path whose end is closest to a goal according to the heuristic function.
- Greedy Best-First Search - Best-First Search selects the next node for expansion using the heuristic function for its evaluation function, i.e. $f(n) = h(n)$
    - $h(n)=0 \quad \Rightarrow n$ is a goal state
    - i.e. greedy search minimises the estimated cost to the goal; it expands whichever node n is estimated to be closest to the goal.
- It treats the frontier as a priority queue ordered by h.
- Greedy: tries to “bite off” as big a chunk of the solution as possible, without worrying about long-term consequences.

Examples of Greedy Best-First Search



$$
\begin{array}{lll}
h(mail) = 26 & h(ts) = 23 & h(o103) = 21 \\
h(o109) = 24 & h(o111) = 27 & h(o119) = 11 \\
h(o123) = 4 & h(o125) = 6 & h(r123) = 0 \\
h(b1) = 13 & h(b2) = 15 & h(b3) = 17 \\
h(b4) = 18 & h(c1) = 6 & h(c2) = 10 \\
h(c3) = 12 & h(storage) = 12 &
\end{array}
$$



Properties of Greedy Best-First Search

Algorithm	Completeness	Admissibility	Space	Time
Greedy Best-First Search	guaranteed, if b is finite and with repeated-state checking	not guaranteed	$O(b^m)$	$O(b^m)$

- Complete: generally no! can get stuck in loops, e.g., but complete in finite space with repeated-state checking
- Time: $O(b^m)$, where m is the maximum depth in search space.
- Space: $O(b^m)$ (retains all nodes in memory)
- Optimal: No!
- Therefore Greedy Search has the same deficits as Depth-First Search. However, a good heuristic can reduce time and memory costs substantially.

A* Search

估计

- Idea: Use both cost of path generated and estimate to goal to order nodes on the frontier
- $g(n)$ = cost of path from start to n; $h(n)$ = estimate from n to goal
- Order priority queue using function $f(n) = g(n) + h(n)$
- $f(n)$ is the estimated cost of the cheapest solution extending this path

A^* Search

- A_* Search uses evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost from initial node to node n
  - $h(n)$ = estimated cost of cheapest path from n to goal
  - $f(n)$=estimated total cost of cheapest solution through node n

- Combines uniform-cost search and greedy search

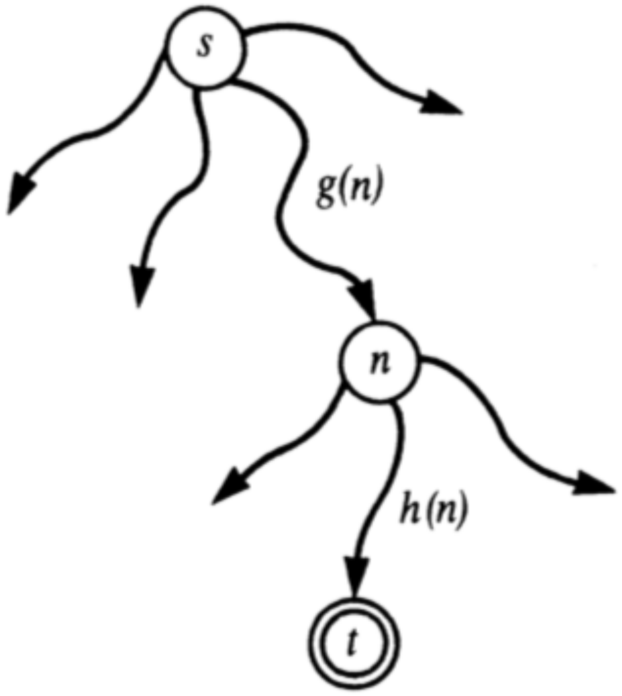
- Greedy Search minimizes $h(n)$
  - efficient but not optimal or complete
- Uniform Cost Search minimizes $g(n)$
  - optimal and complete but not efficient

Heuristic function

Heuristic estimate *f(n) of*
the cost of the cheapest paths from ***s*** to ***t*** via ***n****:*
$f(n) = g(n) + h(n)$

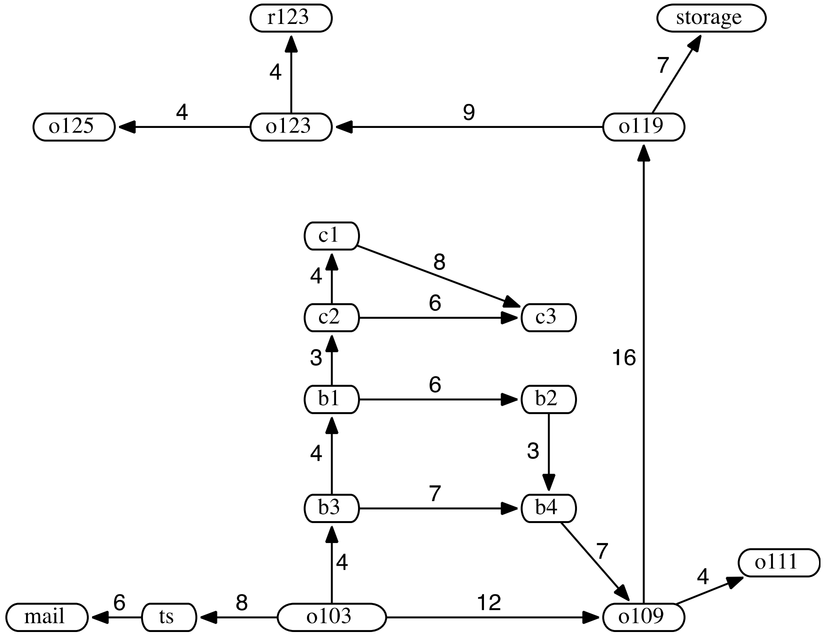
g(n) is an estimate of the cost of an optimal path from s to n

h(n) is an estimate of the cost of an optimal path from n to t.

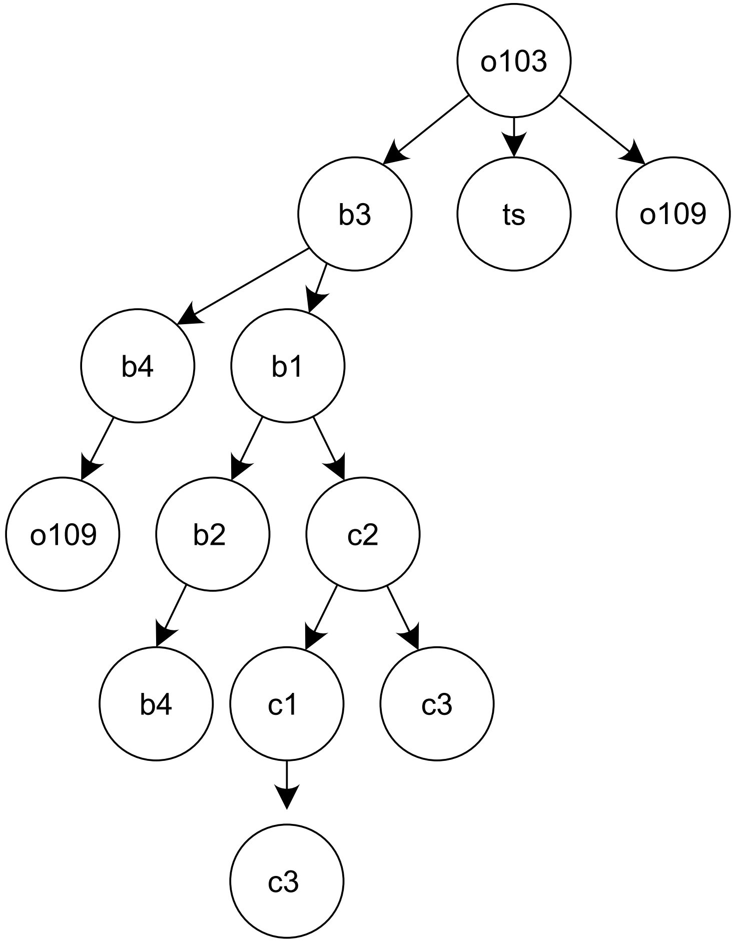


Demonstration - A$*$ Search

A∗ Search - the Delivery Robot



$$
\begin{aligned}
h(mail) &= 26 & h(ts) &= 23 & h(o103) &= 21\\
h(o109) &= 24 & h(o111) &= 27 & h(o119) &= 11\\
h(o123) &= 4 & h(o125) &= 6 & h(r123) &= 0\\
h(b1) &= 13 & h(b2) &= 15 & h(b3) &= 17\\
h(b4) &= 18 & h(c1) &= 6 & h(c2) &= 10\\
h(c3) &= 12 & h(storage) &= 12 & &
\end{aligned}
$$

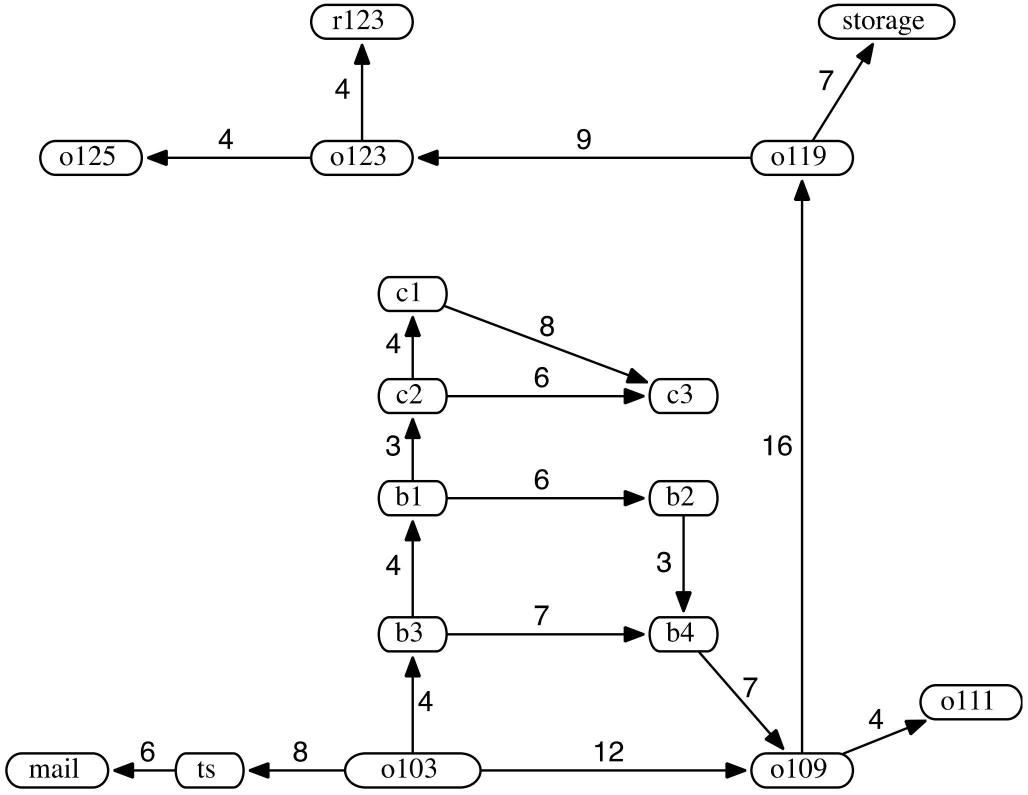


A* Search - the Delivery Robot

$$\begin{array}{rclrclrcl} h(mail) &=& 26 & h(ts) &=& 23 & h(o103) &=& 21 \\ h(o109) &=& 24 & h(o111) &=& 27 & h(o119) &=& 11 \\ h(o123) &=& 4 & h(o125) &=& 6 & h(r123) &=& 0 \\ h(b1) &=& 13 & h(b2) &=& 15 & h(b3) &=& 17 \\ h(b4) &=& 18 & h(c1) &=& 6 & h(c2) &=& 10 \\ h(c3) &=& 12 & h(storage) &=& 12 & & & \end{array}$$

$$f(\langle o103, b3\rangle) = cost(\langle o103, b3\rangle) + h(b3) = 4 + 17 = 21.$$

A lowest-cost path to the goal is eventually found. The algorithm is forced to try many different paths, because several of them temporarily seemed to have the lowest cost. It still does better than either lowest-cost-first search or greedy best-first search.



A$_*$ Search

- A$_*$ Search minimises $f(n) = g(n) + h(n)$
  - idea: preserve efficiency of Greedy Search but avoid expanding paths that are already expensive

- Question: is A* Search optimal and complete?
- Answer: Yes! provided $h()$ is **admissible** in the sense that it never overestimates the cost to reach the goal or **consistent**.

Conditions for optimality: Admissibility and consistency

- The first condition we require for optimality is that h(n) be an admissible heuristic.

- An admissible heuristic is one that *never overestimates* the cost to reach the goal.
  - Because g(n) is the actual cost to reach n along the current path, and
  - f (n) = g(n) + h(n), we have as an immediate consequence that f(n) never overestimates the true cost of a solution along the current path through n.

- Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.

A^* Search – conditions for optimality

- Heuristic ***h*** is called **admissible** if
- $\forall n\ h(n) \leq h^*(n)$ where $h^*(n)$ is true cost from n to the goal

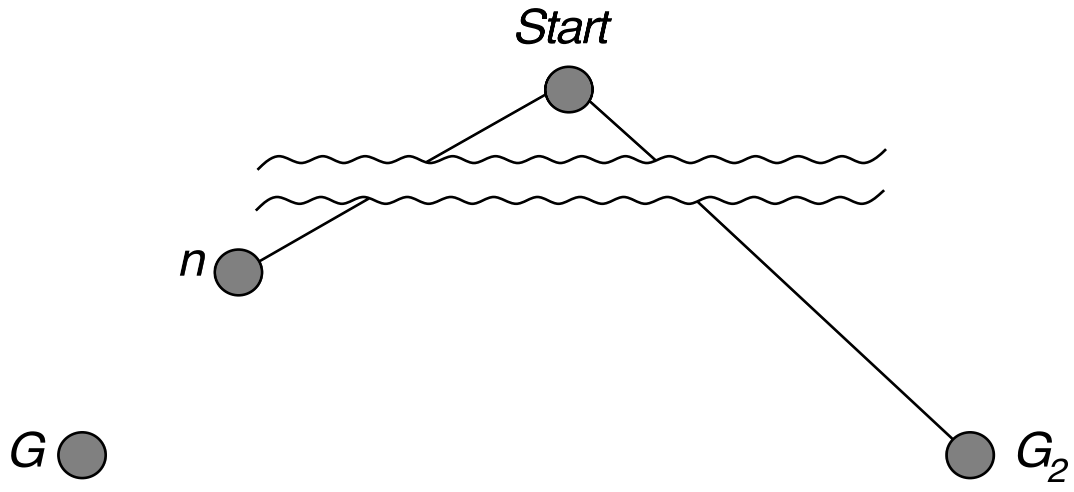
- If ***h*** is **admissible** then ***f (n)*** never overestimates the actual cost of the best solution through *n.*

- Example: $\boldsymbol{h}_{\mathbf{SLD}}$ is admissible for delivery robot problem because the shortest path between any two points is a line.

- Theorem: A^* Search finds optimal solution if ***h()*** is admissible.

Optimality of A* Search

Suppose a suboptimal goal node *G*2 has been generated and is in the queue. Let *n* be the last unexpanded node on a shortest path to an optimal goal node *G*.



$f(G_2) = g(G2)$ since $h(G_2) = 0$

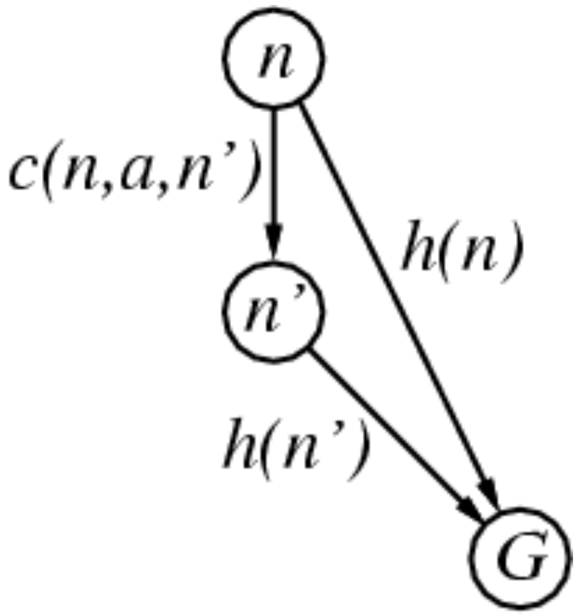
$f(G_2) > g(G)$ since G_2 is suboptimal
$f(G_2) \geq f(n)$ since h is admissible

Optimality of A* Search

- Since $f(G2) > f(n)$, A* will never select $G2$ for expansion.
- Note: suboptimal goal node $G2$ may be generated, but it will never be expanded.
- In other words, even after a goal node has been generated, A* will keep searching so long as there is a possibility of finding a shorter solution.
- Once a goal node is selected for expansion, we know it must be optimal, so we can terminate the search.

Consistent heuristics

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,
  $h(n) \leq c(n,a,n') + h(n')$



If *h* is consistent, we have

$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

Theorem:
If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal

Optimality of A* Search

Algorithm	Completeness	Admissibility	Space	Time
A* Search	guaranteed	guaranteed if heuristic is admissible/consistent	$O(b^d)$	$O(b^d)$

- Complete: Yes, unless there are infinitely many nodes with f ≤ cost of solution
- Time: Exponential
- Space: Keeps all nodes is memory
- Optimal: Yes (assuming *h*() is admissible).

Iterative Deepening A* Search

- Iterative Deepening A* is a low-memory variant of A* which performs a series of depth-first searches but cuts off each search when the sum $f() = g() + h()$ exceeds some pre-defined threshold.

- The threshold is steadily increased with each successive search. It starts at the estimate of the cost at the initial state, and increases for each iteration of the algorithm. At each iteration, the threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold.

- IDA* is asymptotically as efficient as A* for domains where the number of states grows exponentially.

Optimality of Iterative Deepening A* Search

Algorithm	Completeness	Admissibility	Space	Time
Iterative Deepening A* Search	guaranteed	guaranteed if heuristic is admissible/consistent	O(d)	$O(b^d)$

- Complete: Yes, unless there are infinitely many nodes with f ≤ cost of solution
- Time: Exponential
- Space: it need only store a stack of nodes which represents the branch of the tree it is expanding
- Optimal: Yes (assuming *h*() is admissible).

How to Find Heuristic Functions ?

- Admissible heuristics can often be derived from the exact solution cost of a simplified or “relaxed” version of the problem. (i.e. with some of the constraints weakened or removed)

- Dominance: if $h2(n) \geq h1(n)$ for all n (both admissible) then $h2$ dominates $h1$ and is better for search. So the aim is to make the heuristic $h()$ as large as possible, but without exceeding $h*()$.

Composite Heuristic Functions

Let $h_1, h_2, \ldots, h_m$ be admissible heuristics for a given task.

The composite heuristic is defined as:

$$h(n) = max(h_1(n), h_2(n), \ldots, h_m(n))$$

Where h is admissible and h dominates $h_1, h_2, \ldots, h_m$

Summary of Informed Search

- Heuristics can be applied to reduce search cost.
- Greedy Search tries to minimise cost from current node *n* to the goal.
- A^* combines the advantages of Uniform-Cost Search and Greedy Search
- A^* is complete, optimal and optimally efficient among all optimal search algorithms.
- Memory usage is still a concern for A^*. A^* is a low-memory variant.

- Informed search makes use of problem-specific knowledge to guide progress of search

- This can lead to a significant improvement in performance

- Much research has gone into admissible heuristics
    - Even on the automatic generation of admissible heuristics