Module 2

Problem Solving using Search- (Single agent search)

Lesson 5

Informed Search Strategies-I

3.1 Introduction

We have outlined the different types of search strategies. In the earlier chapter we have looked at different blind search strategies. Uninformed search methods lack problem-specific knowledge. Such methods are prohibitively inefficient in many cases. Using problem-specific knowledge can dramatically improve the search speed. In this chapter we will study some informed search algorithms that use problem specific heuristics.

Review of different Search Strategies

1. Blind Search
    a) Depth first search
    b) Breadth first search
    c) Iterative deepening search
    d) Bidirectional search
2. Informed Search

3.1.1 Graph Search Algorithm

We begin by outlining the general graph search algorithm below.

```
                Graph search algorithm
Let fringe be a list containing the initial state
Let closed be initially empty
Loop
    if fringe is empty return failure
    Node ← remove-first (fringe)
                if Node is a goal
                   then return the path from initial state to Node
       else put Node in closed
                      generate all successors of Node S
                      for all nodes m in S
                if m is not in fringe or closed
                            merge m into fringe
End Loop
```

3.1.2 Review of Uniform-Cost Search (UCS)

We will now review a variation of breadth first search we considered before, namely Uniform cost search.

To review, in uniform cost search we enqueue nodes by path cost.

Let g(n) = cost of the path from the start node to the current node n.

The algorithm sorts nodes by increasing value of g, and expands the lowest cost node of the fringe.

Properties of Uniform Cost Search

- Complete
- Optimal/Admissible
- Exponential time and space complexity, O(bd)

The UCS algorithm uses the value of g(n) to select the order of node expansion. We will now introduce informed search or heuristic search that uses problem specific heuristic information. The heuristic will be used to select the order of node expansion.

3.1.3 Informed Search

We have seen that uninformed search methods that systematically explore the state space and find the goal. They are inefficient in most cases. Informed search methods use problem specific knowledge, and may be more efficient. At the heart of such algorithms there is the concept of a heuristic function.

3.1.3.1 Heuristics

Heuristic means "rule of thumb". To quote Judea Pearl, "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". In heuristic search or informed search, heuristics are used to identify the most promising search path.

Example of Heuristic Function

A heuristic function at a node n is an estimate of the optimum cost from the current node to a goal. It is denoted by *h(n)*.
h(n) = estimated cost of the cheapest path from node n to a goal node

Example 1: We want a path from Kolkata to Guwahati
Heuristic for Guwahati may be straight-line distance between Kolkata and Guwahati
h(Kolkata) = euclideanDistance(Kolkata, Guwahati)

Example 2: 8-puzzle: Misplaced Tiles Heuristics is the number of tiles out of place.

2	**8**	**3**
1	**6**	**4**
	7	**5**

Initial State

1	**2**	**3**
8		**4**
7	**6**	**5**

Goal state

Figure 1: 8 puzzle

The first picture shows the current state *n*, and the second picture the goal state.
h(n) = 5
because the tiles 2, 8, 1, 6 and 7 are out of place.

Manhattan Distance Heuristic: Another heuristic for 8-puzzle is the Manhattan distance heuristic. This heuristic sums the distance that the tiles are out of place. The distance of a tile is measured by the sum of the differences in the x-positions and the y-positions.
For the above example, using the Manhattan distance heuristic,
h(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2 = 6

We will now study a heuristic search algorithm best-first search.

3.2 Best First Search

Uniform Cost Search is a special case of the best first search algorithm. The algorithm maintains a priority queue of nodes to be explored. A cost function f(n) is applied to each node. The nodes are put in OPEN in the order of their f values. Nodes with smaller f(n) values are expanded earlier. The generic best first search algorithm is outlined below.

Best First Search
Let *fringe* be a priority queue containing the initial state Loop if *fringe* is empty return failure Node ← remove-first (fringe) if Node is a goal then return the path from initial state to Node else generate all successors of Node, and put the newly generated nodes into fringe according to their f values End Loop

We will now consider different ways of defining the function f. This leads to different search algorithms.

3.2.1 Greedy Search

In greedy search, the idea is to expand the node with the smallest estimated cost to reach the goal.
We use a heuristic function
f(n) = h(n)
h(n) estimates the distance remaining to a goal.
Greedy algorithms often perform very well. They tend to find good solutions quickly, although not always optimal ones.

The resulting algorithm is not optimal. The algorithm is also incomplete, and it may fail to find a solution even if one exists. This can be seen by running greedy search on the following example. A good heuristic for the route-finding problem would be straight-line distance to the goal.

Figure 2 is an example of a route finding problem. S is the starting state, G is the goal state.

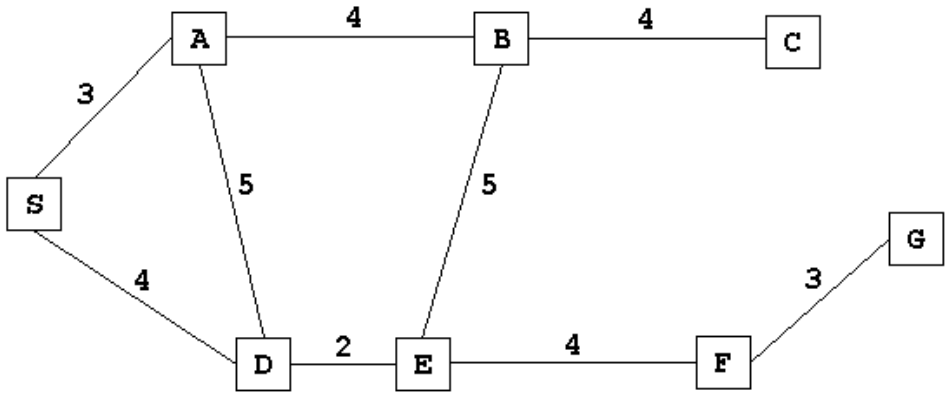


Figure 2

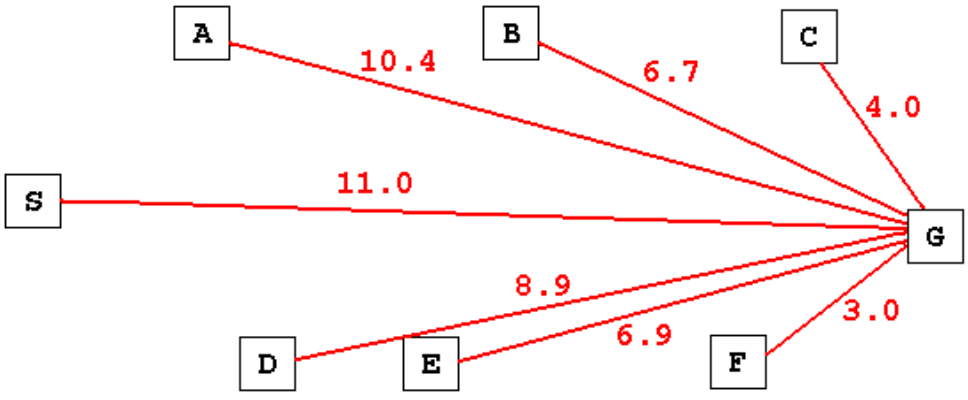
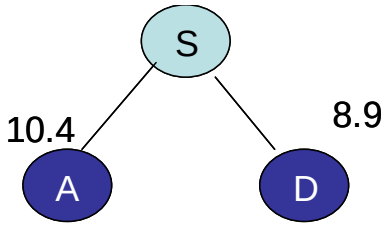


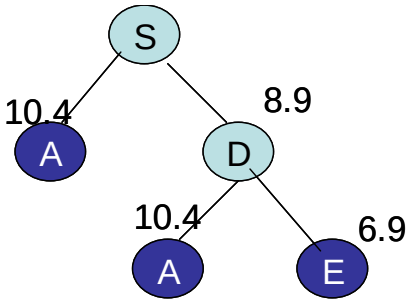
Figure 3

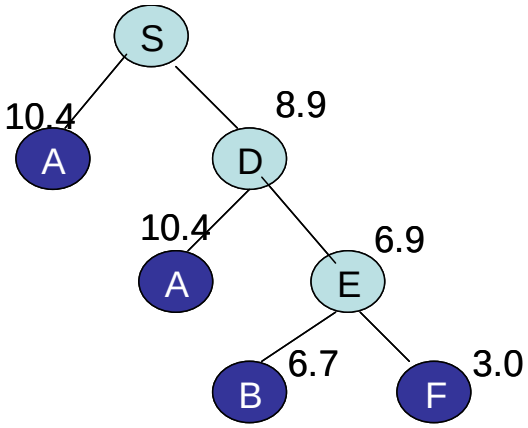
Let us run the greedy search algorithm for the graph given in Figure 2. The straight line distance heuristic estimates for the nodes are shown in Figure 3.

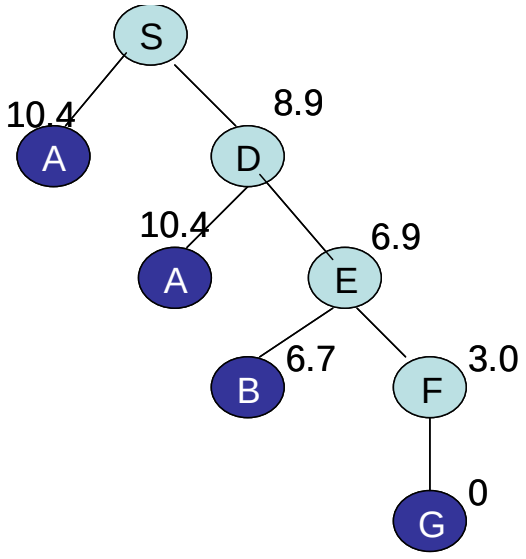
Step 1: S is expanded. Its children are A and D.



Step 2: D has smaller cost and is expanded next.







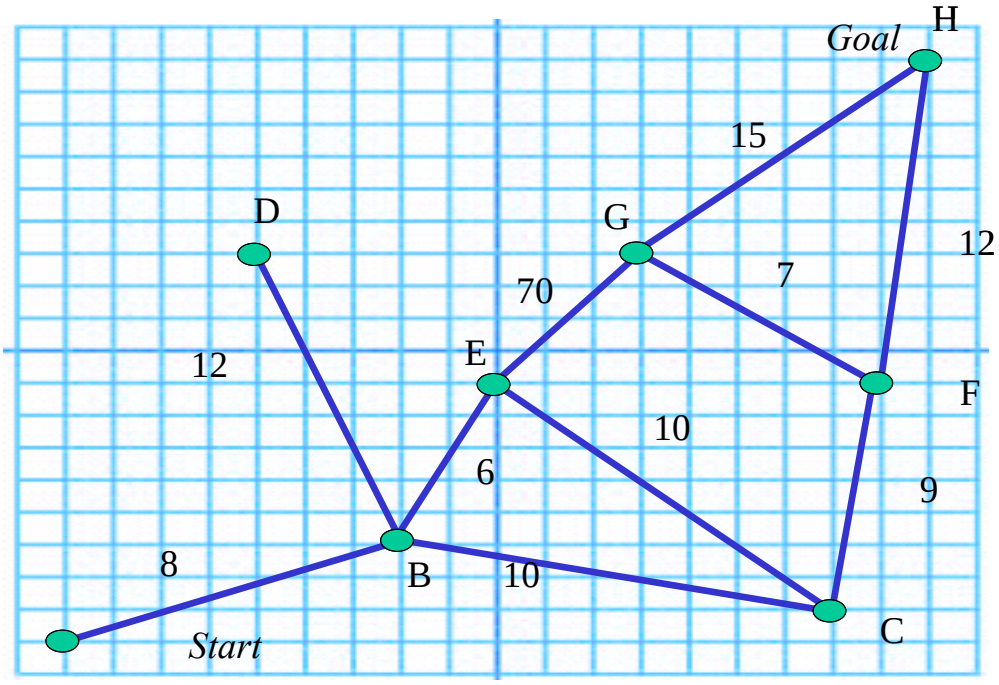


Figure 4

Greedy Best-First Search illustrated

We will run greedy best first search on the problem in Figure 2. We use the straight line heuristic. *h(n)* is taken to be the straight line distance from *n* to the goal position.

The nodes will be expanded in the following order:

A
B
E
G
H

The path obtained is A-B-E-G-H and its cost is 99
Clearly this is not an optimum path. The path A-B-C-F-H has a cost of 39.

3.2.2 A* Search

We will next consider the famous A* algorithm. This algorithm was given by Hart, Nilsson & Rafael in 1968.

A* is a best first search algorithm with
$f(n) = g(n) + h(n)$
where
$g(n)$ = sum of edge costs from start to n
$h(n)$ = estimate of lowest cost path from n to goal

$f(n)$ = actual distance so far + estimated distance remaining

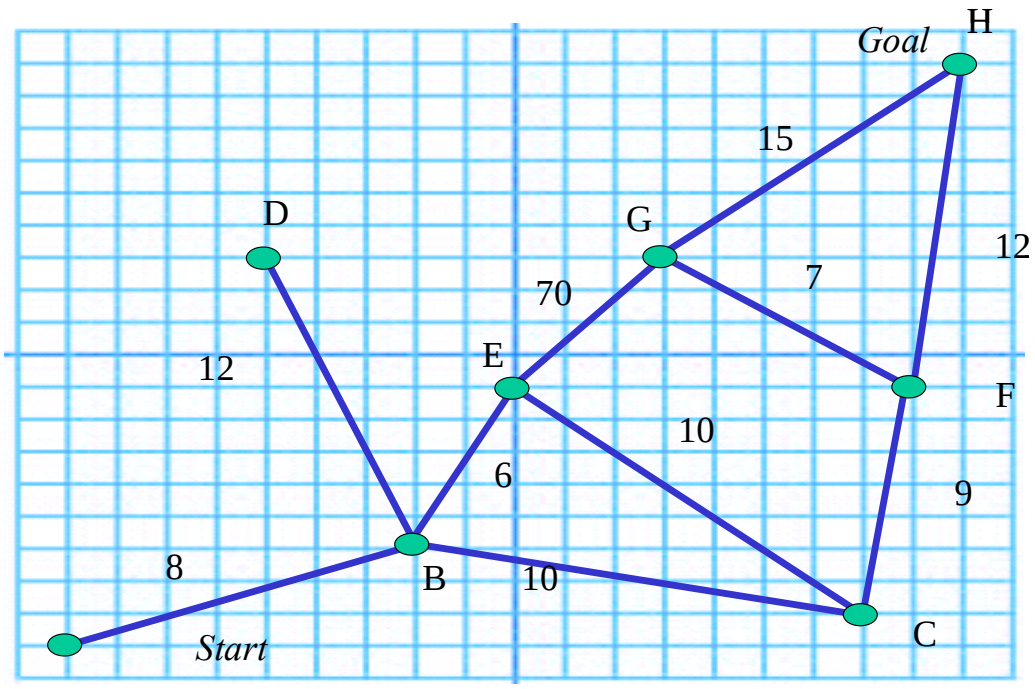
$h(n)$ is said to be admissible if it underestimates the cost of any solution that can be reached from n. If $C^*(n)$ is the cost of the cheapest solution path from n to a goal node, and if h is admissible,
$h(n) <= C^*(n)$.
We can prove that if $h(n)$ is admissible, then the search will find an optimal solution.

The algorithm A* is outlined below:

Algorithm A*
OPEN = nodes on frontier. CLOSED = expanded nodes. OPEN = {<s, nil>} while OPEN is not empty remove from OPEN the node <*n,p*> with minimum *f(n)* place <*n,p*> on CLOSED if *n* is a goal node, return success (path *p*) for each edge connecting *n* & *m* with cost *c* if <*m, q*> is on CLOSED and *{p|e}* is cheaper than *q* then remove *n* from CLOSED, put <*m,{p|e}*> on OPEN else if <*m,q*> is on OPEN and *{p|e}* is cheaper than *q* then replace *q* with *{p|e}* else if *m* is not on OPEN then put <*m,{p|e}*> on OPEN

return failure

3.2.1 A* illustrated



The heuristic function used is straight line distance. The order of nodes expanded, and the status of Fringe is shown in the following table.

Steps	Fringe	Node expanded	Comments
1	A		
2	B(26.6)	A	
3	E(27.5), C(35.1), D(35.2)	B	
4	C(35.1), D(35.2), ~~C(41.2)~~ G(92.5)	E	C is not inserted as there is another C with lower cost.
5	D(35.2), F(37), G(92.5)	C	
6	F(37), G(92.5)	D	
7	H(39), G(42.5)	F	G is replaced with a lower cost node
8	G(42.5)	H	Goal test successful.

The path returned is A-B-C-F-H.
The path cost is 39. This is an optimal path.

3.2.2 A* search: properties

The algorithm A* is admissible. This means that provided a solution exists, the first solution found by A* is an optimal solution. A* is admissible under the following conditions:

- In the state space graph
  - Every node has a finite number of successors
  - Every arc in the graph has a cost greater than some $\varepsilon > 0$
- Heuristic function: for every node *n*, $h(n) \leq h^*(n)$

A* is also complete under the above conditions.

A* is optimally efficient for a given heuristic – of the optimal search algorithms that expand search paths from the root node, it can be shown that no other optimal algorithm will expand fewer nodes and find a solution

However, the number of nodes searched still exponential in the worst case.

Unfortunately, estimates are usually not good enough for A* to avoid having to expand an exponential number of nodes to find the optimal solution. In addition, A* must keep all nodes it is considering in memory.

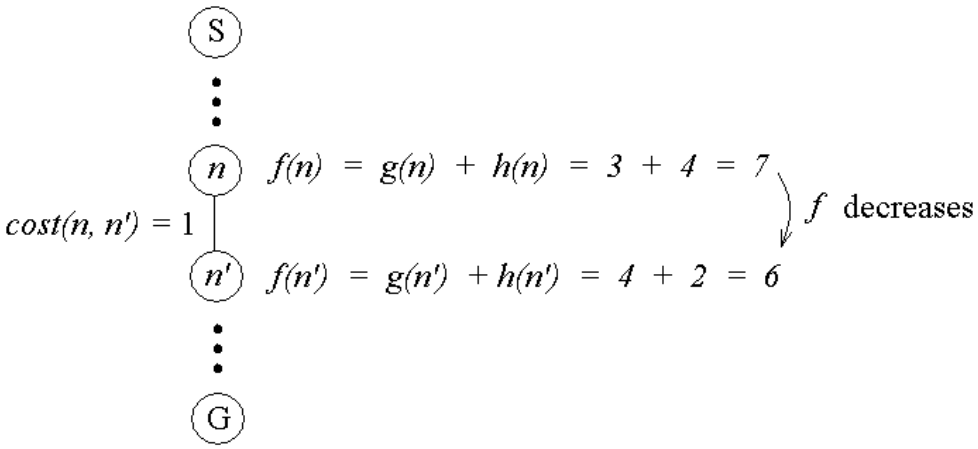
A* is still much more efficient than uninformed methods.

It is always better to use a heuristic function with higher values as long as it does not overestimate.

A heuristic is consistent if:

$h(n) <= cost(n, n') + h(n')$

For example, the heuristic shown below is inconsistent, because h(n) = 4, but $cost(n, n') + h(n') = 1 + 2 = 3$, which is less than 4. This makes the value of f decrease from node n to node n':



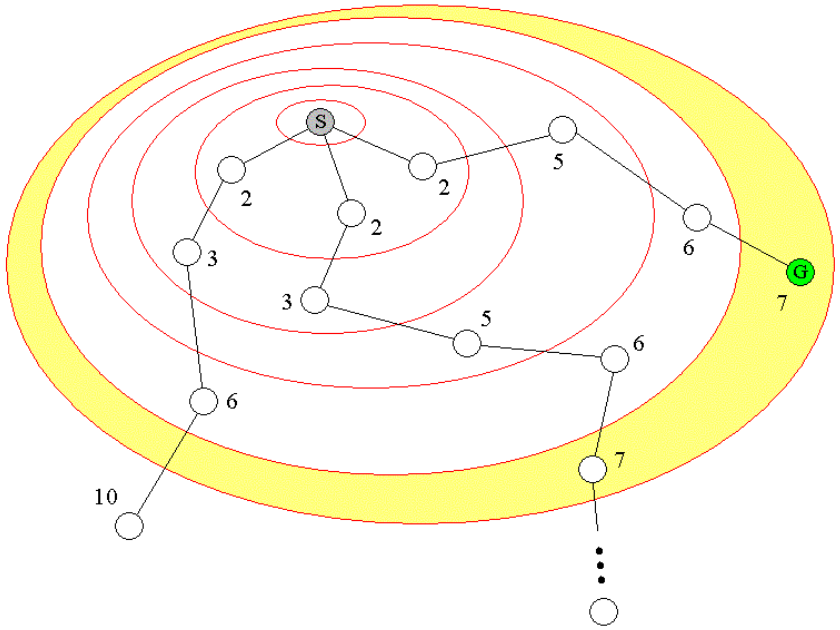
If a heuristic h is consistent, the f values along any path will be nondecreasing:

$f(n')$ = estimated distance from start to goal through n'
= actual distance from start to n + step cost from n to n' + estimated distance from n' to goal
= $g(n) + cost(n, n') + h(n')$
$\geq$ $g(n) + h(n)$ because $cost(n, n') + h(n') \geq h(n)$ by consistency
= $f(n)$

Therefore $f(n') \geq f(n)$, so f never decreases along a path.

If a heuristic h is inconsistent, we can tweak the f values so that they behave as if h were consistent, using the **pathmax equation**:
$f(n') = \max(f(n), g(n') + h(n'))$

This ensures that the f values never decrease along a path from the start to a goal.
Given nondecreasing values of f, we can think of A* as searching outward from the start node through successive **contours** of nodes, where all of the nodes in a contour have the same f value:



For any contour, A* examines all of the nodes in the contour before looking at any contours further out. If a solution exists, the goal node in the closest contour to the start node will be found first.

We will now prove the admissibility of A*.

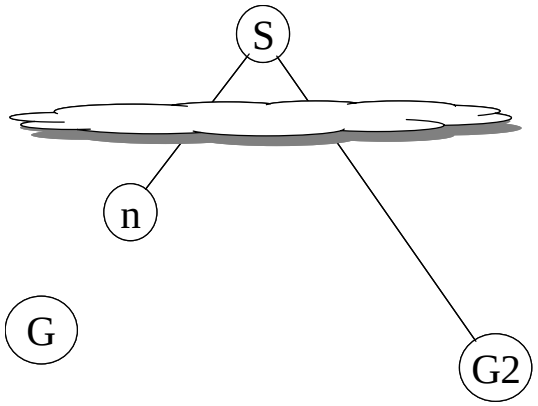
3.2.3 Proof of Admissibility of A*

We will show that A* is admissible if it uses a monotone heuristic.

A monotone heuristic is such that along any path the f-cost never decreases.
But if this property does not hold for a given heuristic function, we can make the f value monotone by making use of the following trick (m is a child of n)
f(m) = max (f(n), g(m) + h(m))

- Let G be an optimal goal state
- C* is the optimal path cost.
- G2 is a suboptimal goal state: g(G2) > C*

Suppose A* has selected G2 from OPEN for expansion.



Consider a node n on OPEN on an optimal path to G. Thus $C^* \geq f(n)$
Since *n* is not chosen for expansion over G2, $f(n) \geq f(G2)$
G2 is a goal state. $f(G2) = g(G2)$
Hence $C^* \geq g(G2)$.
This is a contradiction. Thus A* could not have selected G2 for expansion before reaching the goal by an optimal path.

3.2.4 Proof of Completeness of A*

Let G be an optimal goal state.
A* cannot reach a goal state only if there are infinitely many nodes where $f(n) \leq C^*$.
This can only happen if either happens:

- There is a node with infinite branching factor. The first condition takes care of this.
- There is a path with finite cost but infinitely many nodes. But we assumed that Every arc in the graph has a cost greater than some $\varepsilon > 0$. Thus if there are infinitely many nodes on a path $g(n) > f^*$, the cost of that path will be infinite.

Lemma: A* expands nodes in increasing order of their f values.

A* is thus **complete** and **optimal**, assuming an admissible and consistent heuristic function (or using the pathmax equation to simulate consistency).
A* is also **optimally efficient**, meaning that it expands only the minimal number of nodes needed to ensure optimality and completeness.

3.2.4 Performance Analysis of A*

Model the search space by a uniform b-ary tree with a unique start state s, and a goal state, g at a distance N from s.
The number of nodes expanded by A* is exponential in N unless the heuristic estimate is logarithmically accurate
$|h(n) - h^*(n)| \leq O (\log h^*(n))$

In practice most heuristics have proportional error.

It becomes often difficult to use A* as the OPEN queue grows very large.
A solution is to use algorithms that work with less memory.

3.2.5 Properties of Heuristics

Dominance:
h2 is said to dominate h1 iff $h2(n) \geq h1(n)$ for any node n.
A* will expand fewer nodes on average using h2 than h1.

Proof:

Every node for which $f(n) < C^*$ will be expanded. Thus n is expanded whenever

$$h(n) < f^* - g(n)$$

Since $h2(n) \geq h1(n)$ any node expanded using h2 will be expanded using h1.

3.2.6 Using multiple heuristics

Suppose you have identified a number of non-overestimating heuristics for a problem:
$h1(n), h2(n), \ldots, hk(n)$

Then
$max\ (h1(n), h2(n), \ldots, hk(n))$
is a more powerful non-overestimating heuristic. This follows from the property of dominance