MACHINE INTELLIGENCE

INFORMED SEARCH STARTEGIES

K.S.Srinivas
Department of Computer Science and Engineering

MACHINE INTELLIGENCE

INFORMED SEARCH STARTEGIES

Srinivas K S.
Associate Professor, Department of Computer Science

Informed Search

- Informed Search is the one that uses problem-specific knowledge beyond the definition of the problem itself.
- example-for a path-search in a graph with a geometrical representation-give preference to the neighbour which are closest to the target based on the Eucledian distance
- can find solutions more efficiently than an uninformed strategy.
- The general approach we consider is called best-first search
- best-first search is an algorithm in which the node is selected based on the evaluation function ***f*(n).**
- the evaluation function **f(n)** is a function that takes a node as a input and gives out a positive number similar to the g(n) function we used in UCS.
- so the node with the lowest evaluation is expanded first.

ARE WE DOING UNIFORM COST SEARCH AGAIN

The implementation of best-first graph search is identical to that for uniform-cost search. except for the use of f instead of g to order the priority queue. The choice of f determines the search strategy. .

Tip touch on Heuristic Function

Most best-first algorithms include as a component of f a heuristic function, denoted h(n):

h(n)= estimated cost of the cheapest path from the state at node n to a goal state

Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.

basically Heuristic is what you call as अनुमान

for now

Lets now convince ourselves for the below mentioned about HEURISTIC FUNCTION

it is a arbitrary, non negative, problem-specific functions, with one constraint: if n is a goal node, then *h*(n)=0.

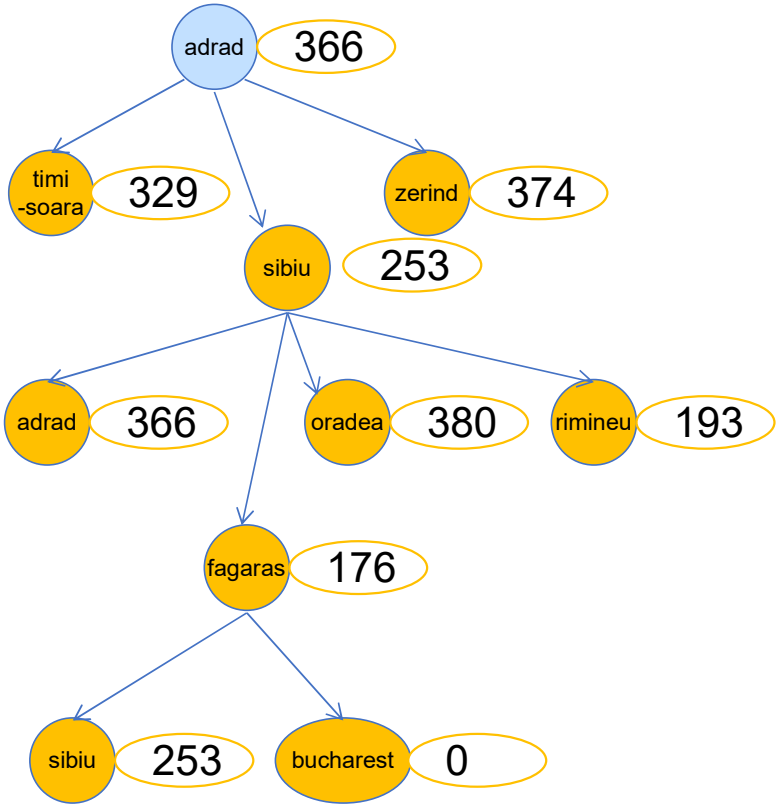
Greedy Best First Search

- Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, f(n)=h(n).
- Let us analyse the PSEUDO CODE

```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
   (s , h ) := Pop-least(PQ)
   foreach s' in succs(s)
   if s' is not already in PQ and s' never previously been visited
      Insert-PriQueue(PQ,s',h(s'))
```

- Let us now directly travel to Romania and solve some problem for better understanding

Adrad-->>>Bucharest



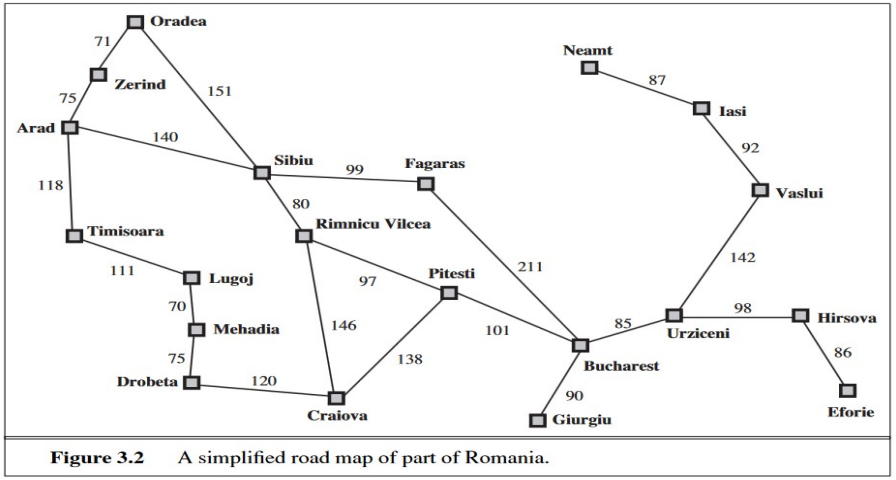


Figure 3.2 A simplified road map of part of Romania.

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Lets Analyse

- For this particular problem, greedy best-first search using hSLD finds a solution without ever expanding a node that is not on the solution path; hence, its search cost is minimal.
- It is not optimal, however: the path via Sibiu and Fagaras to Bucharest is 32 kilometers longer than the path through Rimnicu Vilcea and Pitesti. This shows why the algorithm is called “greedy”—at each step it tries to get as close to the goal as it can.
- Greedy best-first tree search is also incomplete even in a finite state space, much like depth-first search.

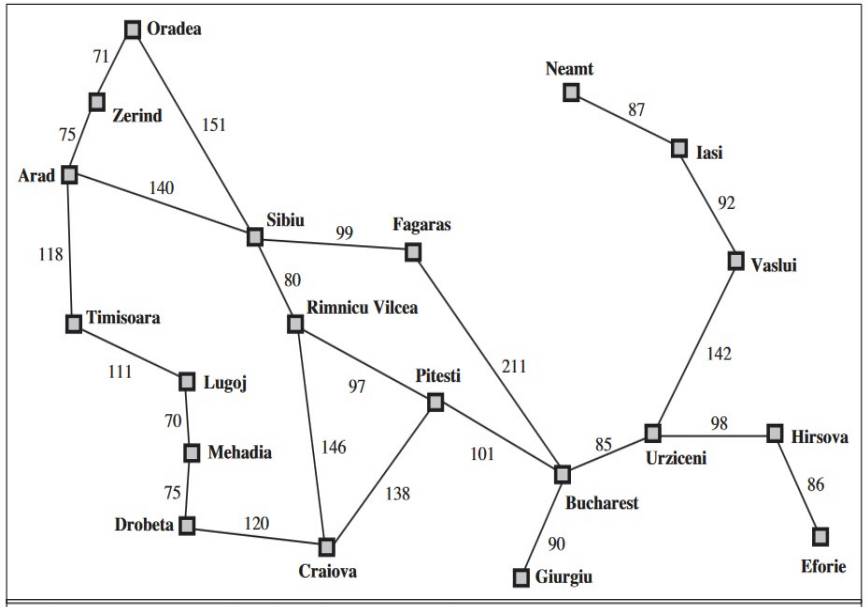


Figure 3.2 A simplified road map of part of Romania.

Lets Analyse

- Consider the problem of getting from Iasi to Fagaras
- The heuristic suggests that Neamt be expanded first because it is closest to Fagaras, but it is a dead end
- The solution is to go first to Vaslui—a step that is actually farther from the goal according to the heuristic—and then to continue to Urziceni, Bucharest, and Fagaras.
- The algorithm will never find this solution, however, because expanding Neamt puts Iasi back into the frontier, Iasi is closer to Fagaras than Vaslui is, and so Iasi will be expanded again, leading to an infinite loop.
- The worst-case time and space complexity for the tree version is O(bm),wherem is the maximum depth of the search space. With a good heuristic function, however, the complexity can be reduced substantially. The amount of the reduction depends on the particular problem and on the quality of the heuristic

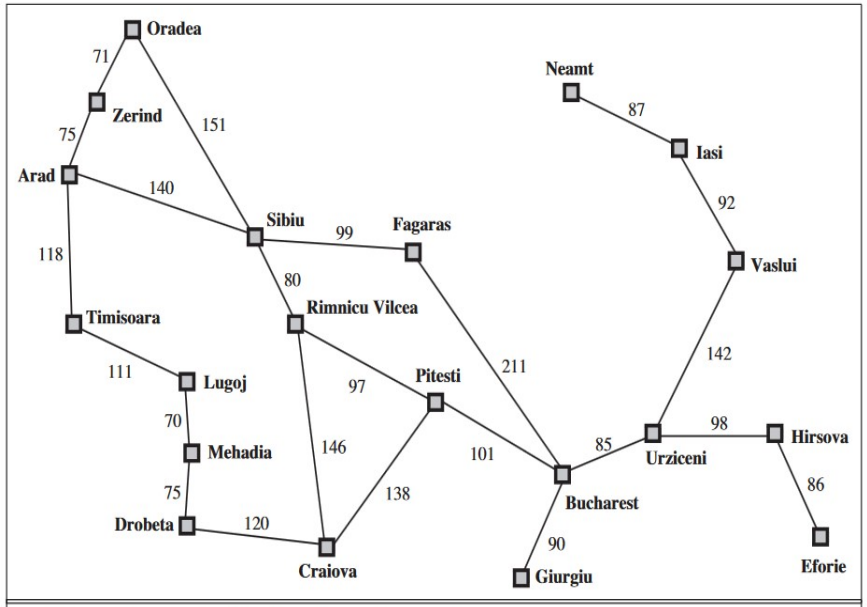
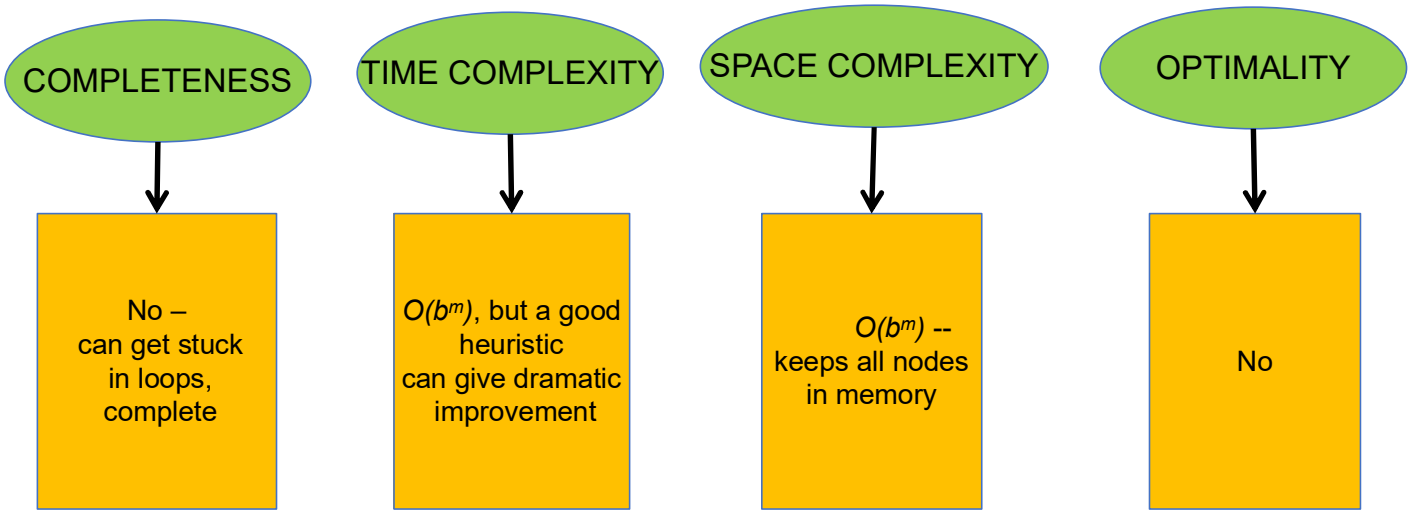


Figure 3.2 A simplified road map of part of Romania.

How good is it???



A* Search

- The most widely known form of best-first search is called A* search
- The most important point to notice is the function f(n)

$$f(n)=g(n)+h(n)$$

- Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have

f(n)= estimated cost of the cheapest solution through n.

- The algorithm is identical to **UNIFORM-COST-SEARCH** except that A* uses g + h instead of g.
- It turns out that this strategy is more than just reasonable: provided that the heuristic function h(n) **satisfies certain conditions**, A* search is both complete and optimal.

Conditions for optimality

Admissibility

- 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.

EXAMPLE:
straight-line distance h_{SLD} Straight-line distance is admissible because the shortest path between any two points is a straight line, so the straight line cannot be an overestimate.

Conditions for optimality

Consistency

- A heuristic h(n) is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n'

- consistent heuristic is also admissible

- Consistency is therefore a stricter requirement than admissibility,

h(n) <= c(n,a,n')+h(n')
this is form of triangle inequality, which stipulates that each side of a triangle cannot be longer than the sum of the other two sides. Here, the triangle is formed by n, n', and the goal Gn closest to n.

Pseudo-Code

- lets analyse the pseudo code and jump to solve a prob

The A* Algorithm

Reminder: $g(n)$ is cost of shortest known path to n

Reminder: $h(n)$ is a heuristic estimate of cost to a goal from n

- Priority queue PQ begins empty.
- V (= set of previously visited ($state$,f,$backpointer$)-triples) begins empty.
- Put S into PQ and V with priority $f(s) = g(s) + h(s)$
- Is PQ empty?
  - Yes? Sadly admit there's no solution
  - No? Remove node with lowest $f(n)$ from queue. Call it n.
  - If n is a goal, stop and report success.
  - "expand" n : For each n' in **successors**(n)....
    - Let $f' = g(n') + h(n') = g(n) + cost(n,n') + h(n')$
    - **If** n' not seen before, or n' previously expanded with $f(n') > f'$, or n' currently in PQ with $f(n') > f'$
    - **Then** Place/promote n' on priority queue with priority f' and update V to include ($state$=n', f', $BackPtr$=n).
    - **Else** Ignore n'

= $h(s)$ because $g(\text{start}) = 0$

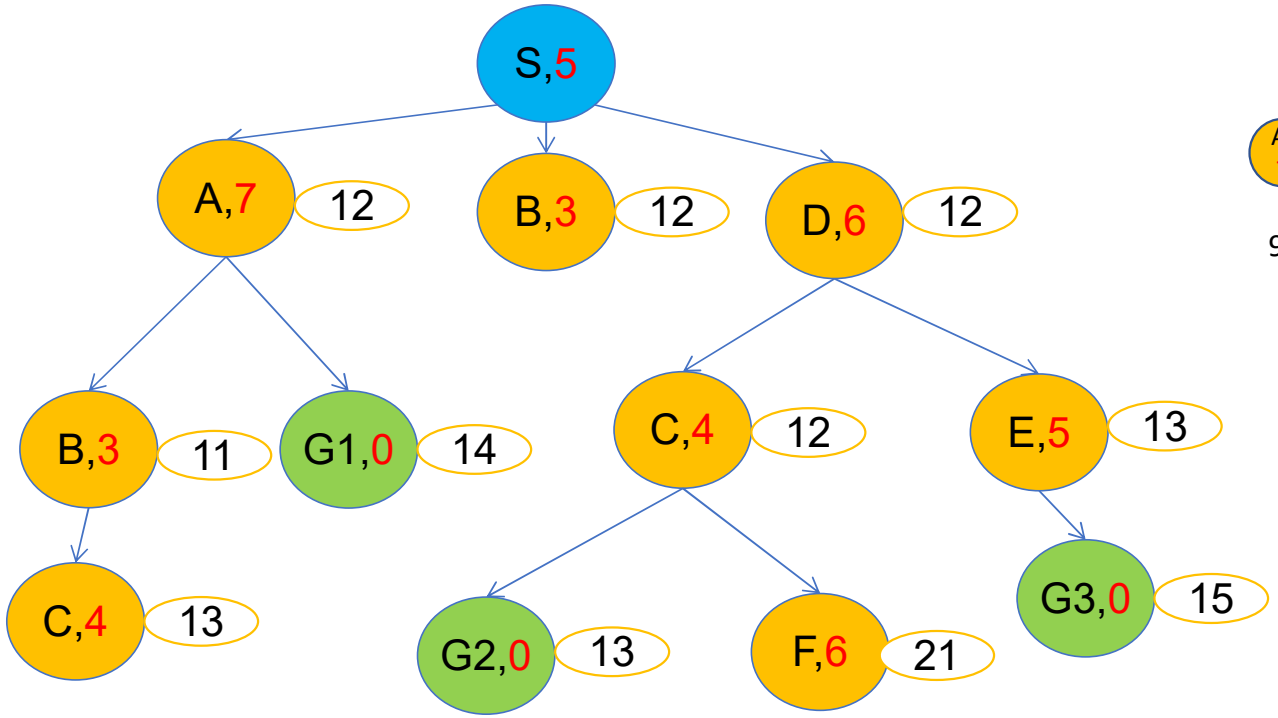
use sneaky trick to compute $g(n)$

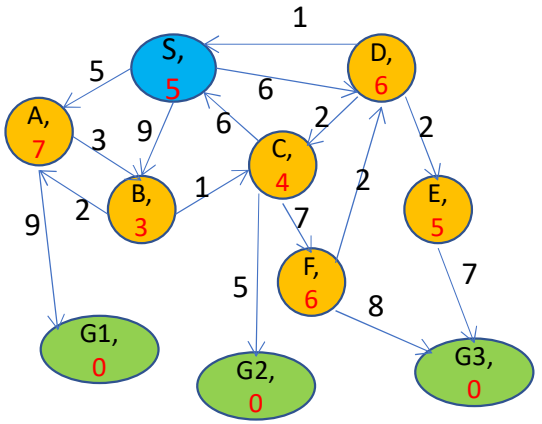
Problem

now we add A to visited along wit its A* score

Fo[illegible] goal from S

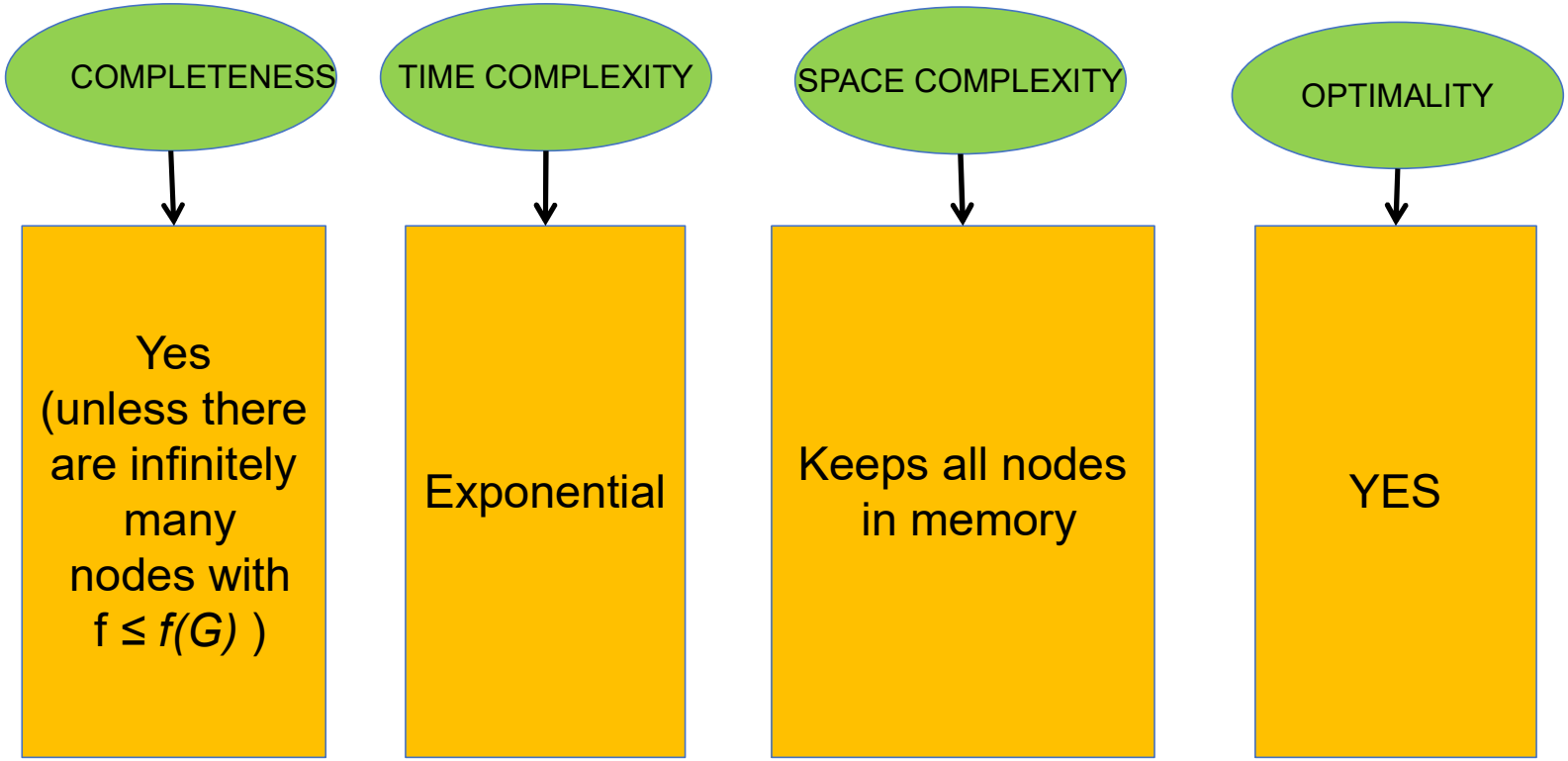
[illegible] as visited





S,0,5	A,7,12	B,3,11	D,6,12	C,4,12	E,5,13

How good is it??



Heuristic is the controller of A*

- The selection of the heuristic function is crucial in determining the efficiency of A*.
- Using h≡ 0 assures admissibility but results in a uniform-cost search and is thus usually inefficient.
- Setting h equal to the highest possible lower bound on h expands the fewest nodes consistent with maintaining admissibility.
- In the Eight-puzzle, for example, the function h(n)=W(n) (where W (n) is the number of tiles in the wrong place) is a lower bound on h (n), but it does not provide a very good estimate of the difficulty (in terms of number of steps to the goal) of a tile configuration. A better estimate is the function h(n)=P(n), where P(n) is the sum of the (**Manhattan**) distances that each tile is from "home" (ignoring intervening pieces).
- In selecting an h function, we must take into consideration the amount of effort involved in calculating it.
- There is usually a trade off between the benefits gained by an accurate h function and the cost of computing it.
- Another possibility is to modify the relative weights of g and h in the evaluation function . That is, we use f=g+w.h , where w is a positive number. Very large values of w overemphasize the heuristic component, whereas very small values of w give the search a predominantly breadth-first character.

Heuristic is the controller of A*

- A* ability to vary its behavior based on the heuristic and cost functions can be very useful.
- For most games, you don't really need the best path between two points. You need something that's close. What you need may depend on what's going on in the game, or how fast the computer is. Using a function that guarantees it never overestimates the cost means that it will sometimes underestimate the cost by quite a bit
- Suppose your game has two types of terrain, Flat and Mountain, and the movement costs are 1 for flat land and 3 for mountains, A* is going to search three times as far along flat land as it does along mountainous land. This is because it's possible that there is a path along flat terrain that goes around the mountains. You can speed up A*'s search by using 1.5 as the heuristic distance between two map spaces. A* will then compare 3 to 1.5, and it won't look as bad as comparing 3 to 1. It is not as dissatisfied with mountainous terrain, so it won't spend as much time trying to find a way around it.

Choice of a Heuristic

- On a grid, there are well-known heuristic functions to use.
- On a square grid that allows 4 directions of movement, use Manhattan distance (L1).
- On a square grid that allows 8 directions of movement, use Diagonal distance (L∞).
- On a square grid that allows any direction of movement, you might or might not want Euclidean distance (L2). If A* is finding paths on the grid but you are allowing movement not on the grid, you may want to consider other representations of the map.
- On a hexagon grid that allows 6 directions of movement, use Manhattan distance adapted to hexagonal grids.
- If you want to search for any of several goals, construct a heuristic h'(x) that is the minimum of h1(x), h2(x), h3(x), ... where h1, h2, h3 are heuristics to each of the nearby spots.

THANK YOU

K.S.Srinivas

srinivasks@pes.edu

+91 80 2672 1983 Extn 701