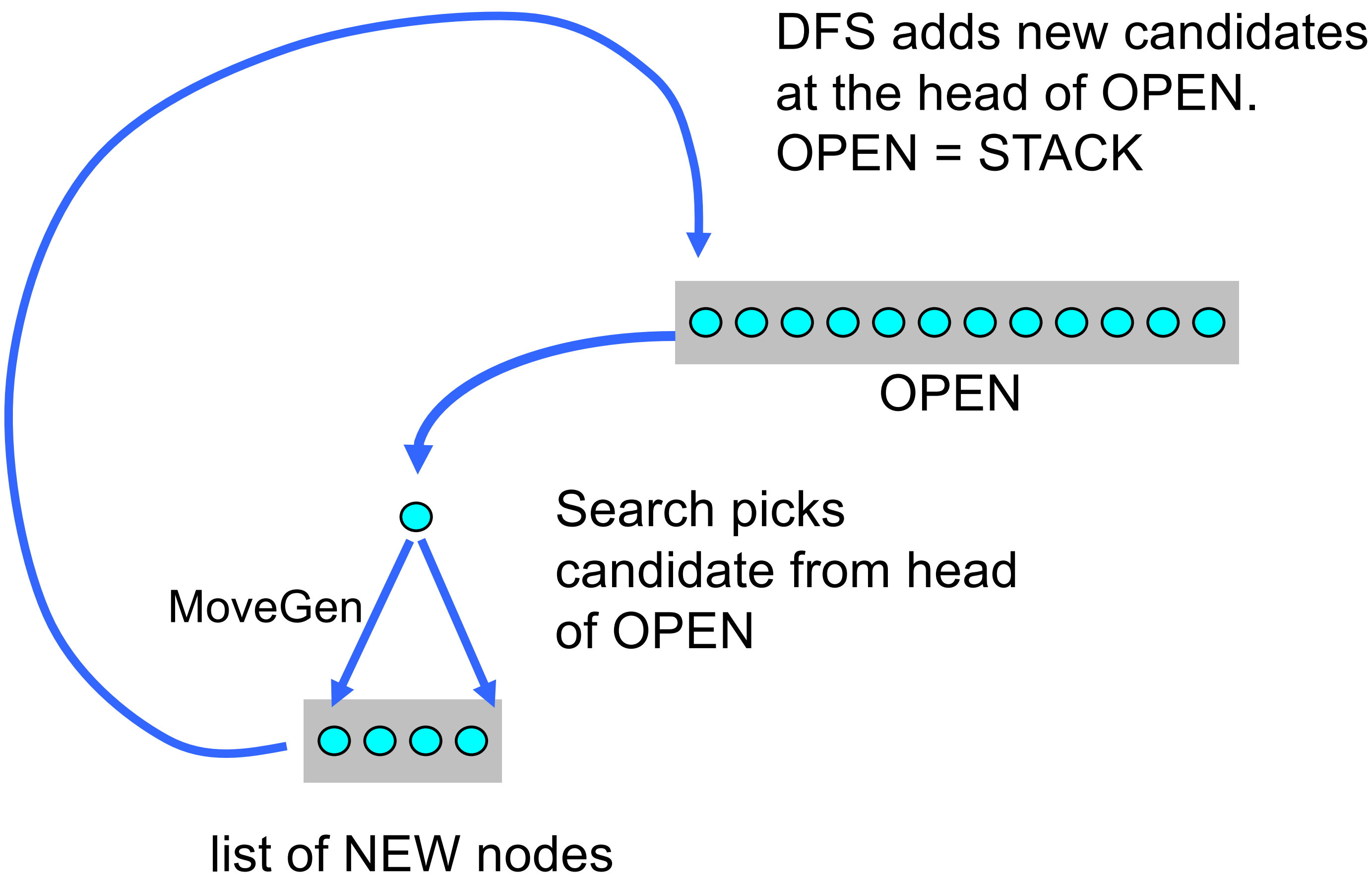
Search Methods in Artificial Intelligence

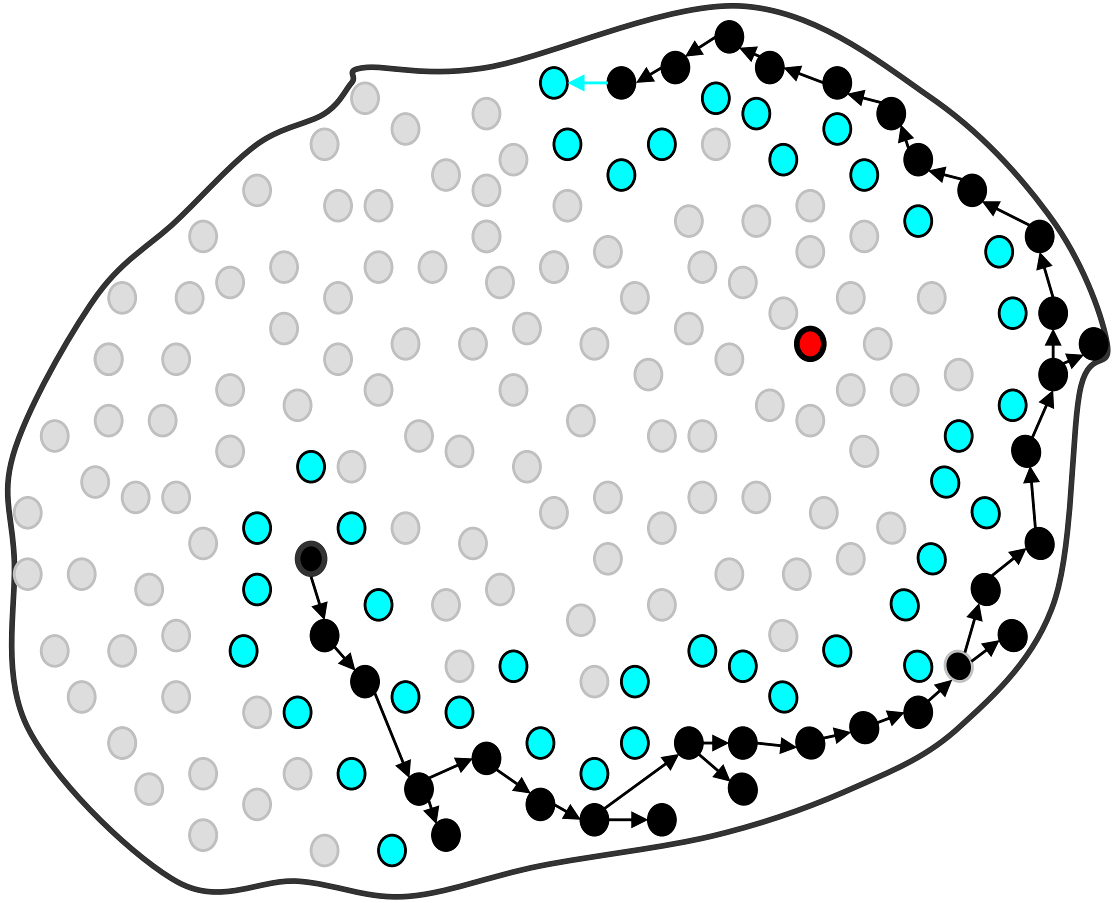
Heuristic Search

Deepak Khemani
Plaksha University

Depth First Search

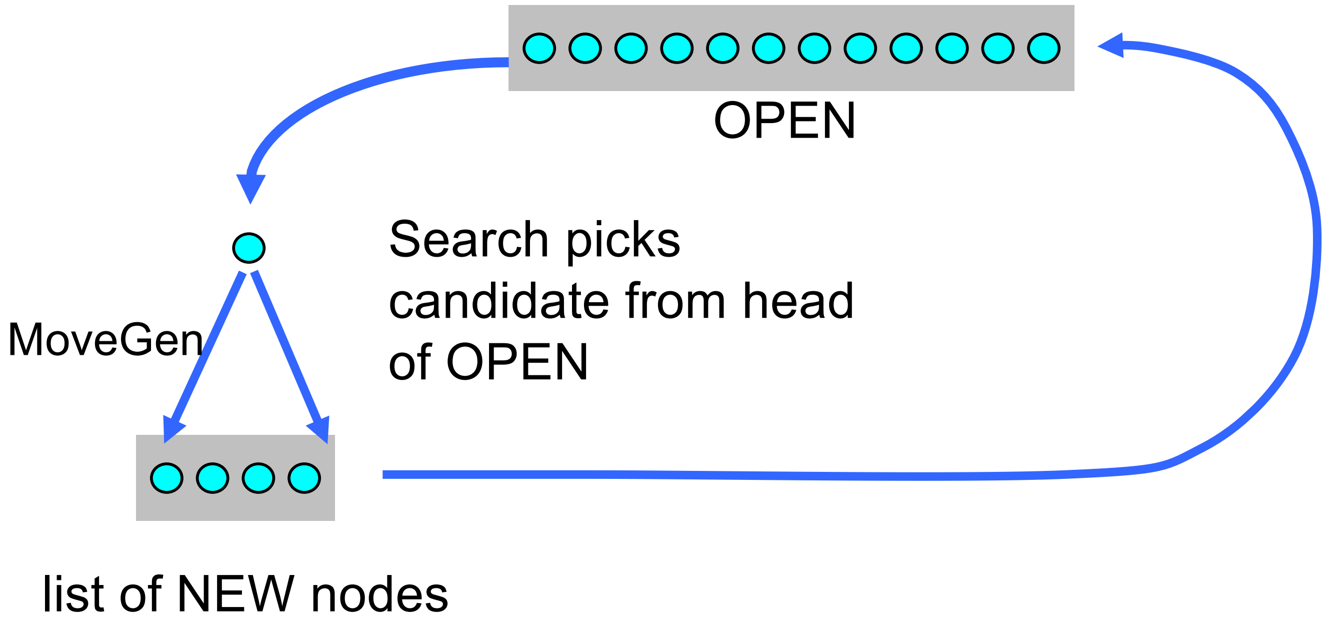


Depth First Search dives into the search space

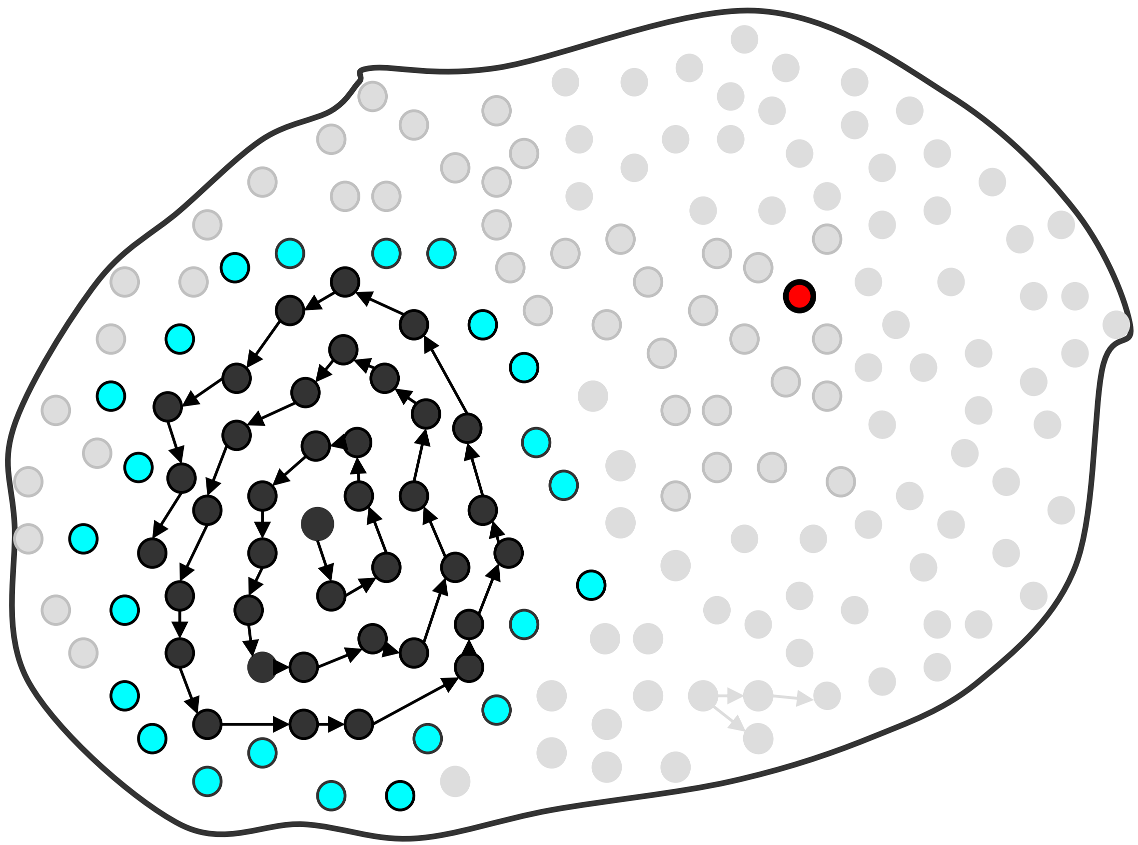


Breadth First Search

Breadth First Search adds new candidates at the head of OPEN.
OPEN = QUEUE



Breadth First Search sticks close to the start state



Blind / Uninformed Search

Both
Depth First Search and Breadth First Search
are oblivious of the goal.

Irrespective of where the goal is in the state space both the algorithms set out on the same predetermined trajectories every time.

What is needed is a search algorithm with a sense of direction.

The pull of gravity

The gradient on a mountain side gives *a sense of direction* to water flowing down a stream.

Water always takes the *steepest gradient* descent path.

We can simulate this *sense of direction* in search algorithms.

Imagine a physical agent wanting to reach the bottom of a valley. It can do the following.

- Test the landscape by “taking a step” in each direction
- Choose the direction with the steepest gradient

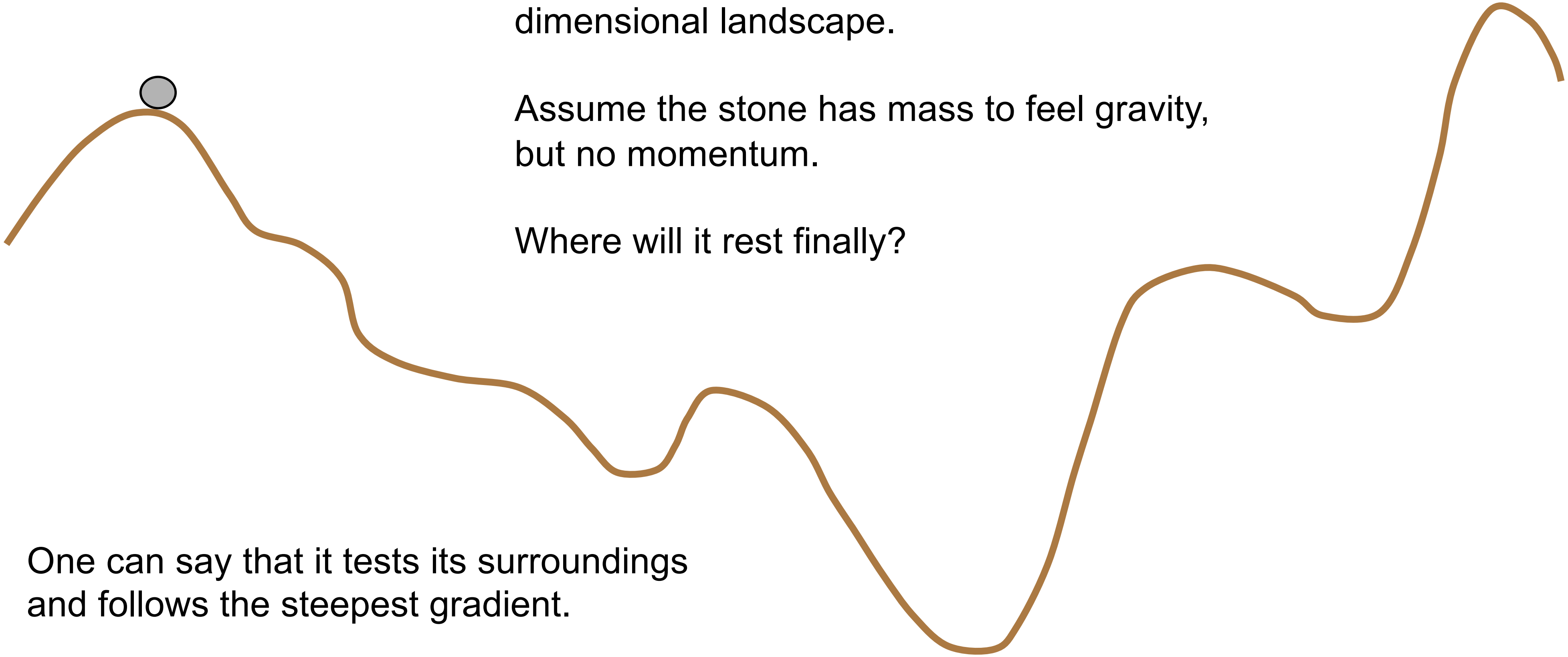
The question is what defines the landscape we are traversing? How is the gradient defined?

A Rolling Stone…

Consider a stone rolling down a two dimensional landscape.

Assume the stone has mass to feel gravity, but no momentum.

Where will it rest finally?



One can say that it tests its surroundings and follows the steepest gradient.

Local search: it looks only at its immediate neighbourhood!

The idea of a heuristic function

Testing the neighbourhood and following the steepest gradient identifies which of the neighbours is the lowest – or closest to the bottom of the valley.

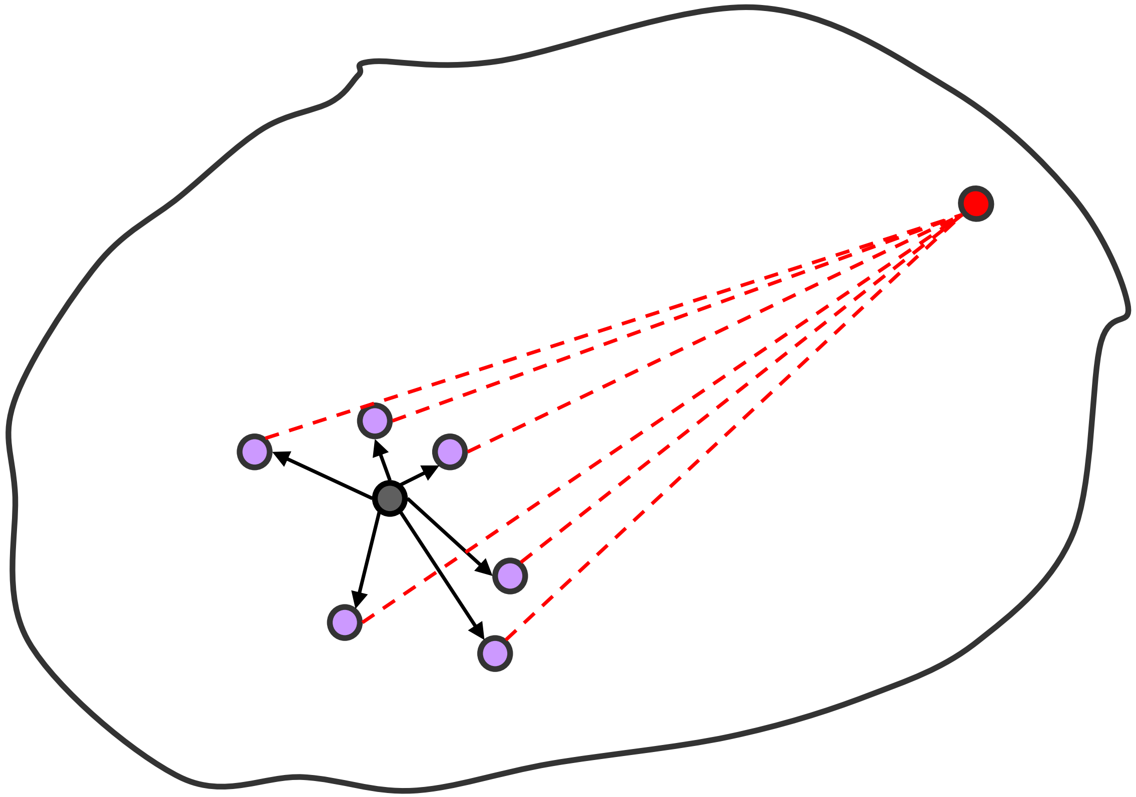
The idea of a heuristic function is that it takes a state or a node as input and computes a number which is an estimate of the distance of that node from the goal.

Then a search algorithm can pick that node from OPEN which has the lowest heuristic value.

The heuristic function *h(N)* is typically a user defined function.
- in addition to the *MoveGen* and the *GoalTest* functions

Since h(goal) = 0 we are effectively seeking to minimize h(N)

Heuristic functions



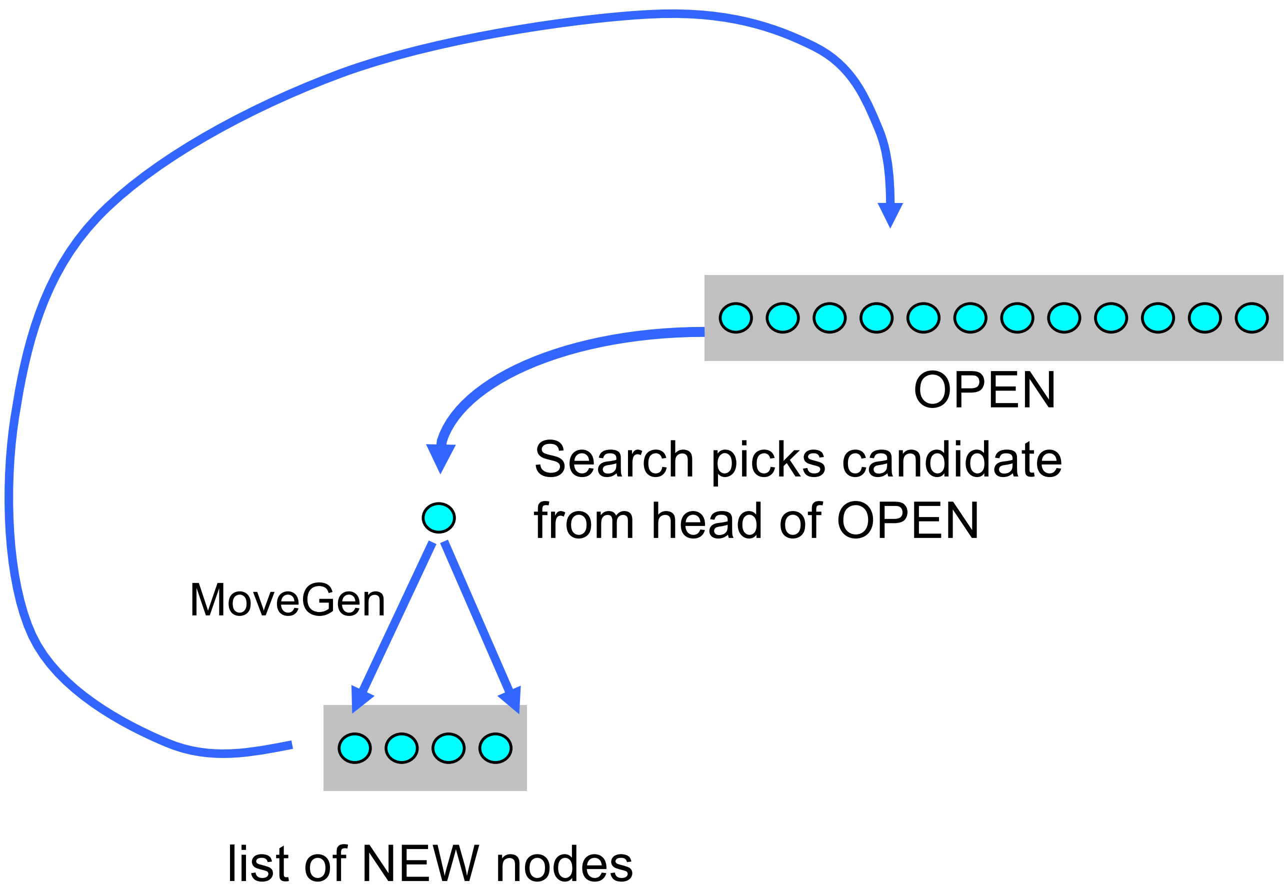
The heuristic function **estimates** the distance to the goal.

This estimate, h(n), can be used to decide **which** node to pick from OPEN

Best First Search

Best First Search **inserts** new candidates into OPEN **sorted** on **h(n)**

OPEN = PRIORITY QUEUE



Best First Search picks the node with lowest *h(n)*

~~Depth~~ Best First Search

```
    OPEN ← ((Start, Nil)) ; CLOSED ← ()
    While not null (OPEN) Do
            nodePair ← head (OPEN) ; node ← head(nodePair)
            IF goalTest (node) = True THEN
                    return reconstructPath(nodePair, CLOSED)
            ELSE
                    CLOSED ← cons (nodePair, CLOSED)
                    CHILDREN ← moveGen (node)
                    NOLOOPS ← removeSeen (CHILDREN, OPEN, CLOSED)
                    NEW ← makePairs(NOLOOPS, node)
                    OPEN ← append (NEW, tail(OPEN))
    endWhile
Return "No solution found"
End
```

The nodePair is to be transformed into a nodeTriple to include h(n)

OPEN ← sort_h (append (NEW, tail(OPEN)))

or OPEN ← merge (sort_h(NEW), tail(OPEN)))

In practice OPEN is maintained as a priority queue

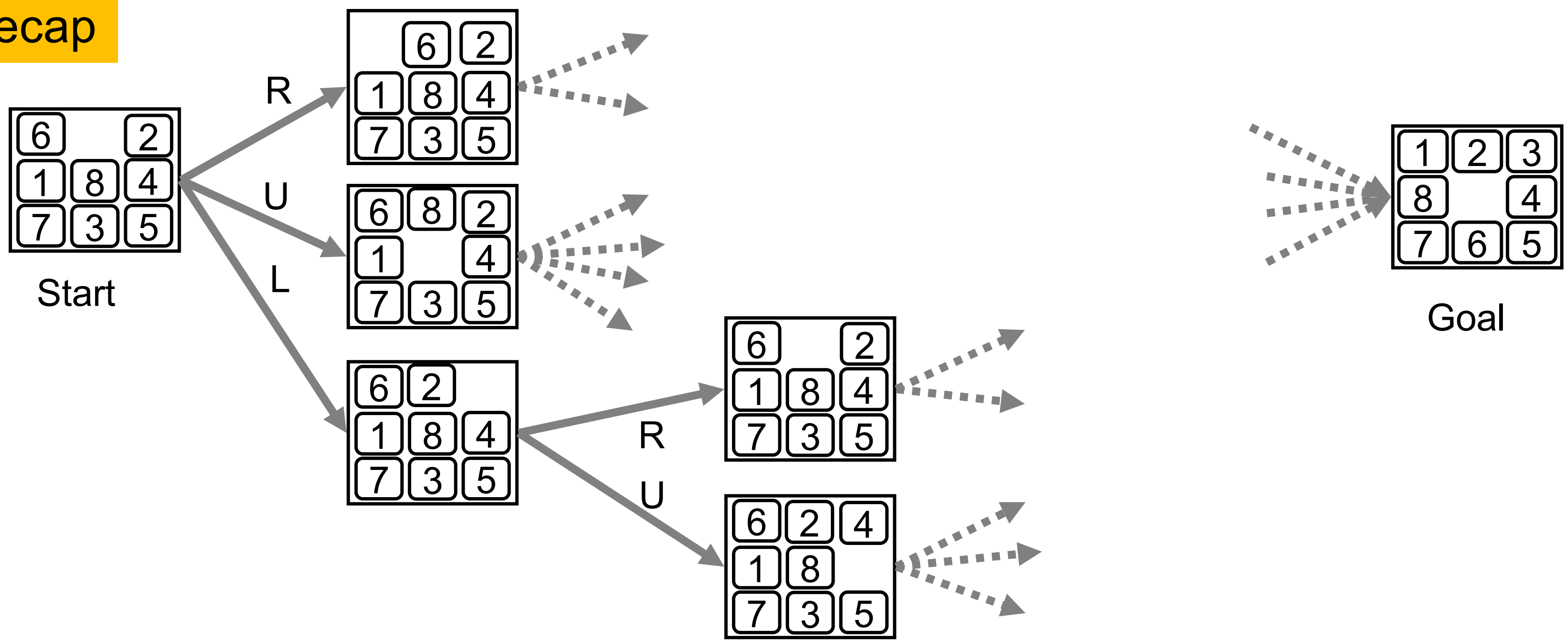
Best First Search sorts OPEN on *h(N)*

```
BEST-FIRST-SEARCH(S)
OPEN ← (S, null, h(S)) : [ ]
CLOSED ← empty list
while OPEN is not empty
      nodePair ← head OPEN
      (N, _ , _ ) ← nodePair
      if GOALTEST(N) = TRUE
            return RECONSTRUCTPATH(nodePair, CLOSED)
      else CLOSED ← nodePair : CLOSED
            children ← MOVEGEN(N)
            newNodes ← REMOVESEEN(children, OPEN, CLOSED)
            newPairs ← MAKEPAIRS(newNodes, N)
            OPEN ← sort_h( newPairs ++ tail OPEN )
return empty list
```

A triple, even if we still call it a nodePair

The Eight-puzzle

Recap

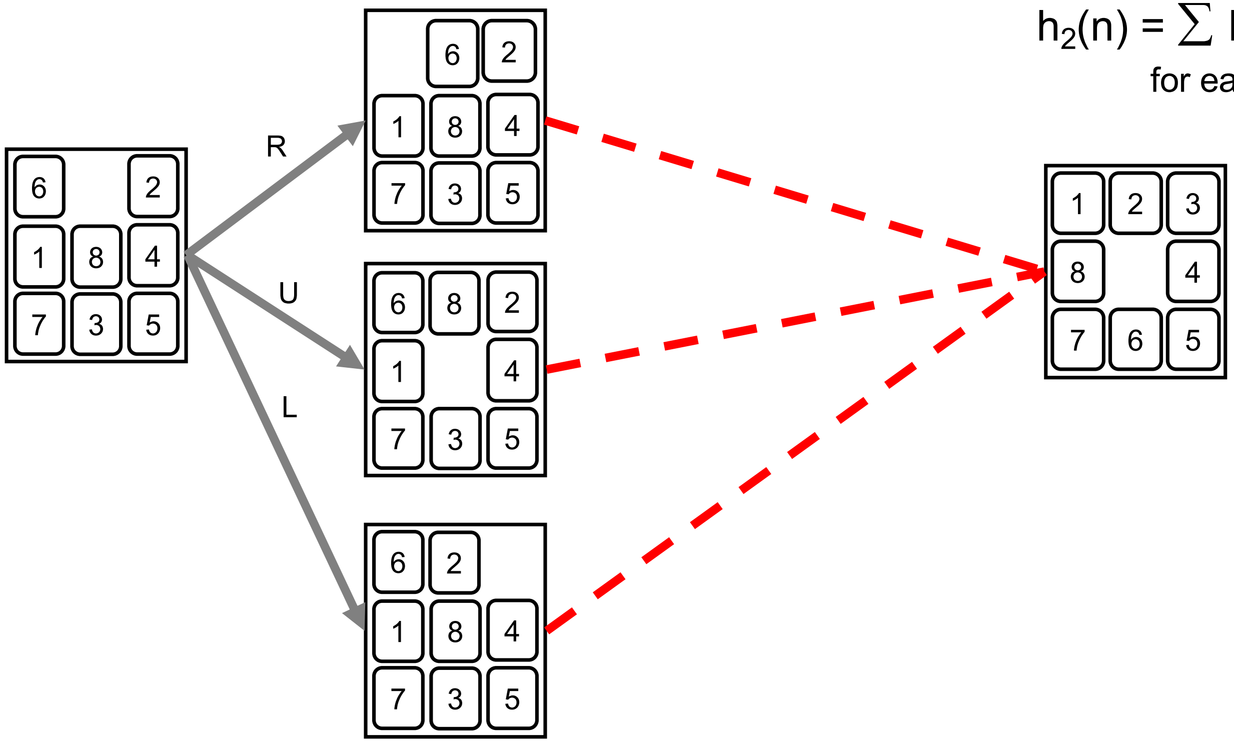


The Eight puzzle consists of eight tiles on a 3x3 grid. A tile can slide into an adjacent location if it is empty. A move is labeled R if a tile moves right, and likewise for up (U), down (D) and left (L).

The Eight-puzzle

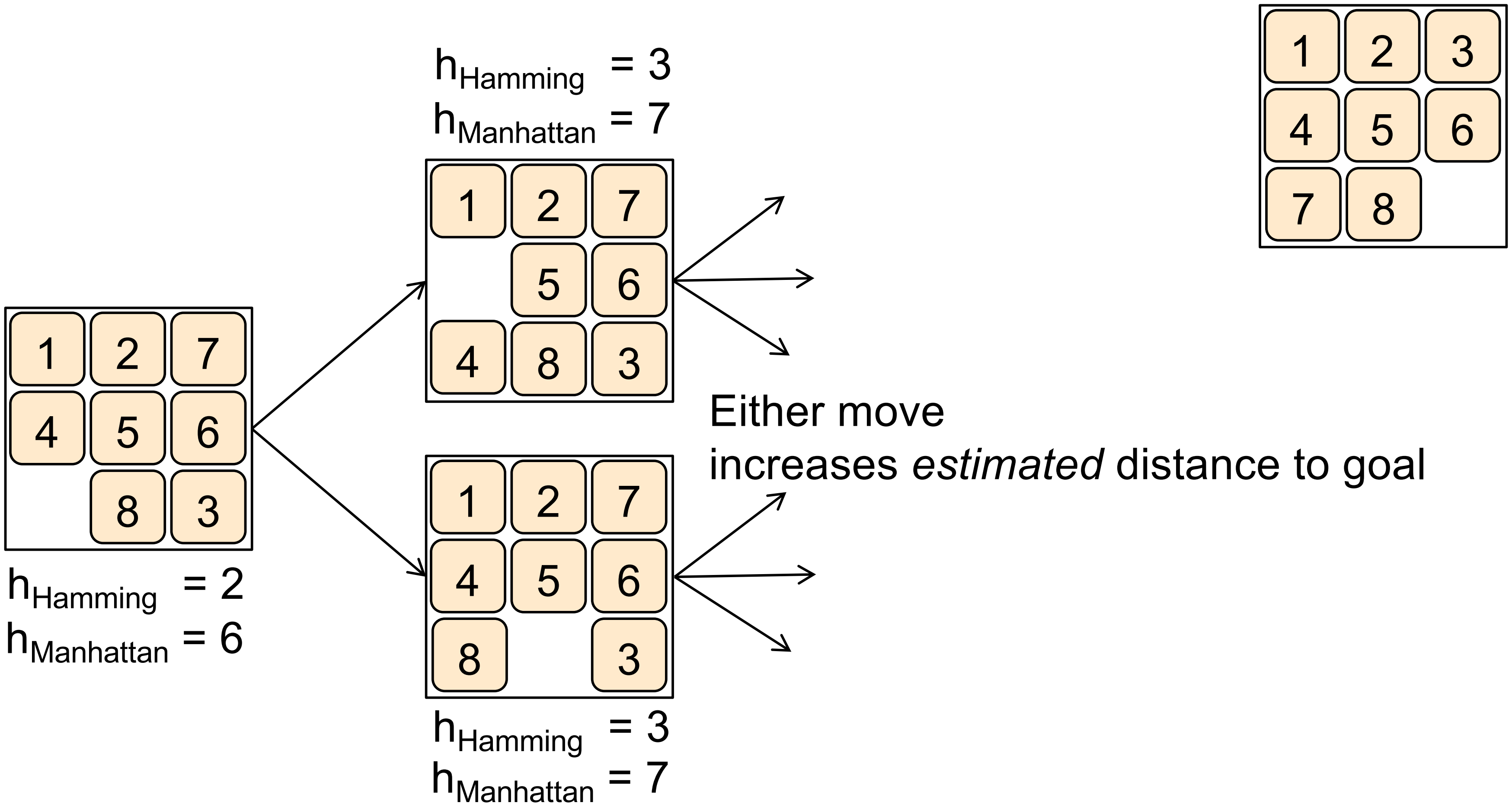
$h_1(n)$ = number of tiles out of place
Hamming distance

$h_2(n) = \sum$ Manhattan distance to its destination
for each tile



Which state is closest to the goal?

8-puzzle: A local minimum

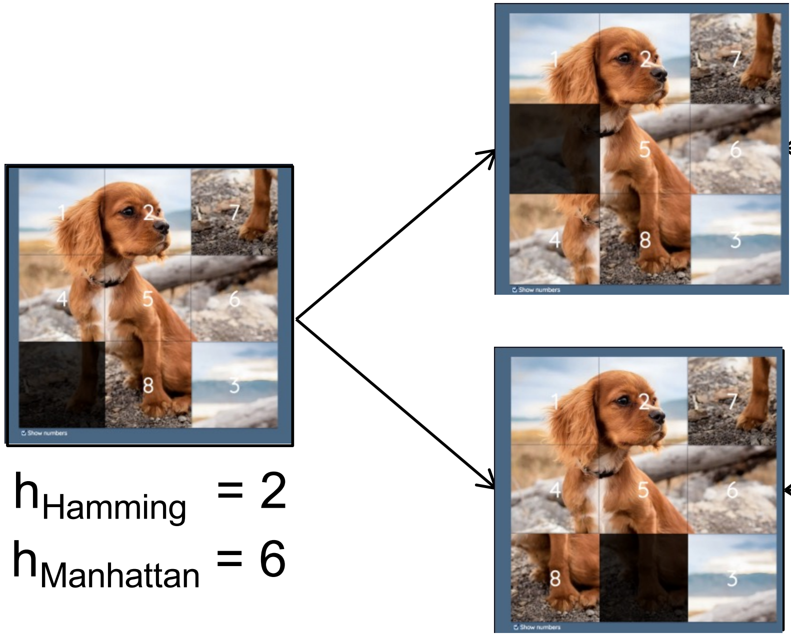


Either move
increases *estimated* distance to goal

8-puzzle: Similarity is inverse of distance

$h_{Hamming} = 3$

$h_{Manhattan} = 7$



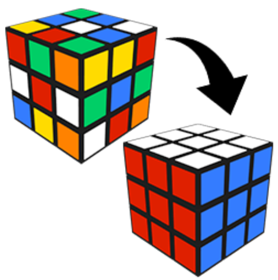
Either move decreases similarity with goal

$h_{Hamming} = 2$

$h_{Manhattan} = 6$

$h_{Hamming} = 3$

$h_{Manhattan} = 7$

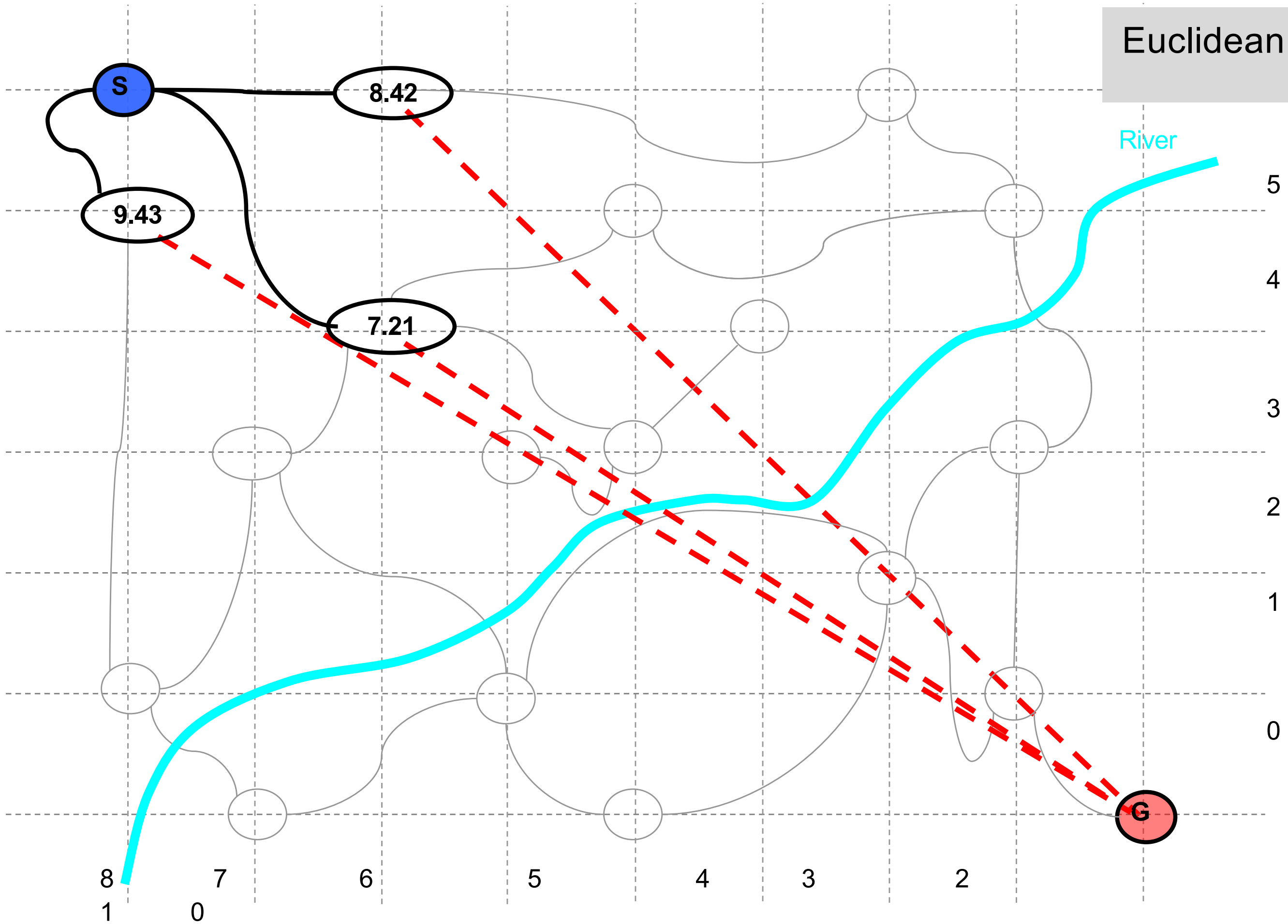


Rubik's cube

https://murhafsousli.github.io/8puzzle/#/

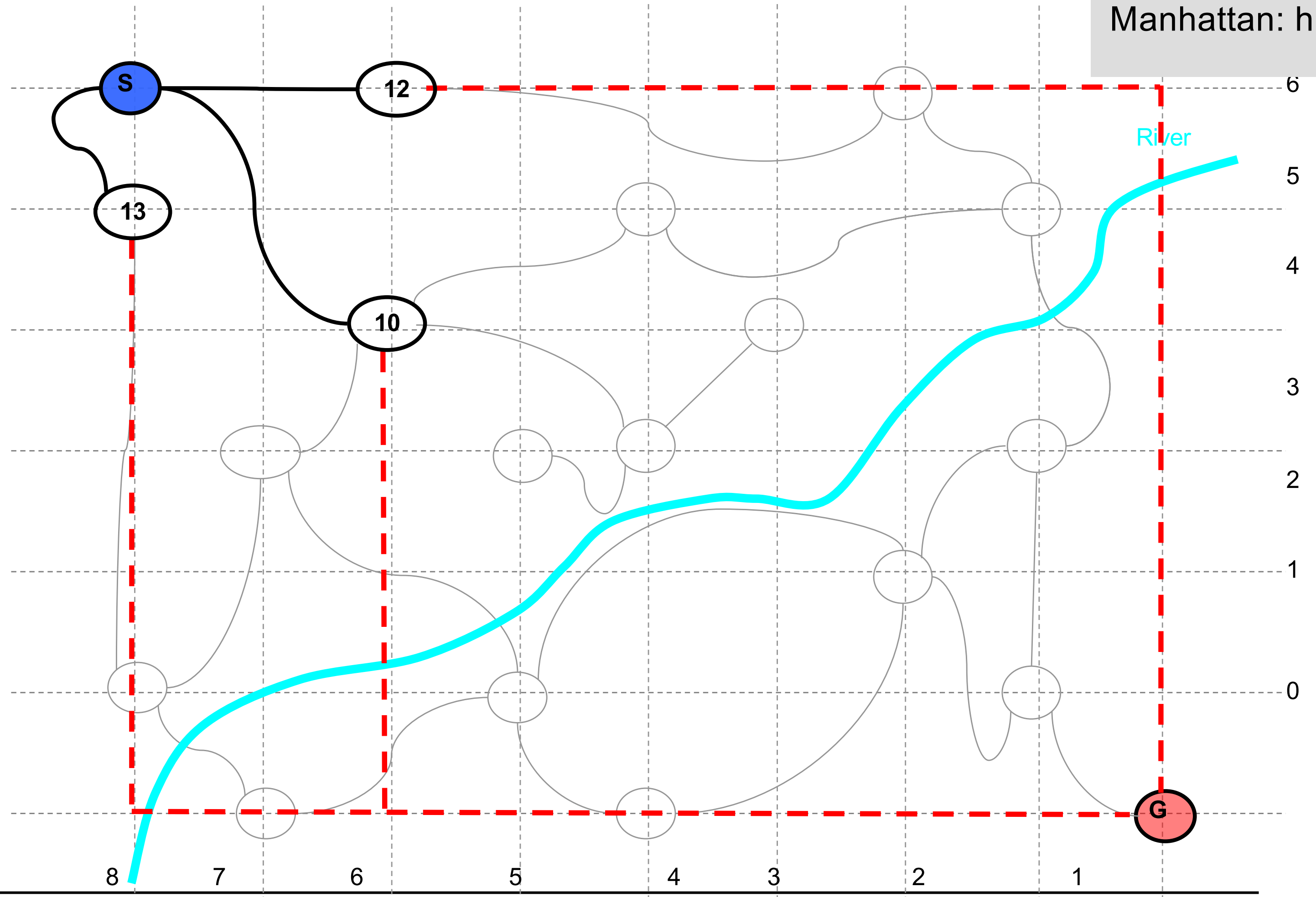
Geographical route finding : h(n) = Euclidean distance

Euclidean: $h(n) = \sqrt{(x_{Goal} - x_n)^2 + (y_{Goal} - y_n)^2}$

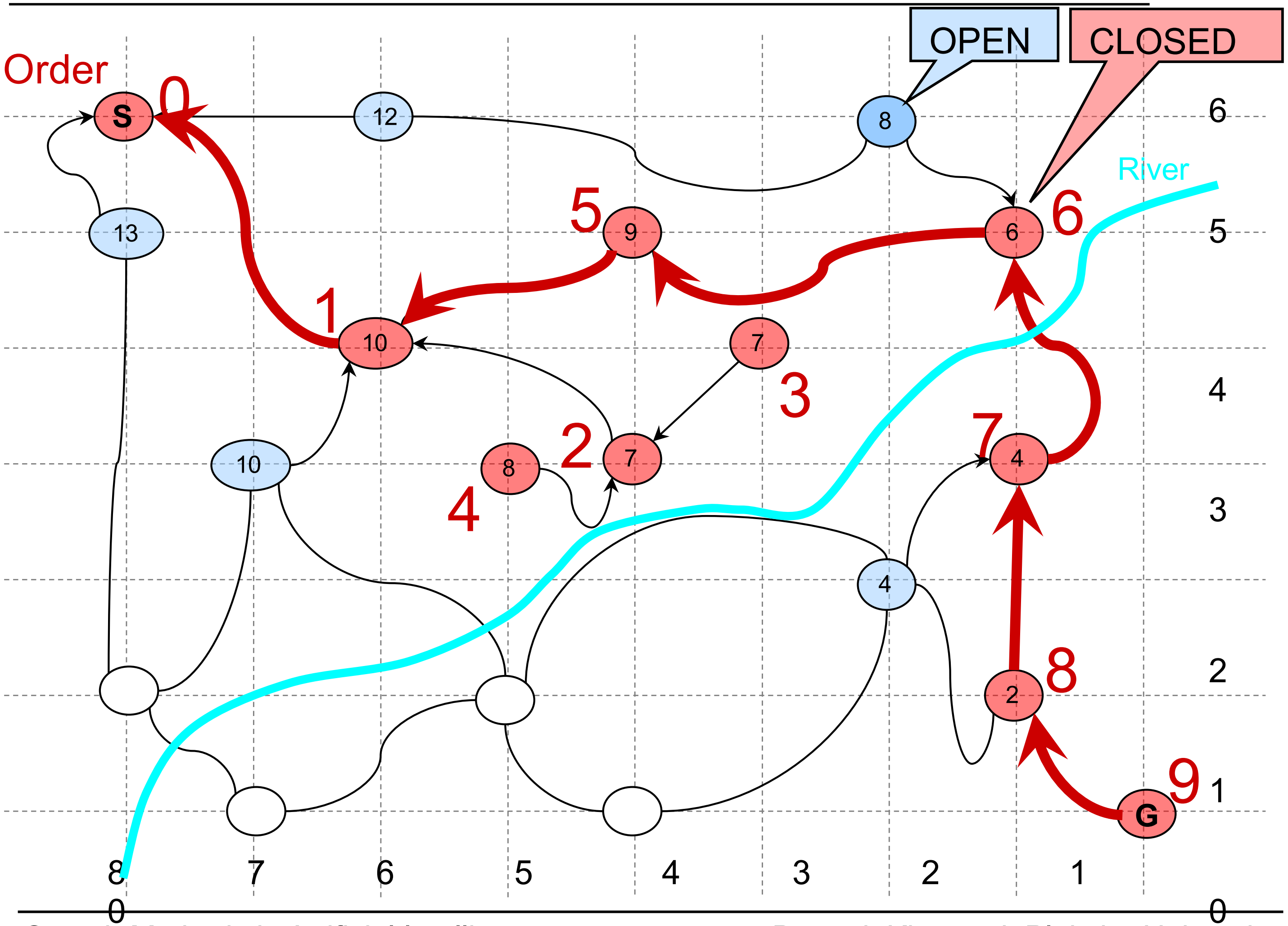


Geographical route finding : h(n) = Manhattan /city-block distance

Manhattan: $h(n) = |x_{Goal} - x_n| + |y_{Goal} - y_n|$

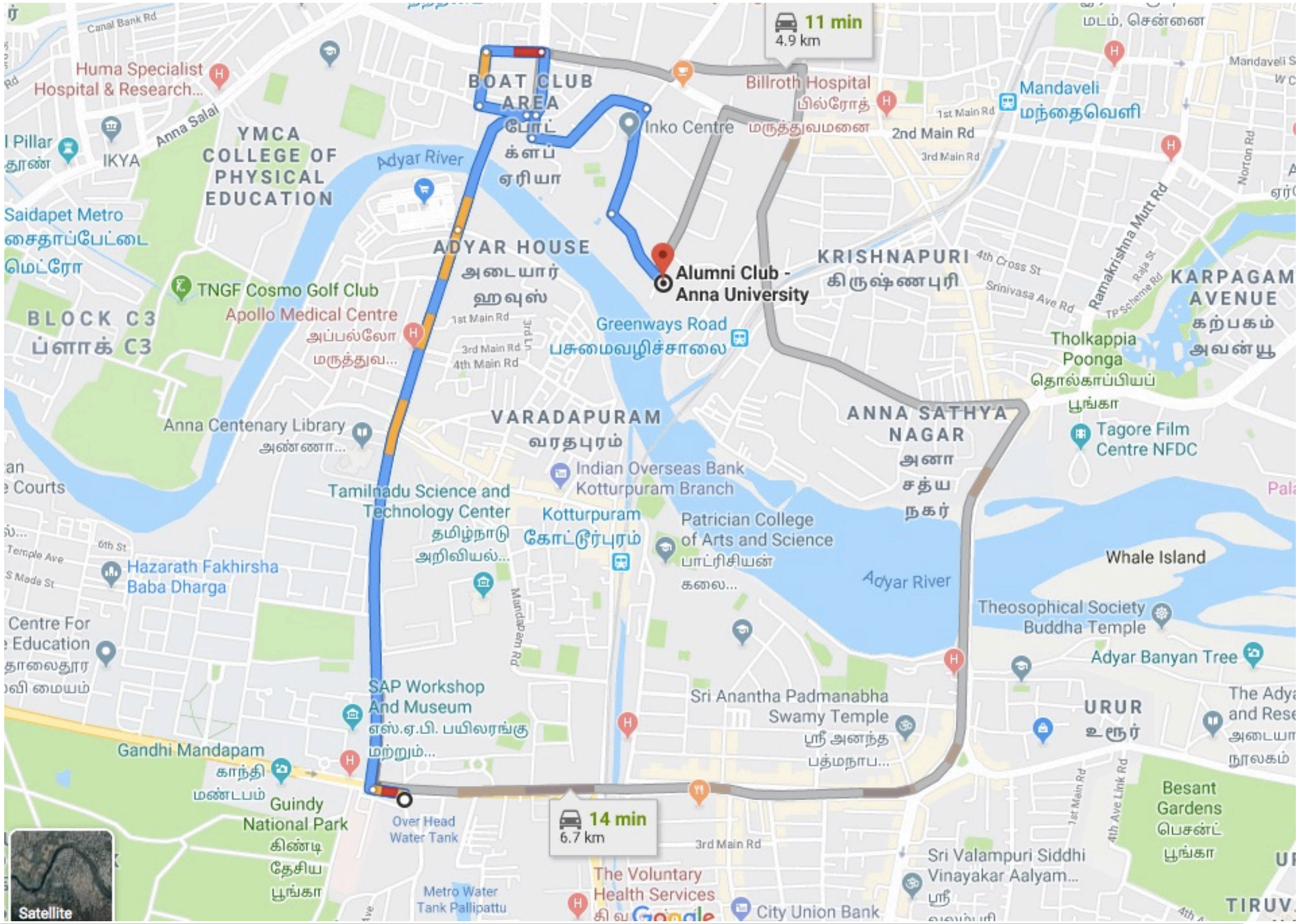


Best First Search has a sense of direction

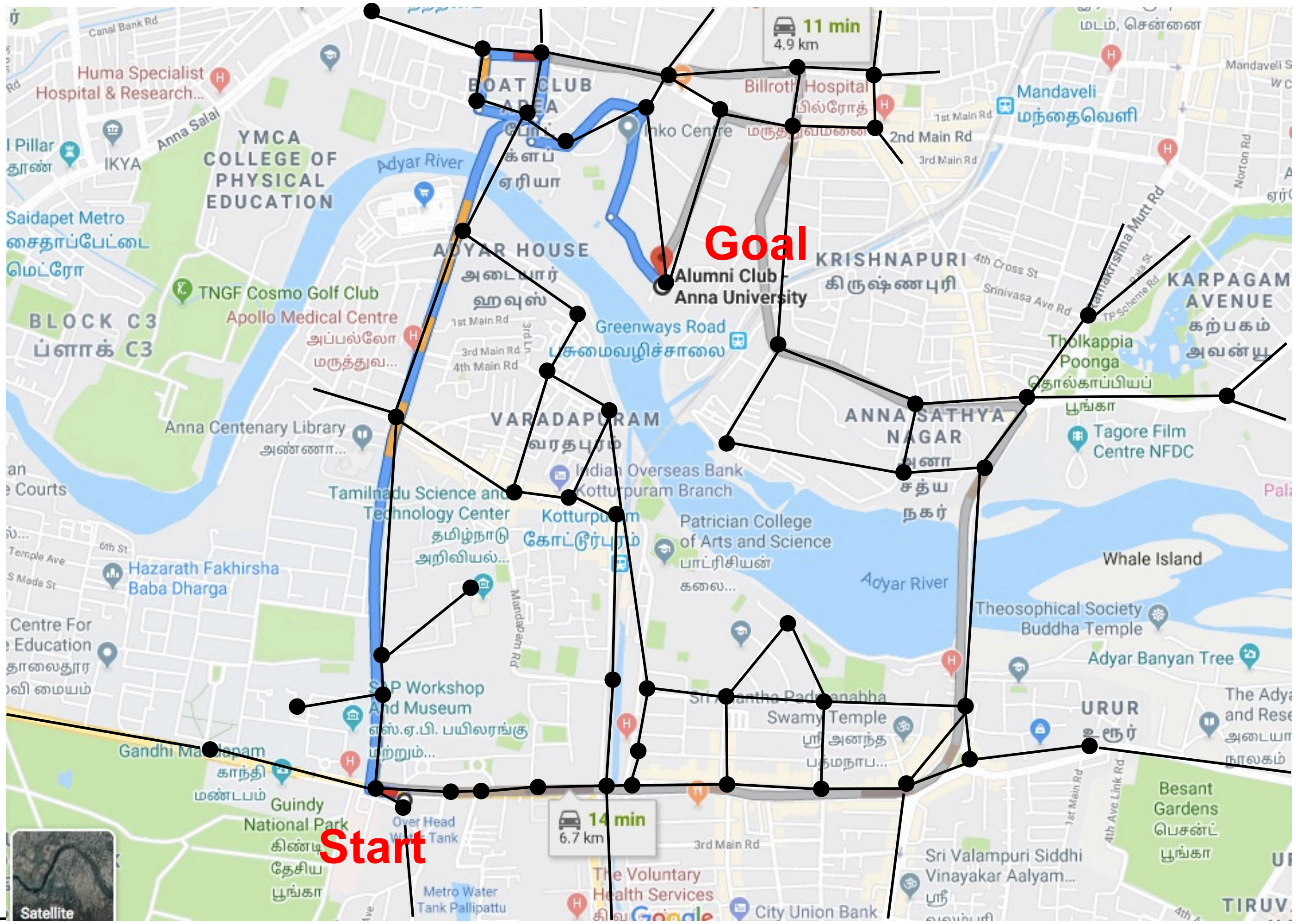


The **order** shows its progress – it first hits a dead end (steps 2-4)

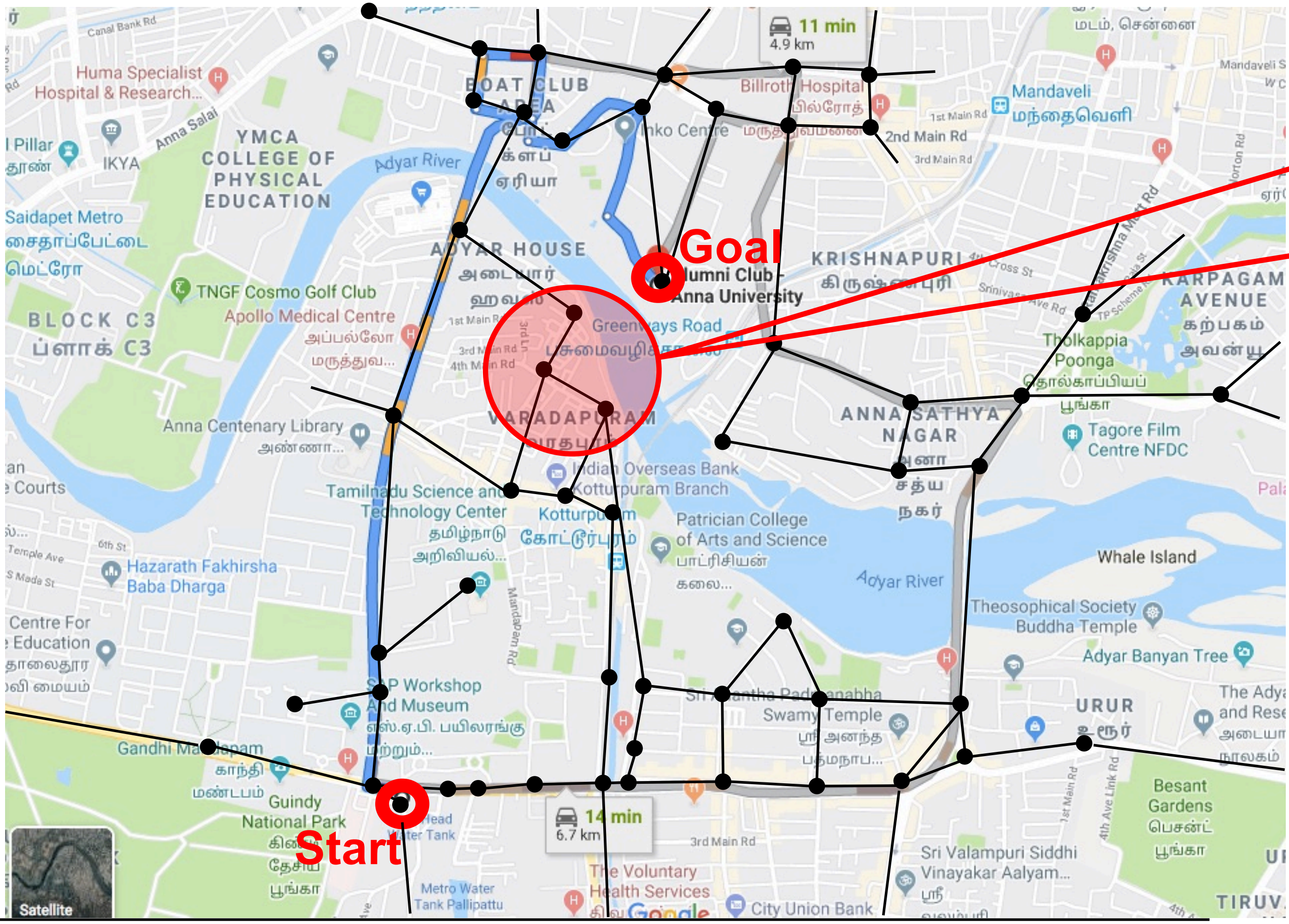
IIT Madras to Anna Alumni Club – Google Maps



The underlying graph

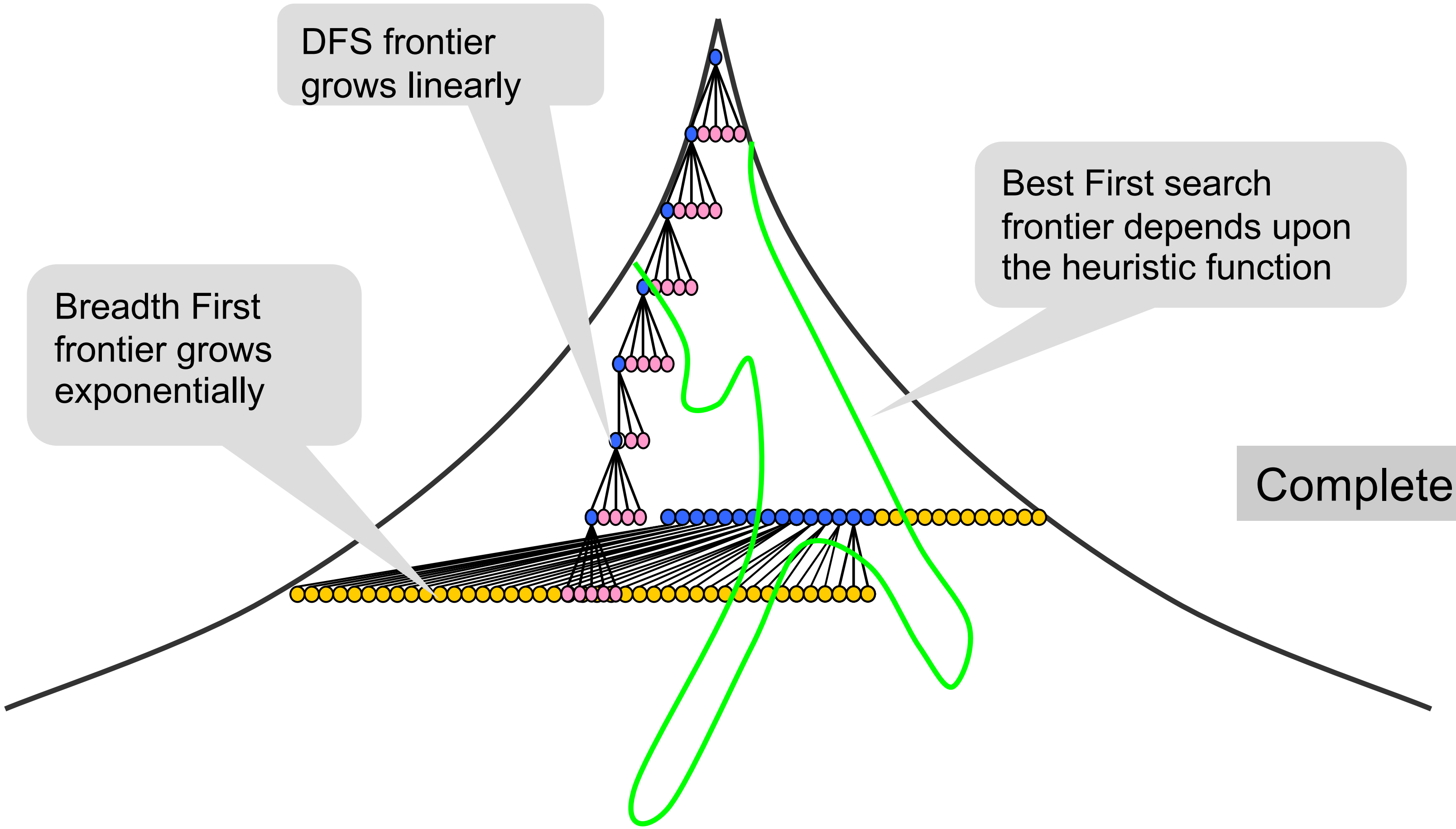


Best First Search



A heuristic function would drive the search towards *these* nodes and then would have to *move away*

Search Frontiers

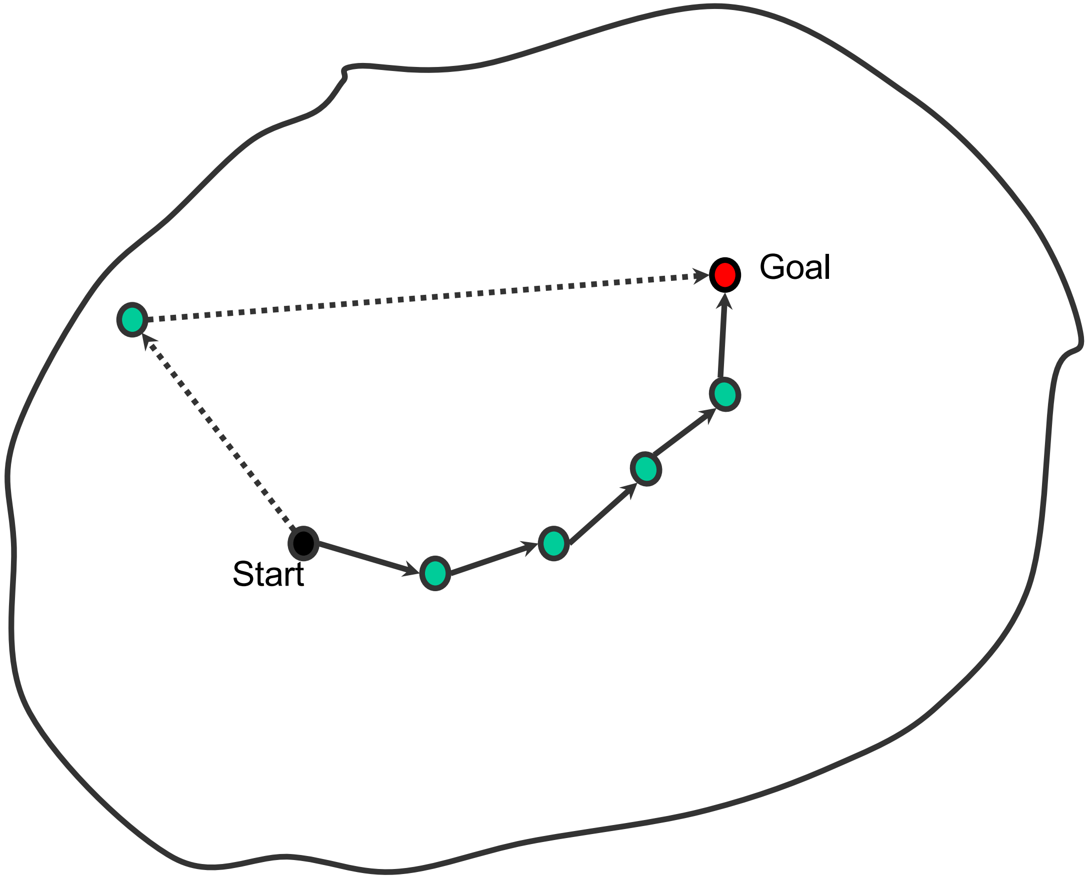


Completeness?

The Search Frontier is an indication of space requirement

Quality of solution

Best First Search may choose
a solution with five moves



Hill Climbing – a local search algorithm

Move to the best neighbour if it is better, else terminate

```
Hill Climbing
    node ← Start
    newNode ← head(sort_h moveGen (node)))
    While h(newNode) < h(node) Do
            node ← newNode
            newNode ← head(sort_h (moveGen (node))))
    endWhile
    return node
End

Algorithm Hill Climbing
```

In practice sorting is not needed, only the best node

Change of termination criterion

Local search – Hill Climbing has burnt its bridges by not storing OPEN

Hill Climbing – a constant space algorithm

HC only looks at local neighbours of a node. It's space requirement is thus *constant* !

A vast improvement!

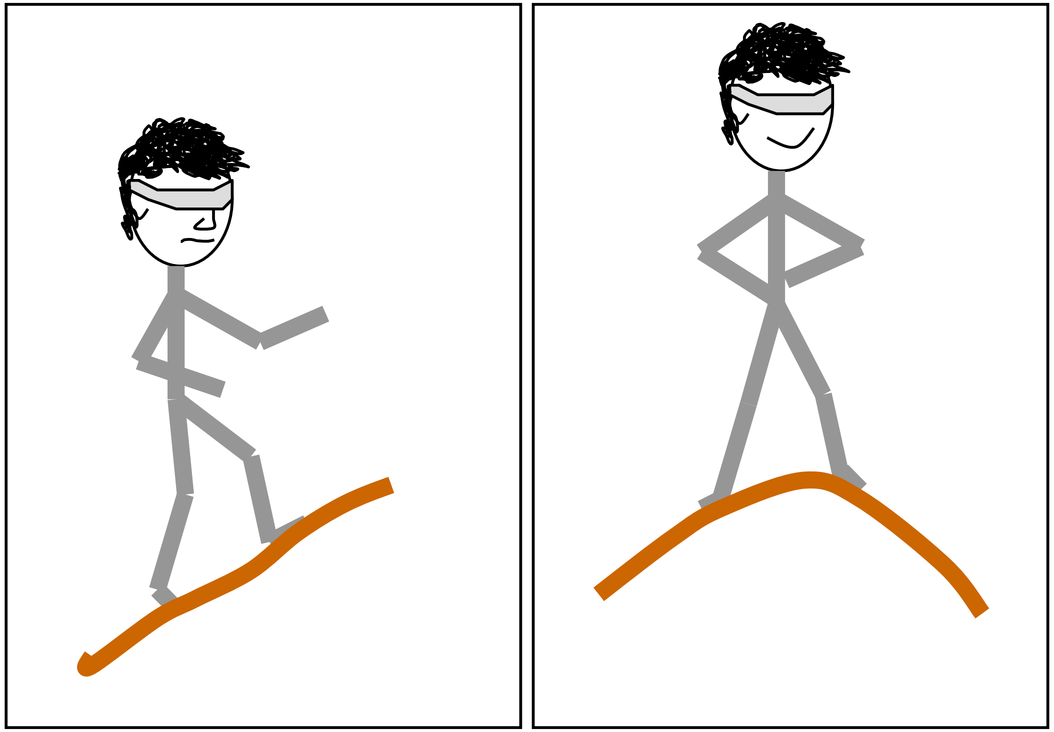
HC only moves to a better node. It terminates if it cannot.

Consequently the *time complexity is linear*.

It's *termination criterion is different*. It stops when no better neighbour is available. It treats the problem as an *optimization problem*.

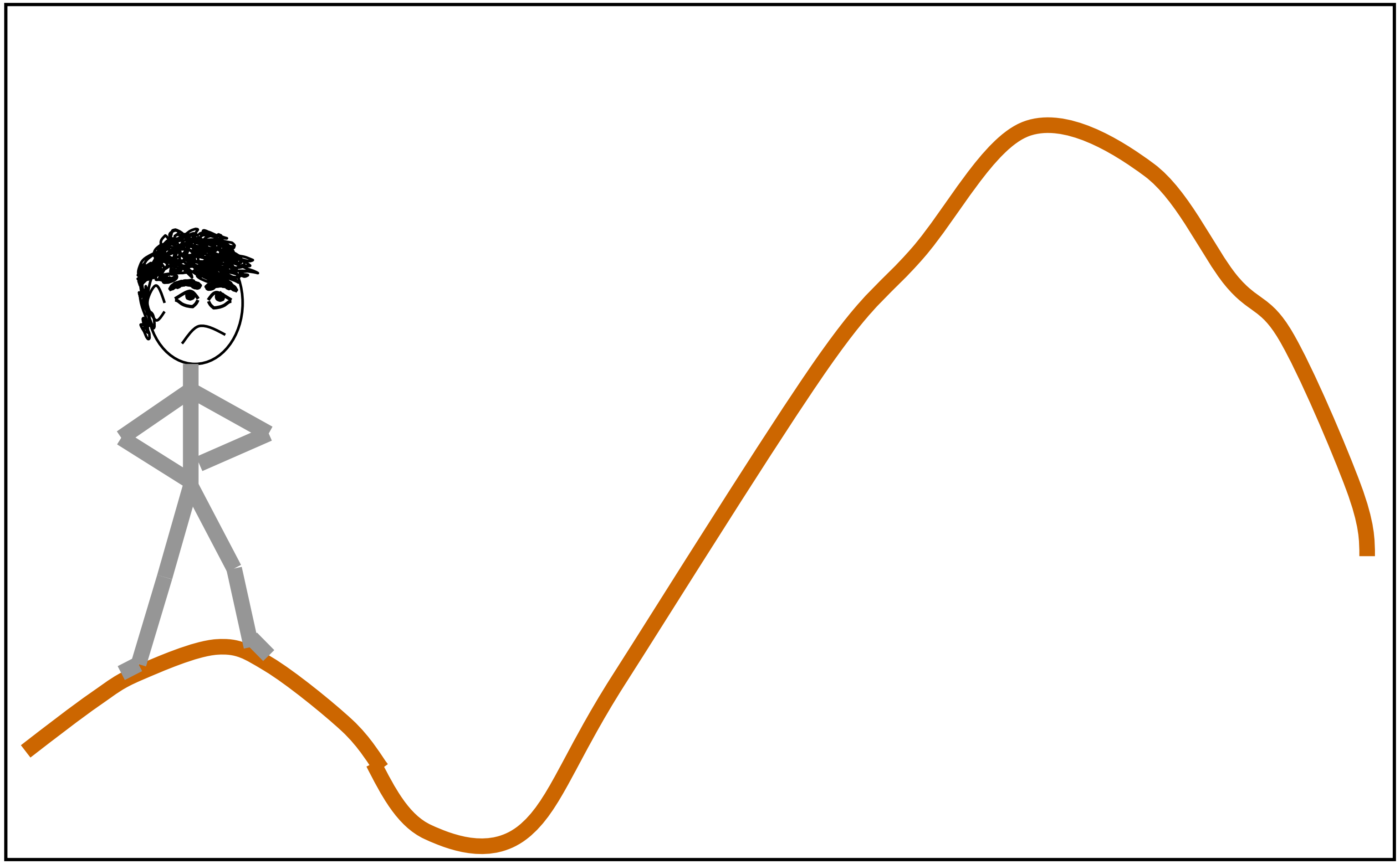
However, it is not complete, and may not find the global optimum which corresponds to the solution!

Steepest gradient ascent

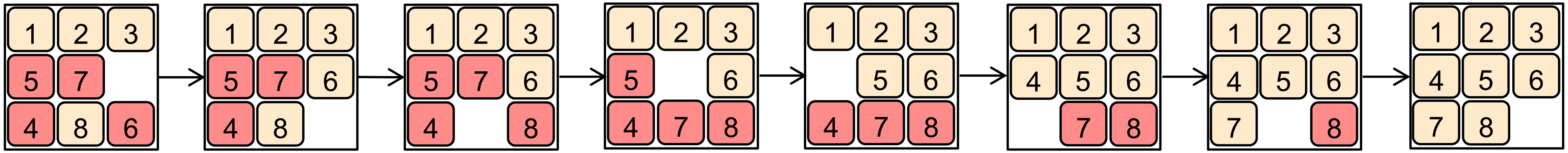


Hill Climbing (for a maximization problem)

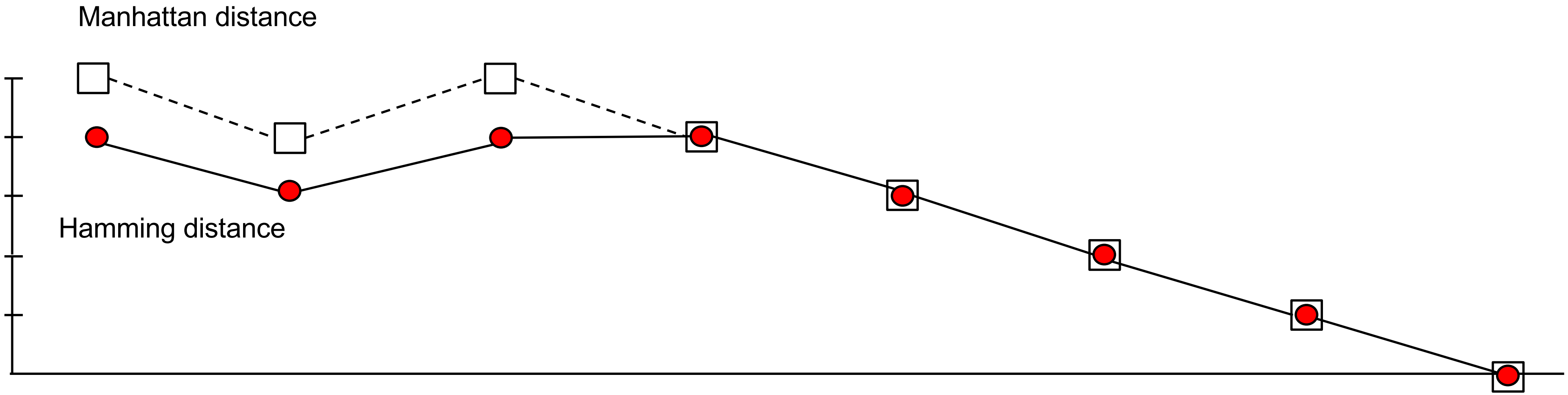
May end on a local maximum



8-puzzle: A Solution



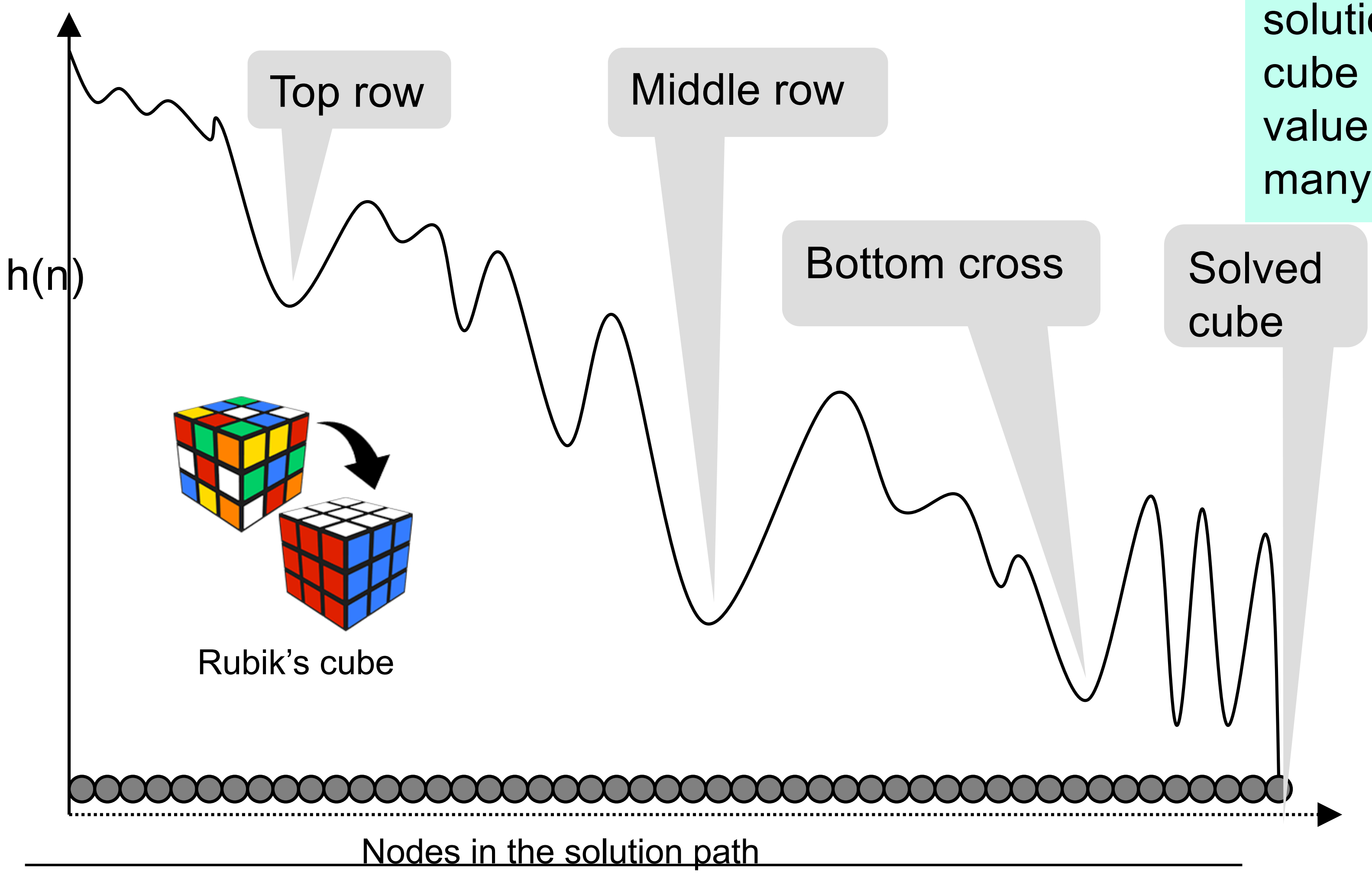
$h_{Hamming}$ = 4	$h_{Hamming}$ = 3	$h_{Hamming}$ = 4	$h_{Hamming}$ = 4	$h_{Hamming}$ = 3	$h_{Hamming}$ = 2	$h_{Hamming}$ = 1	$h_{Hamming}$ = 0
$h_{Manhattan}$ = 5	$h_{Manhattan}$ = 4	$h_{Manhattan}$ = 5	$h_{Manhattan}$ = 4	$h_{Manhattan}$ = 3	$h_{Manhattan}$ = 2	$h_{Manhattan}$ = 1	$h_{Manhattan}$ = 0



A plot of heuristic value along the solution path

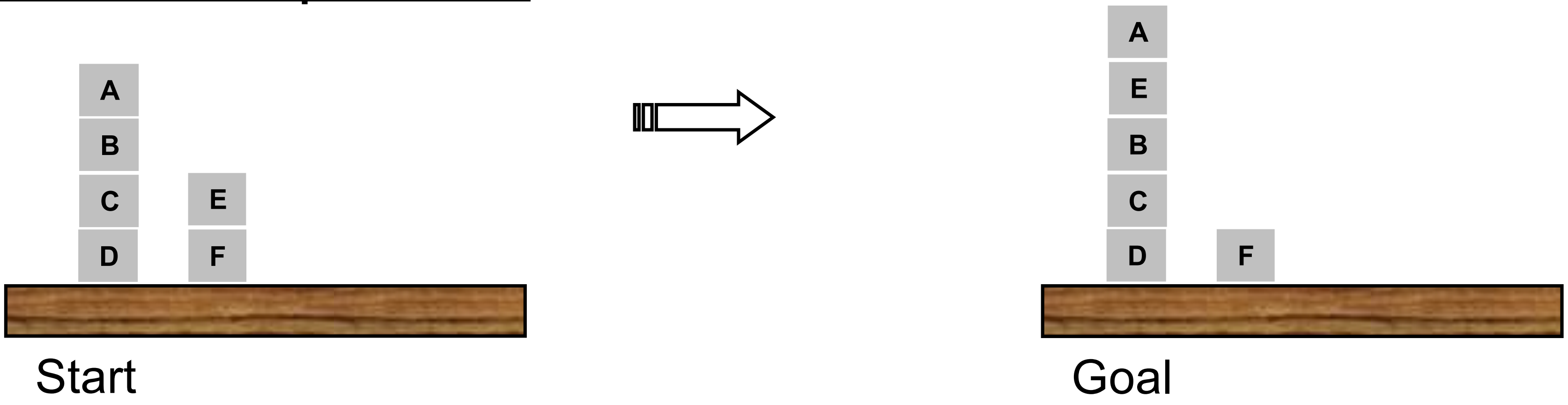
The Rubik's cube – fluctuating heuristic values

For human generated solutions of the Rubik's cube a simple heuristic value passes through many local minima



A blocks world problem

Move: move block X to loc Y



$h_1(\text{start}) = 2$
$h_1(\text{goal}) = 6$

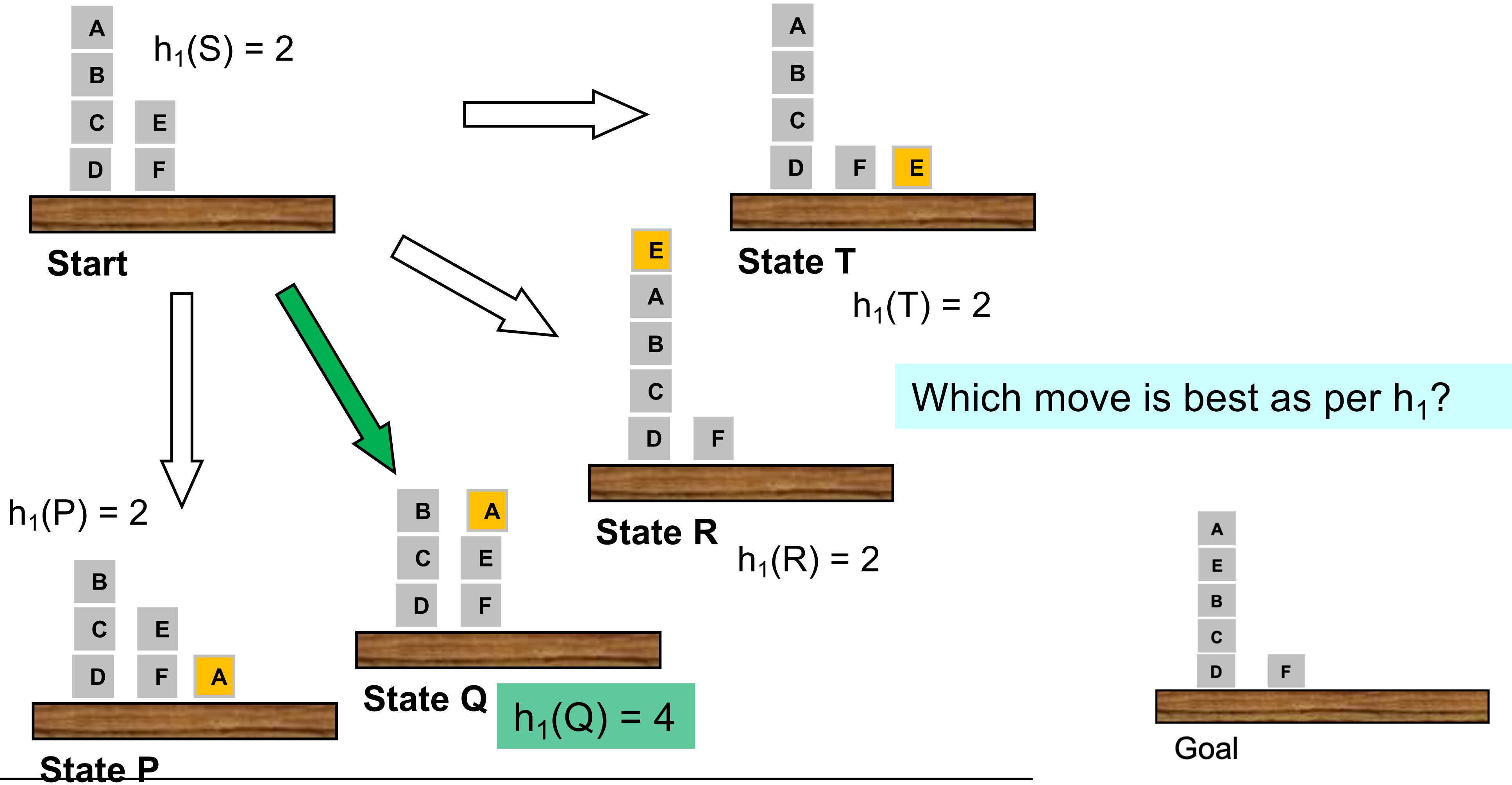
Two heuristic functions:

- h_1 : Add 1 for every block that is on the correct block/table
  Subtract 1 for every block on a wrong block/table
- h_2 : Add *n* if block is on a correct structure of *n* blocks
  Subtract *n* if block is on wrong structure of *n* blocks

$h_2(\text{start}) = 1$
$h_2(\text{goal}) = 16$

Note: a maximization problem

4 Moves from the Start state



$\underline{h_1(n)}$

For the five states, *S* (start) and its successors *P, Q, R,* and *T* the values are,

$h_1(S)$	$= (-1) + 1 + 1 + 1 + (-1) + 1$	=	2
$h_1(P)$	$= (-1) + 1 + 1 + 1 + (-1) + 1$	=	2
$h_1(Q)$	$= 1 + 1 + 1 + 1 + (-1) + 1$	=	4
$h_1(R)$	$= (-1) + 1 + 1 + 1 + (-1) + 1$	=	2
$h_1(T)$	$= (-1) + 1 + 1 + 1 + (-1) + 1$	=	2

where,

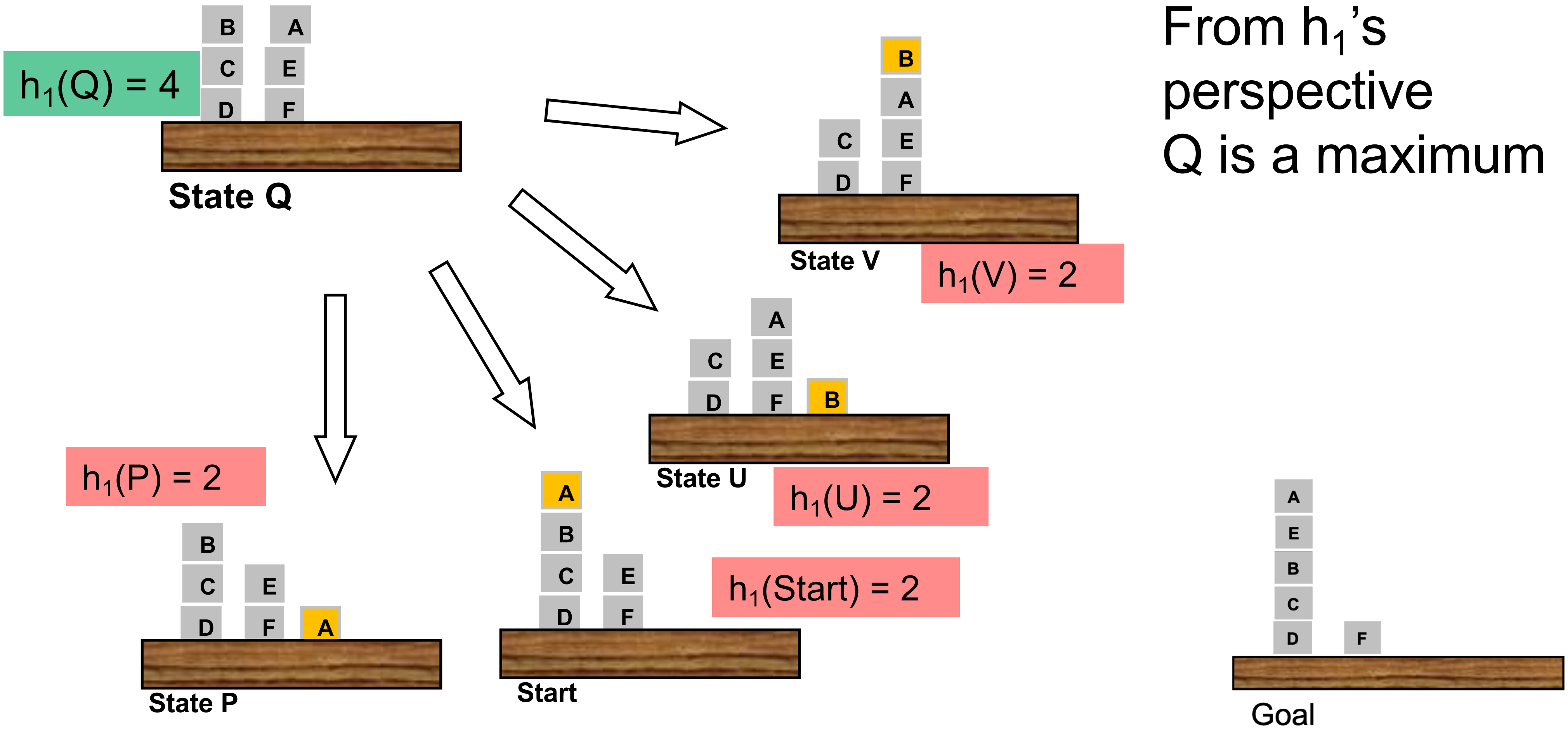
$h_1(n) = val_A + val_B + val_C + val_D + val_E + val_F$

Clearly h_1 thinks that moving to state *Q* is the best idea, because in that state block *A* is on block *E*.

The choices from state Q

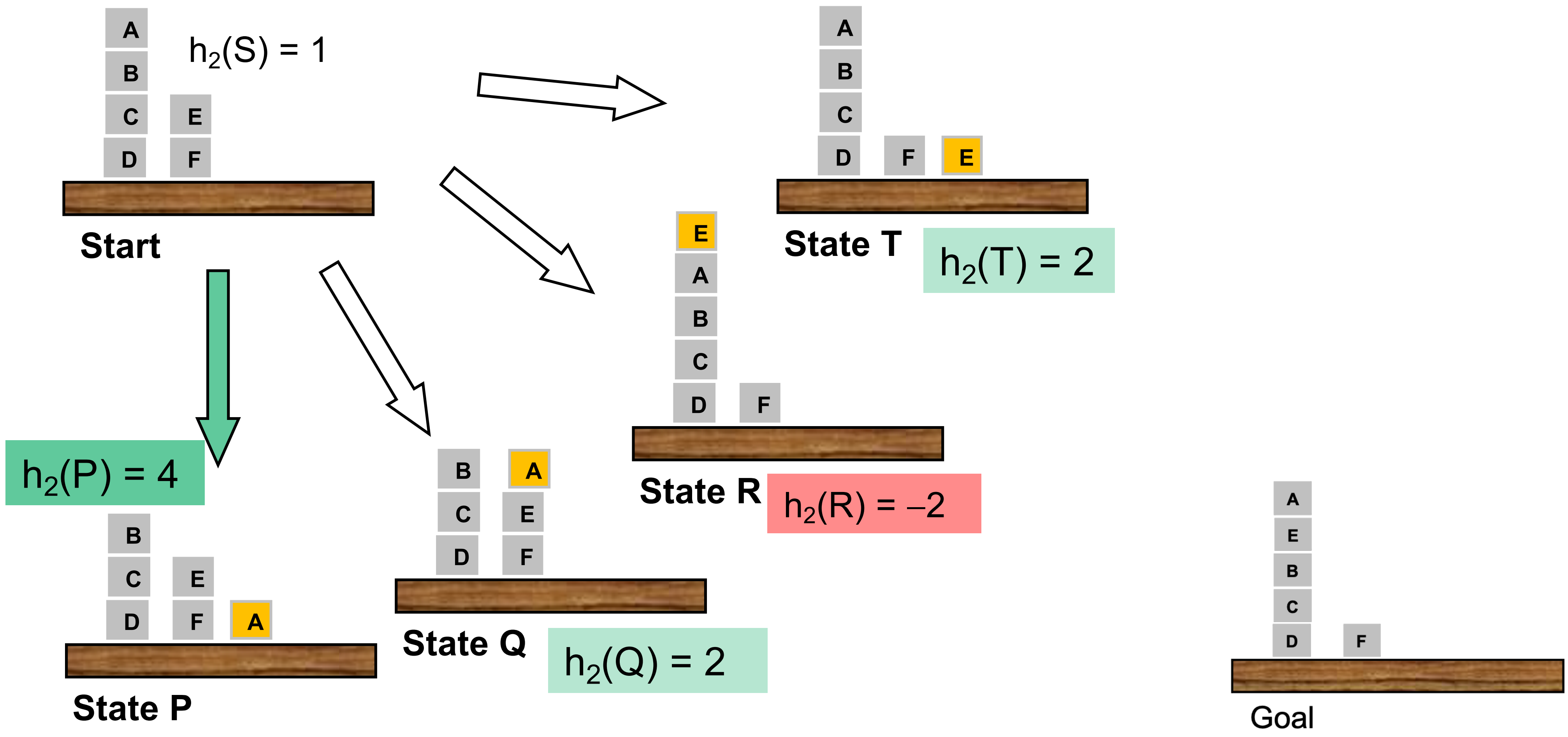
…are all worse than Q

From h_1's perspective Q is a maximum



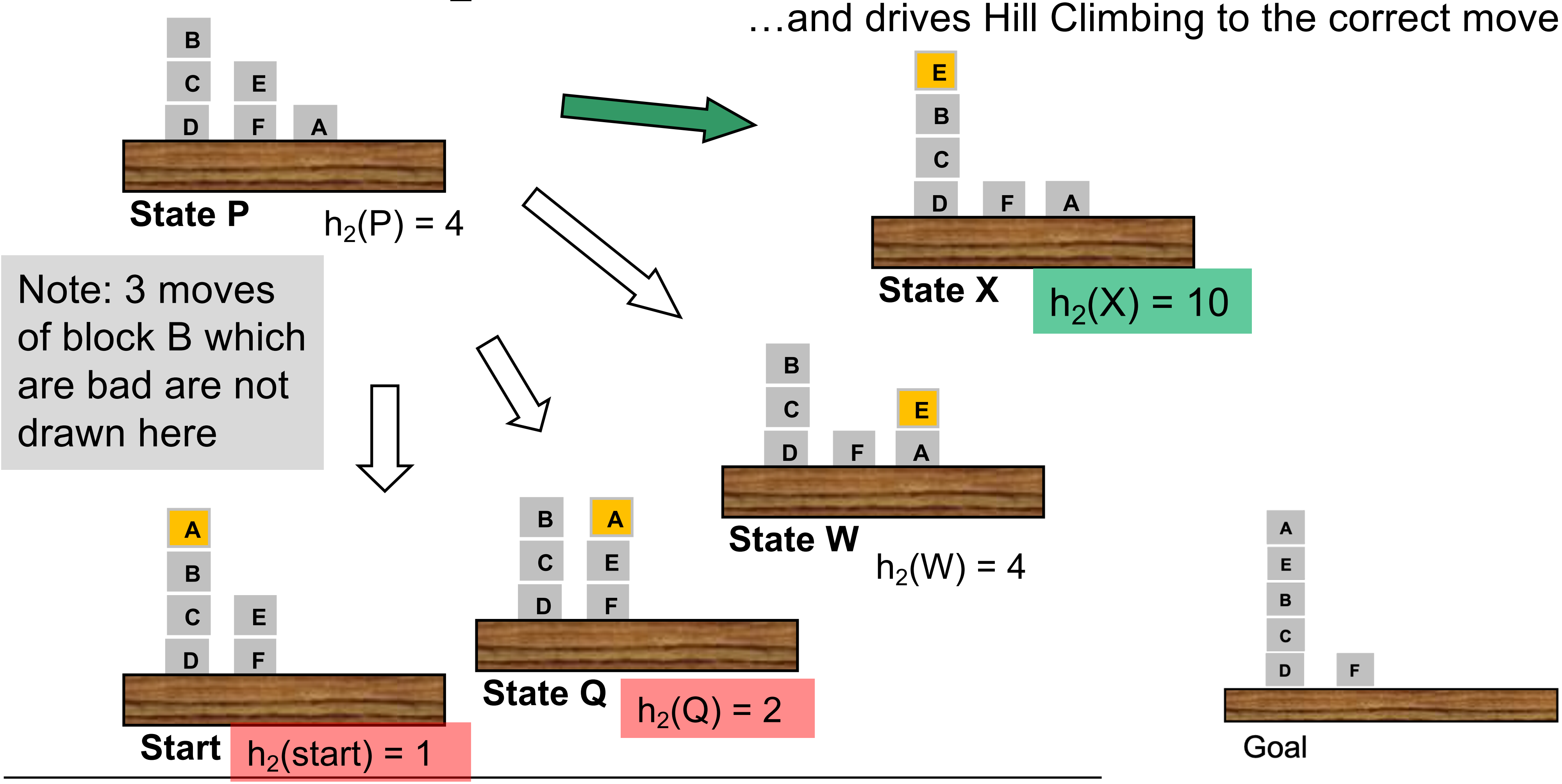
4 Moves from the Start state

Which move is best as per h_2?

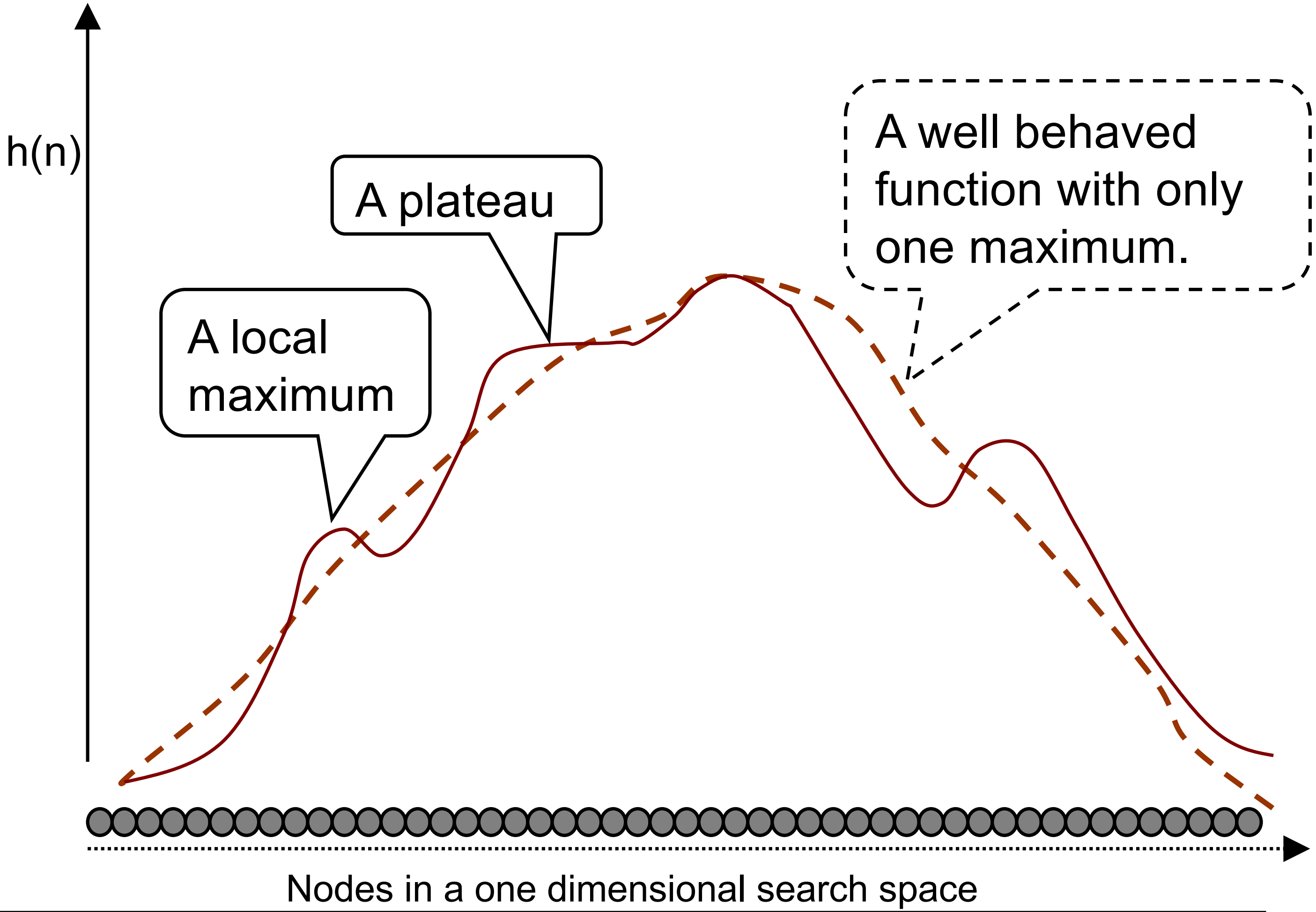


Heuristic function h_2 is more discriminative

…and drives Hill Climbing to the correct move



Heuristic functions *define* the terrain



Escaping local optima

Given that

it is difficult to define heuristic functions
that are monotonic and well behaved

the alternative is

to look for algorithms that can do better than Hill Climbing.

We look at three deterministic methods next,

and will look at randomized methods later

Watch this space….