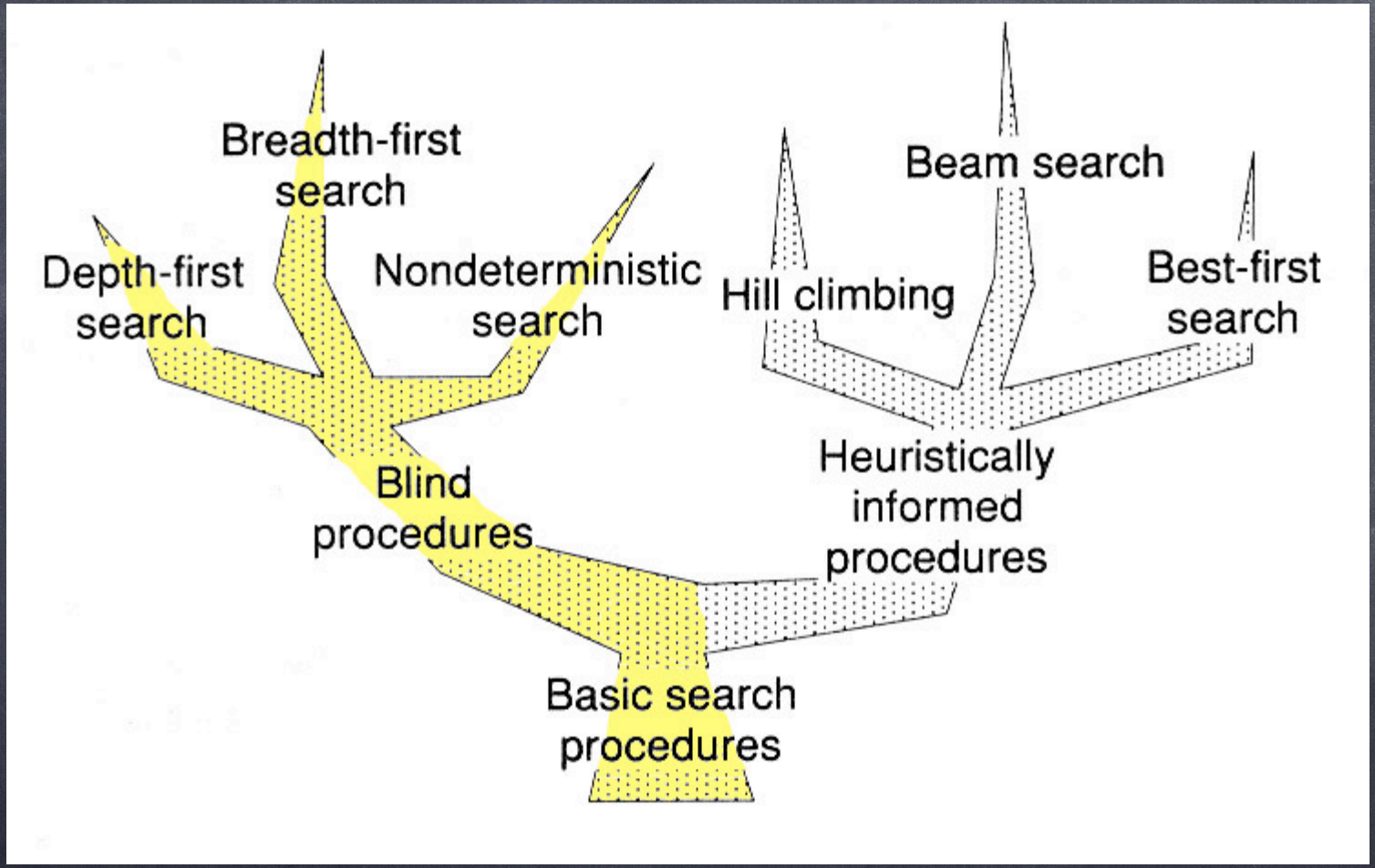
# Informed Search

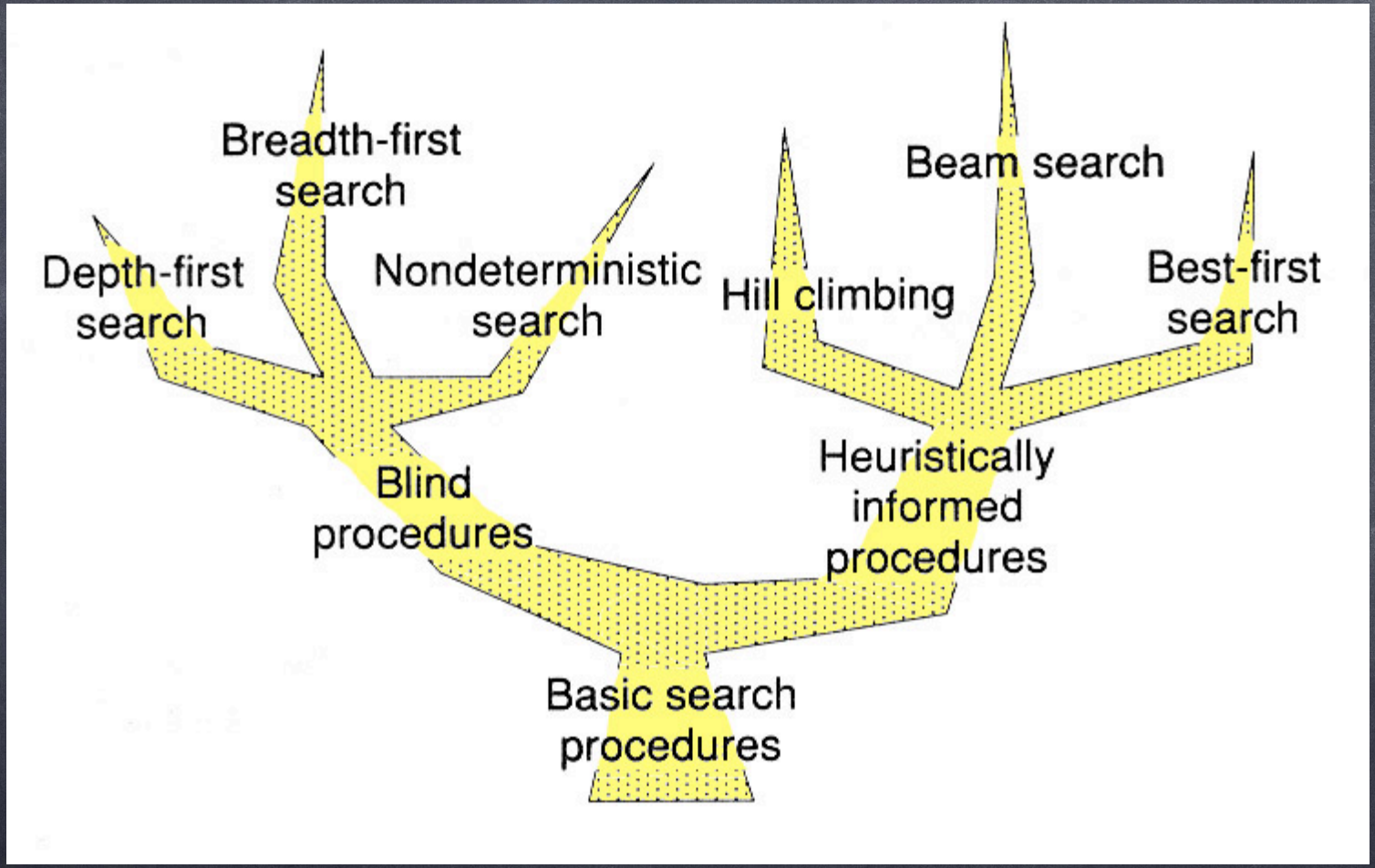
M. Anthony Kapolka III
Wilkes University
CS 340 - Fall 2019

The task that a symbol system is faced with when it is presented with a problem and a problem space, is to use its limited processing resources to generate possible solutions until it finds one that satisfies the problem. If the symbol system could control the order in which potential solutions were generated it would exhibit intelligence if it generated them in an order so actual solutions had a high likelihood of appearing early.

Intelligence for a system with limited processing resources consists in making wise choices of what to do next...

a paraphrase from Newell and Simon's 1976 Turing Award Lecture


Breadth-first search
Depth-first search
Nondeterministic search
Beam search
Hill climbing
Best-first search
Blind procedures
Heuristically informed procedures
Basic search procedures


Breadth-first search
Depth-first search
Nondeterministic search
Beam search
Hill climbing
Best-first search
Blind procedures
Heuristically informed procedures
Basic search procedures


ˈευρισκω

# Heuristic

from the greek `ευρισκω meaning to find

a rule of thumb, a guideline
rather than a strict algorithm.

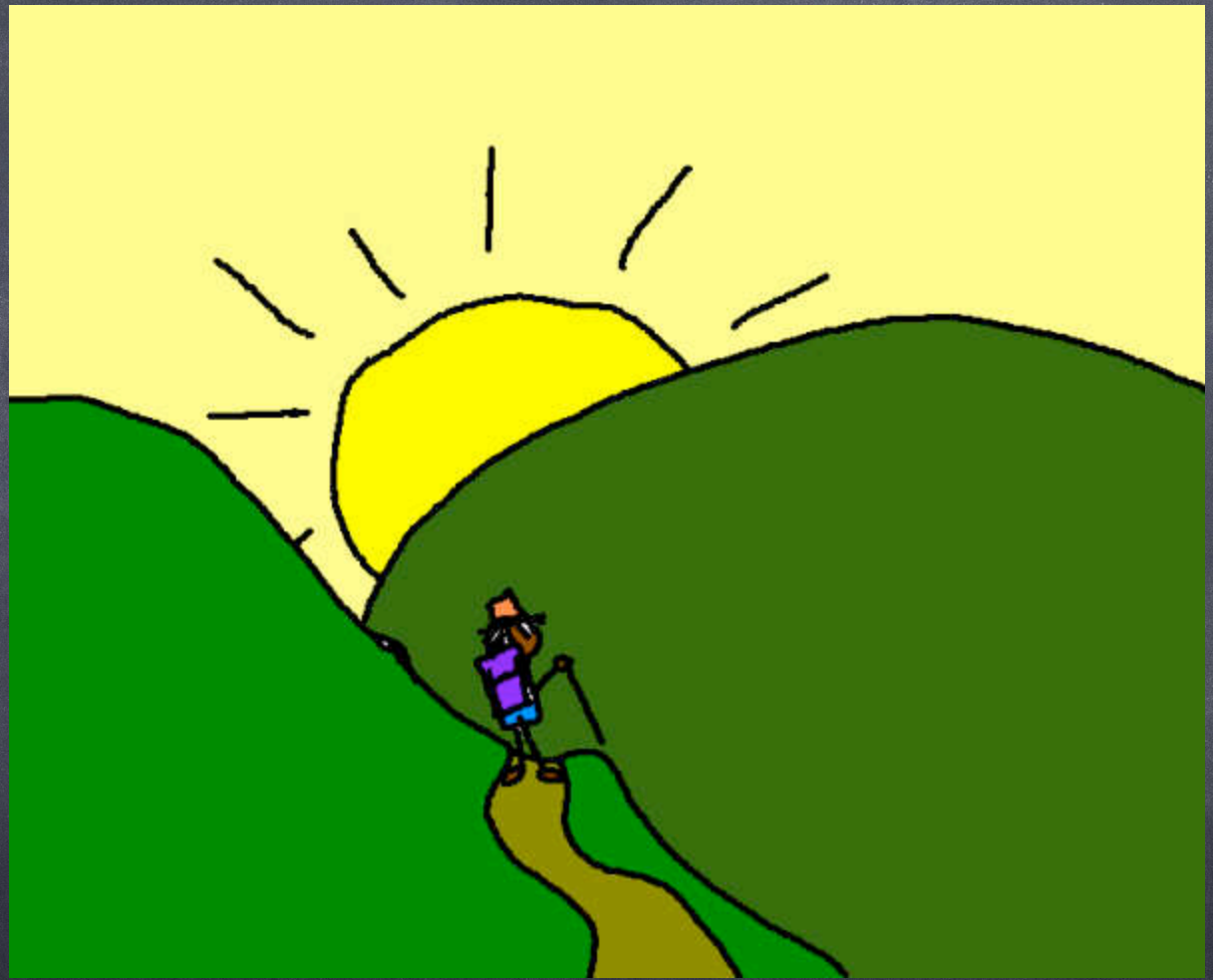
- control the center of the chess board.
- more expensive beer tastes better.

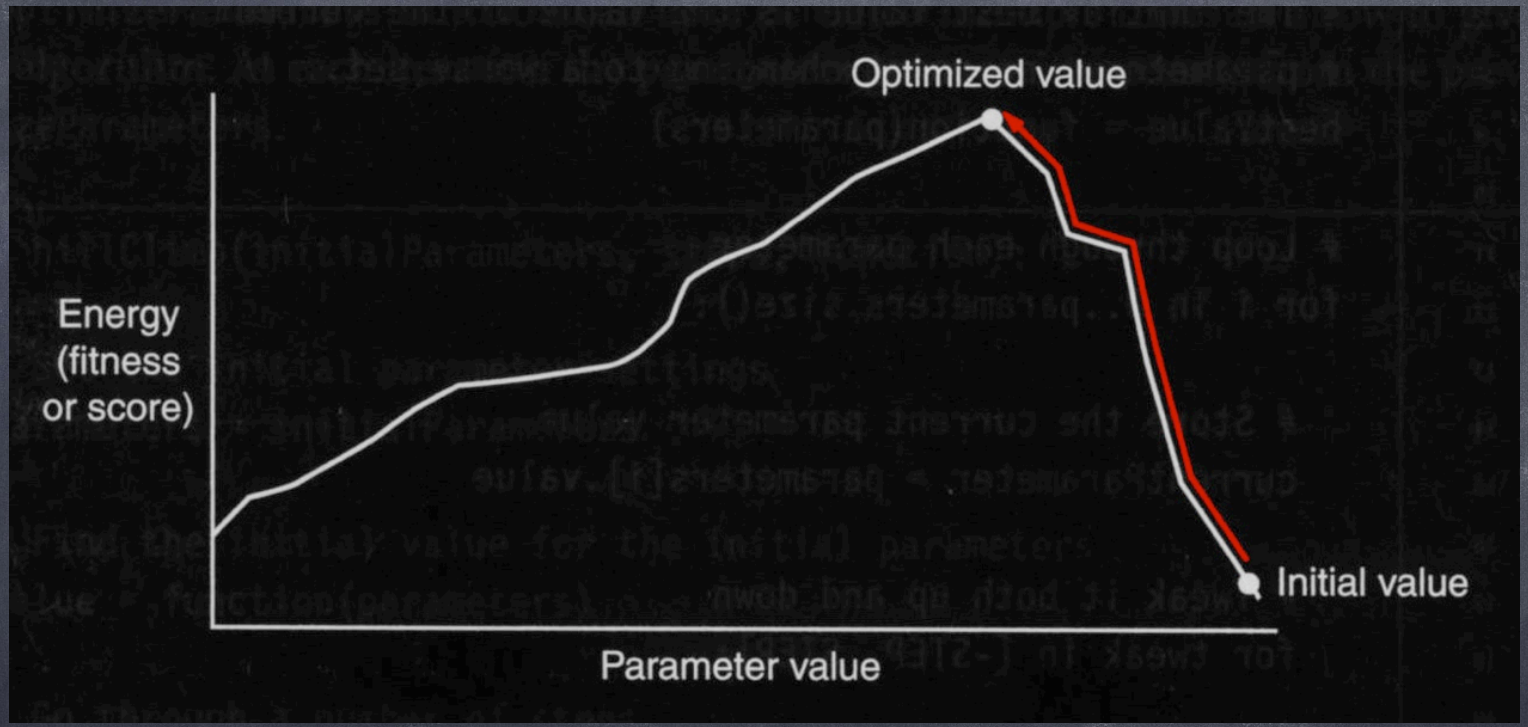
# Use heuristics if...

- no exact solution exists
- finding exact solution too expensive

heuristics are fallible

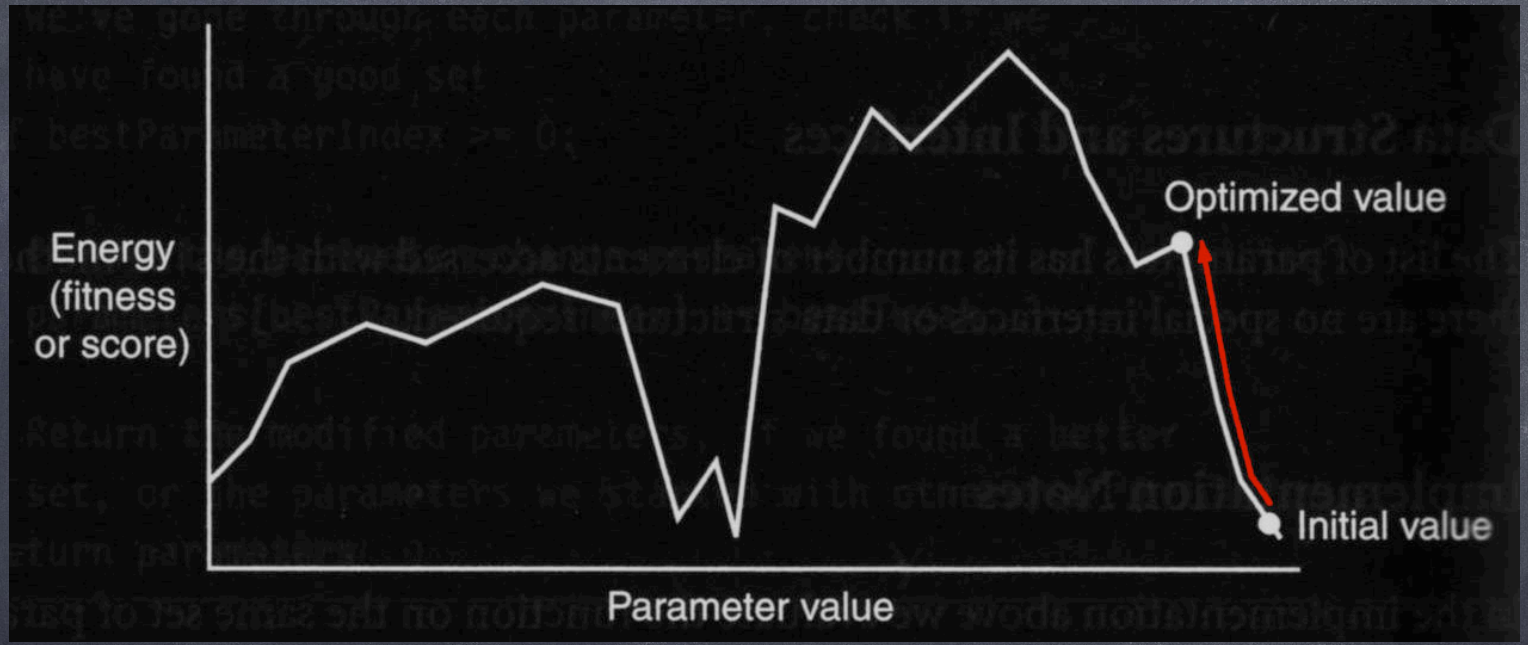
# Hill Climbing



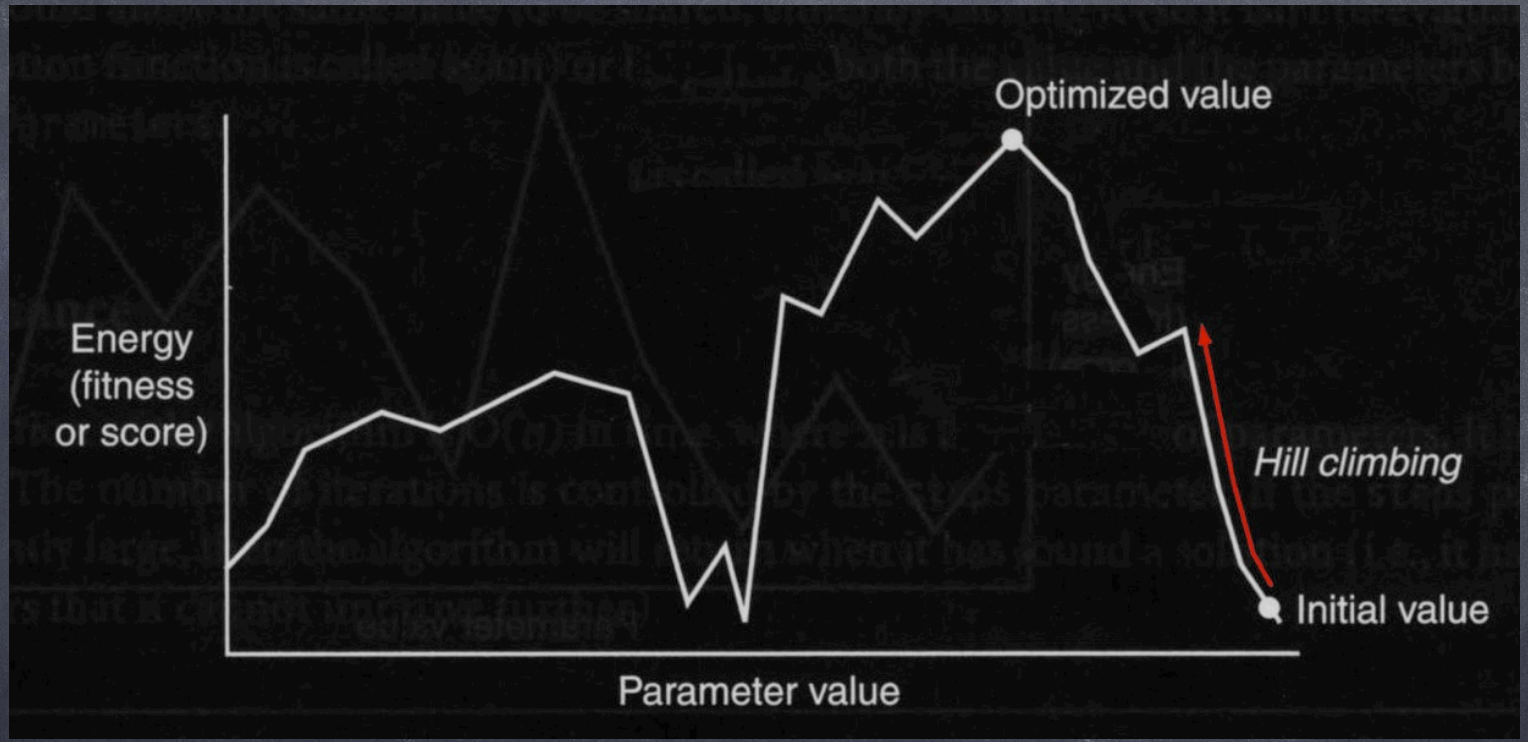



AI for Games, Millington & Funge: Figure 7.2

# Sub-optimal Results

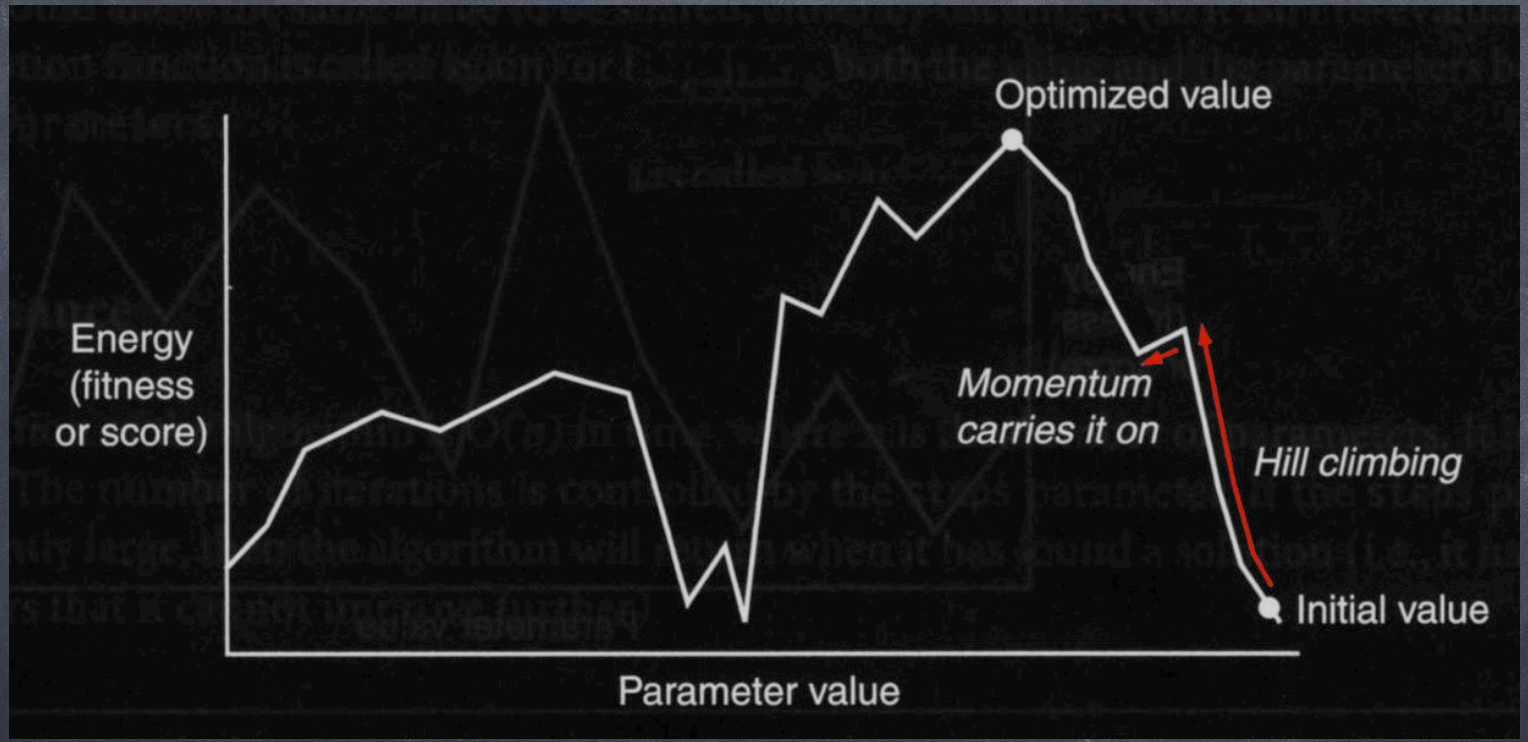



AI for Games, Millington & Funge: Figure 7.3



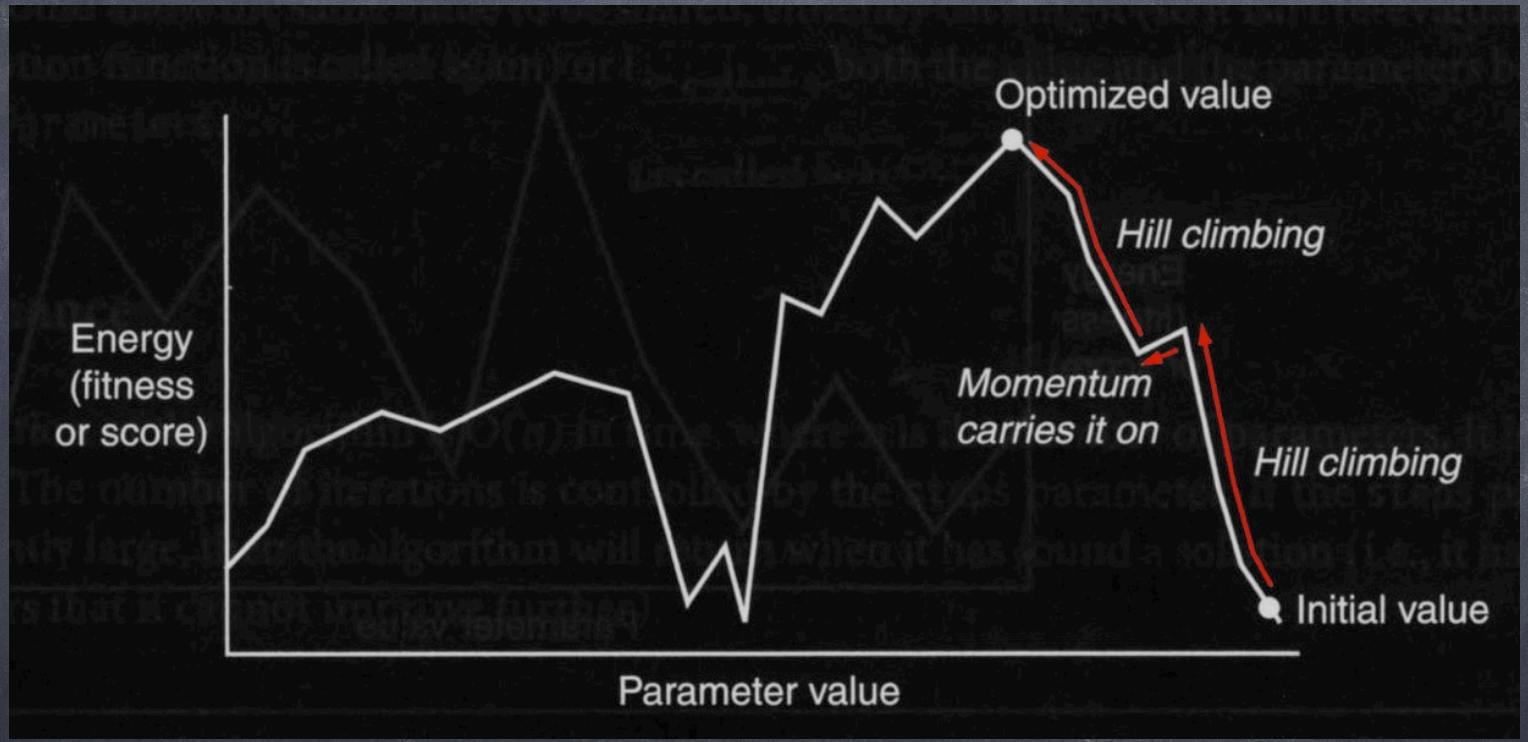

AI for Games, Millington & Funge: Figure 7.5

# Introduce Momentum...



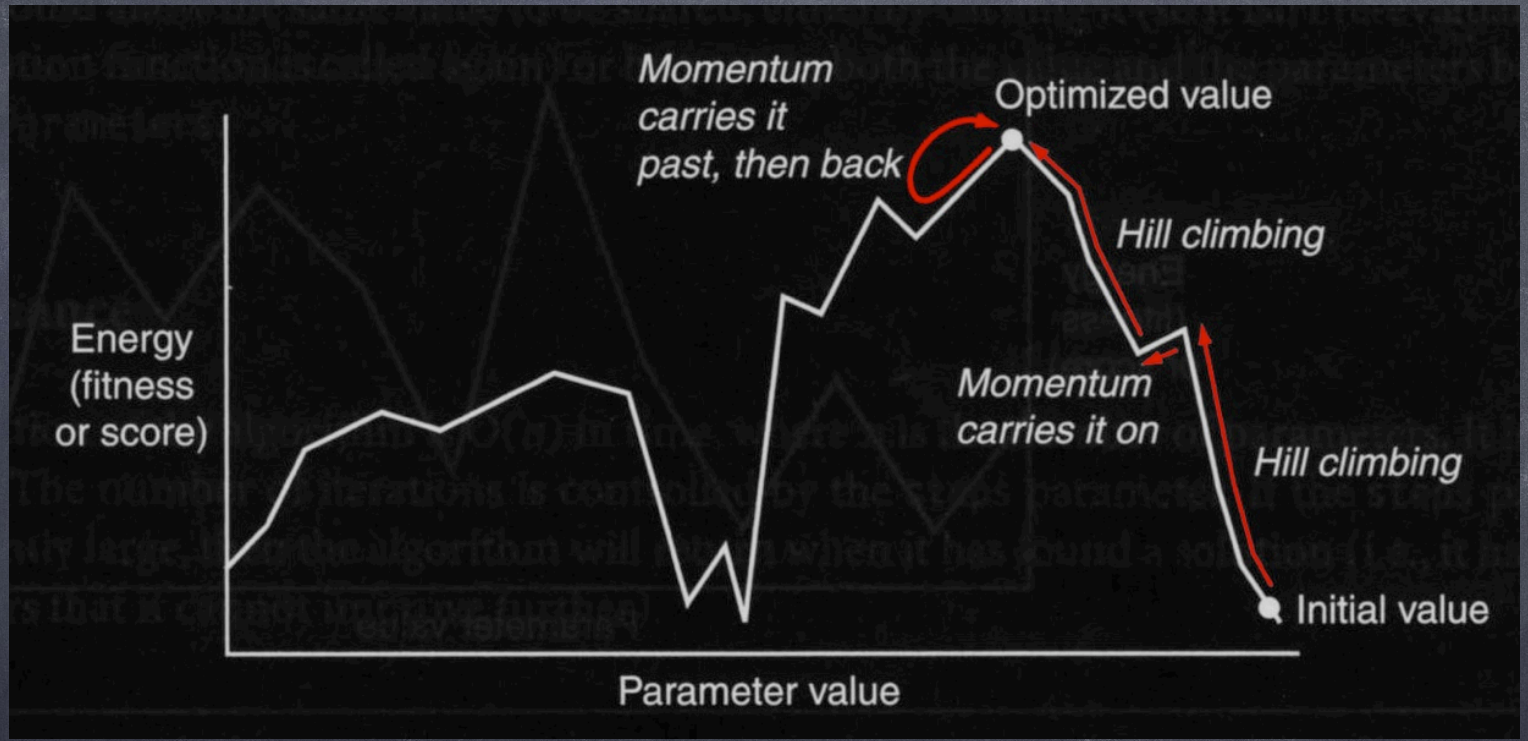

AI for Games, Millington & Funge: Figure 7.5

# Introduce Momentum…



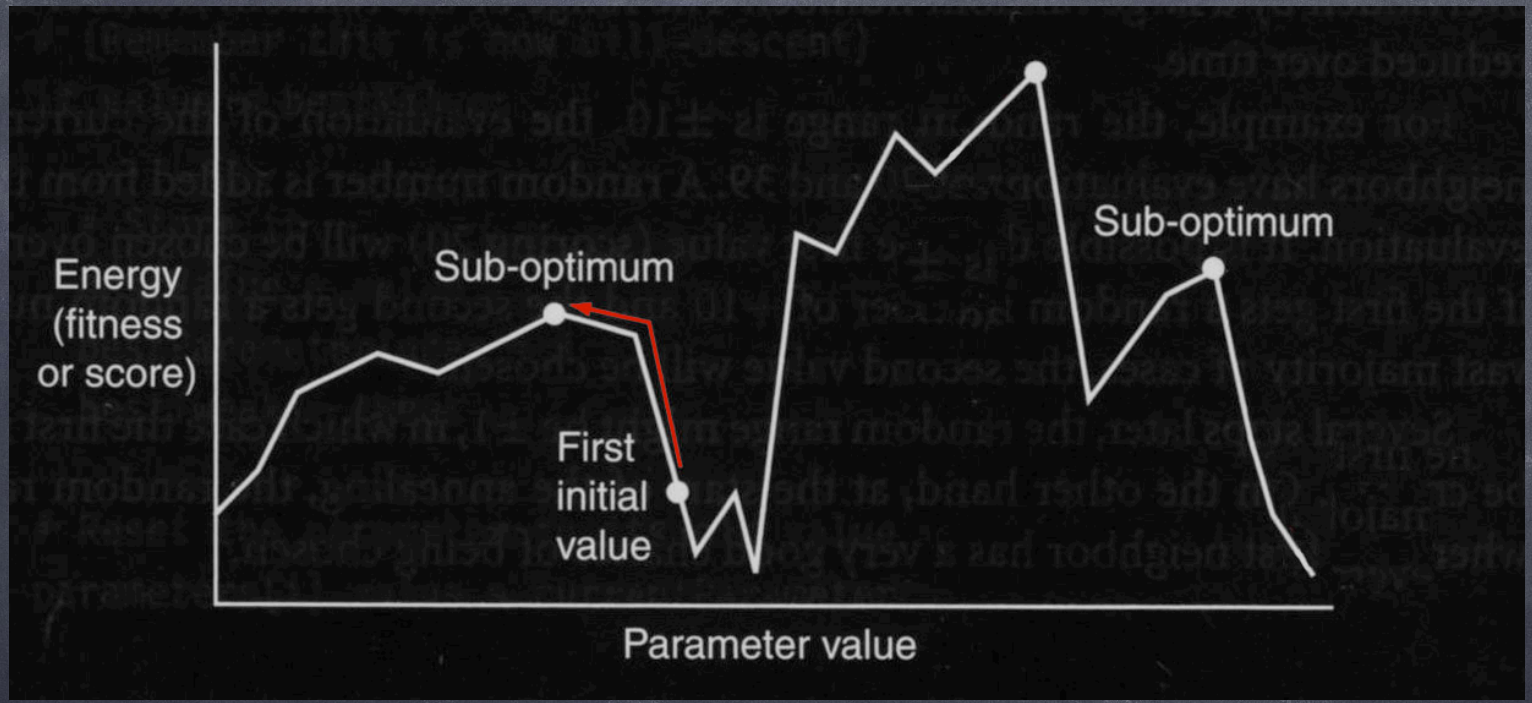

AI for Games, Millington & Funge: Figure 7.5

# Introduce Momentum…



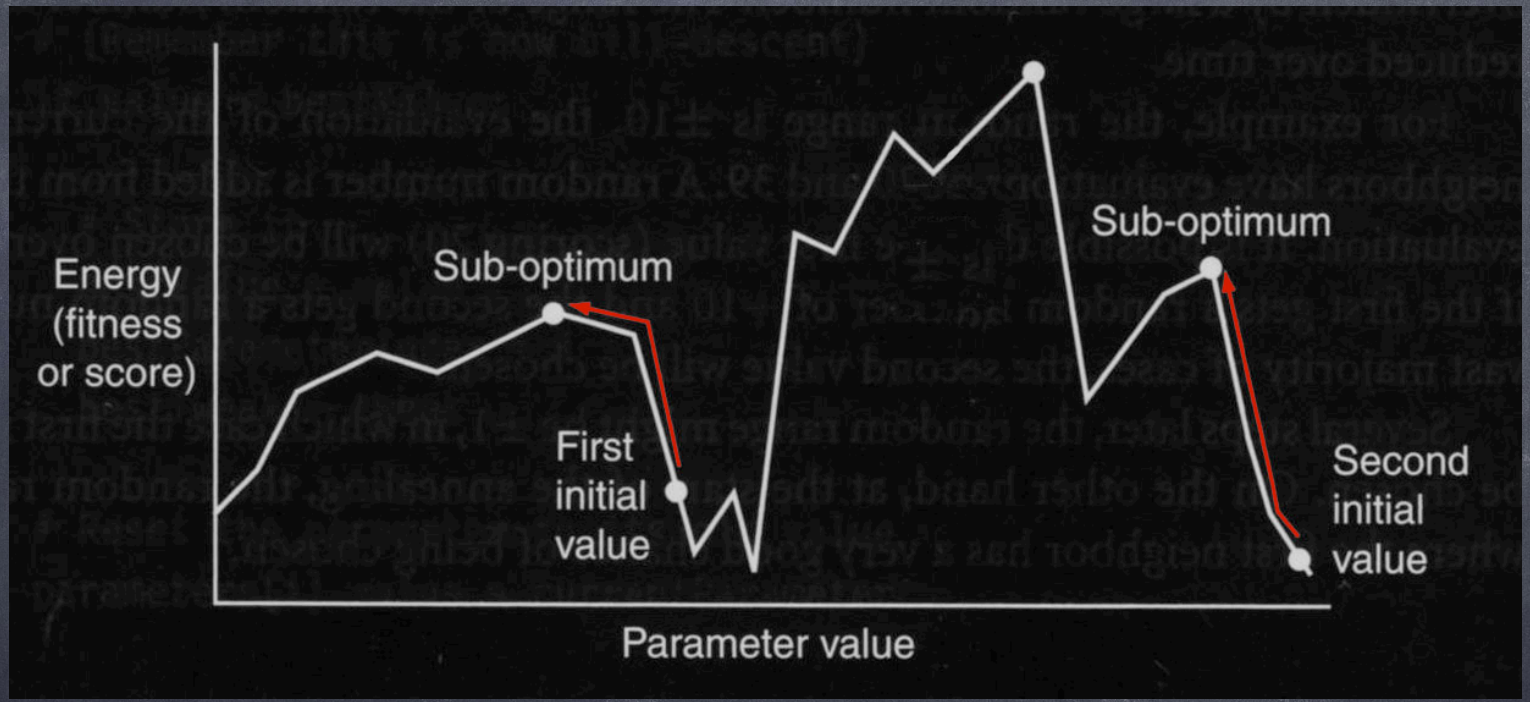

AI for Games, Millington & Funge: Figure 7.5

# 'Best of Three' Heuristic



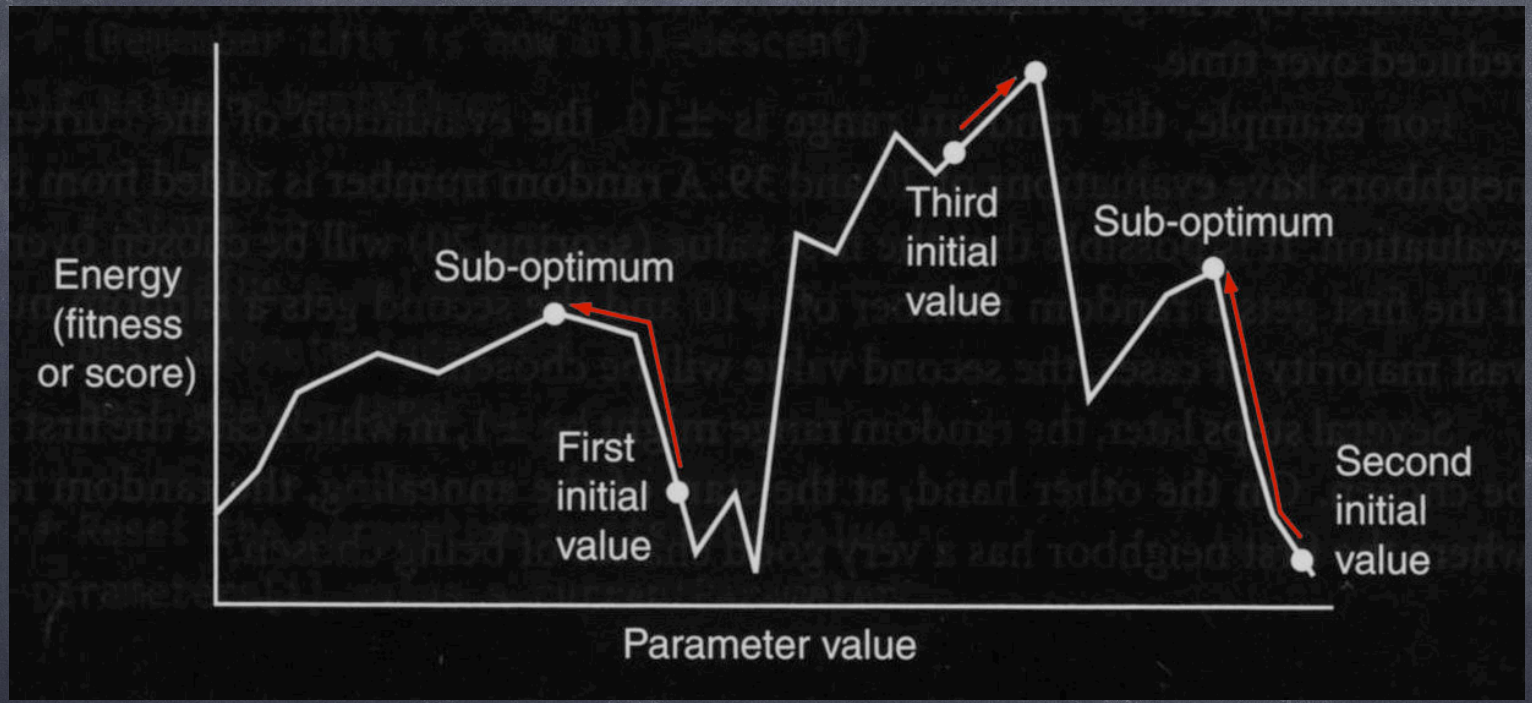

AI for Games, Millington & Funge: Figure 7.6

# 'Best of Three' Heuristic




AI for Games, Millington & Funge: Figure 7.6

# 'Best of Three' Heuristic




AI for Games, Millington & Funge: Figure 7.6

# Beam Search

# Beam Search

- Uses a heuristic evaluation function
- Like breadth-first search
  but explores only M best nodes
  prunes nodes not explored
- Not optimal, not complete

```
function beam-search
   open := [Start]
   done := [ ]
   while open != [ ] do
      X := open.dequeue()
      if (X = Goal) return success
         else
          generate children states of X
          discard any children in open, done
          for best M children C
              open.enqueue(C)
          done.add(X)
return failure
```

# Best First Search

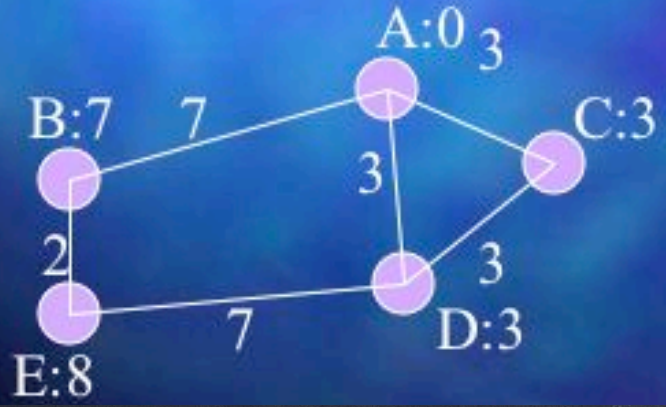
save all the children!

# Best First Search

- Expands tree heuristically
- Nodes to expand in best-first order
- can be like depth or breadth first
- Keeps all nodes (unlike beam search)

# Best-First Searches

- More accurately, '***estimated* best-first searches'.**
- Estimate the ***cost*** of the next nodes from a given node: e.g., $h(n)$ = the straight-line distance to goal, or something more complex.
- Are ***complete***: eventually finds the goal if its in state space
- Now, however, they ***queue*** all the nodes ***in order*** of the estimated best first, where 'best' is defined according to some heuristic. E.g. consider this graph:

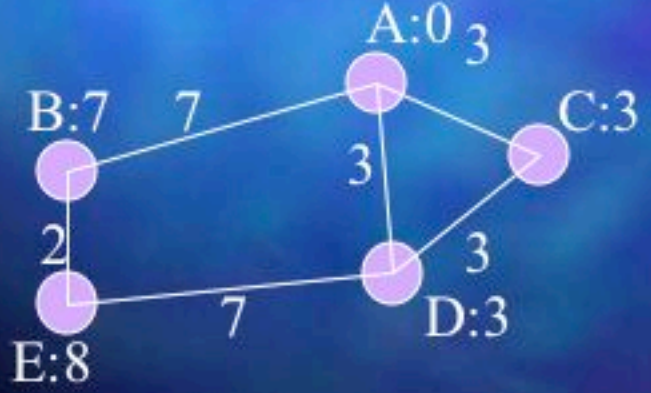



| Node distance |
| --- |
| A: C:3, D:3, B:7 |
| B: E:2, A:7 |
| C: A:3, D:3 |
| D: C:3, A:3, E:7 |
| E: B:2, D:7 |

| Straight Line Dist. from A |
| --- |
| B: A:0, E:8 |
| C: A:0,D:3 |
| D: A:0,C:3,E:8 |
| E: D:3,B:7 |

# Greedy Search

- Choose the least cost node first, here using $h(n_i)$ = sld.
- If we start from E, and the goal is A, then the route is E, D, A:
  - The nodes available from E are B (cost 7) and D (cost 3), so lower cost D is chosen. Remembers B's cost, in case the route via D is bad.
  - The nodes available from D are A (cost 0), C (cost 3) and E (cost 8), so lowest cost A is chosen and the goal is reached.

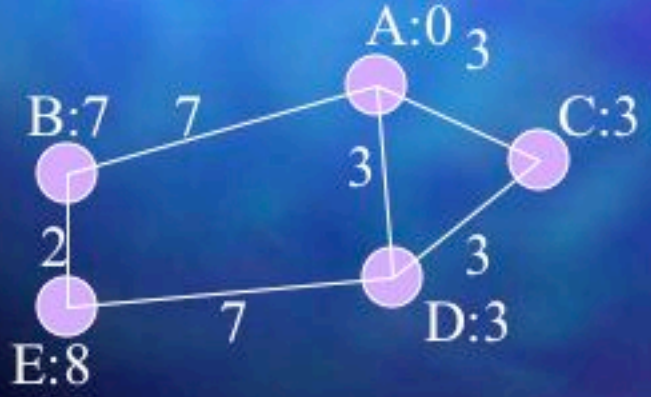



| Node distance |
| --- |
| A: C:3, D:3, B:7 |
| B: E:2, A:7 |
| C: A:3, D:3 |
| D: C:3, A:3, E:7 |
| E: B:2, D:7 |

| Straight Line Dist. from A |
| --- |
| B: A:0, E:8 |
| C: A:0,D:3 |
| D: A:0,C:3,E:8 |
| E: D:3,B:7 |

# Problem with Greedy Search

- The problem with greedy search is that it doesn't take notice of the distance it travels to get to a low cost node and can end up with sub-optimal solutions.
- In this case the route E,D,A is a little longer (10) than route E,B,A (9).



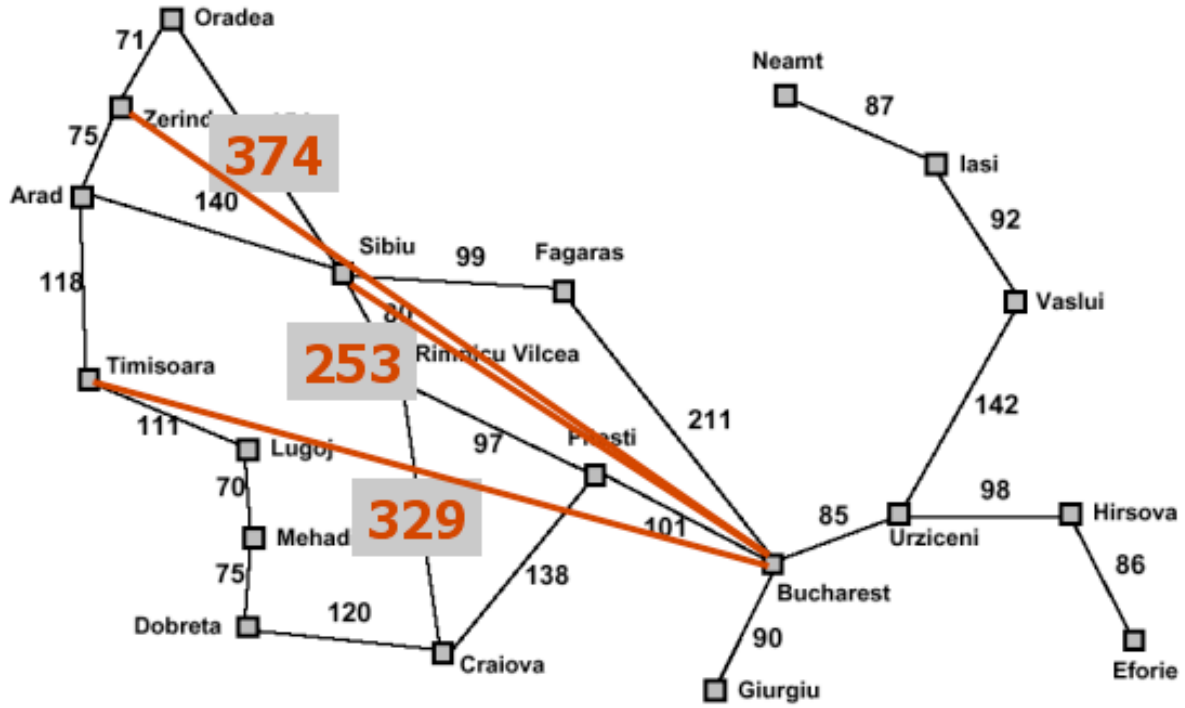

| Node distance |
| --- |
| A: C:3, D:3, B:7<br>B: E:2, A:7<br>C: A:3, D:3<br>D: C:3, A:3, E:7<br>E: B:2, D:7 |

| Straight Line Dist. from A |
| --- |
| B: A:0, E:8<br>C: A:0,D:3<br>D: A:0,C:3,E:8<br>E: D:3,B:7 |

Another example...

let's travel from Arad to Bucharest

# Romania with step costs in km




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 |

## Greedy search

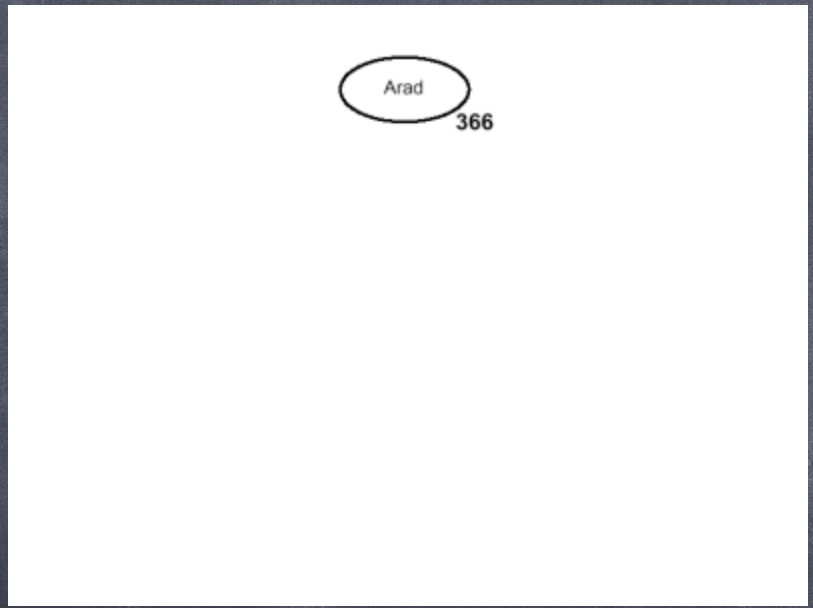
- Estimation function:

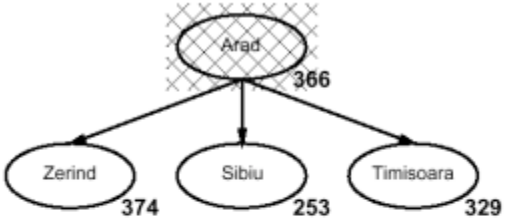
  $h(n)$ = estimate of cost from $n$ to goal (heuristic)

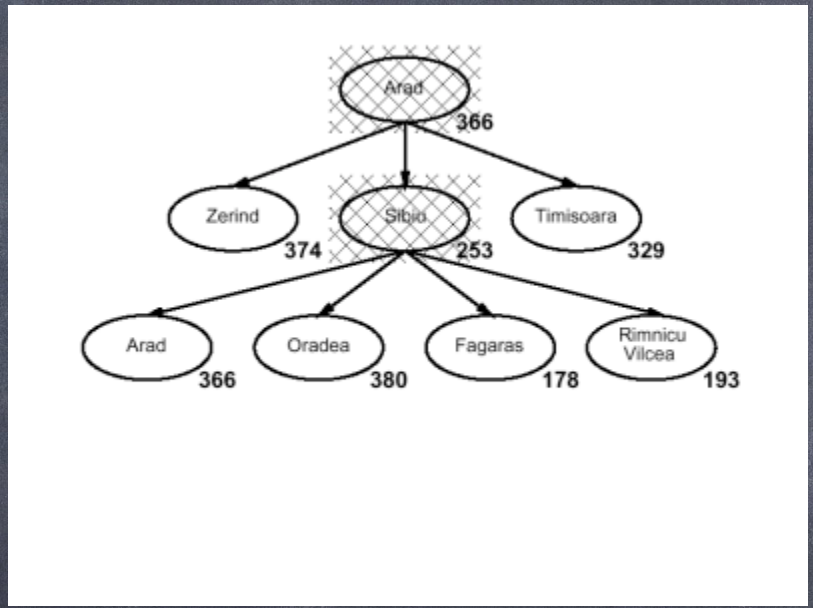
- For example:

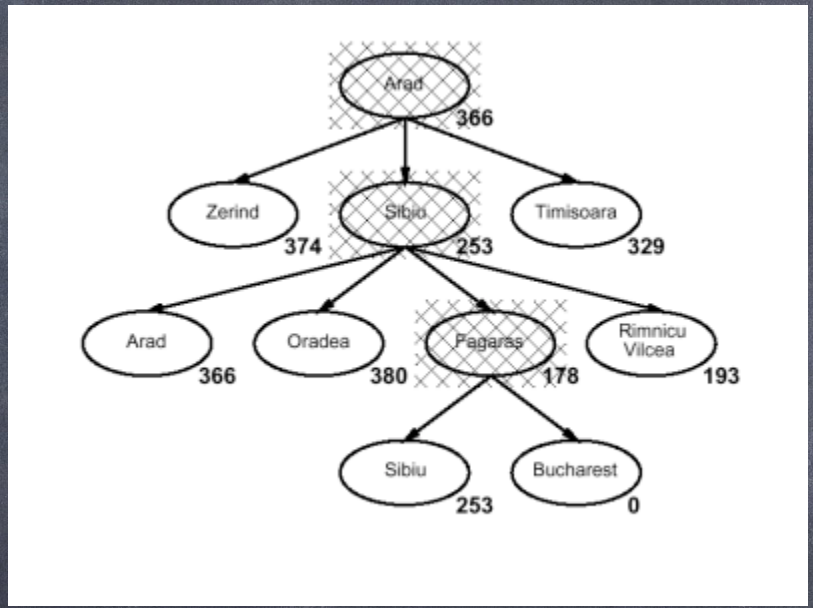
  $h_{SLD}(n)$ = straight-line distance from $n$ to Bucharest

- Greedy search expands first the node that appears to be closest to the goal, according to $h(n)$.


Arad
366


Arad
366
Zerind
374
Sibiu
253
Timisoara
329

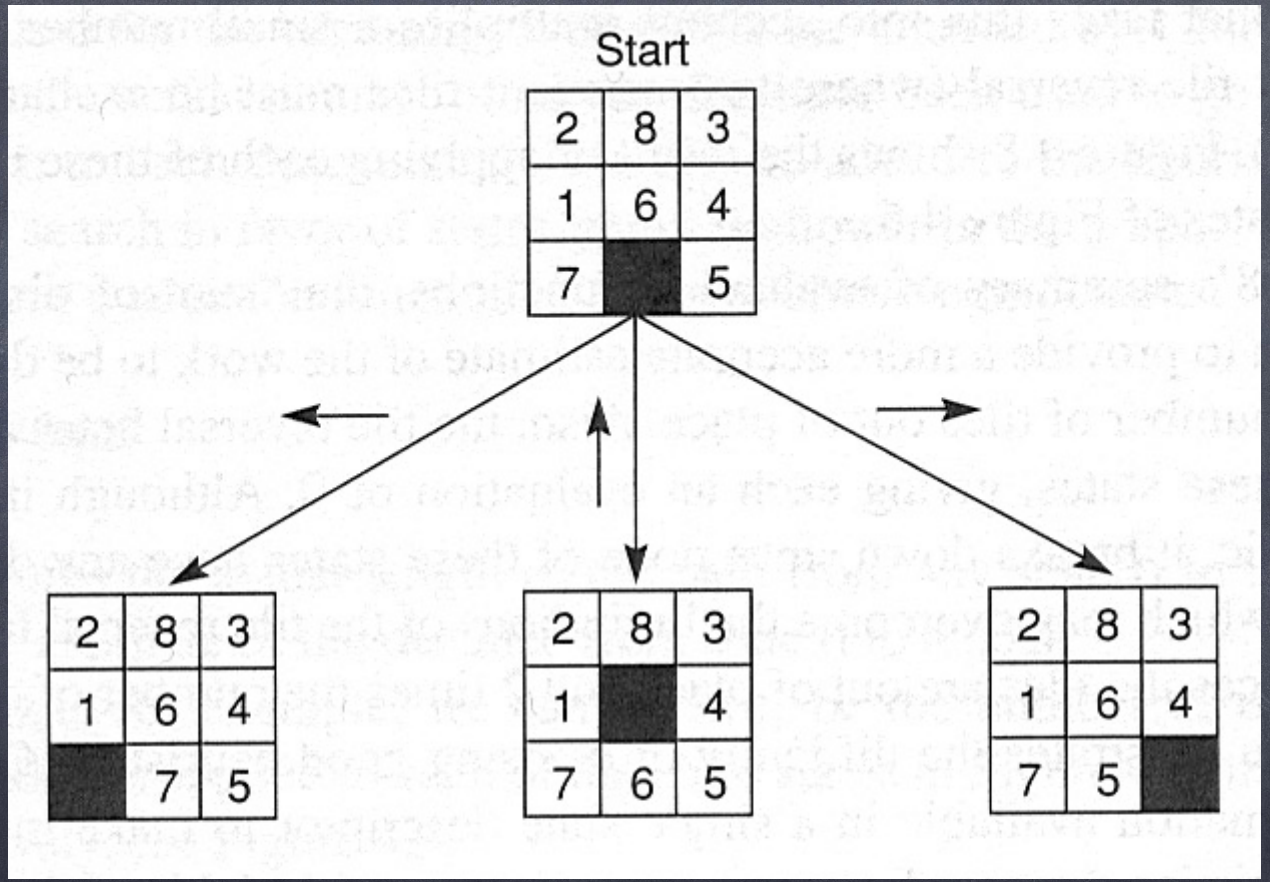

Arad
366
Zerind
374
Sibiu
253
Timisoara
329
Arad
366
Oradea
380
Fagaras
178
Rimnicu
Vilcea
193


Arad
366
Zerind
374
Sibiu
253
Timisoara
329
Arad
366
Oradea
380
Pagaras
178
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0

# Properties of Greedy Search

- Complete ?
  - Can get stuck in loops without repeat checking
  - Complete in finite space with repeat checking
- Time ?
  - $O(b^m)$, but good heuristic improves performance
- Space ?
  - $O(b^m)$, keeps all nodes in memory
- Optimal ? - No

# 8 PUZZLE




[Luger 2002]

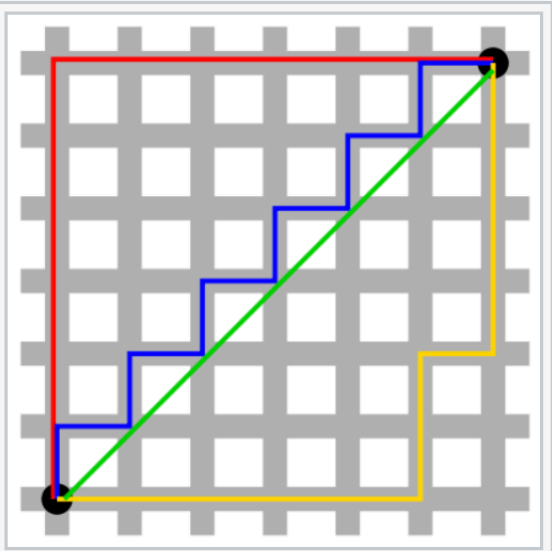
# Possible Heuristics

- Count number of tiles out of place
- Sum all the distances the tiles must move

# Taxicab geometry

From Wikipedia, the free encyclopedia

A **taxicab geometry** is a form of geometry in which the usual distance function or metric of Euclidean geometry is replaced by a new metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. The **taxicab metric** is also known as **rectilinear distance**, $L_1$ **distance**, $L^1$ **distance** or $\ell_1$ **norm** (see $L^p$ space), **snake distance**, **city block distance**, **Manhattan distance** or **Manhattan length**, with corresponding variations in the name of the geometry.[1] The latter names allude to the grid layout of most streets on the island of Manhattan,

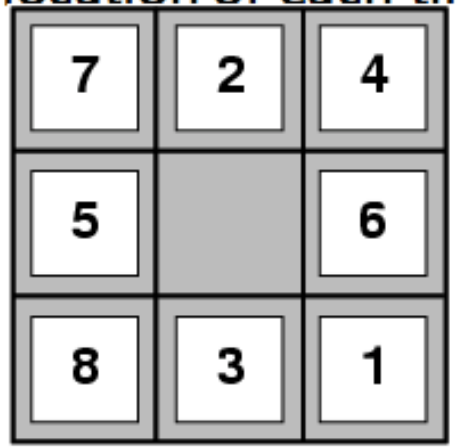


Taxicab geometry versus Euclidean distance: In taxicab geometry, the red, yellow, and blue paths all have the same shortest path length of 12. In Euclidean geometry, the green line has length $6\sqrt{2} \approx 8.49$ and is the unique shortest path.

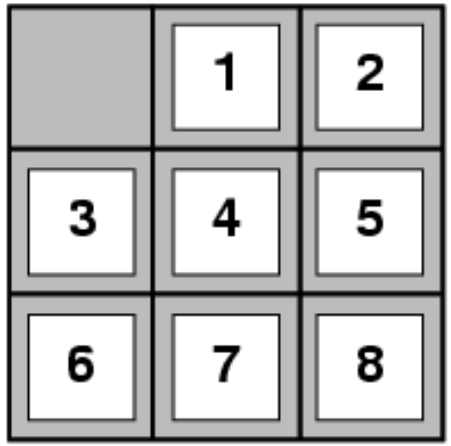
E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)

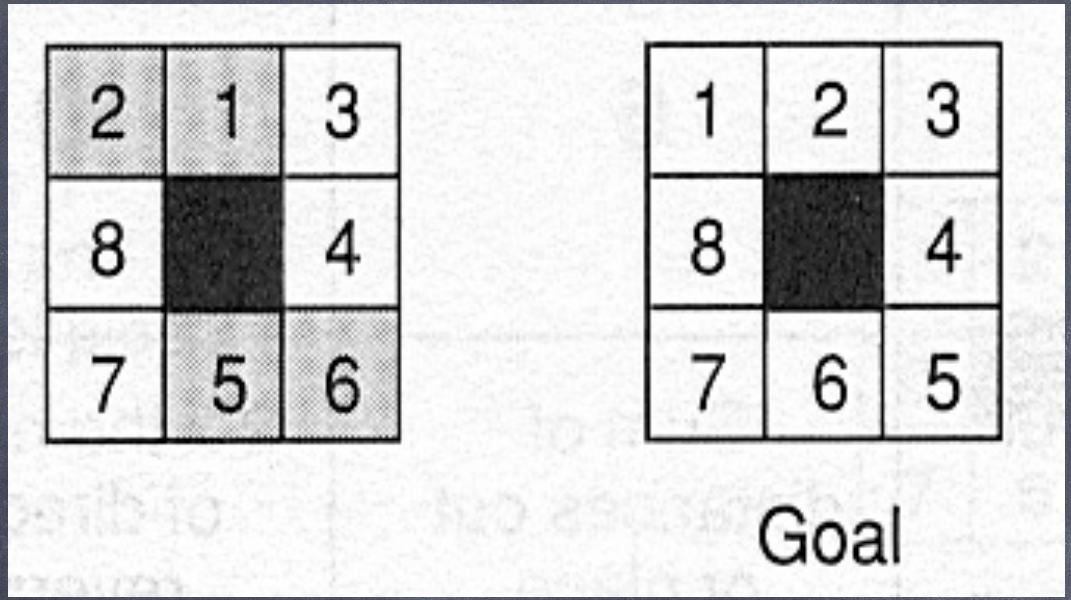



Start State




Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

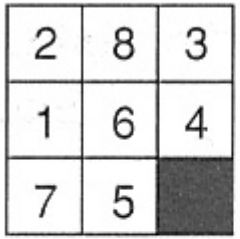



# difficult reversals!

[Luger 2002]

# Possible Heuristics

- Count number of tiles out of place
- Sum all the distances the tiles must move
- Recognize very hard to swap adjoining tiles

| | | | |
|---|---|---|---|
|  | 5 | 6 | 0 |
|  | 3 | 4 | 0 |
|  | 5 | 6 | 0 |
| | Tiles out of place | Sum of distances out of place | 2 x the number of direct tile reversals |

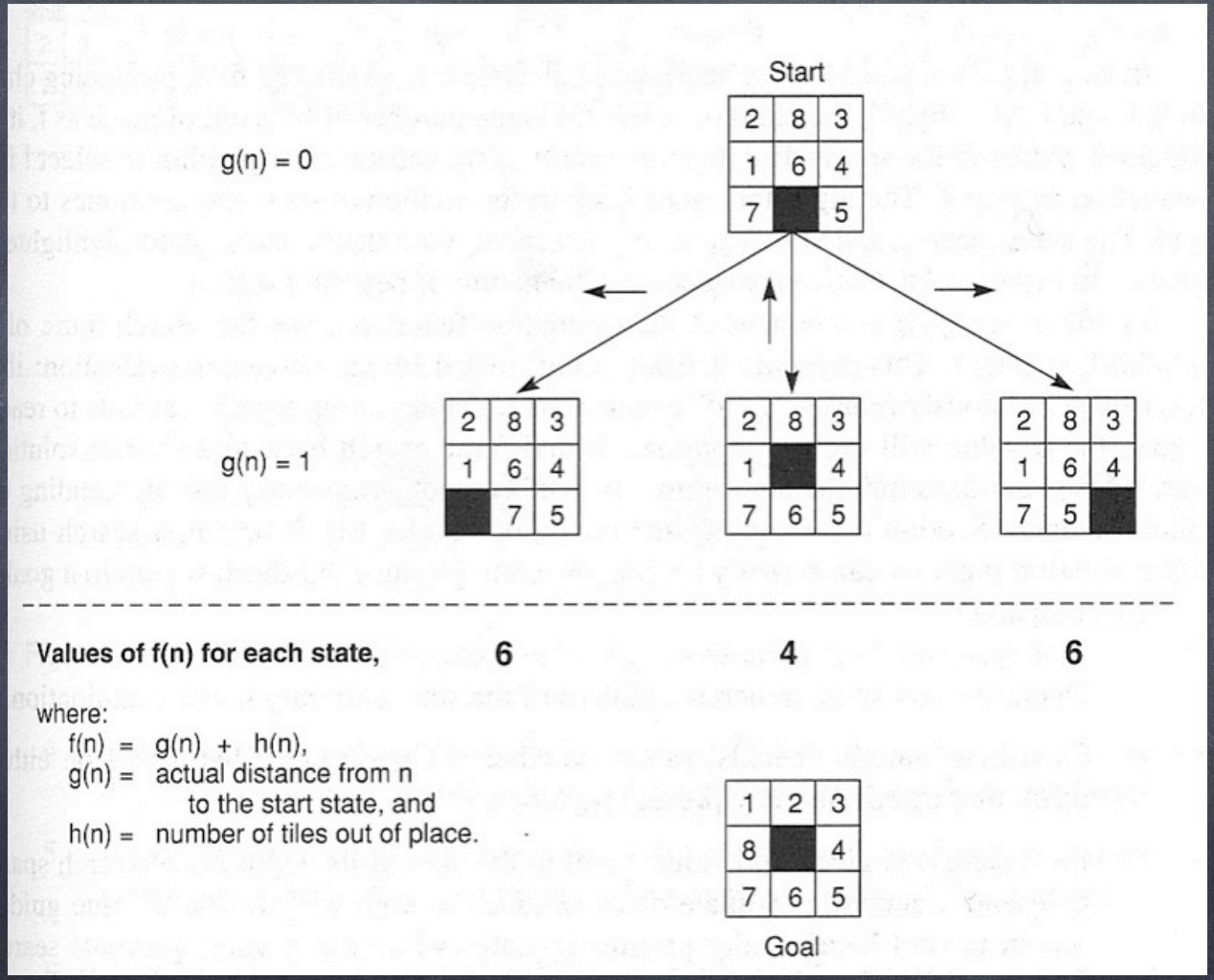
[Luger 2002]

# Evaluation function

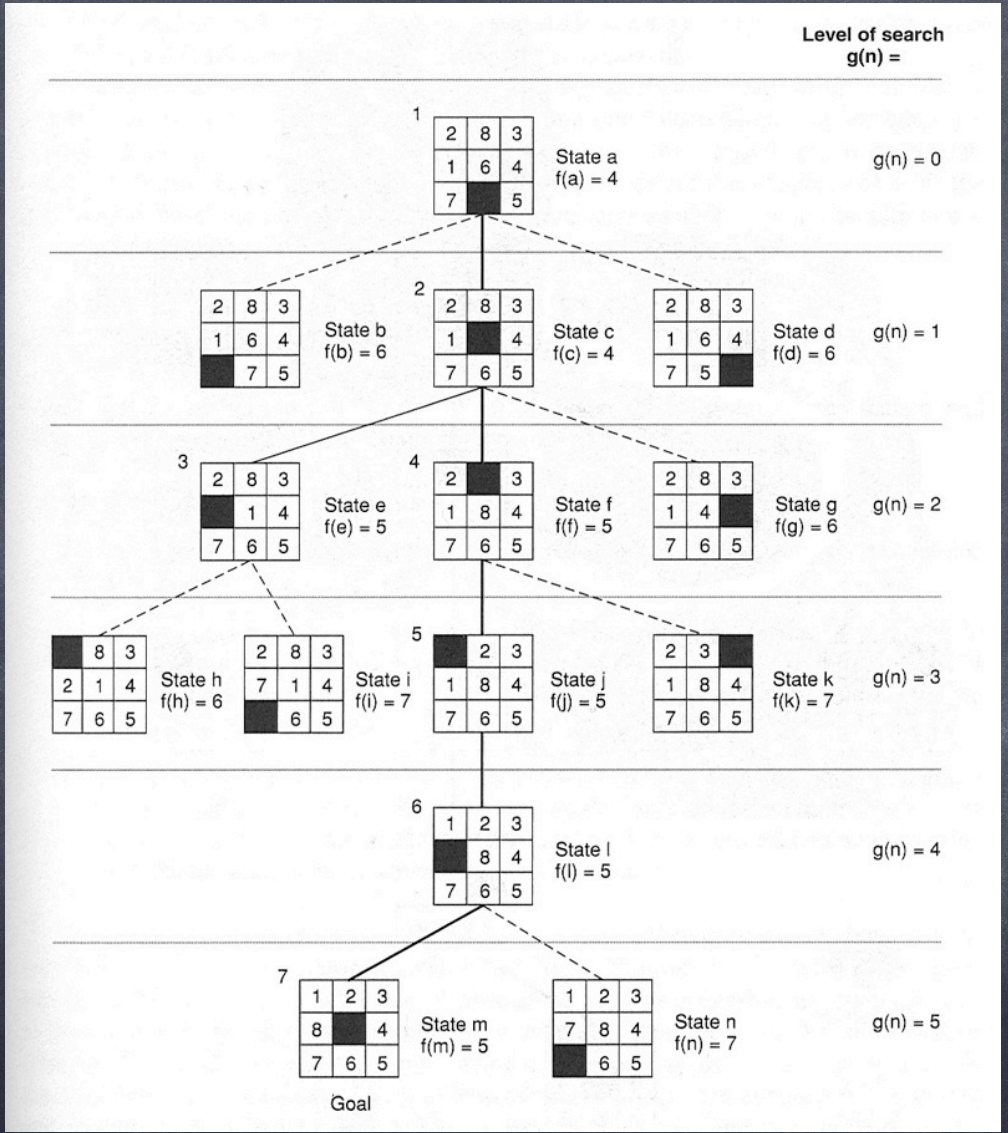
- Can combine multiple heuristics

  e.g. distance + (2 x reversals)

- Force exploration of shallower solutions by adding a path-length cost

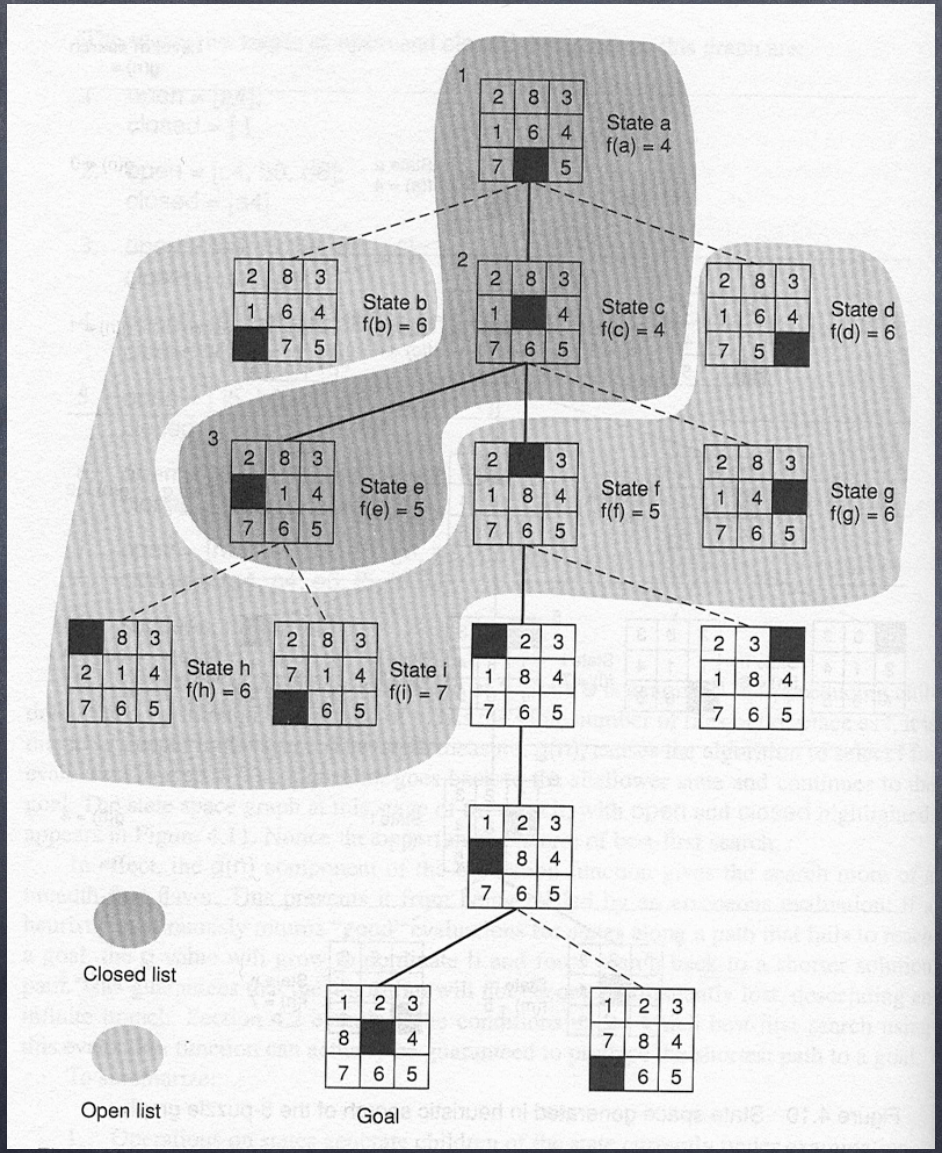

Start
g(n) = 0
2 8 3
1 6 4
7 5
g(n) = 1
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 6 4
7 5
Values of f(n) for each state, 6 4 6
where:
f(n) = g(n) + h(n),
g(n) = actual distance from n to the start state, and
h(n) = number of tiles out of place.
1 2 3
8 4
7 6 5
Goal

[Luger 2002]

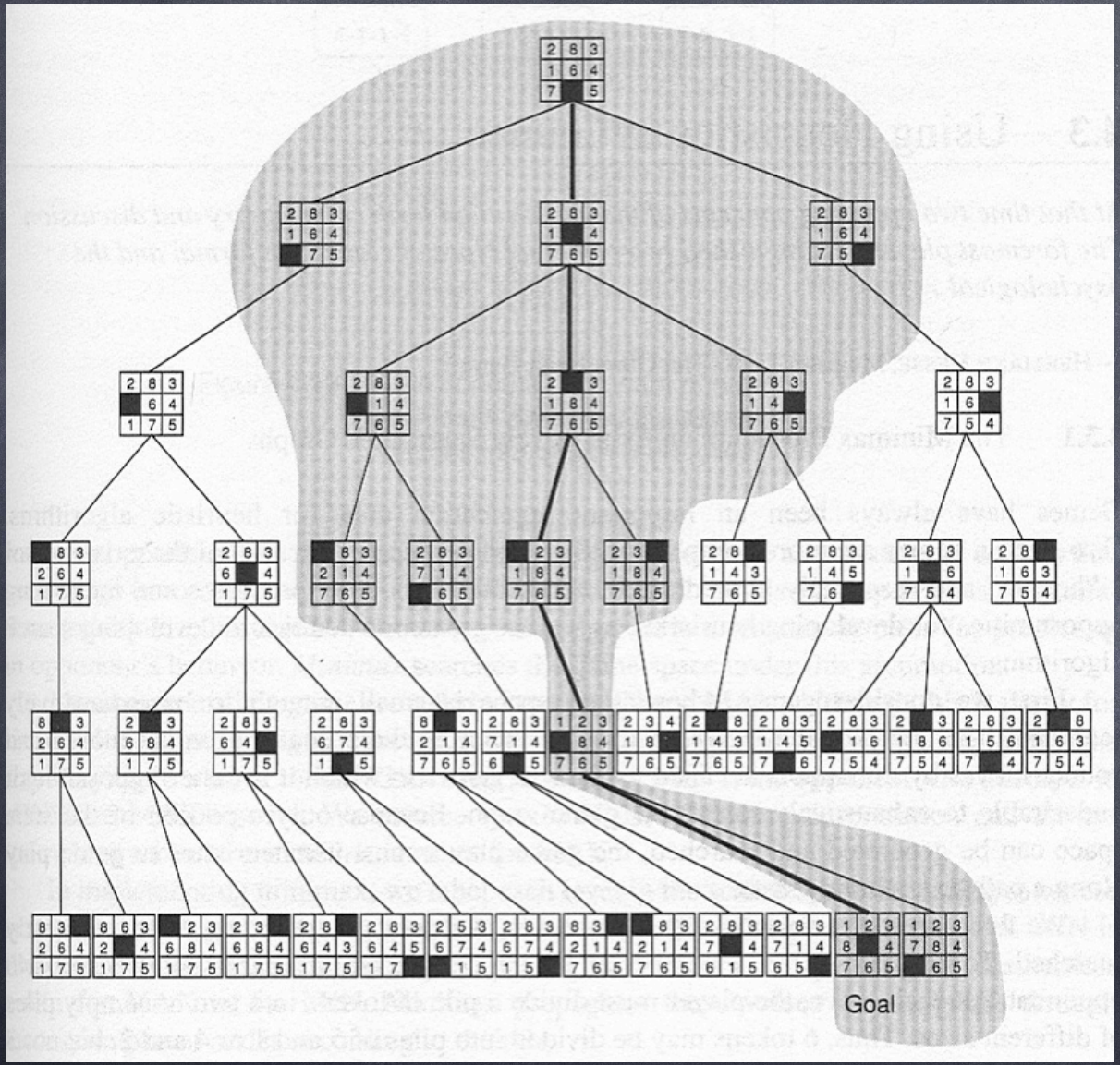

Level of search
g(n) =
1
State a
f(a) = 4
g(n) = 0
2
State b
f(b) = 6
State c
f(c) = 4
State d
f(d) = 6
g(n) = 1
3
State e
f(e) = 5
4
State f
f(f) = 5
State g
f(g) = 6
g(n) = 2
State h
f(h) = 6
State i
f(i) = 7
5
State j
f(j) = 5
State k
f(k) = 7
g(n) = 3
6
State l
f(l) = 5
g(n) = 4
7
State m
f(m) = 5
State n
f(n) = 7
g(n) = 5
Goal

[Luger 2002]

```
function best-first-search
   open := [Start]
   done := [ ]
   while open != [ ] do
      X := best state in open
      if (X = Goal) return success
         else
               generate children states of X
               discard any children in open, done
               for each child
                   compute heuristic value
                   add to open (keeping sorted)
               done.add(X)
   return failure
```


1
2 8 3
1 6 4
7 5
State a
f(a) = 4
2
2 8 3
1 4
7 6 5
State c
f(c) = 4
2 8 3
1 6 4
7 5
State b
f(b) = 6
2 8 3
1 6 4
7 5
State d
f(d) = 6
3
2 8 3
1 4
7 6 5
State e
f(e) = 5
2 3
1 8 4
7 6 5
State f
f(f) = 5
2 8 3
1 4
7 6 5
State g
f(g) = 6
8 3
2 1 4
7 6 5
State h
f(h) = 6
2 8 3
7 1 4
6 5
State i
f(i) = 7
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
1 2 3
8 4
7 6 5
Closed list
Open list
1 2 3
8 4
7 6 5
Goal
1 2 3
7 8 4
6 5

[Luger 2002]




[Luger 2002]

# Admissibility

a search algorithm is admissible if it is guaranteed to find a minimal path when it exists.

breadth-first search is admissible

is best-first search?

a heuristic is admissible when it finds the shortest path.

Our heuristic:

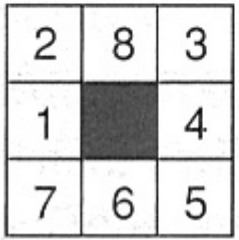
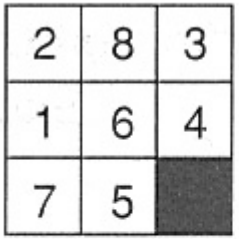
$$f(n) = g(n) + h(n)$$

takes into account path length and 'goodness'

'goodness' is really an estimated distance to goal

as long as heuristic does not OVERESTIMATE distance,
it is admissible.

(an overestimate might cause the best path to be ignored.)

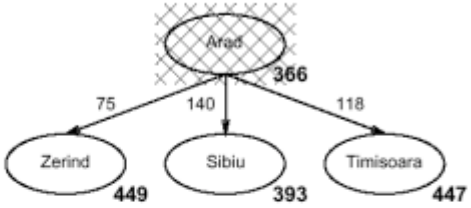
| | | | |
|---|---|---|---|
|  | **5** | **6** | **0** |
|  | **3** | **4** | **0** |
|  | **5** | **6** | **0** |
| | Tiles out of place | Sum of distances out of place | 2 x the number of direct tile reversals |

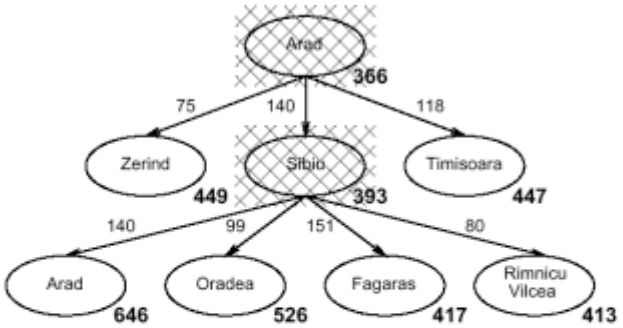
- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

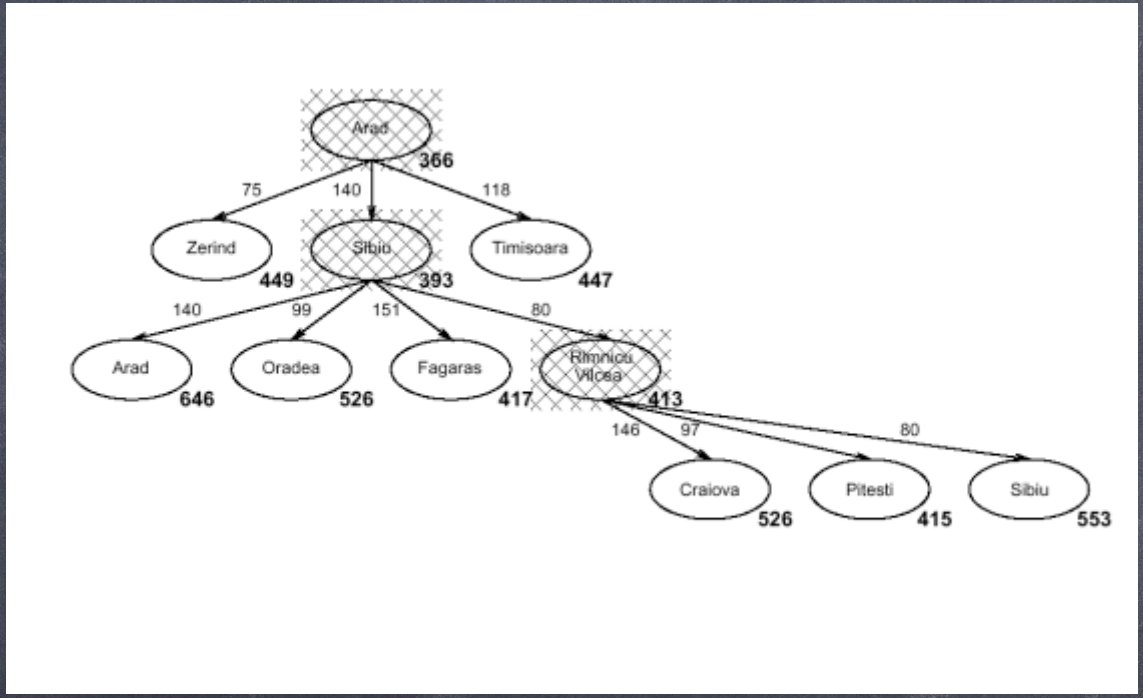
https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

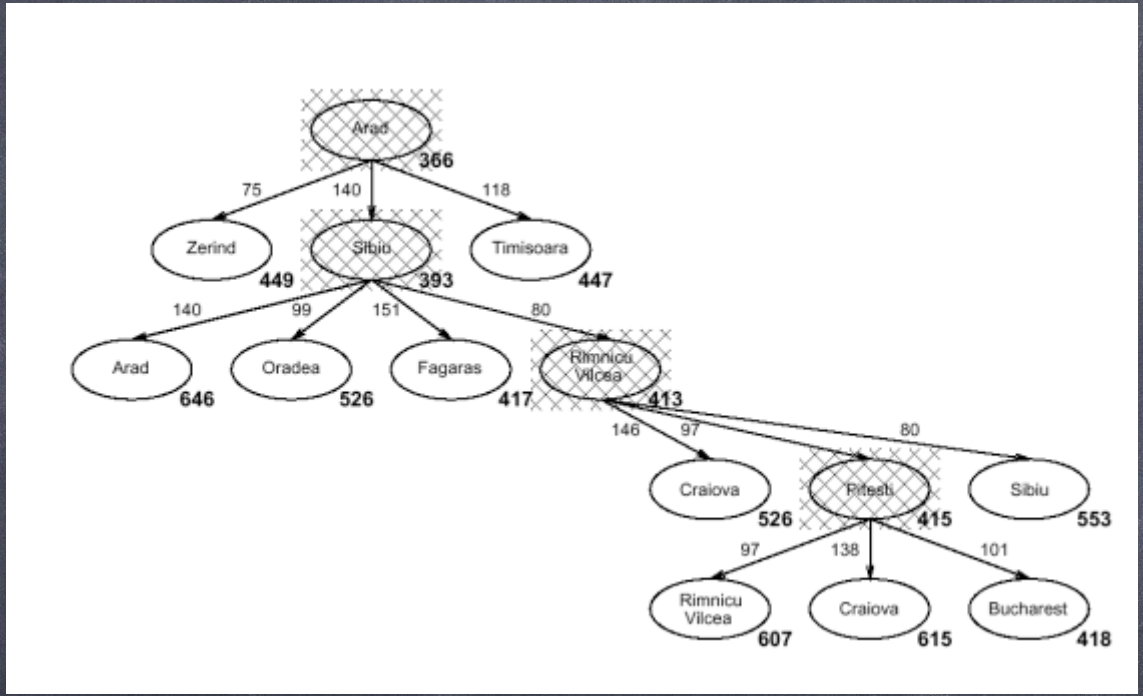
Best-first search with an admissible heuristic is called the A* Algorithm

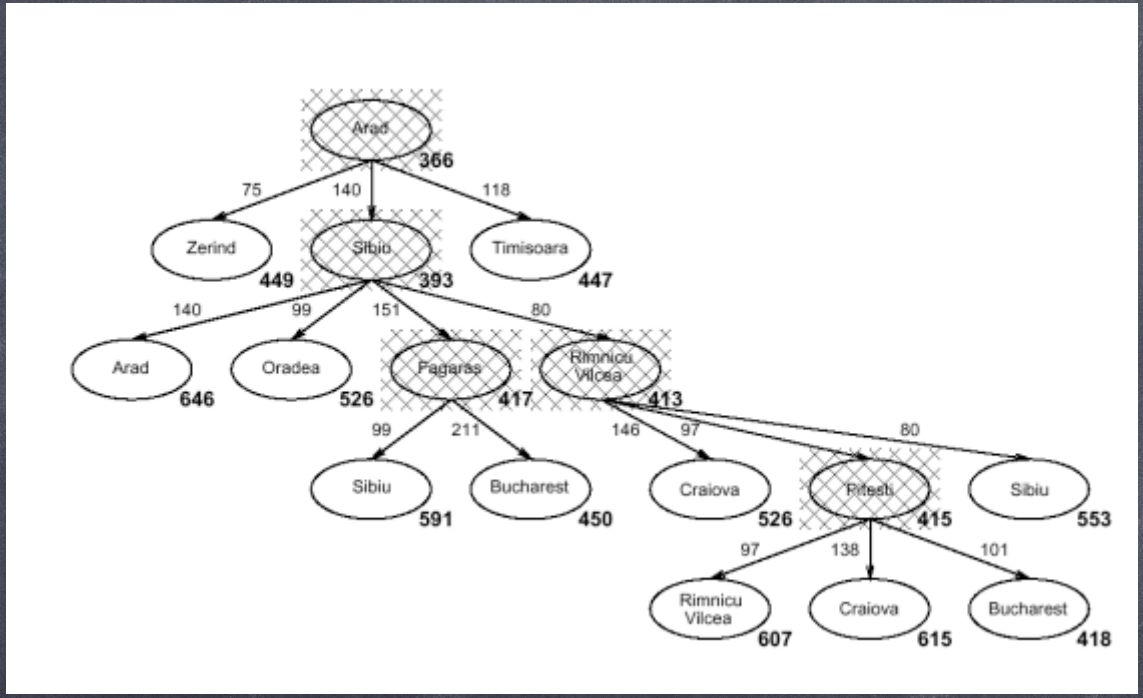

Arad
366

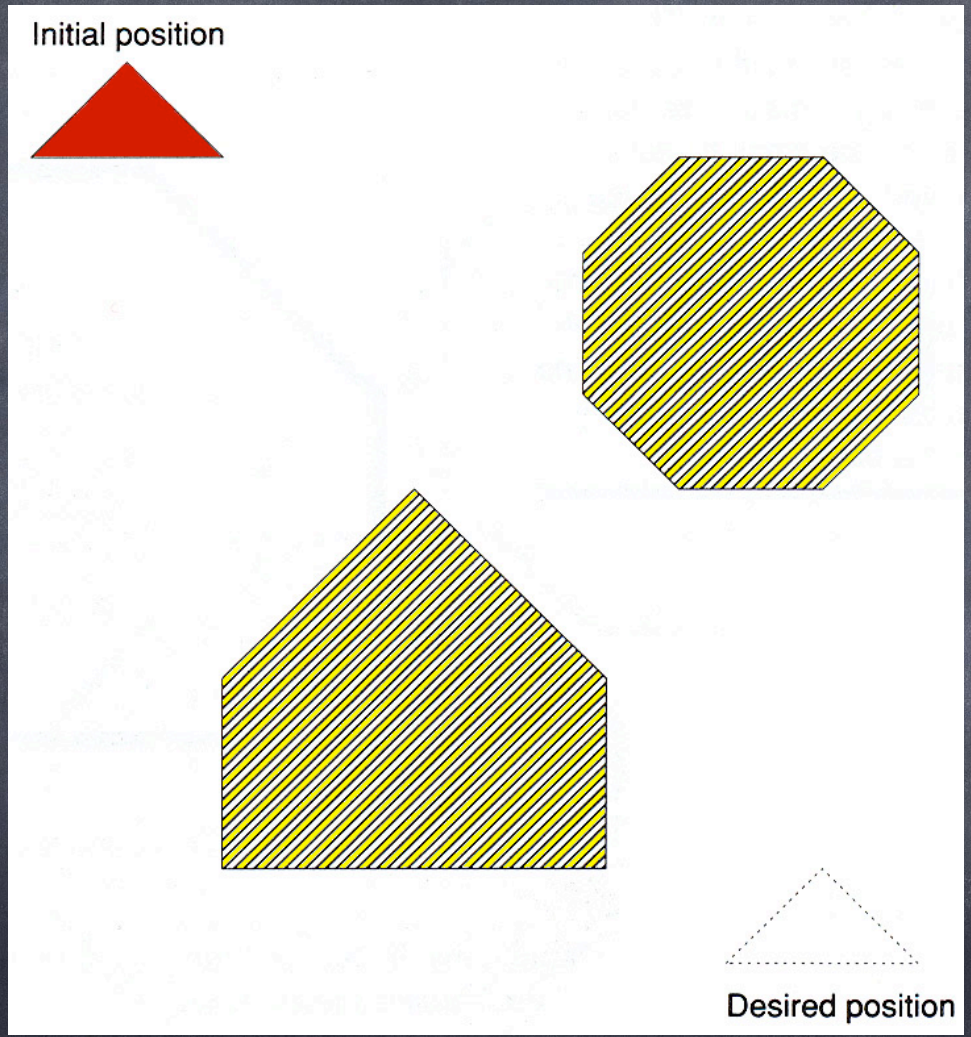

Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447

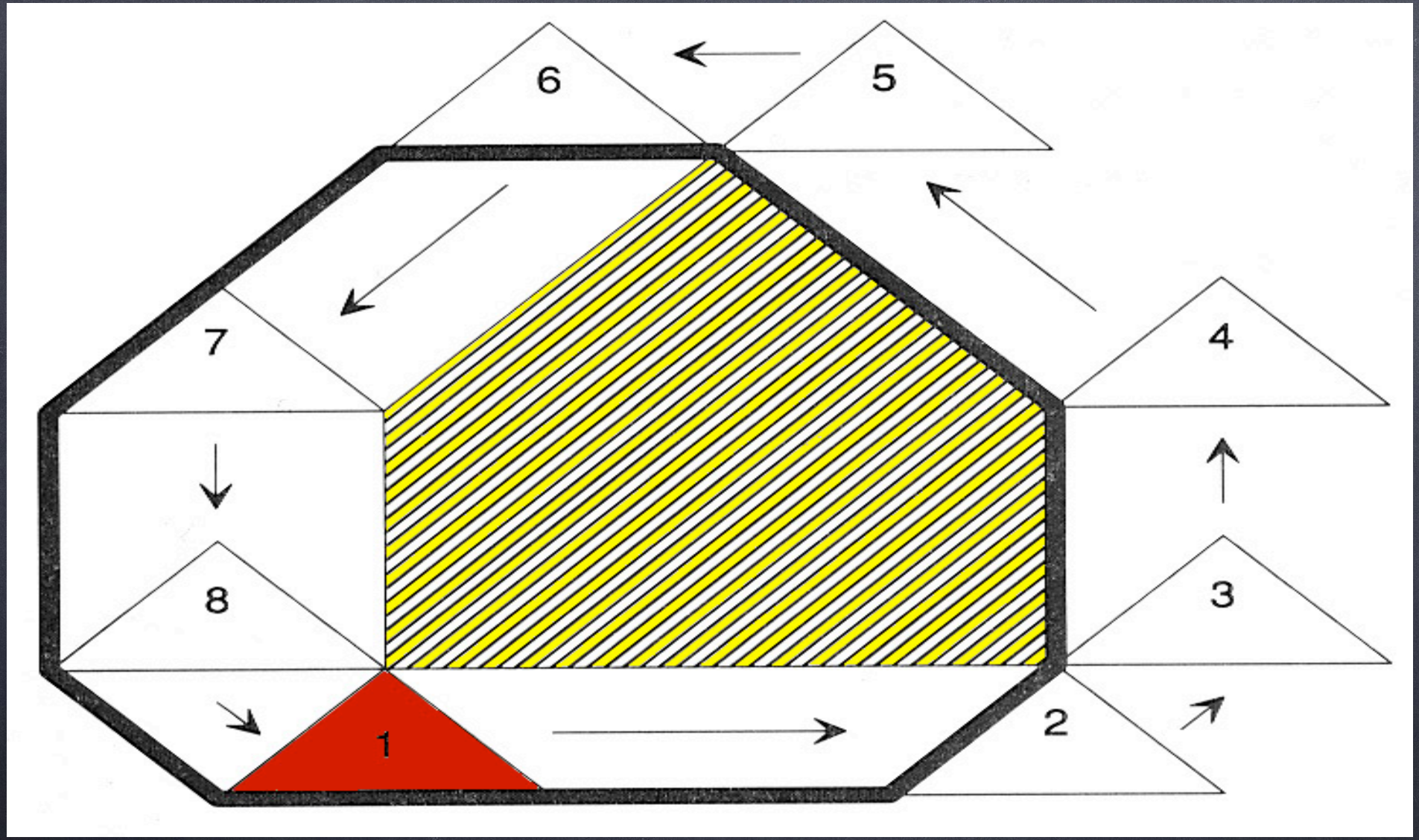

Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu
Vilcea
413

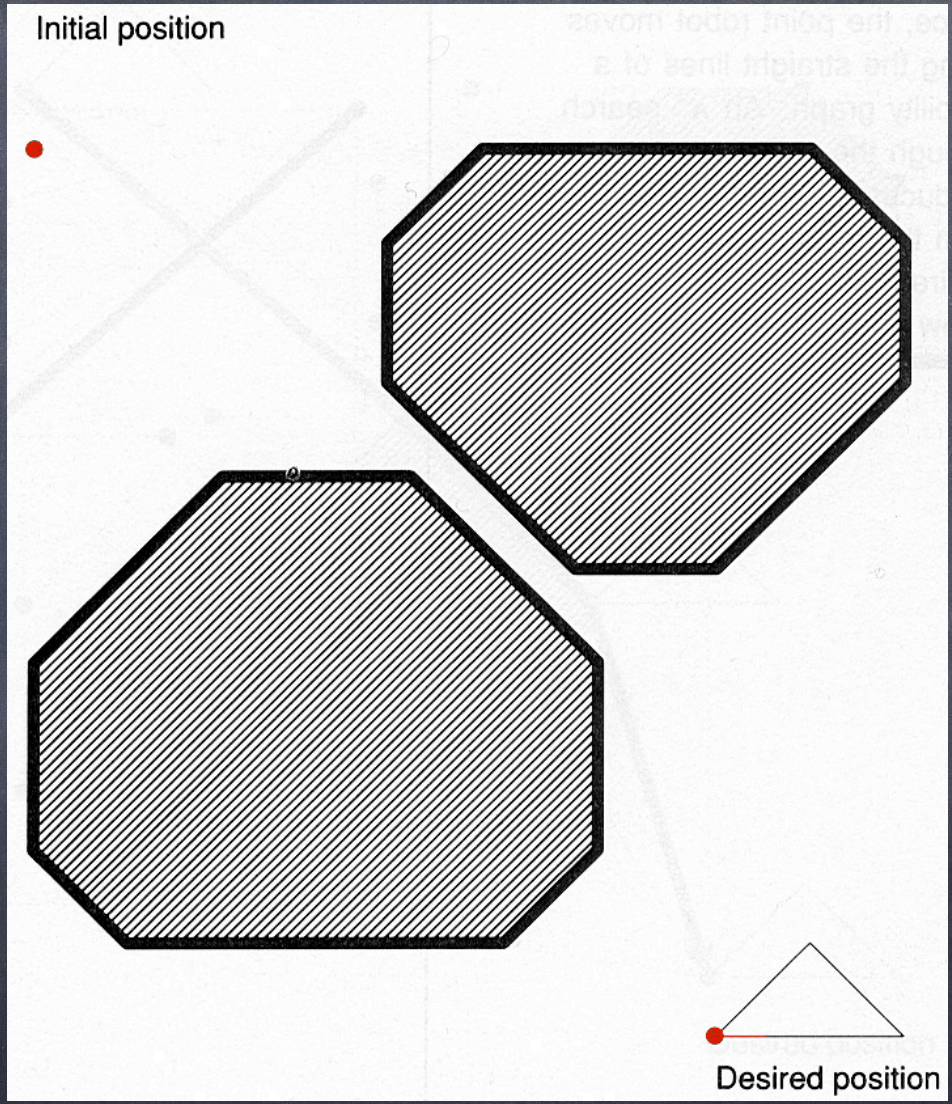

Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
146
97
80
Craiova
526
Pitesti
415
Sibiu
553

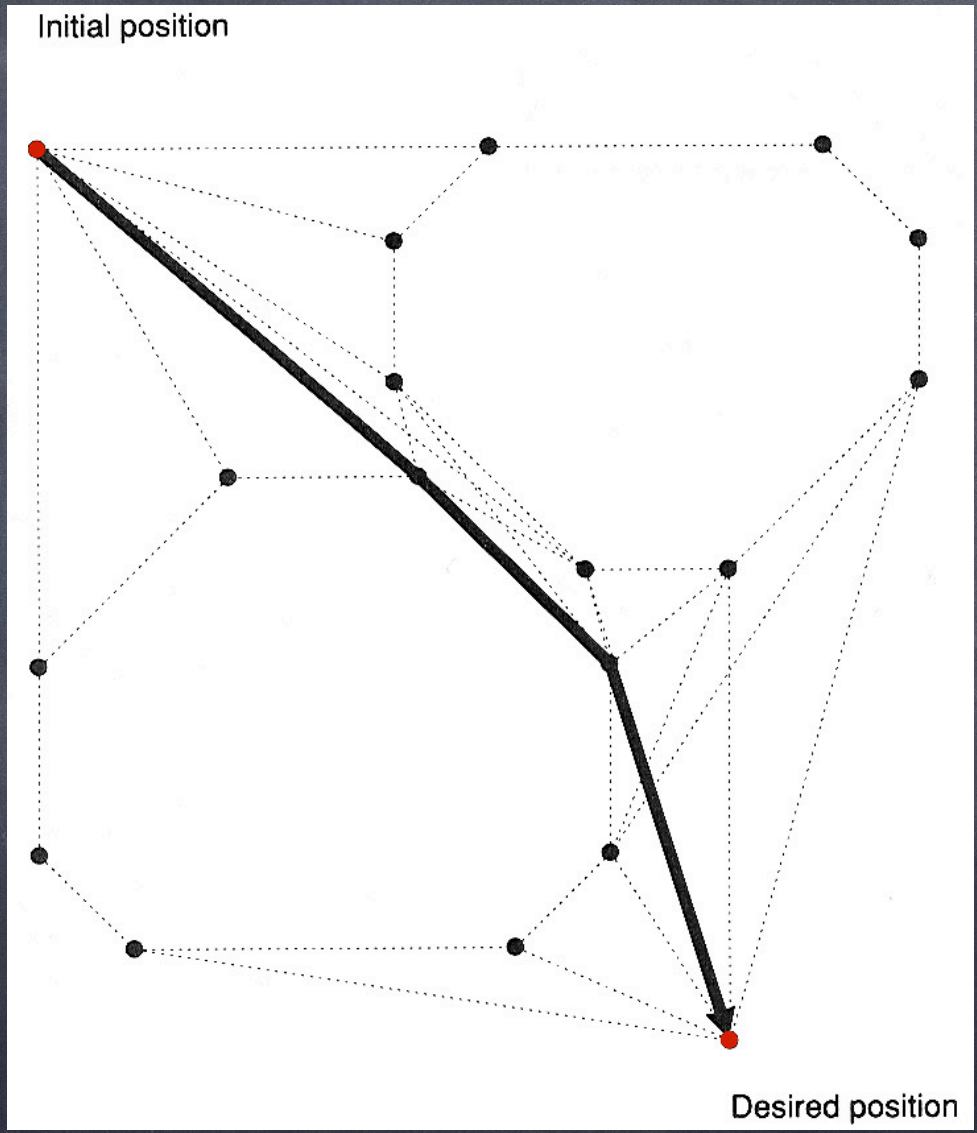

Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
146
97
80
Craiova
526
Pitesti
415
Sibiu
553
97
138
101
Rimnicu Vilcea
607
Craiova
615
Bucharest
418

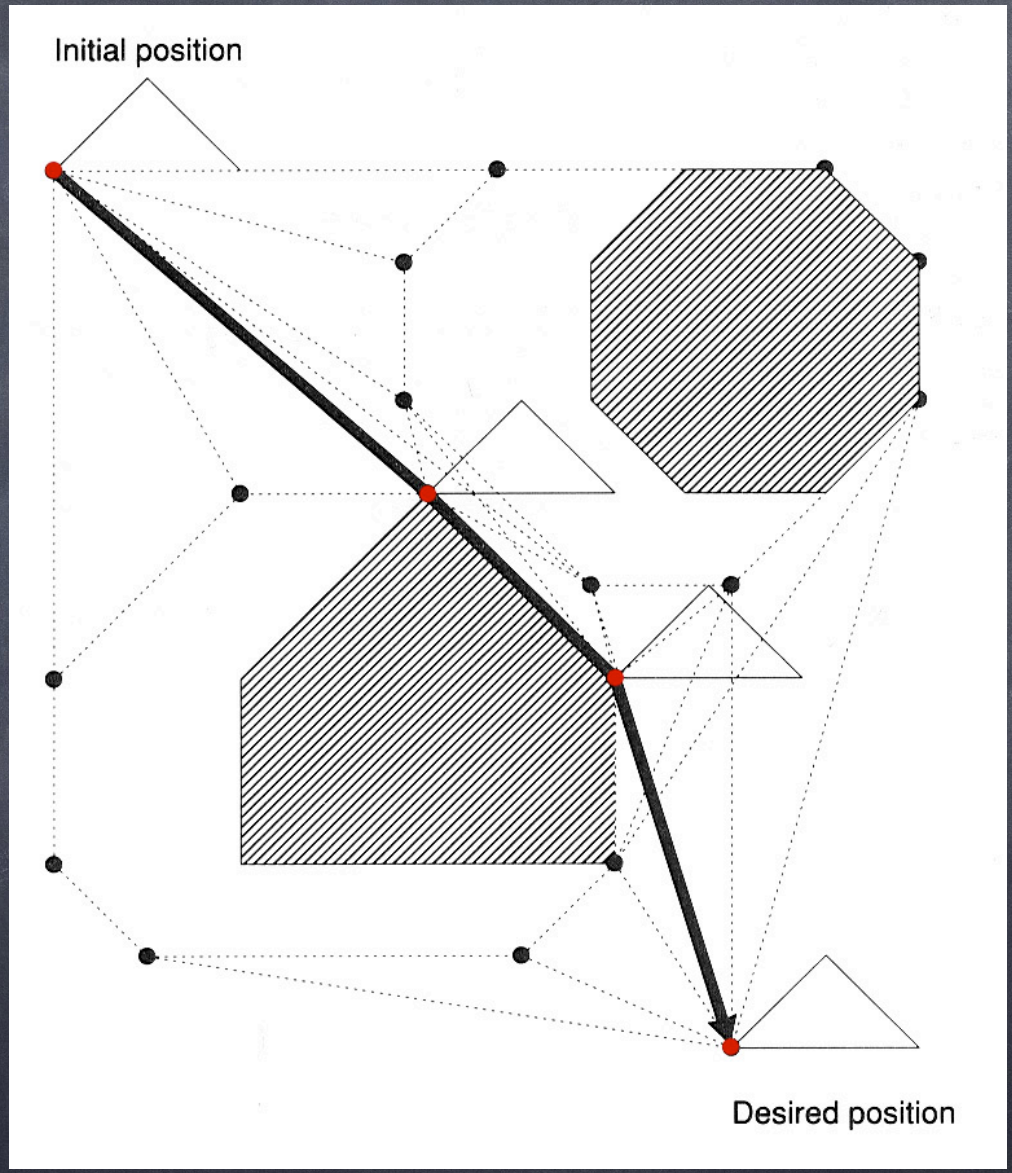

Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
99
211
146
97
80
Sibiu
591
Bucharest
450
Craiova
526
Pitesti
415
Sibiu
553
97
138
101
Rimnicu Vilcea
607
Craiova
615
Bucharest
418

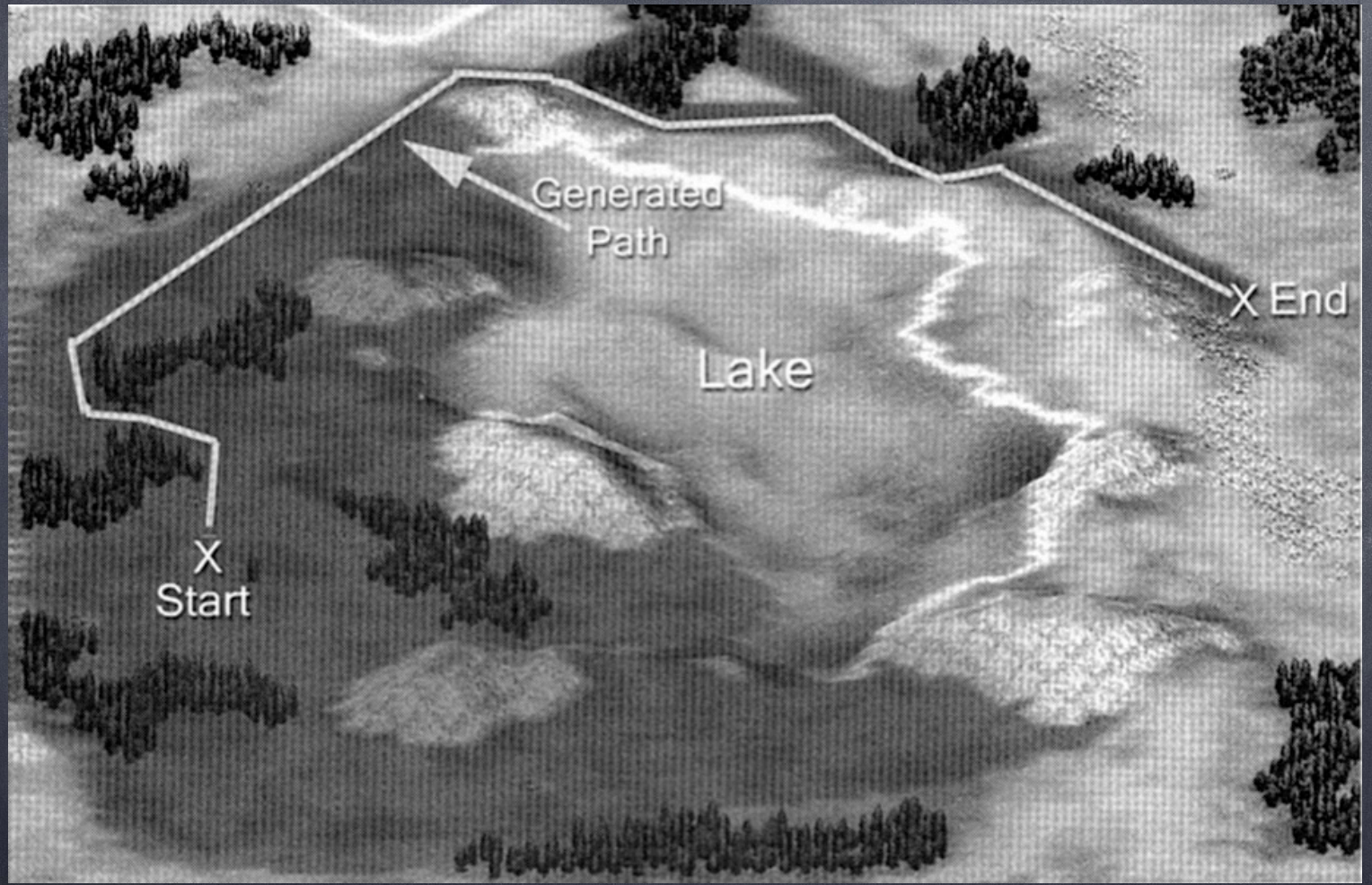

Initial position
Desired position


6
5
7
4
8
3
1
2


Initial position
Desired position


Initial position
Desired position

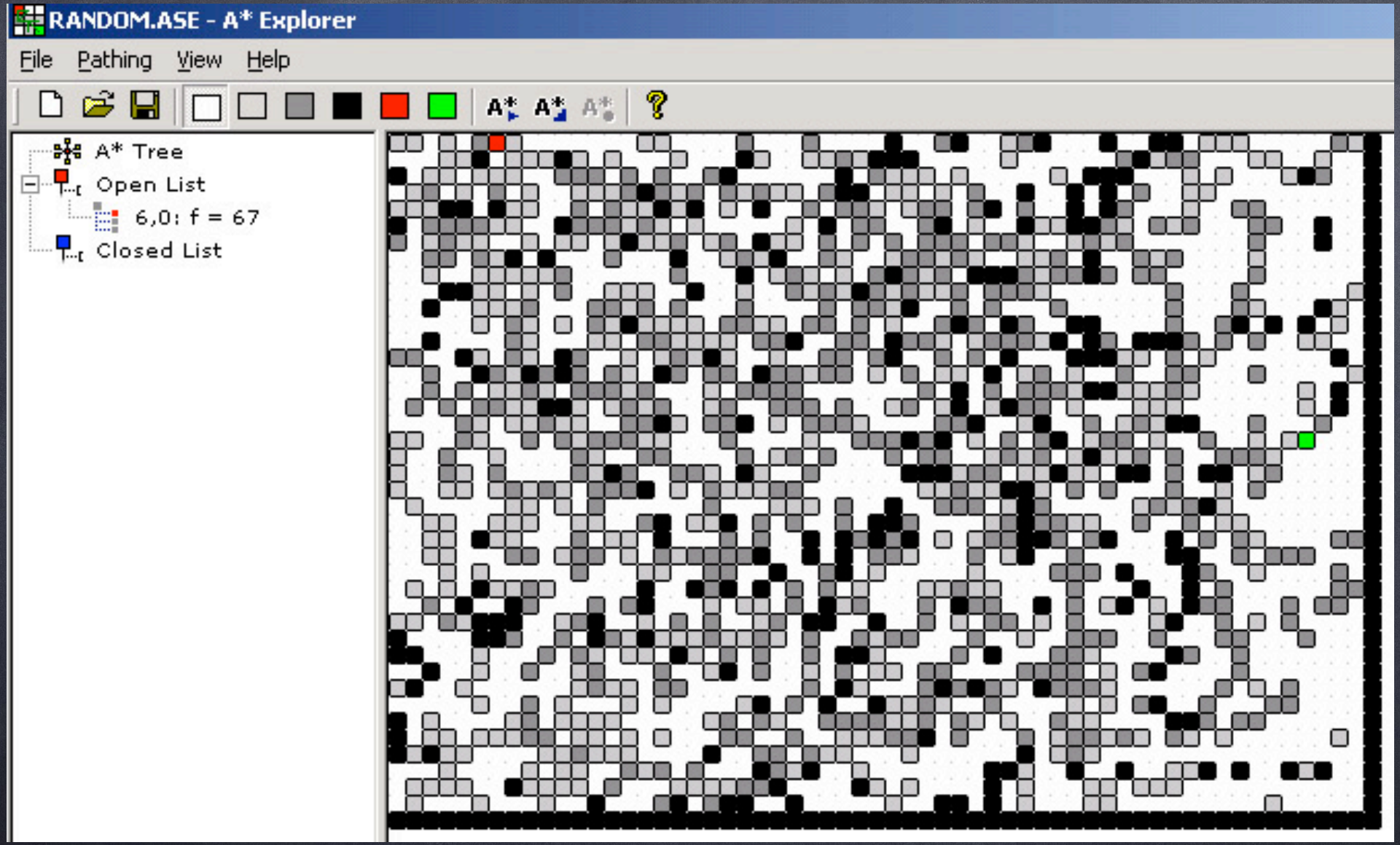

Initial position
Desired position

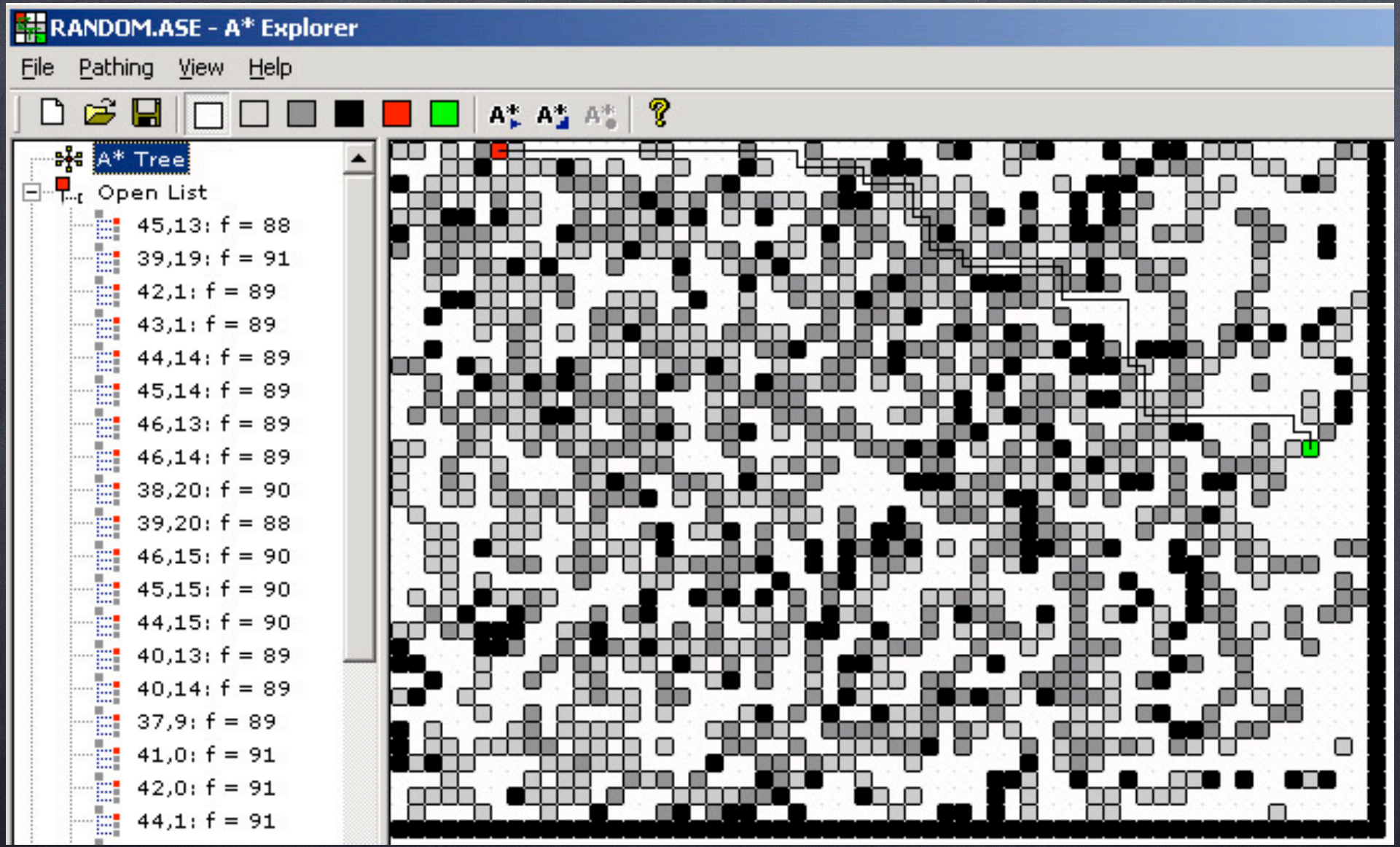

Generated
Path
X End
Lake
X
Start

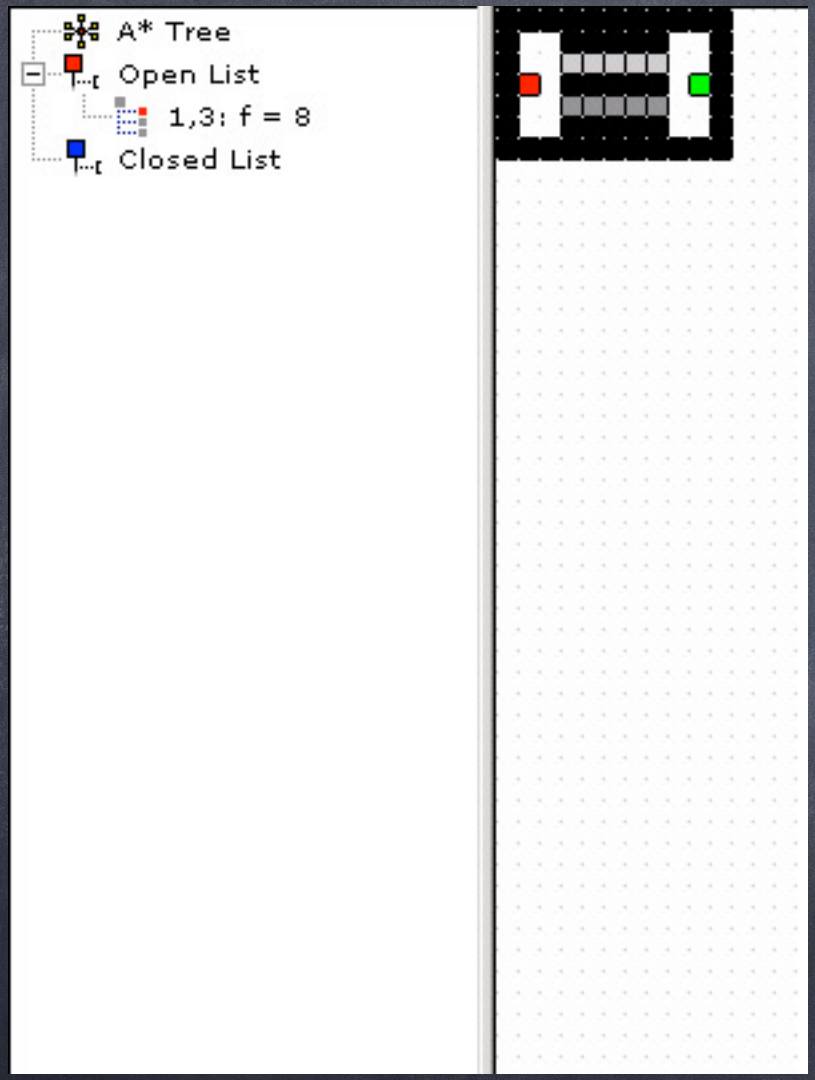
# Example: A* explorer

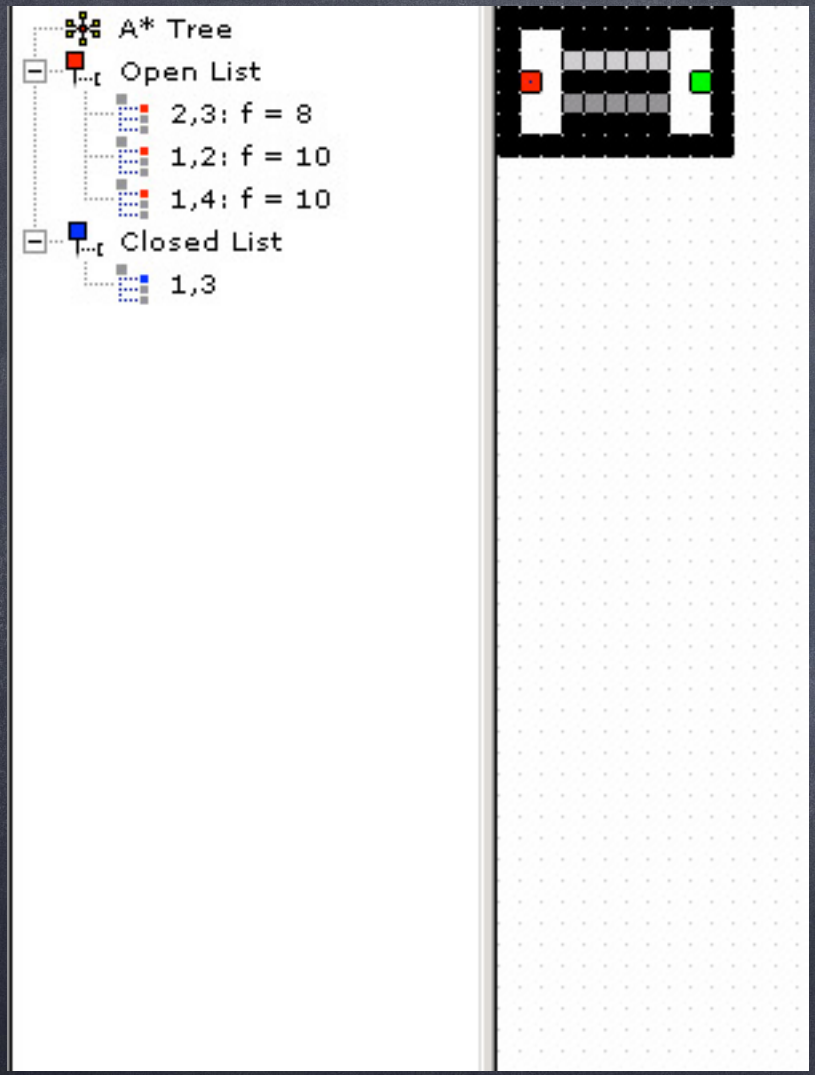
www.generation5.org

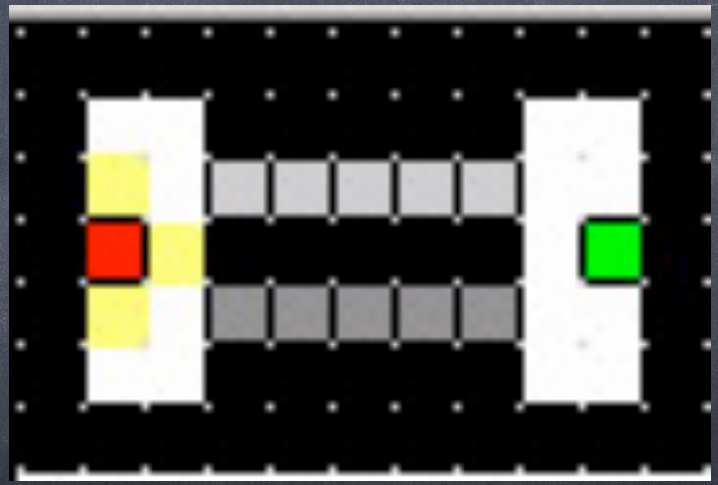
provides A* implementation in C++

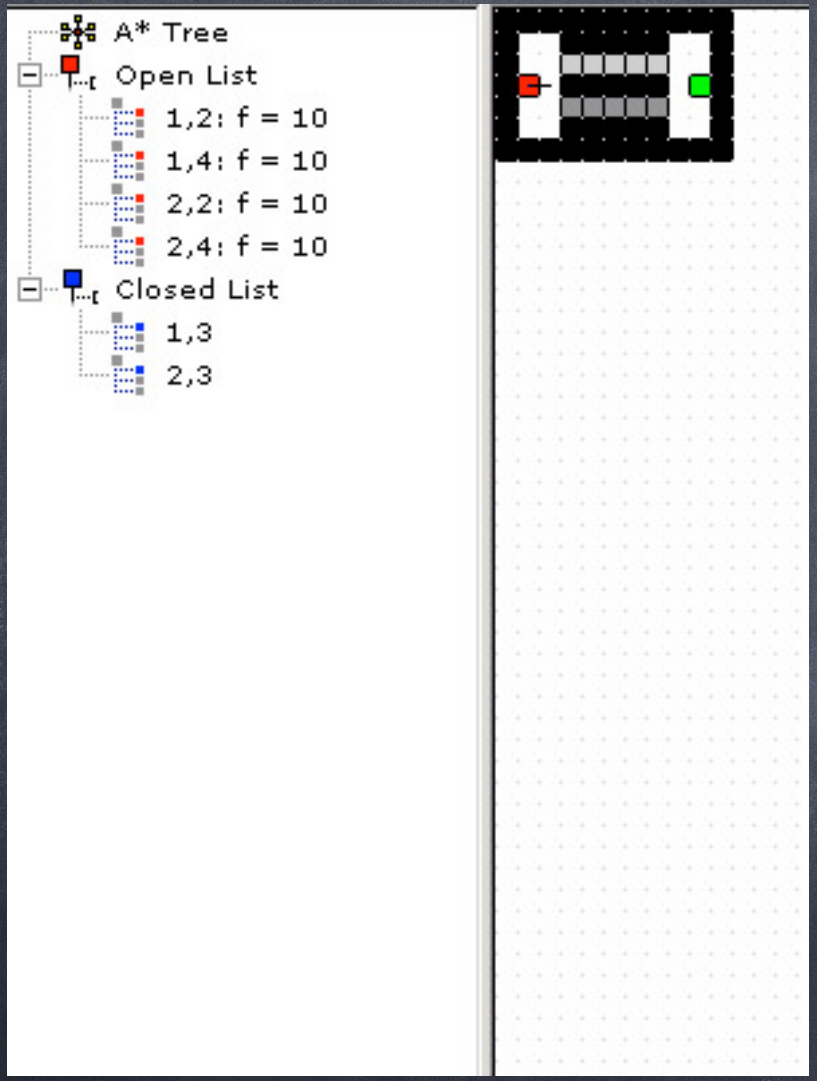

RANDOM.ASE - A* Explorer
File
Pathing
View
Help
A* Tree
Open List
6,0: f = 67
Closed List

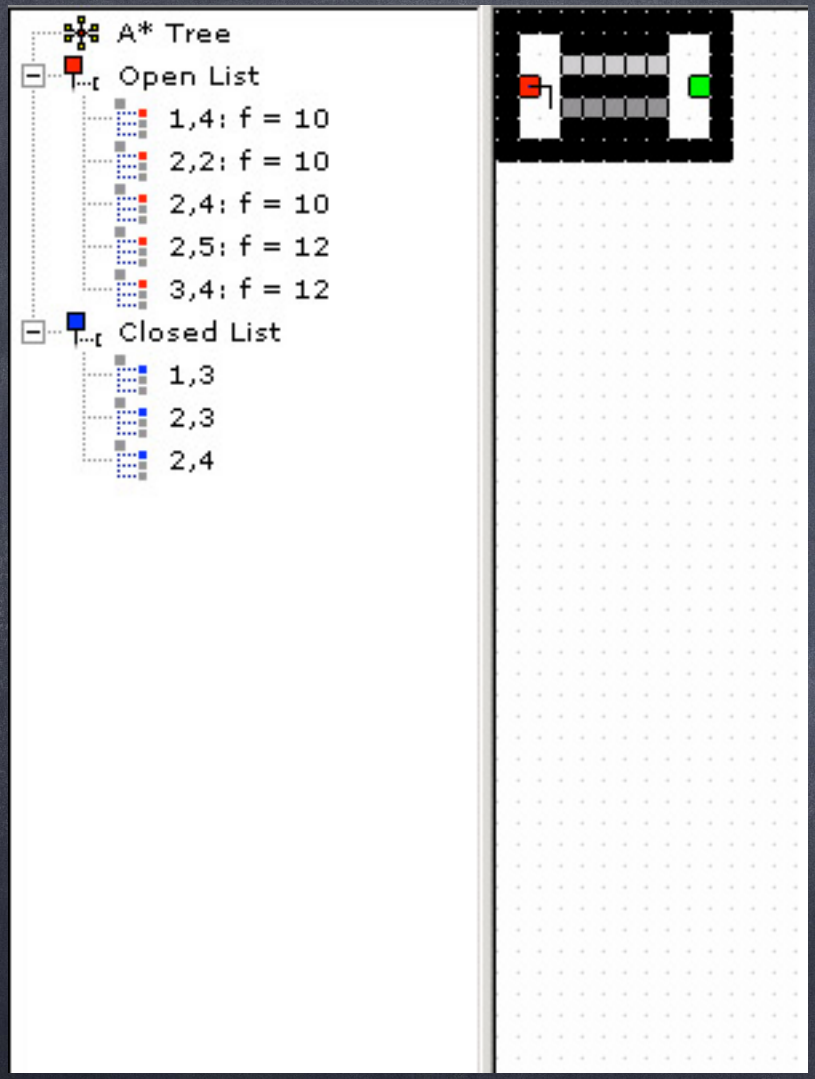

RANDOM.ASE - A* Explorer
File
Pathing
View
Help
A* Tree
Open List
45,13: f = 88
39,19: f = 91
42,1: f = 89
43,1: f = 89
44,14: f = 89
45,14: f = 89
46,13: f = 89
46,14: f = 89
38,20: f = 90
39,20: f = 88
46,15: f = 90
45,15: f = 90
44,15: f = 90
40,13: f = 89
40,14: f = 89
37,9: f = 89
41,0: f = 91
42,0: f = 91
44,1: f = 91

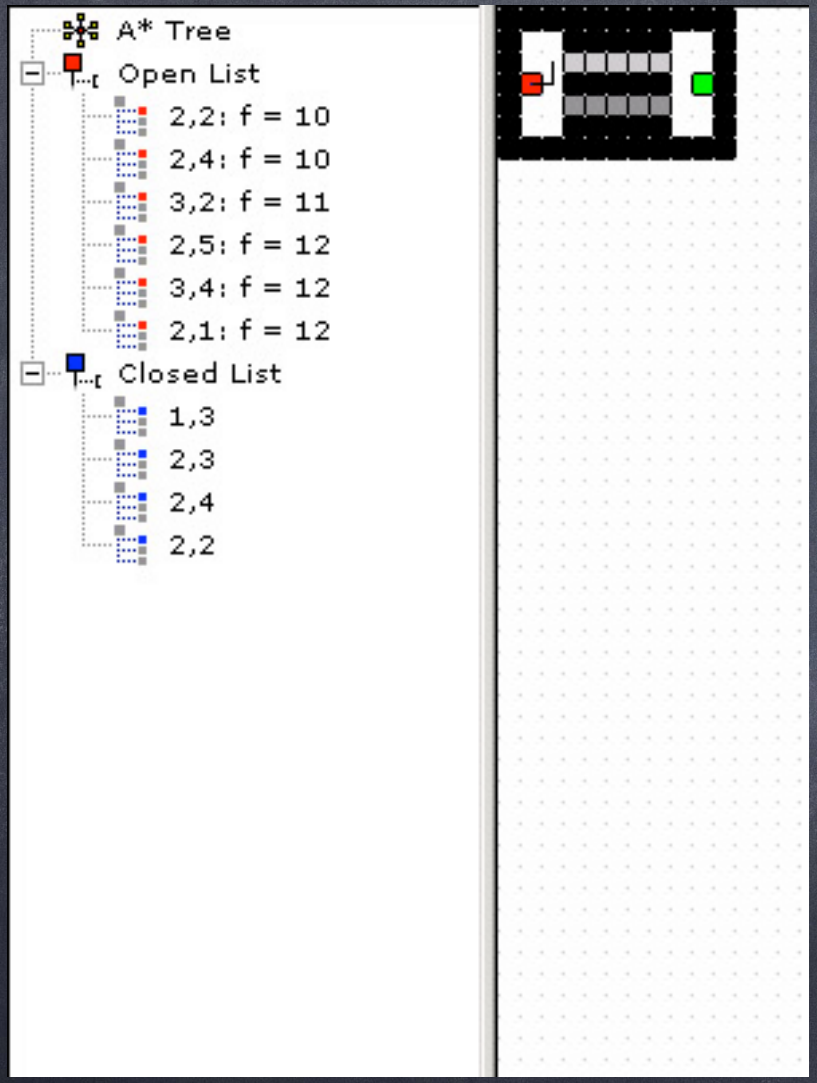

A* Tree
Open List
1,3: f = 8
Closed List

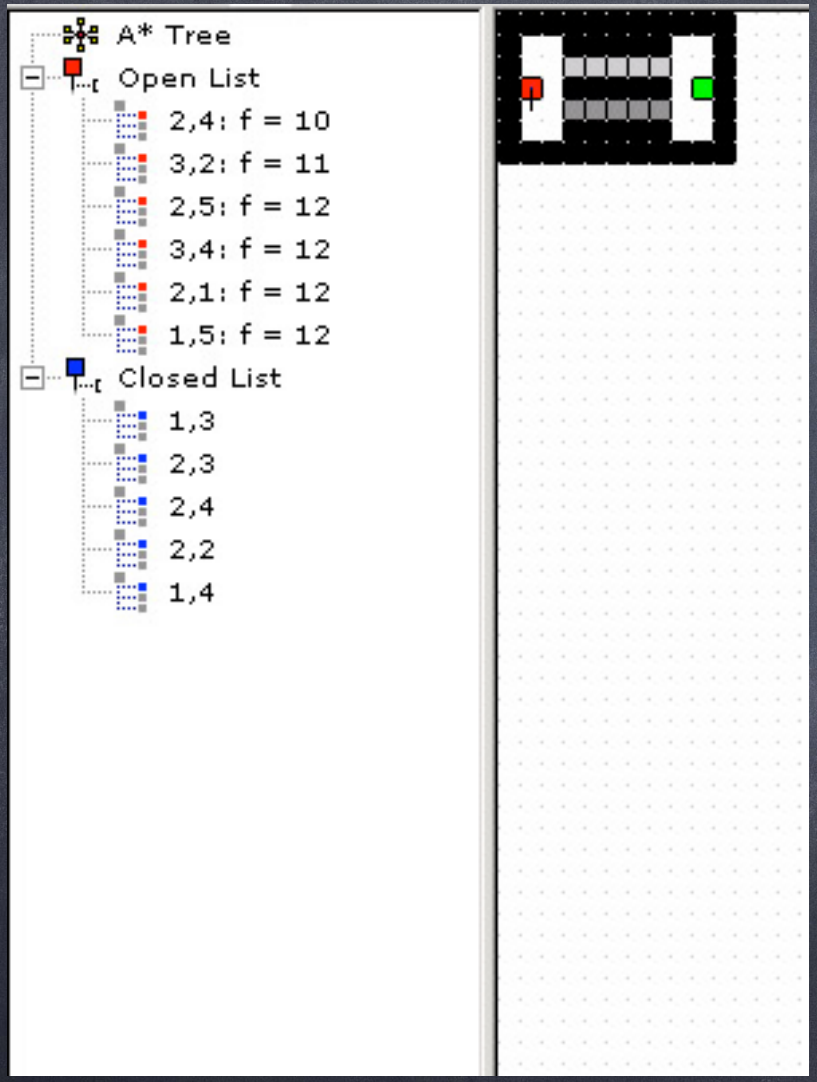

A* Tree
Open List
2,3: f = 8
1,2: f = 10
1,4: f = 10
Closed List
1,3



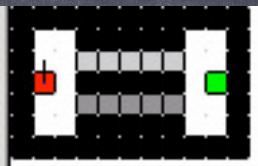

A* Tree
Open List
1,2: f = 10
1,4: f = 10
2,2: f = 10
2,4: f = 10
Closed List
1,3
2,3

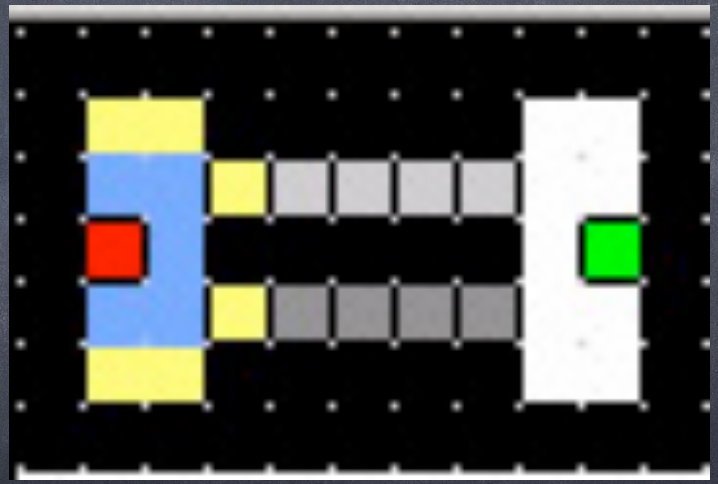

A* Tree
Open List
1,4: f = 10
2,2: f = 10
2,4: f = 10
2,5: f = 12
3,4: f = 12
Closed List
1,3
2,3
2,4


A* Tree
Open List
2,2: f = 10
2,4: f = 10
3,2: f = 11
2,5: f = 12
3,4: f = 12
2,1: f = 12
Closed List
1,3
2,3
2,4
2,2

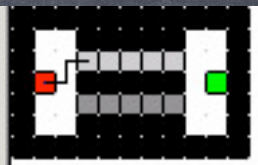

A* Tree
Open List
2,4: f = 10
3,2: f = 11
2,5: f = 12
3,4: f = 12
2,1: f = 12
1,5: f = 12
Closed List
1,3
2,3
2,4
2,2
1,4

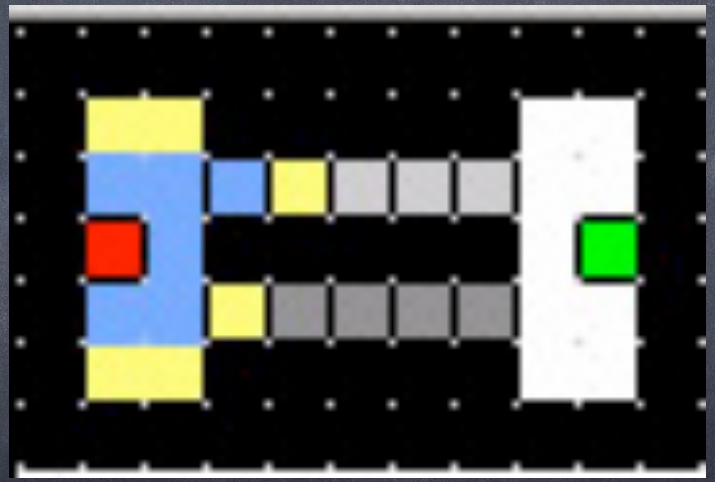
- A* Tree
- Open List
  - 3,2: f = 11
  - 2,5: f = 12
  - 3,4: f = 12
  - 2,1: f = 12
  - 1,5: f = 12
  - 1,1: f = 12
- Closed List
  - 1,3
  - 2,3
  - 2,4
  - 2,2
  - 1,4
  - 1,2

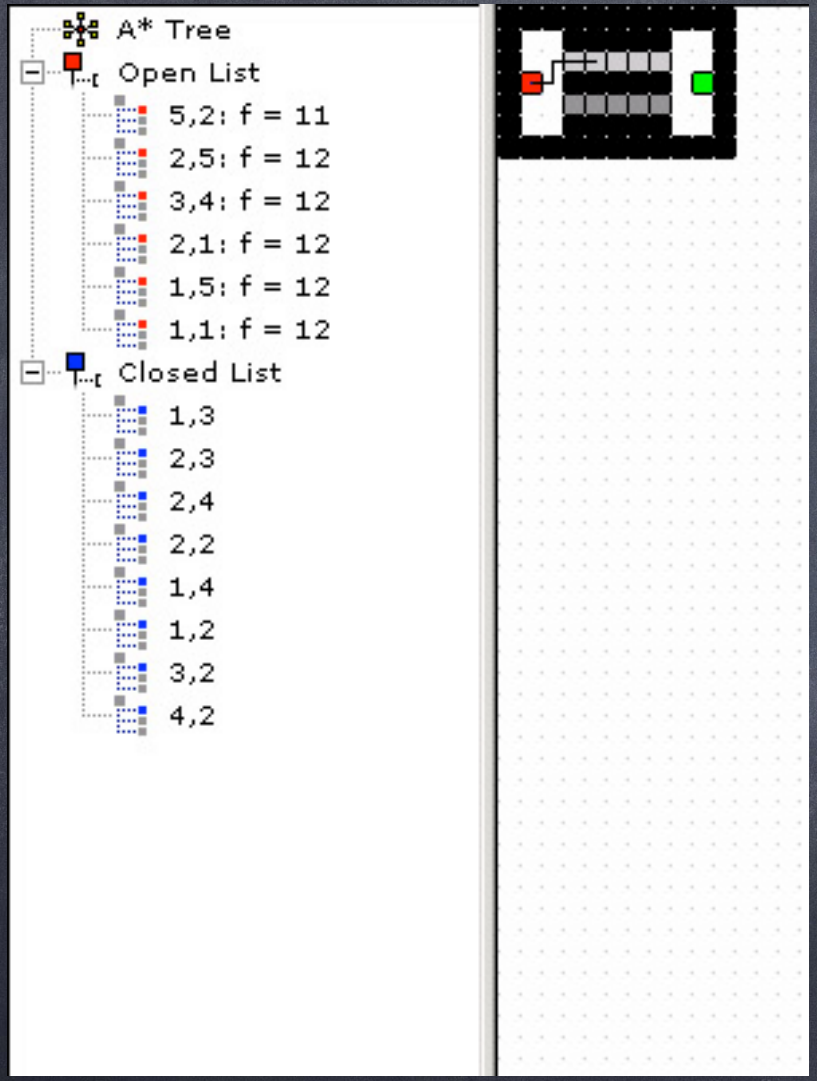


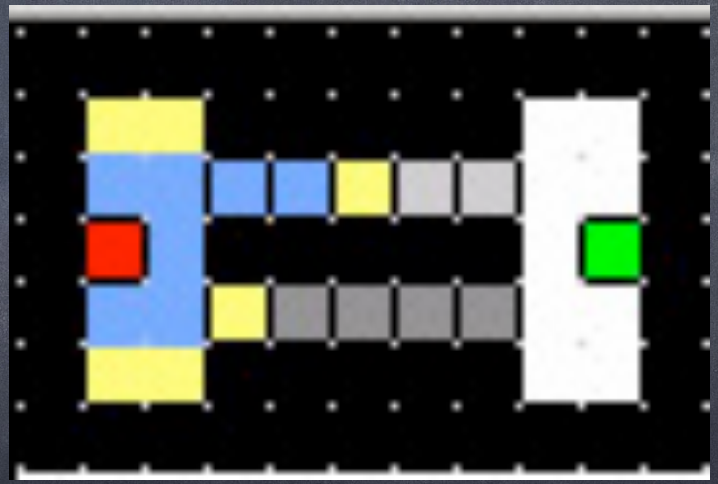


- A* Tree
- Open List
  - 4,2: f = 11
  - 2,5: f = 12
  - 3,4: f = 12
  - 2,1: f = 12
  - 1,5: f = 12
  - 1,1: f = 12
- Closed List
  - 1,3
  - 2,3
  - 2,4
  - 2,2
  - 1,4
  - 1,2
  - 3,2

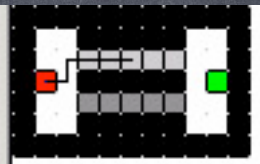


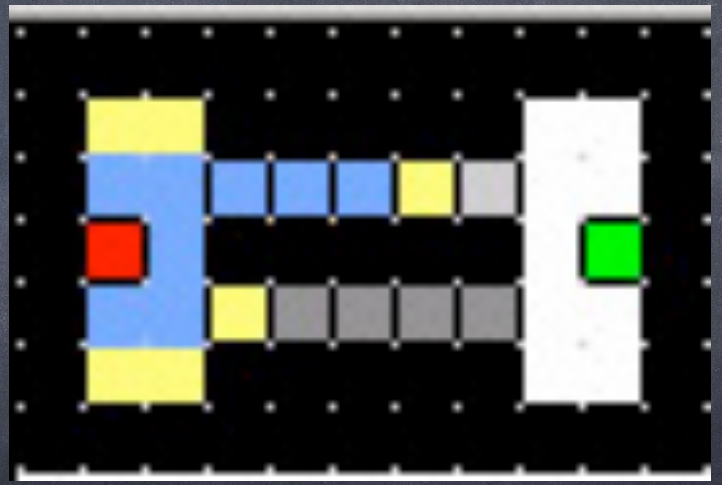



A* Tree
Open List
5,2: f = 11
2,5: f = 12
3,4: f = 12
2,1: f = 12
1,5: f = 12
1,1: f = 12
Closed List
1,3
2,3
2,4
2,2
1,4
1,2
3,2
4,2

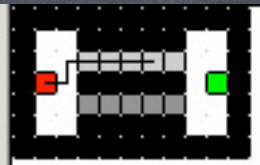


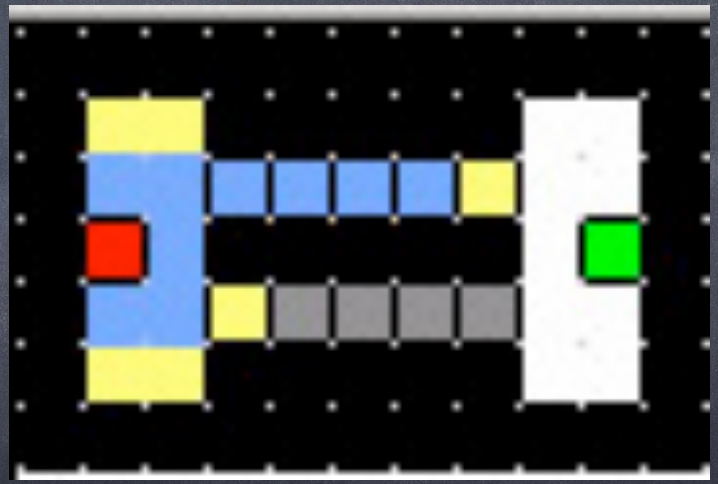
- A* Tree
- Open List
  - 6,2: f = 11
  - 2,5: f = 12
  - 3,4: f = 12
  - 2,1: f = 12
  - 1,5: f = 12
  - 1,1: f = 12
- Closed List
  - 1,3
  - 2,3
  - 2,4
  - 2,2
  - 1,4
  - 1,2
  - 3,2
  - 4,2
  - 5,2

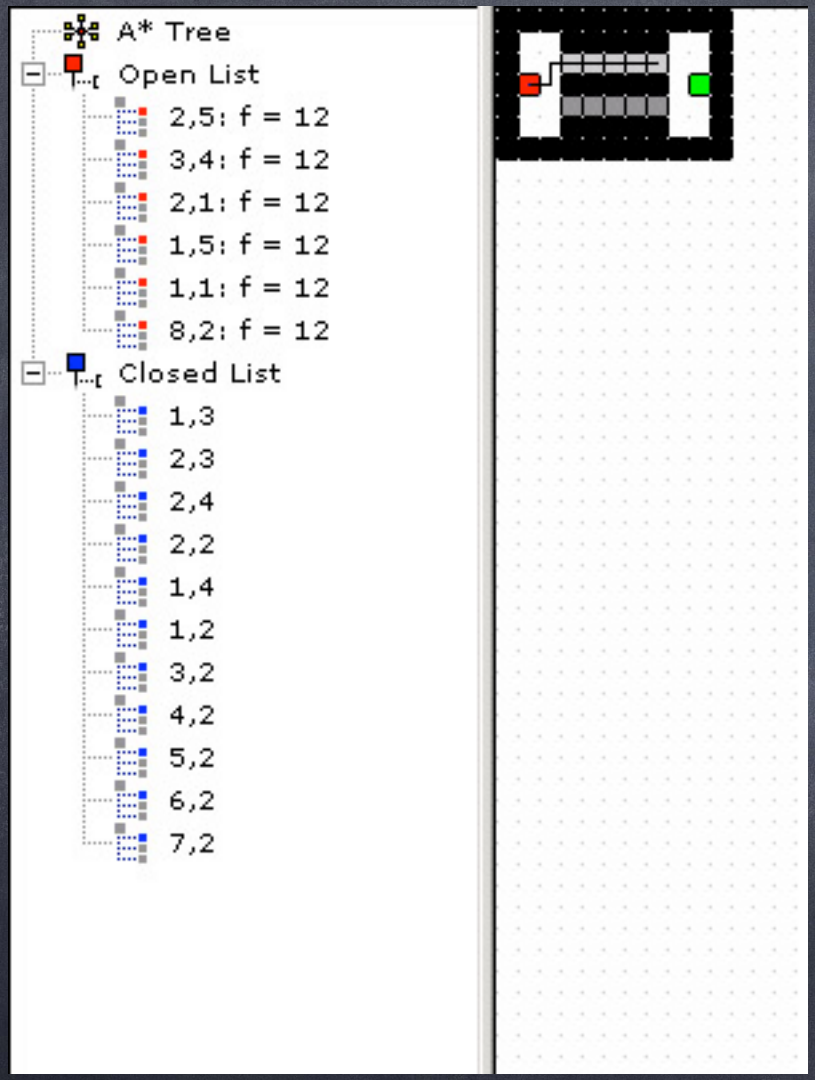


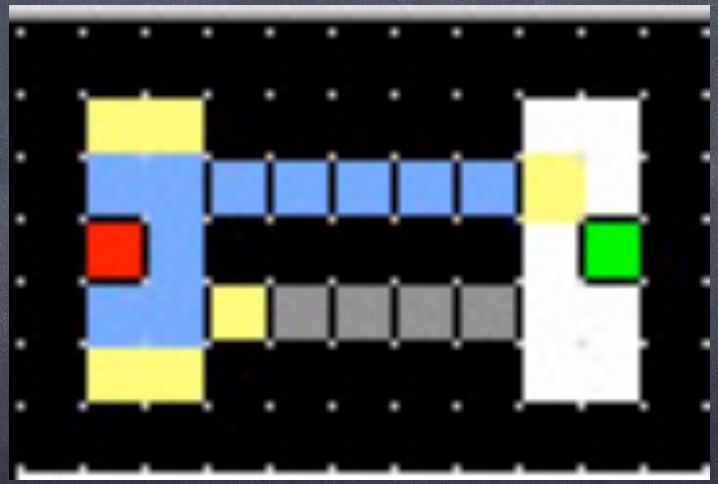


- A* Tree
- Open List
  - 7,2: f = 11
  - 2,5: f = 12
  - 3,4: f = 12
  - 2,1: f = 12
  - 1,5: f = 12
  - 1,1: f = 12
- Closed List
  - 1,3
  - 2,3
  - 2,4
  - 2,2
  - 1,4
  - 1,2
  - 3,2
  - 4,2
  - 5,2
  - 6,2






A* Tree
Open List
2,5: f = 12
3,4: f = 12
2,1: f = 12
1,5: f = 12
1,1: f = 12
8,2: f = 12
Closed List
1,3
2,3
2,4
2,2
1,4
1,2
3,2
4,2
5,2
6,2
7,2



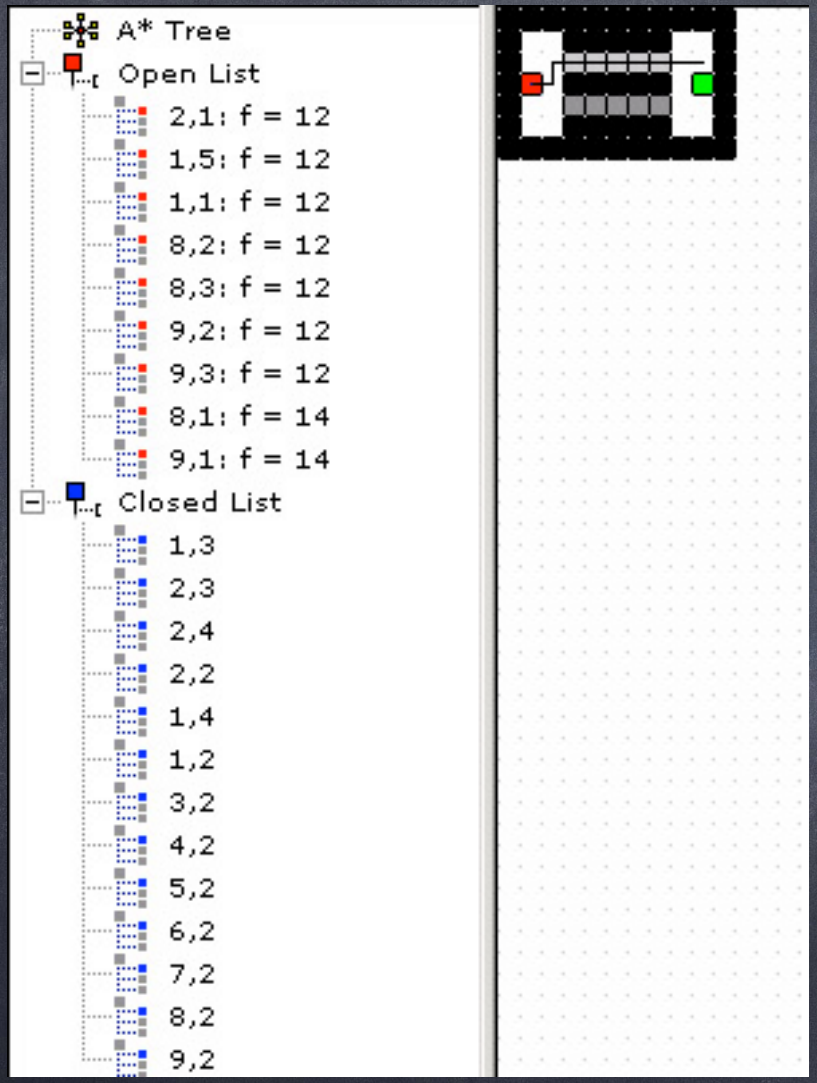
A* Tree

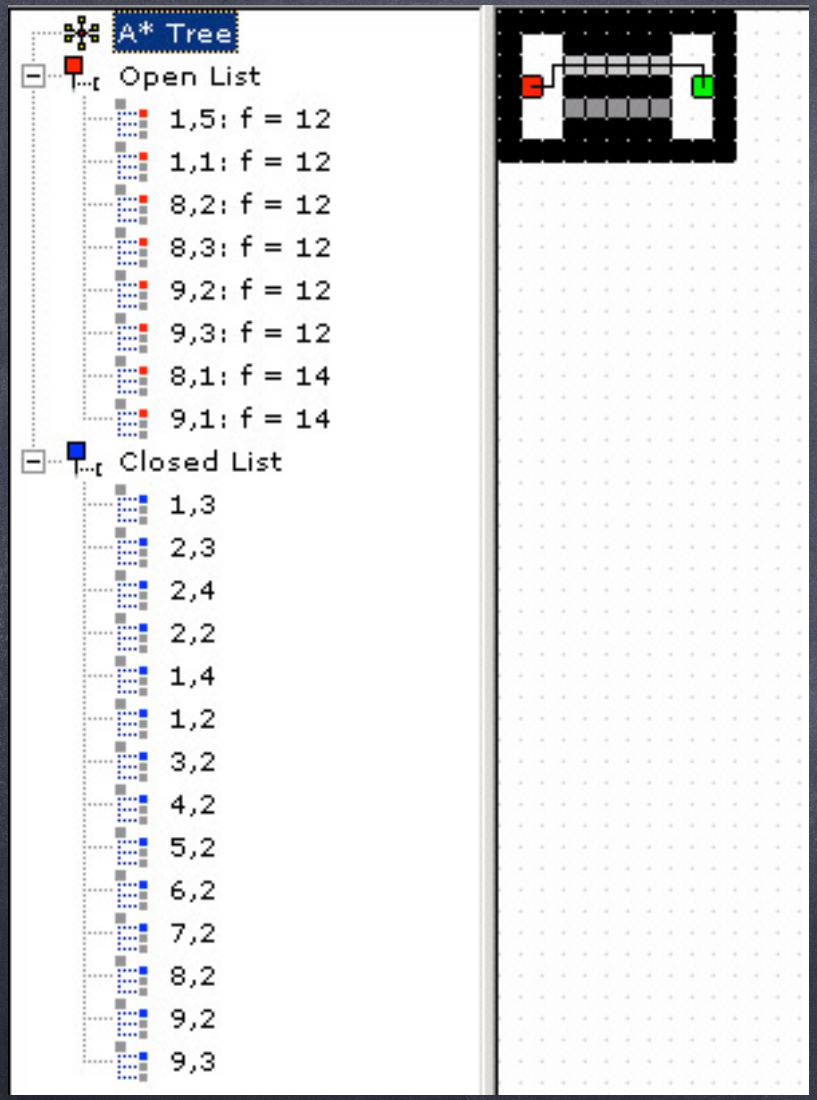
Open List

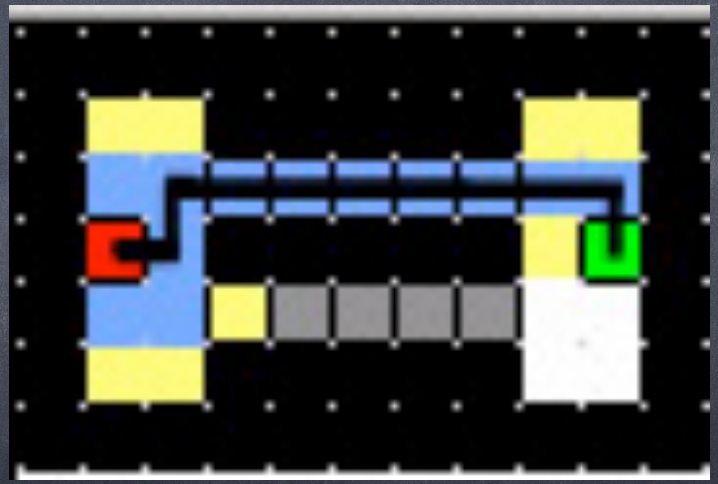
- 3,4: f = 12
- 2,1: f = 12
- 1,5: f = 12
- 1,1: f = 12
- 8,2: f = 12
- 8,3: f = 12
- 9,2: f = 12
- 8,1: f = 14

Closed List

- 1,3
- 2,3
- 2,4
- 2,2
- 1,4
- 1,2
- 3,2
- 4,2
- 5,2
- 6,2
- 7,2
- 8,2


A* Tree
Open List
2,1: f = 12
1,5: f = 12
1,1: f = 12
8,2: f = 12
8,3: f = 12
9,2: f = 12
9,3: f = 12
8,1: f = 14
9,1: f = 14
Closed List
1,3
2,3
2,4
2,2
1,4
1,2
3,2
4,2
5,2
6,2
7,2
8,2
9,2

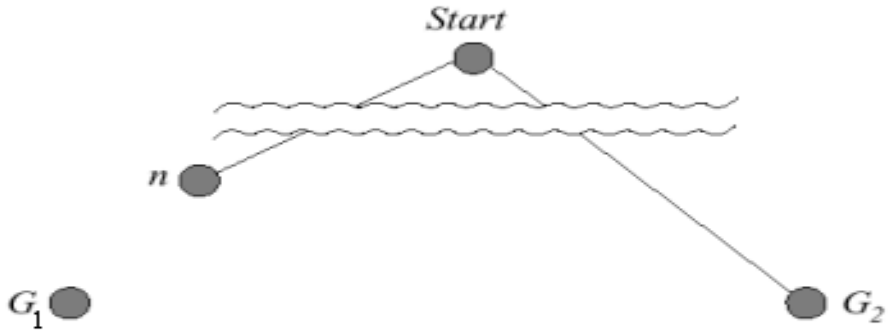

A* Tree
Open List
1,5: f = 12
1,1: f = 12
8,2: f = 12
8,3: f = 12
9,2: f = 12
9,3: f = 12
8,1: f = 14
9,1: f = 14
Closed List
1,3
2,3
2,4
2,2
1,4
1,2
3,2
4,2
5,2
6,2
7,2
8,2
9,2
9,3



https://www.youtube.com/watch?v=huJEgJ82360

# Optimality of A* (standard proof)

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.



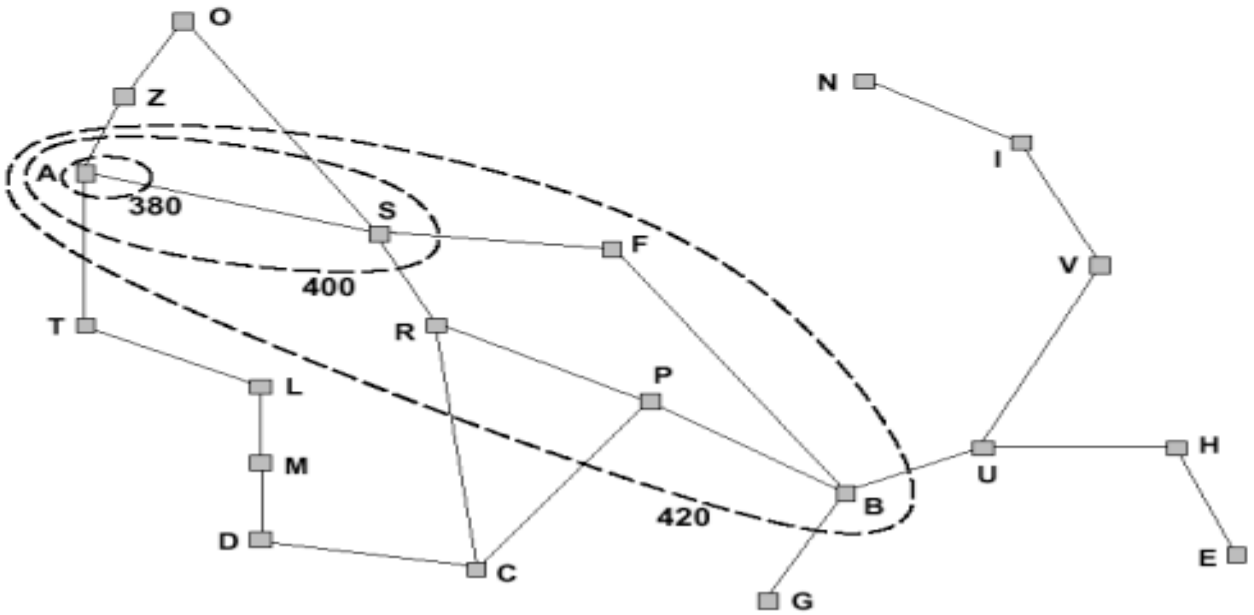

$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion

# Optimality of A* (more useful proof)

Lemma: A* expands nodes in order of increasing $f$ value

Gradually adds "$f$-contours" of nodes (cf. breadth-first adds layers)
Contour $i$ has all nodes with $f = f_i$, where $f_i < f_{i+1}$



Memory Problems?

Implement A* with Iterative deepening.

# Sizes of Problem Spaces

| Problem | Nodes | Brute-Force Search Time (10 million nodes/second) |
|---|---|---|
| • 8 Puzzle: | $10^5$ | .01 seconds |
| • $2^3$ Rubik's Cube: | $10^6$ | .2 seconds |
| • 15 Puzzle: | $10^{13}$ | 6 days |
| • $3^3$ Rubik's Cube: | $10^{19}$ | 68,000 years |
| • 24 Puzzle: | $10^{25}$ | 12 billion years |

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

# Performance of IDA* on 15 Puzzle

- Random 15 puzzle instances were first solved optimally using IDA* with Manhattan distance heuristic (Korf, 1985).
- Optimal solution lengths average 53 moves.
- 400 million nodes generated on average.
- Average solution time is about 50 seconds on current machines.

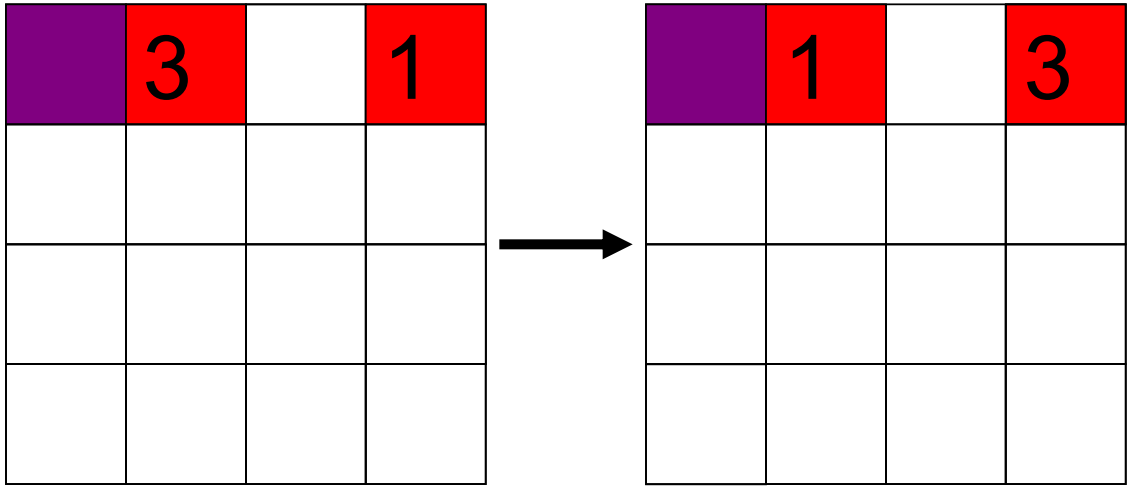
https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

# Limitation of Manhattan Distance

- To solve a 24-Puzzle instance, IDA* with Manhattan distance would take about 65,000 years on average.
- Assumes that each tile moves independently
- In fact, tiles interfere with each other.
- Accounting for these interactions is the key to more accurate heuristic functions.

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

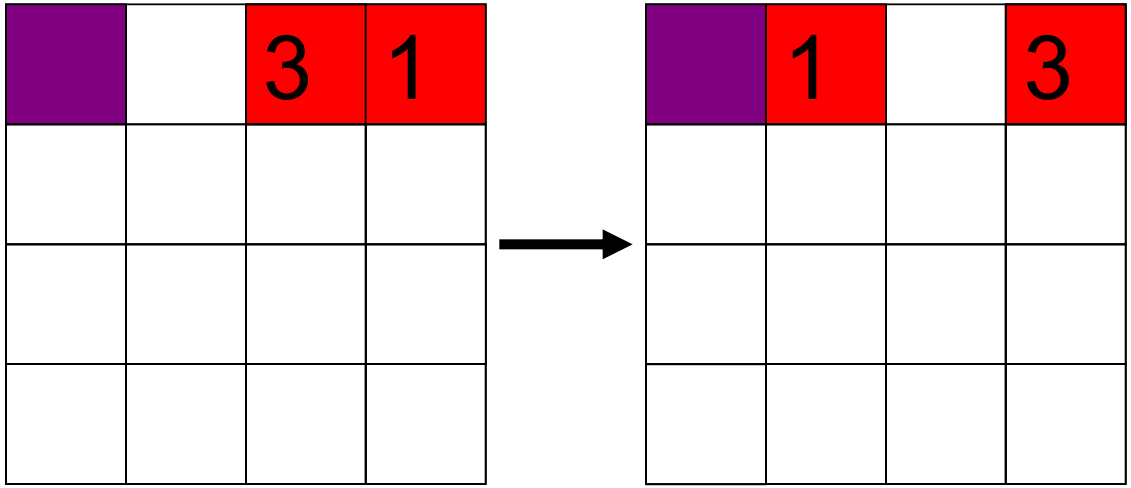
# Example: Linear Conflict




Manhattan distance is 2+2=4 moves

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

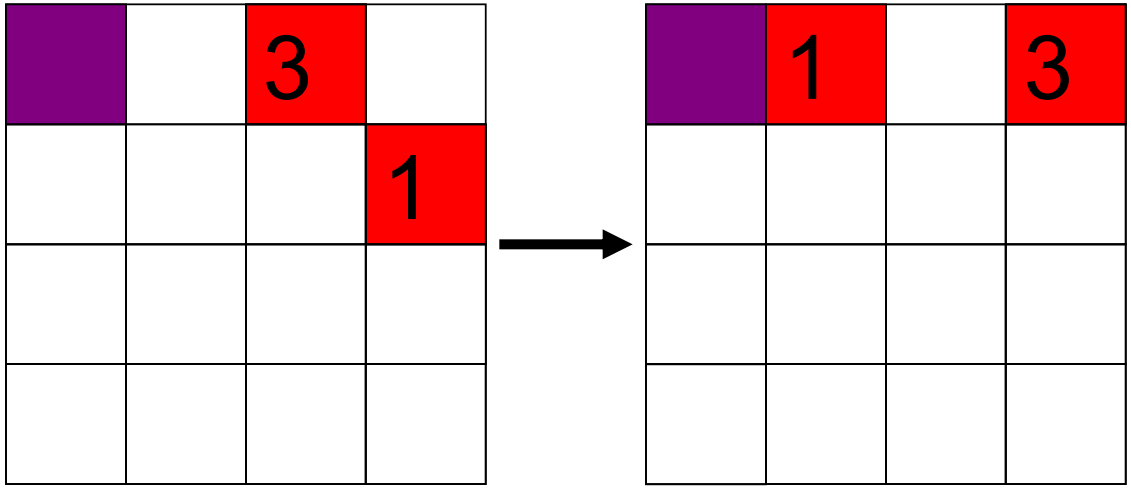
# Example: Linear Conflict




Manhattan distance is 2+2=4 moves

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

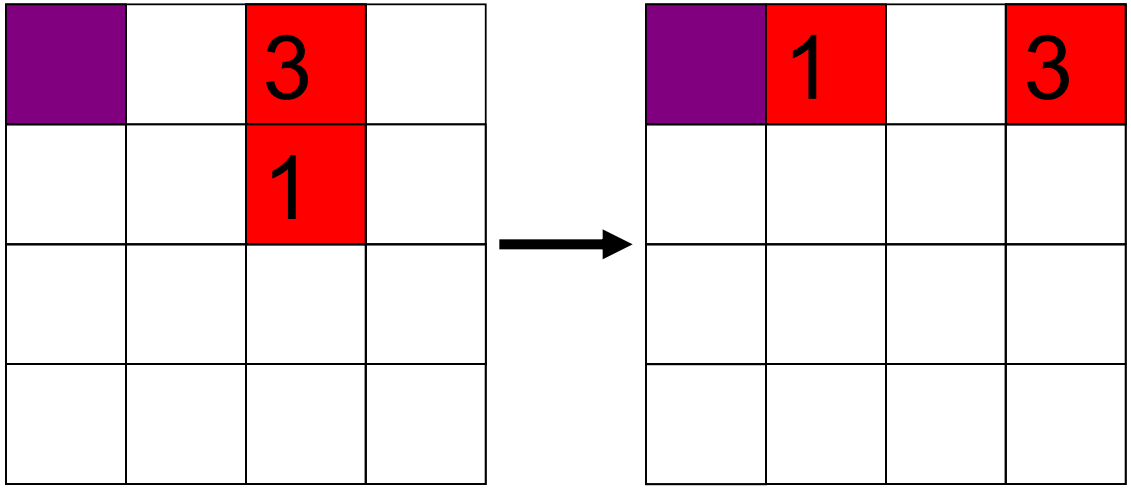
# Example: Linear Conflict




Manhattan distance is 2+2=4 moves

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

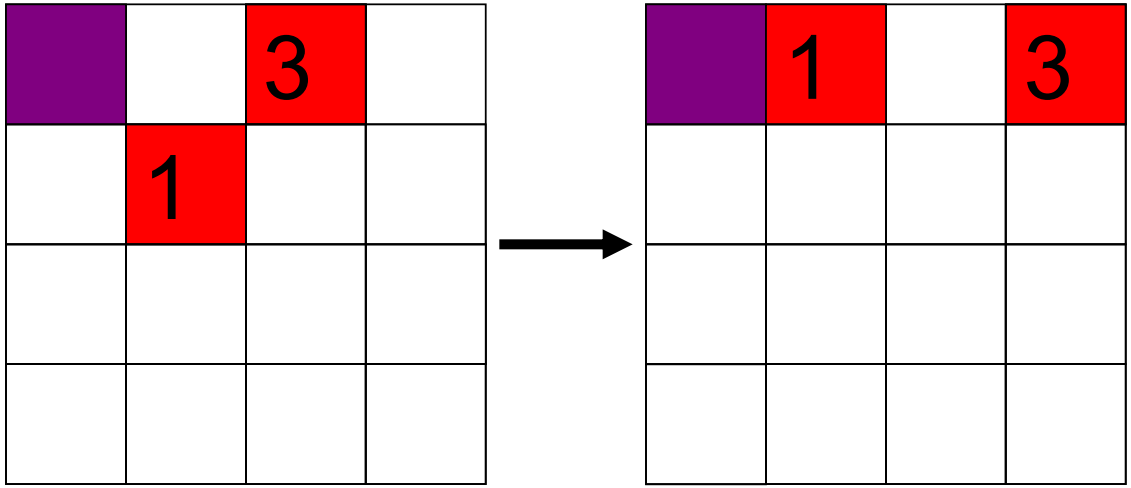
# Example: Linear Conflict




Manhattan distance is 2+2=4 moves

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

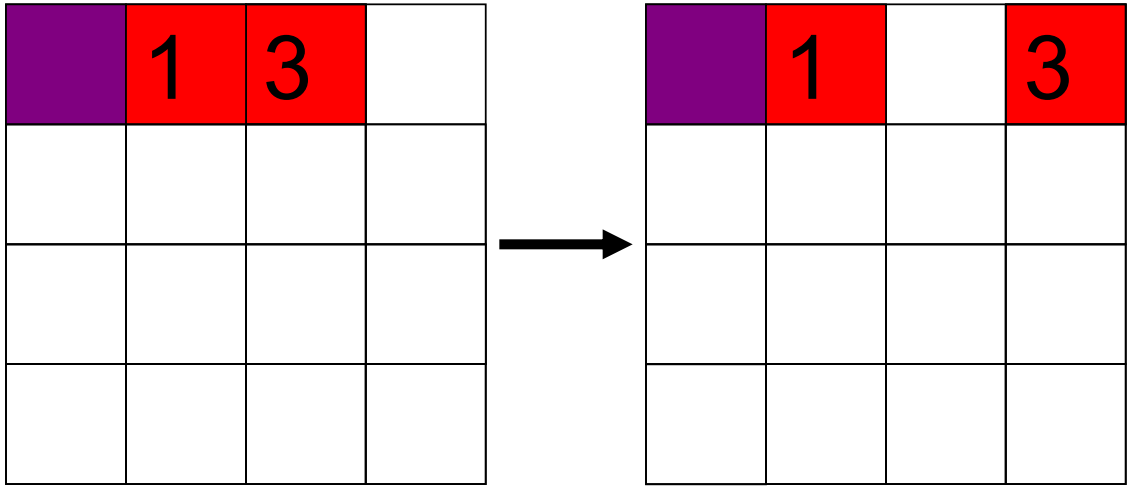
# Example: Linear Conflict




Manhattan distance is 2+2=4 moves

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

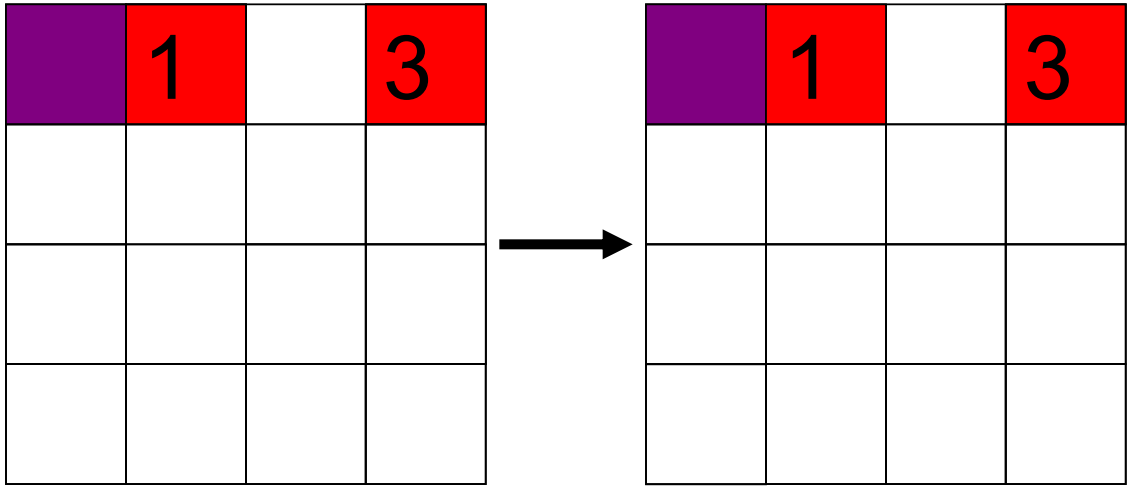
# Example: Linear Conflict




Manhattan distance is 2+2=4 moves

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

# Example: Linear Conflict




Manhattan distance is 2+2=4 moves, but linear conflict adds 2 additional moves.

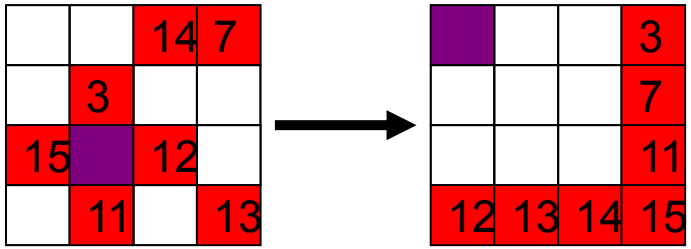
https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

# Linear Conflict Heuristic

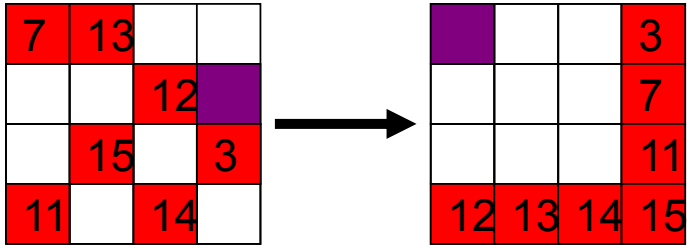
- Hansson, Mayer, and Yung, 1991
- Given two tiles in their goal row, but reversed in position, additional vertical moves can be added to Manhattan distance.
- Still not accurate enough to solve 24-Puzzle
- We can generalize this idea further.

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

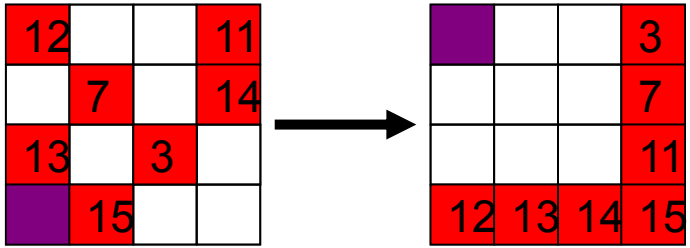
# More Complex Tile Interactions




M.d. is 19 moves, but 31 moves are needed.




M.d. is 20 moves, but 28 moves are needed




M.d. is 17 moves, but 27 moves are needed

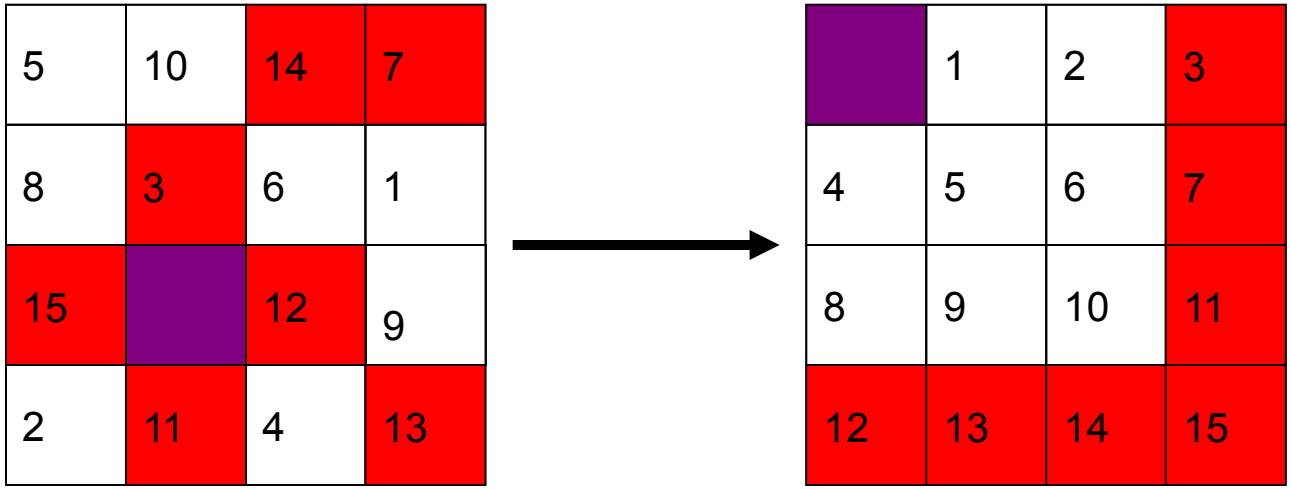
https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

# Pattern Database Heuristics

- Culberson and Schaeffer, 1996
- A pattern database is a complete set of such positions, with associated number of moves.
- e.g. a 7-tile pattern database for the Fifteen Puzzle contains 519 million entries.

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

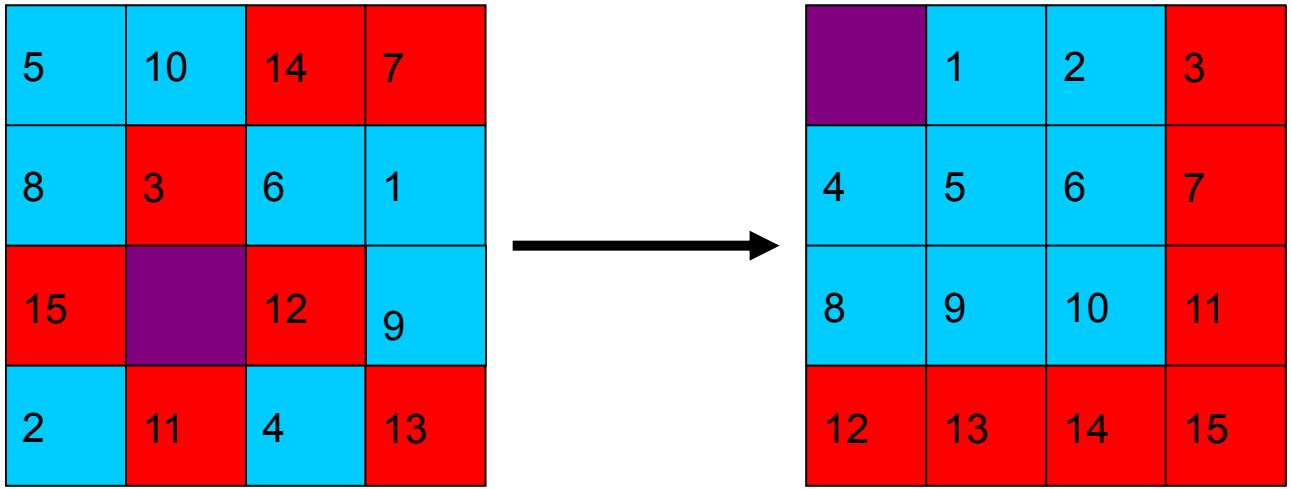
# Heuristics from Pattern Databases




31 moves is a lower bound on the total number of moves needed to solve this particular state.

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

# Combining Multiple Databases




31 moves needed to solve red tiles

22 moves need to solve blue tiles

Overall heuristic is maximum of 31 moves

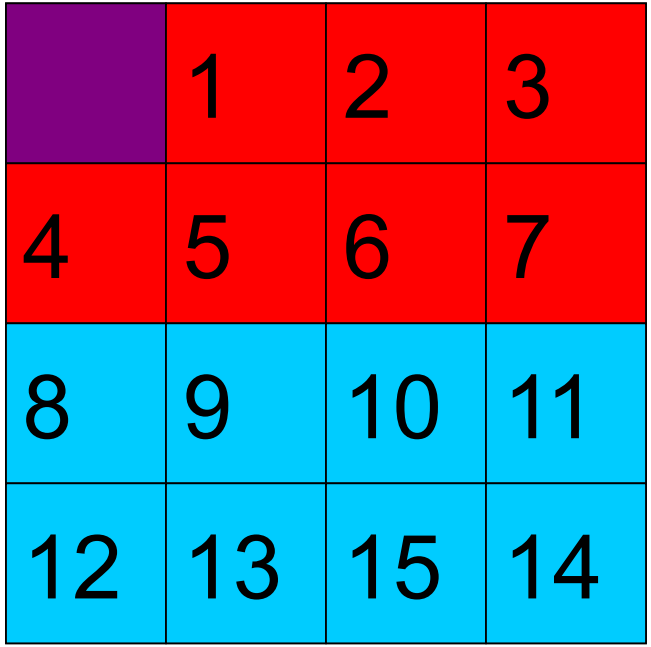
https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

# Additive Pattern Databases

- Culberson and Schaeffer counted all moves needed to correctly position the pattern tiles.
- In contrast, we count only moves of the pattern tiles, ignoring non-pattern moves.
- If no tile belongs to more than one pattern, then we can add their heuristic values.
- Manhattan distance is a special case of this, where each pattern contains a single tile.

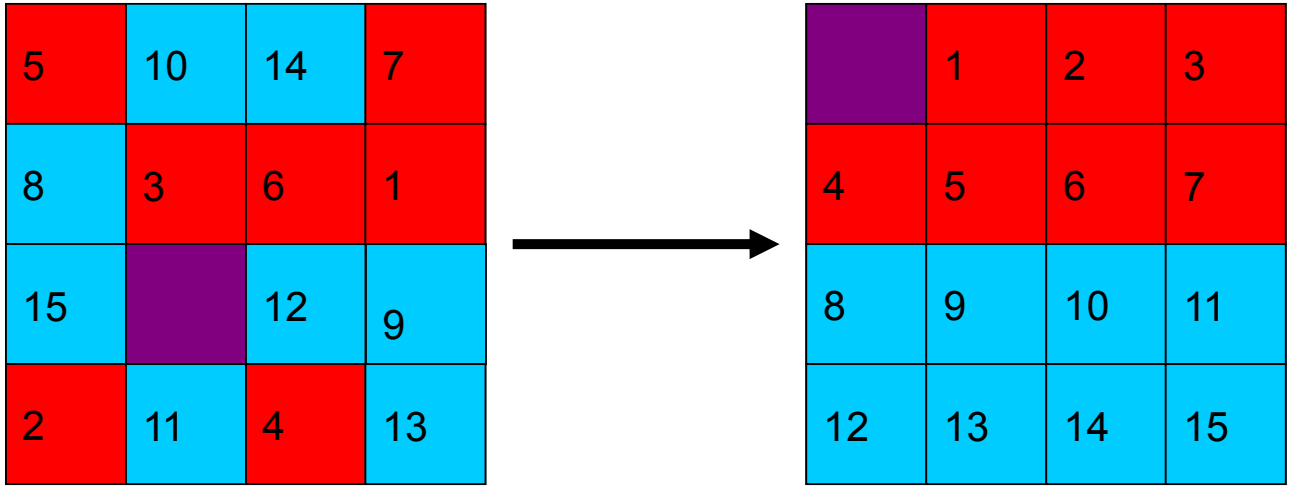
https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

# Example Additive Databases




The 7-tile database contains 58 million entries. The 8-tile database contains 519 million entries.

# Computing the Heuristic




20 moves needed to solve red tiles

25 moves needed to solve blue tiles

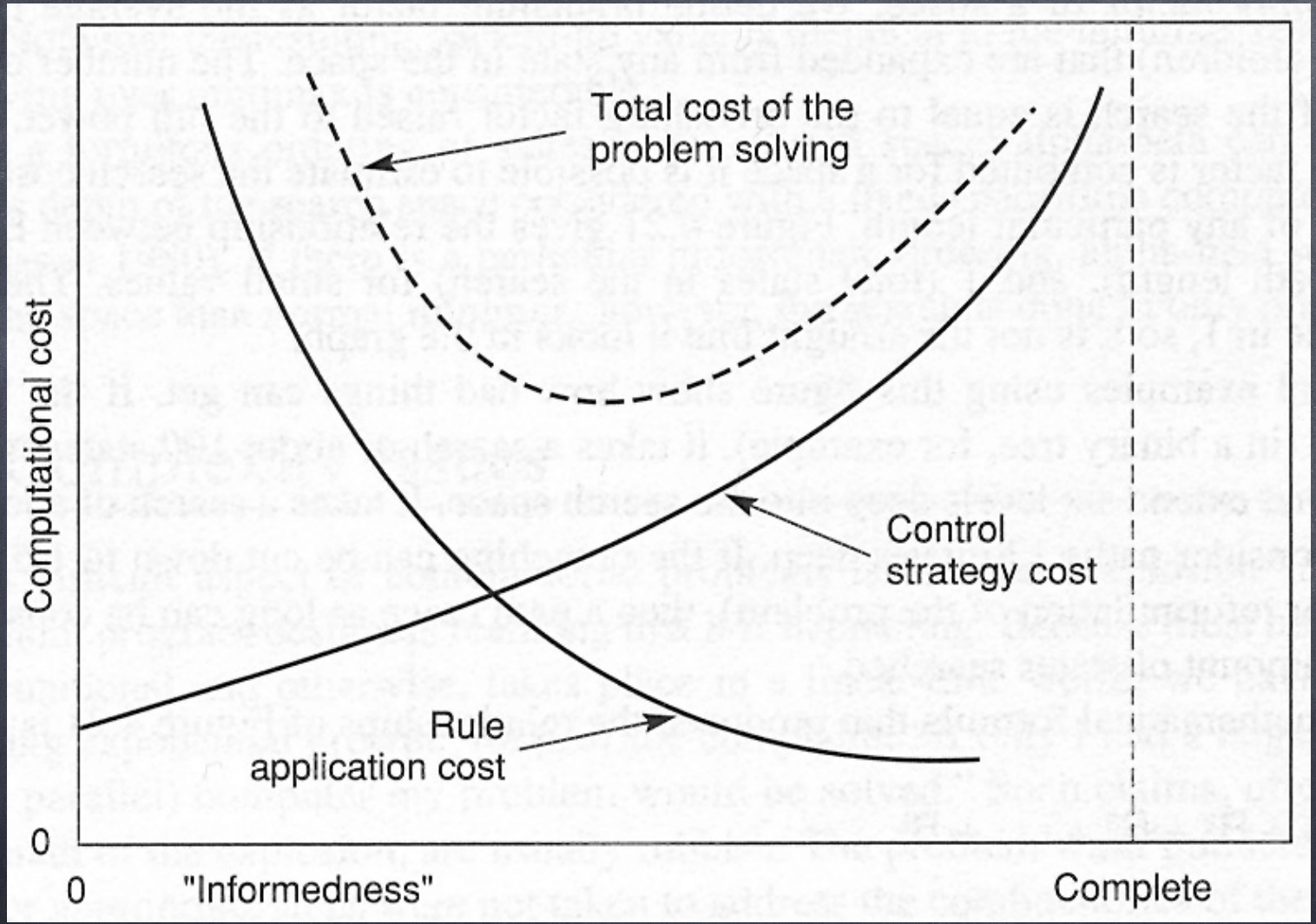
Overall heuristic is sum, or 20+25=45 moves

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf

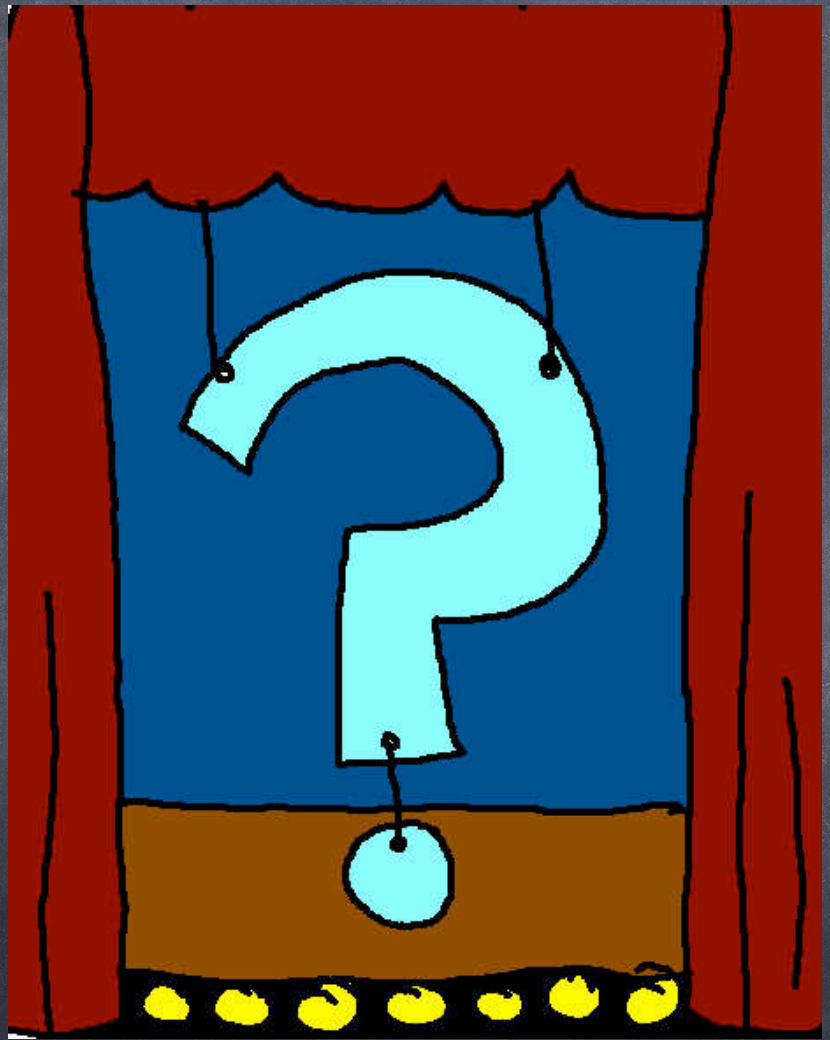
# Performance on 15 Puzzle

- IDA* with a heuristic based on these additive pattern databases can optimally solve random 15 puzzle instances in less than 29 milliseconds on average.
- This is about 1700 times faster than with Manhattan distance on the same machine.

https://courses.cs.washington.edu/courses/csep573/11wi/lectures/03-hsearch.pdf


Total cost of the problem solving
Control strategy cost
Rule application cost
Computational cost
0
0
"Informedness"
Complete

[Luger 2002]




Read Basic A* essay

Try A*

# Study Questions

- Distinguish between blind and informed search algorithms.
- Explain the three main problems with Hill Climbing. How are they solved by A*?
- What is an admissible heuristic?