# Heuristic (Informed) Search

(Where we try to be smarter in how we choose among alternatives)

R&N III: Chapter 3.5
R&N II: Chap. 4, Sect. 4.1–3

## Search Algorithm

```
1. INSERT(initial-node,FRINGE)
2. Repeat:
   a. If empty(FRINGE) then return failure
   b. n ← REMOVE(FRINGE)
   c. s ← STATE(n)
   d. If GOAL?(s) then return path or goal state
   e. For every state s' in SUCCESSORS(s)
      i.  Create a node n' as a successor of n
      ii. INSERT(n',FRINGE)
```

## Best-First Search

- It exploits state description to estimate how promising each search node is
- An evaluation function f maps each search node N to positive real number f(N)
- Traditionally, the smaller f(N), the more promising N
- Best-first search sorts the fringe in increasing f [random order is assumed among nodes with equal values of f]

## Best-First Search

- It exploits state description to estimate how promising each search node is
- An evaluation function f maps every search node N f(N)
- Usually, the s promising N
- Best-first search sorts the fringe in increasing f [random order is assumed among nodes with equal values of f]

"Best" only refers to the value of f, not to the quality of the actual path. Best-first search does not generate optimal paths in general

## How to construct an evaluation function?

- There are no limitations on f. Any function of your choice is acceptable. But will it help the search algorithm?
- The classical approach is to construct f(N) as an estimator of a solution path through N

## Heuristic Function

- The heuristic function h(N) estimates the distance of STATE(N) to a goal state

  Its value is **independent of the current search tree**; it depends only on STATE(N) and the goal test
- Example:

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

STATE(N)

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

Goal state

  - $h_1(N)$ = number of misplaced tiles = 6

## Other Examples

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

STATE(N)

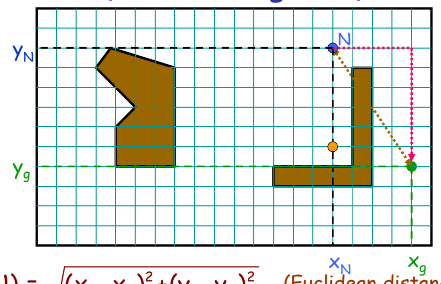
| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

Goal state

- $h_1(N)$ = number of misplaced tiles = 6
- h2(N) = sum of the (Manhattan) distances of every tile to its goal position = 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13
- h3(N) = sum of permutation inversions = 4 + 0 + 3 + 1 + 0 + 1 + 0 + 0 = 9

7

## Other Examples (Robot Navigation)



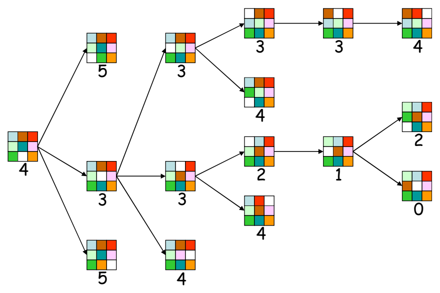
$h_1(N) = \sqrt{(x_N-x_g)^2+(y_N-y_g)^2}$ (Euclidean distance)

$h_2(N) = |x_N-x_g| + |y_N-y_g|$ (Manhattan distance)

8

## 8-Puzzle

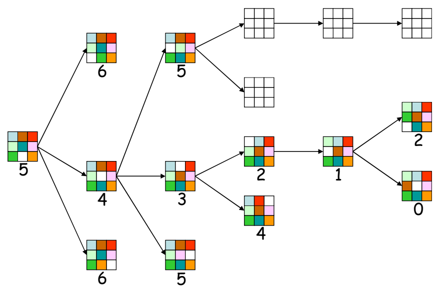
f(N) = h(N) = number of misplaced tiles



9

## 8-Puzzle

f(N) = h(N) = Σ distances of tiles to goal



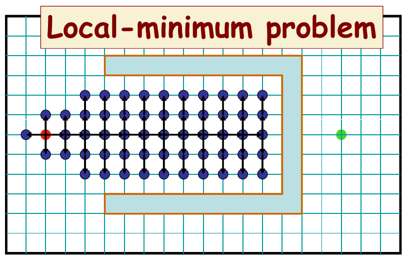
10

## Can we Prove Anything?

- If the state space is finite and we discard nodes that revisit states, the search is complete, but in general is not optimal
- If the state space is finite and we do not discard nodes that revisit states, in general the search is not complete
- If the state space is infinite, in general the search is not complete

11

## Best-First $\not\rightarrow$ Efficiency



f(N) = h(N) = straight distance to the goal

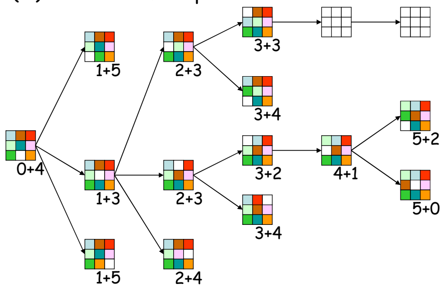
12

## Classical Evaluation Functions

- h(N): heuristic function
  [Independent of search tree]
- g(N): cost of the best path found so far between the initial node and N
  [Dependent on search tree]
- f(N) = h(N) → greedy best-first search
- f(N) = g(N) + h(N)

## 8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



## Algorithm A Search

- Orders open list according to
  F(n) = G(n) + H(n)
- Must have search package keep a record of what the best path found so far is so that the G(n) is as accurate as possible.

(go to handout on Algorithm A Search)

## Admissible Heuristic

- Let h*(N) be the cost of the optimal path from N to a goal node
- The heuristic function h(N) is admissible if:
  $0 \leq h(N) \leq h^*(N)$
- An admissible heuristic function is always optimistic !

## Admissible Heuristic

- Let h*(N) be the cost of the optimal path from N to a goal node
- The heuristic function h(N) is admissible if:
  $0 \leq h(N) \leq h^*(N)$
- An admissible heuristic function is always optimistic !

G is a goal node ➔ h(G) = 0

## Algorithm A*

- Algorithm A Search where we can prove that the heuristic function is admissible.
- This search is guaranteed to find an optimal solution if a solution exists!

## 8-Puzzle Heuristics

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

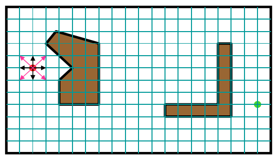
STATE(N)

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

Goal state

- $h_1(N)$ = number of misplaced tiles = 6
  is admissible
- h2(N) = sum of the (Manhattan) distances of every tile to its goal position
  = 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13
  is admissible
- h3(N) = sum of permutation inversions
  = 4 + 0 + 3 + 1 + 0 + 1 + 0 + 0 = 9
  is ??? [left as an exercise]

## Robot Navigation Heuristics



Cost of one horizontal/vertical step = 1
Cost of one diagonal step = $\sqrt{2}$

$$\left.\begin{array}{l} h_1(N) = \sqrt{(x_N-x_g)^2+(y_N-y_g)^2} \\ h_2(N) = |x_N-x_g| + |y_N-y_g| \end{array}\right\}$$ are both admissible

## A* Search
## (most popular algorithm in AI)

- f(N) = g(N) + h(N), where:
  - g(N) = cost of best path found so far to N
  - h(N) = admissible heuristic function
- for all arcs: $0 < \varepsilon \leq c(N,N')$
- "modified" search algorithm is used

➔ Best-first search is called *A* search*

## Result #1

A* is complete and optimal

[This result holds if nodes revisiting states are not discarded]

## Proof (1/2)

1) **If a solution exists, A* terminates and returns a solution**

For each node N on the fringe, $f(N) \geq d(N) \times \varepsilon$, where d(N) is the depth of N in the tree

As long as A* hasn't terminated, a node K on the fringe lies on a solution path

Since each node expansion increases the length of one path, K will eventually be selected for expansion

## Proof (2/2)

2) **Whenever A* chooses to expands a goal node, the path to this node is optimal**

$C^*$= h*(initial-node)

G': non-optimal goal node in the fringe
$f(G') = g(G') + h(G') = g(G') > C^*$

A node K in the fringe lies on an optimal path:
$f(K) = g(K) + h(K) < C^*$

So, G' is not be selected for expansion

## Time Limit Issue

- When a problem has no solution, A* runs for ever if the state space is infinite or states can be revisited an arbitrary number of times (the search tree can grow arbitrarily large). In other case, it may take a huge amount of time to terminate
- So, in practice, A* must be given a time limit. If it has not found a solution within this limit, it stops. Then there is no way to know if the problem has no solution or A* needed more time to find it
- In the past, when AI systems were "small" and solving a single search problem at a time, this was not too much of a concern. As AI systems become larger, they have to solve a multitude of search problems concurrently. Then, a question arises: **What should be the time limit for each of them?** More on this in the lecture on Motion Planning ...

25

## Time Limit Issue

- When a problem has no solution, A* runs for ever if the state space is infinite or states can be revisited an arbitrary number of times (the search tree can

Hence, the usefulness of a simple test, like in the (2n-1)-puzzle, that determines if the goal is reachable

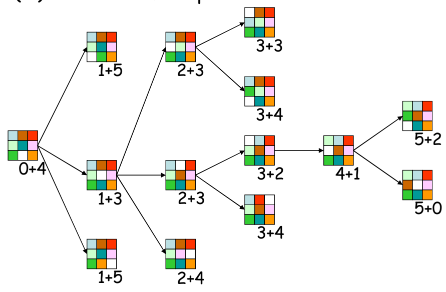
Unfortunately, such a test rarely exists

ing a single search problem at a time, this was not too much of a concern. As AI systems become larger, they have to solve a multitude of search problems concurrently. Then, a question arises: **What should be the time limit for each of them?** More on this in future lectures ...

26

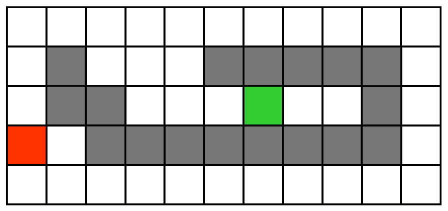
## 8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



27

## Robot Navigation



28

## Robot Navigation

f(N) = h(N), with h(N) = Manhattan distance to the goal
(not A*)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 3 | 4 | 5 | 6 |
| 7 | | 5 | 4 | 3 | | | | | | 5 |
| 6 | | | 3 | 2 | 1 | 0 | 1 | 2 | | 4 |
| 7 | 6 | | | | | | | | | 5 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 3 | 4 | 5 | 6 |

29

## Robot Navigation

f(N) = h(N), with h(N) = Manhattan distance to the goal
(not A*)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 3 | 4 | 5 | 6 |
| 7 | | 5 | 4 | 3 | | | | | | 5 |
| 6 | | | 3 | 2 | 1 | 0 | 1 | 2 | | 4 |
| 7 | 6 | | | | | | | | | 5 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 3 | 4 | 5 | 6 |

30

## Robot Navigation

f(N) = g(N)+h(N), with h(N) = Manhattan distance to goal (A*)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8+3 | 7+4 | 6+3 | 5+6 | 4+7 | 3+8 | 2+9 | 3+10 | 4 | 5 | 6 |
| 7+2 | ▨ | 5+6 | 4+7 | 3+8 | ▨ | ▨ | ▨ | ▨ | ▨ | 5 |
| 6+1 | ▨ | ▨ | 3 | 2+9 | 1+10 | 0+11 | 1 | 2 | ▨ | 4 |
| 7+0 | 6+1 | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | 5 |
| 8+1 | 7+2 | 6+3 | 5+4 | 4+5 | 3+6 | 2+7 | 3+8 | 4 | 5 | 6 |

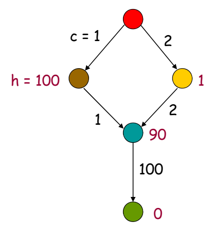
31

## How to create an admissible h?

- An admissible heuristic can usually be seen as the cost of an optimal solution to a relaxed problem (one obtained by removing constraints)
- In robot navigation:
  - The Manhattan distance corresponds to removing the obstacles
  - The Euclidean distance corresponds to removing both the obstacles and the constraint that the robot moves on a grid
- More on this topic later

32

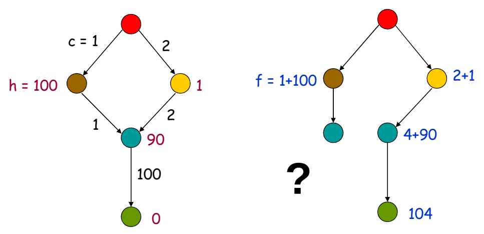
## What to do with revisited states?



The heuristic h is clearly admissible

33

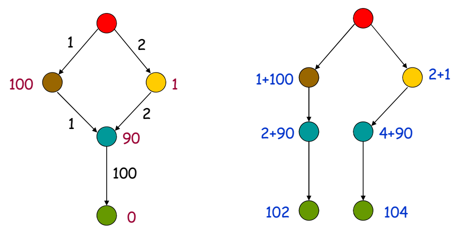
## What to do with revisited states?



If we discard this new node, then the search algorithm expands the goal node next and returns a non-optimal solution

34

## What to do with revisited states?

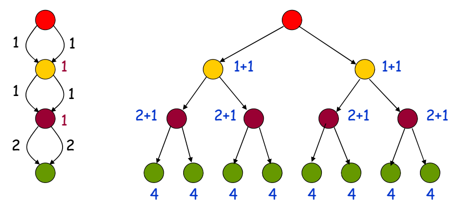


Instead, if we do not discard nodes revisiting states, the search terminates with an optimal solution

35

## But ...

If we do not discard nodes revisiting states, the size of the search tree can be exponential in the number of visited states



36

- It is not harmful to discard a node revisiting a state if the new path to this state has higher cost than the previous one
- A* remains optimal, but the size of the search tree can still be exponential in the worst case
- Fortunately, for a large family of admissible heuristics – consistent heuristics – there is a much easier way of dealing with revisited states

37

## Consistent Heuristic

A heuristic h is consistent if

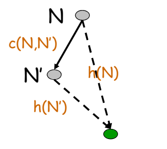
1) for each node N and each child N' of N:

$h(N) \le c(N,N') + h(N')$

[Intuition: h gets more and more precise as we get deeper in the search tree]

2) for each goal node G:

$h(G) = 0$



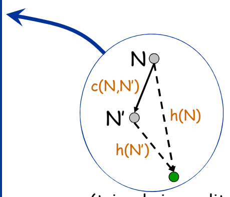
(triangle inequality)

The heuristic is also said to be monotone

38

## Consistency Violation

If h tells that N is 100 units from the goal, then moving from N along an arc costing 10 units should **not** lead to a node N' that h estimates to be 10 units away from the goal



(triangle inequality)

39

## Admissibility and Consistency

- A consistent heuristic is also admissible
- An admissible heuristic may not be consistent, but many admissible heuristics are consistent

40

## 8-Puzzle

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

STATE(N)

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

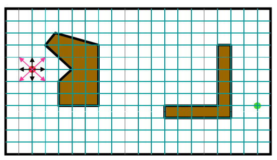
goal

- h1(N) = number of misplaced tiles
- h2(N) = sum of the (Manhattan) distances of every tile to its goal position

are both consistent

41

## Robot navigation



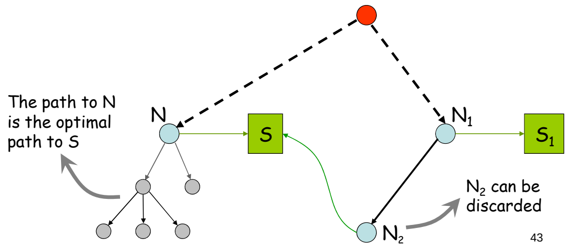
Cost of one horizontal/vertical step = 1
Cost of one diagonal step = √2

$$\left.\begin{aligned} h_1(N) &= \sqrt{(x_N-x_g)^2+(y_N-y_g)^2} \\ h_2(N) &= |x_N-x_g| + |y_N-y_g| \end{aligned}\right\}$$ are both consistent

42

## Result #2

If h is consistent, then whenever A* expands a node, it has already found an optimal path to this node's state



## Proof

1) Consider a node N and its child N'
   Since h is consistent: $h(N) \leq c(N,N')+h(N')$

   $f(N) = g(N)+h(N) \leq g(N)+c(N,N')+h(N') = f(N')$
   So, f is non-decreasing along any path

2) If K is selected for expansion, then any other node K' in the fringe verifies $f(K') \geq f(K)$

   So, if one node K' lies on another path to the state of K, the cost of this other path is no smaller than the path to K

## Revisited States with Consistent Heuristic

- When a node is expanded, store its state into CLOSED
- When a new node N is generated:
  - If STATE(N) is in CLOSED, discard N
  - If there exits a node N' in the fringe such that STATE(N') = STATE(N), discard the node – N or N' – with the largest f

## Worst-Case Complexity of A* when State Space is Finite (1/3)

- Assume a state graph of n states and r arcs
- Two cases:
  a) If the number of successors of any state is O(n), then $r = O(n^2)$; the state graph is dense
  b) If it is O(1), then r = O(n); the graph is sparse
  [In most search problem, the graph is sparse]
- CLOSED is implemented as a hash-table with O(1) access time
- Heuristic h is consistent

## Worst-Case Complexity of A* when State Space is Finite (2/3)

- The fringe is implemented as a list
  → linear scan to find best node

| | |
|---|---|
| ▪ Number of attempted add-to-fringe operations: | O(r) |
| ▪ Time to add a node to the fringe: | O(1) |
| ▪ Number of node expansions: | O(n) |
| ▪ Time to select a node from the fringe: | O(n) |
| ▪ Total time complexity: | $O(r + n^2) = O(n^2)$ |
| ▪ Space complexity: | O(n) |

[A node need not pointing to its children]

## Worst-Case Complexity of A* when State Space is Finite (3/3)

- The fringe is implemented as a priority queue

| | |
|---|---|
| ▪ Number of attempted add-to-fringe operations: | O(r) |
| ▪ Time to add a node to the fringe | O(log n) |
| ▪ Number of node expansions | O(n) |
| ▪ Time to select a node from the fringe | O(log n) |
| ▪ Total time complexity: | O(r log n + n log n) |
| • If dense state graph: | $O(n^2 \log n)$ |
| • If sparse state graph: | O(n log n) |
| ▪ Space complexity: | O(n) |

## Worst-Case Complexity of A* when State Space is Finite (3/3)

- The fringe is implemented as a priority queue

So, for large state spaces with reasonable branching factors (→ sparse state graphs), it is preferable to implement the fringe as a priority queue

- If dense state graph: $O(n^2 \log n)$
- If sparse state graph: $O(n \log n)$
- Space complexity: $O(n)$

49

## Is A* with some consistent heuristic all what we need?

No !

- The previous result only says that A*'s worst-case time complexity is low-polynomial in the size of the state space, but this size may be exponential in other parameters (e.g., path lengths) depending on the input description
- The state space can even be infinite
- There are **very dumb** consistent heuristics

50

## $h \equiv 0$

- It is consistent (hence, admissible) !
- A* with $h \equiv 0$ is uniform-cost search
- Breadth-first and uniform-cost are particular cases of A*

51

## Heuristic Accuracy

Let $h_1$ and $h_2$ be two consistent heuristics such that for all nodes N:

$$h_1(N) \le h_2(N)$$

$h_2$ is said to be more accurate (or more informed) than $h_1$

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

STATE(N)

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

Goal state

- $h_1(N)$ = number of misplaced tiles
- $h_2(N)$ = sum of distances of every tile to its goal position
- $h_2$ is more accurate than $h_1$

52

## Result #3

- Let $h_2$ be more informed than $h_1$
- Let $A_1$* be A* using $h_1$ and $A_2$* be A* using $h_2$
- Whenever a solution exists, all the nodes expanded by $A_2$*, except possibly the goal node, are also expanded by $A_1$*

53

## Proof

- $C^* = h^*(\text{initial-node})$
- Every node N such that $f(N) < C^*$ is eventually expanded
- Every node N such that $h(N) < C^* - g(N)$ is eventually expanded
- Since $h_1(N) \le h_2(N)$, every non-goal node expanded by $A_2$* is also expanded by $A_1$*
- If $f(N) = C^*$, N is a goal node; only one such node is expanded [it may not be the same one for $A_1$* and $A_2$*]

54

## Effective Branching Factor

- It is used as a measure the effectiveness of a heuristic
- Let n be the total number of nodes generated by A* for a particular problem and d the depth of the solution
- The effective branching factor b* is defined by $n = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$

55

## Experimental Results
(see R&N for details)

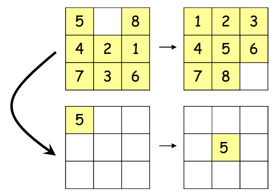
- 8-puzzle with:
  - $h_1$ = number of misplaced tiles
  - $h_2$ = sum of distances of tiles to their goal positions
- Random generation of many problem instances
- Average branching factors (number of expanded nodes):

| d | IDS | $A_1$* | $A_2$* |
|---|---|---|---|
| 2 | 2.45 | 1.79 | 1.79 |
| 6 | 2.73 | 1.34 | 1.30 |
| 12 | 2.78 (3,644,035) | 1.42 (227) | 1.24 (73) |
| 16 | -- | 1.45 | 1.25 |
| 20 | -- | 1.47 | 1.27 |
| 24 | -- | 1.48 (39,135) | 1.26 (1,641) |

56

## How to create good heuristic?

- By solving relaxed problems at each node
- In the 8-puzzle, the sum of the distances of each tile to its goal position ($h_2$) corresponds to solving 8 simple problems:

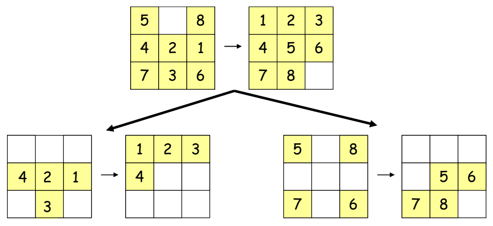


- It ignores negative interactions among tiles

57

## Can we do better?

- For example, we could consider two more complex relaxed problems:

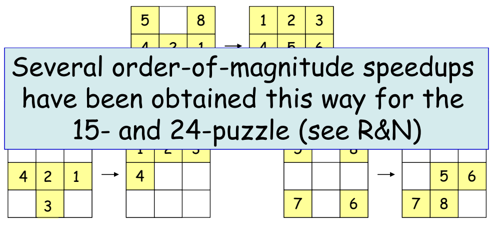


- → $h = d_{1234} + d_{5678}$ [disjoint pattern heuristic]
- These distances could have been precomputed in a database [left as an exercise]

58

## Can we do better?

- For example, we could consider two more complex relaxed problems:



Several order-of-magnitude speedups have been obtained this way for the 15- and 24-puzzle (see R&N)

- → $h = d_{1234} + d_{5678}$
- These distances could have been precomputed [left as an exercise]

59

## On Completeness and Optimality

- A* with a consistent heuristic has nice properties: completeness, optimality, no need to revisit states
- Theoretical completeness does not mean "practical" completeness if you must wait too long to get a solution (remember the time limit issue)
- So, if one can't design an accurate consistent heuristic, it may be better to settle for a non-admissible heuristic that "works well in practice", even completeness and optimality are no longer guaranteed

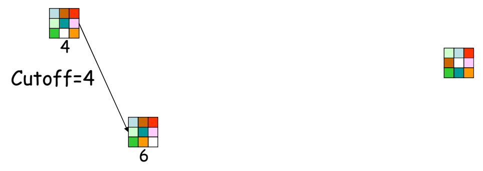
60

## Iterative Deepening A* (IDA*)

- Idea: Reduce memory requirement of A* by applying cutoff on values of f
- Consistent heuristic h
- Algorithm IDA*:
  1. Initialize cutoff to f(initial-node)
  2. Repeat:
     a. Perform depth-first search by expanding all nodes N such that f(N) < cutoff
     b. Reset cutoff to smallest value f of non-expanded (leaf) nodes

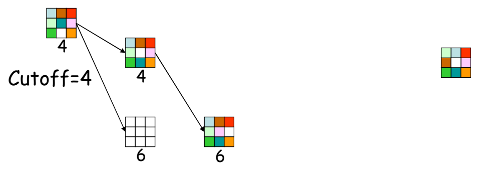
## 8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



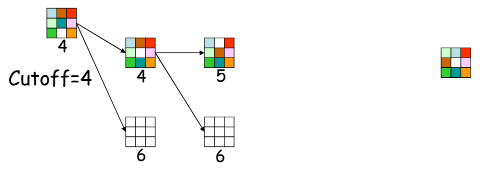
## 8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



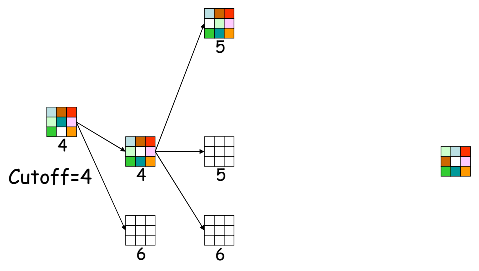
## 8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



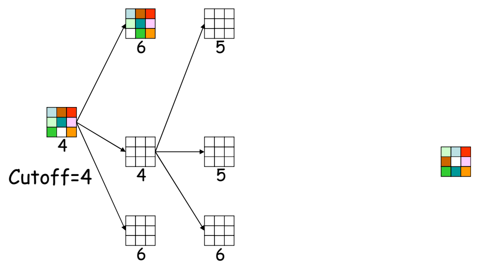
## 8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



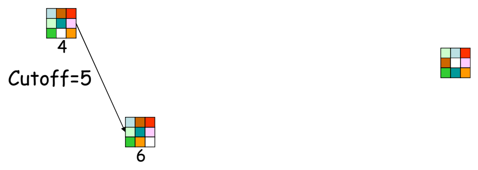
## 8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



## 8-Puzzle

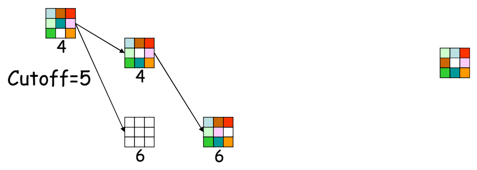
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



67

## 8-Puzzle

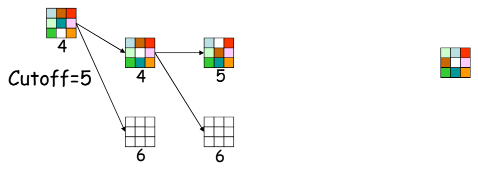
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



68

## 8-Puzzle

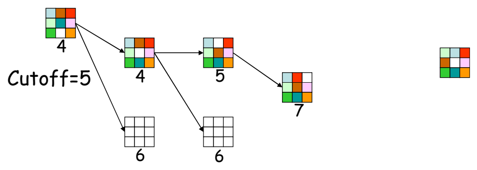
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



69

## 8-Puzzle

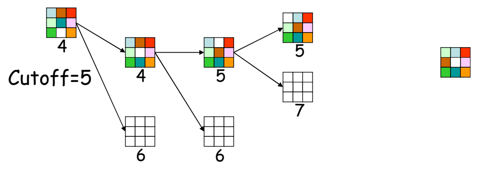
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



70

## 8-Puzzle

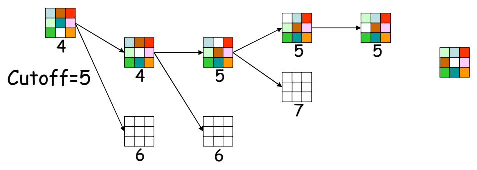
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



71

## 8-Puzzle

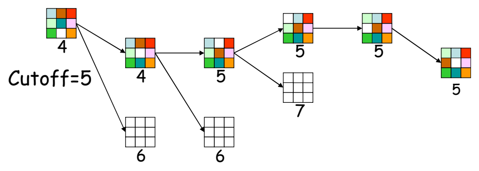
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



72

## 8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles

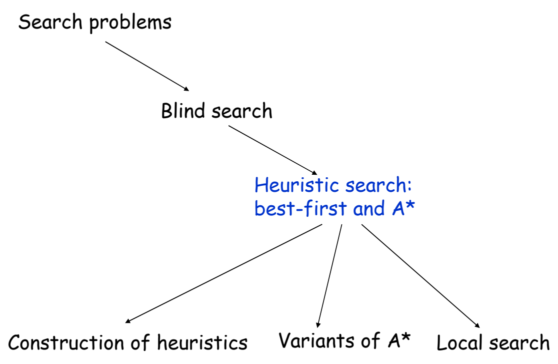


## Advantages/Drawbacks of IDA*

- Advantages:
  - Still complete and optimal
  - Requires less memory than A*
  - Avoid the overhead to sort the fringe
- Drawbacks:
  - Can't avoid revisiting states not on the current path
  - Available memory is poorly used

## SMA* (Simplified Memory-bounded A*)

- Works like A* until memory is full
- Then SMA* drops the node in the fringe with the largest f value and "backs up" this value to its parent
- When all children of a node N have been dropped, the smallest backed up value replaces f(N)
- In this way, SMA* the root of an erased subtree remembers the best path in that subtree
- SMA* will regenerate this subtree only if all other nodes in the fringe have greater f values
- SMA* generates the best solution path that fits in memory
- SMA* can't completely avoid revisiting states, but it can do a better job at this that IDA*



Search problems
Blind search
Heuristic search: best-first and A*
Construction of heuristics
Variants of A*
Local search

## When to Use Search Techniques?

1) The search space is small, and
   - No other technique is available, or
   - Develop a more efficient technique is not worth the effort
2) The search space is large, and
   - No other available technique is available, and
   - There exist "good" heuristics