Theory and Practice of Artificial Intelligence

Search

Daniel Polani

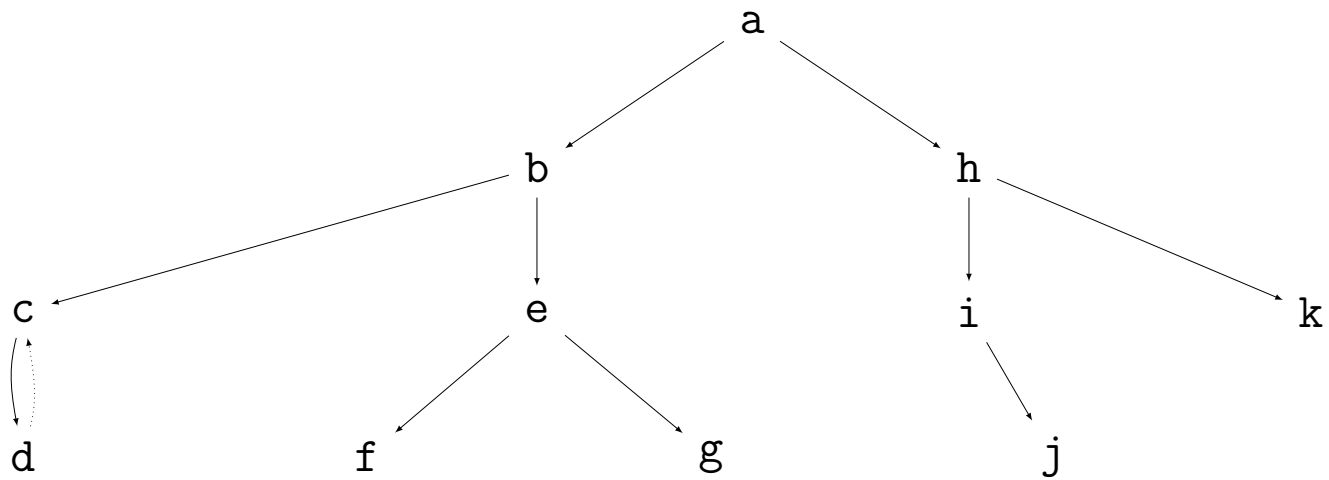
School of Computer Science
University of Hertfordshire

March 9, 2017

Depth-First Search

finds solution path `Sol` from given node `N` to some goal node (there can be several):

- if `N` is a goal node, `Sol=[N]` else
- if there is a successor node `N1` of `N` such that there is a path `Sol1` from `N1` to a goal node `Sol=[N,Sol1]`



Breadth-First Search

finds solution path `Sol` from given node `N` to some goal node:

- if `N` is a goal node, `Sol=[N]` else
- generate one-step extension of paths in candidate lists, adding extensions to that list.
- more detailed: given list of candidate paths
  - if first path contains goal node as head, solution
  - else
    - remove first path from candidate list
    - generate set of one-step extensions
    - append them to list of candidates
    - call breadth-first search on this list

Search III

Best-First Search

- breadth-first selects the shortest path
- best-first selects the least "costly" path

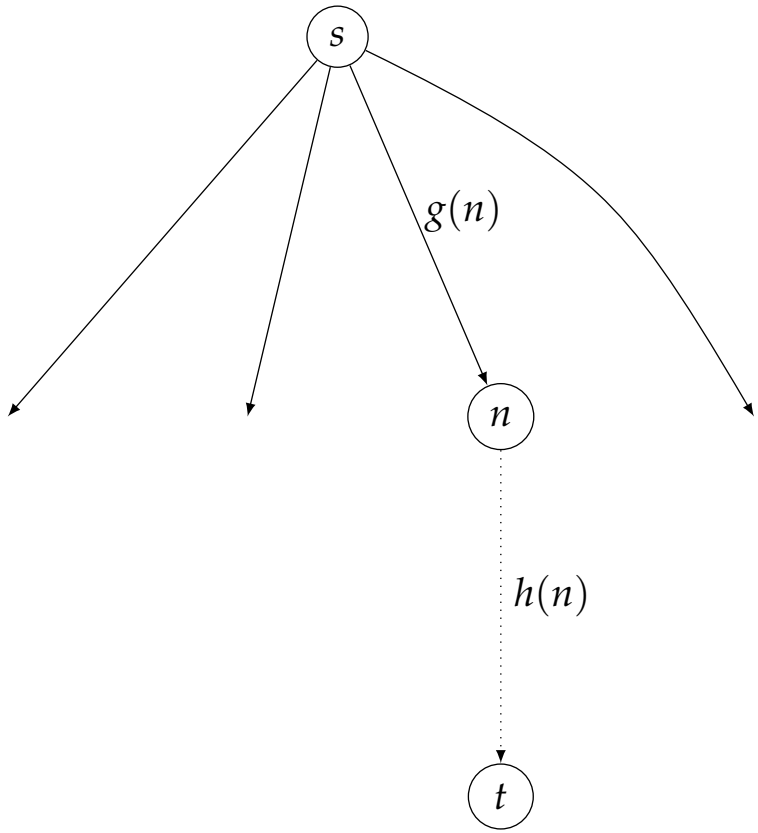
For that

define $c(n, n')$ as cost for moving from node n to n'

Heuristics

- consider a heuristic estimator f
- $f(n)$ estimates "cost" of n, i.e. the cost of best solution path from start node s to some goal node, say t, provided that path goes via n

$$f(n) = \underbrace{g(n)}_{\text{actual cost from } s \text{ to } n \text{ not necessarily optimal, estimate of minimal cost from } s \text{ to } n} + \underbrace{h(n)}_{\text{guesswork! no universal solution}}$$

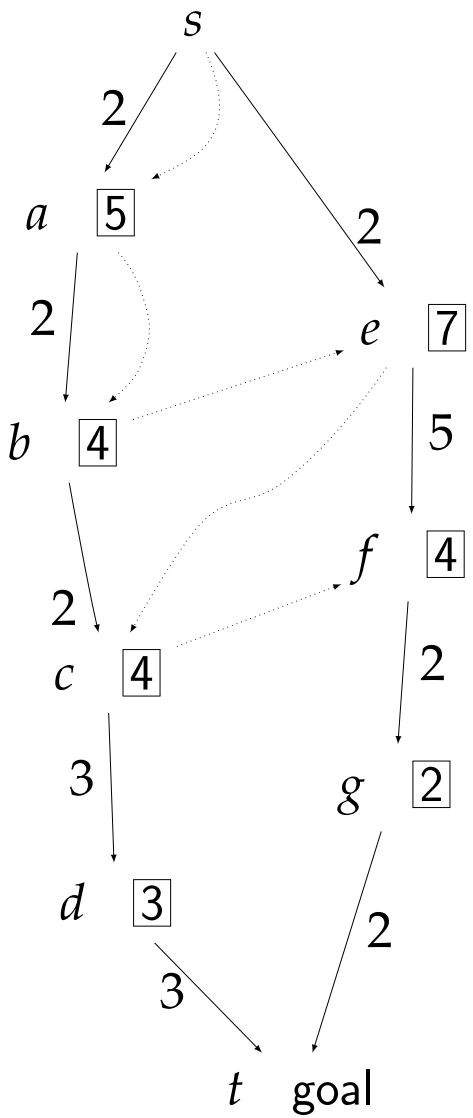


Best-First Idea

Competing Subtrees: among competitors, only one subtree is active at a time: the most promising, i.e. that with the lowest f-value switch to alternative, if that changes

Example: $f(X) = g(X) + h(X)$

Budget of subtree spent until exhausted, switching to other tree



Best-First Search

Best-First Search

finds cheapest solution path `Sol` from given node `N` to some goal node:

1. if `N` is a goal node, `Sol=[N]` else
2. initialize heap of candidate paths (sorted according to cost, *best first*), containing `[N]`
3. pop head of heap (best candidate solution so far) as candidate solution `C`
4. if `C` has node as head, solution
5. generate all one-step extensions of `C`, add them to heap
6. go to 3

Admissibility

Def.: A search algorithm is *admissible* if it always produces an optimal solution.

Note: Above algorithm immediately produces an optimal solution if for each node n, $h^*(n)$ is the cost of an optimal path from n to some goal node.

Note: The Best-First algorithm is a variant of the noted A^* algorithm.

Theorem: Best-First is admissible if it uses an *optimistic* heuristic h, i.e. h with

$$h(n) \leq h^*(n)$$

Default: setting maximally optimistic $h(n) := 0$ is always admissible (gives Breadth-First-Search), but has no predictive power.

Heuristic Construction I

Note

Consider: formalization of a search problem

- often one needs to fulfil set of constraints
- which make calculating the optimal moves difficult

Idea:

- release one or more of these constraints
- the search problem then often becomes much easier
- ensuing h' for the less or unconstrained problem typically admissible for the original problem
- since removing the constraint does not increase the path length to the solution

Thus: releasing constraints on the original problem leads to admissible heuristics

(read up details on the heuristics for the 8-puzzle in Pearl and Russell/Norvig)

Mechanical Generation of Admissible Heuristics I

(Pearl 1984)

- consider **8-puzzle**
- move tile to adjacent empty tile
- until tiles all in order

- some start position

4	5	1
3	2	6
7		8

Mechanical Generation of Admissible Heuristics II

(Pearl 1984)

- consider **8-puzzle**
- move tile to adjacent empty tile
- until tiles all in order

- some start position
- ordered end position

1	2	3
4	5	6
7	8	

Mechanical Generation of Admissible Heuristics III

(Pearl 1984)

Formalize Idea:

- relax rules describing systems
- formalism to describe rules

Nilsson's STRIPS Formal Rule System

Preconditions: necessary predicates for invoking action

Additions: predicates to be added after action

Deletions: predicates no longer true after action

Mechanical Generation of Admissible Heuristics IV

(Pearl 1984)

Formalize Idea:
- relax rules describing systems
- formalism to describe rules

Nilsson's STRIPS Formal Rule System

Preconditions: necessary predicates for invoking action

Additions: predicates to be added after action

Deletions: predicates no longer true after action

For 8-Puzzle

ON(x,y)**:** tile x is on cell y

CLEAR(y)**:** cell y is clear of tiles

ADJ(y,z)**:** cell y is adjacent to cell z

Mechanical Generation of Admissible Heuristics V

(Pearl 1984)

State Description:

$$\texttt{ON}(X_1, C_1), \texttt{ON}(X_2, C_2), \ldots, \texttt{ON}(X_8, C_8), \texttt{CLEAR}(C_9)$$

Board Description:

$$\texttt{ADJ}(C_1, C_2), \texttt{ADJ}(C_1, C_4), \ldots$$

$\texttt{MOVE}(x, y, z)$

Precondition: $\texttt{ON}(x, y), \texttt{CLEAR}(z), \texttt{ADJ}(y, z)$

Add List: $\texttt{ON}(x, z), \texttt{CLEAR}(y)$

Delete List: $\texttt{ON}(x, y), \texttt{CLEAR}(z)$

We Seek

a sequence of $\texttt{MOVE}(x, y, z)$ transforming initial state to a state satisfying the goal criteria

Modification: $\texttt{MOVE}(x, y, z)$

Precondition: $\texttt{ON}(x, y), \texttt{CLEAR}(z), \texttt{ADJ}(y, z)$

Add List: $\texttt{ON}(x, z), \texttt{CLEAR}(y)$

Delete List: $\texttt{ON}(x, y), \texttt{CLEAR}(z)$

Modification: $\texttt{MOVE}'(x,y,z)$

Precondition: $\texttt{ON}(x,y), \sout{\texttt{CLEAR}(z), \texttt{ADJ}(y,z)}$

Add List: $\texttt{ON}(x,z), \texttt{CLEAR}(y)$

Delete List: $\texttt{ON}(x,y), \texttt{CLEAR}(z)$

Relaxation

- delete $\texttt{CLEAR}(z), \texttt{ADJ}(y,z)$ from precondition
- i.e. a move now does not require a cell to be adjacent and empty
- successively "jump" tiles to target positions, "on top" of other tiles

Heuristic h_1

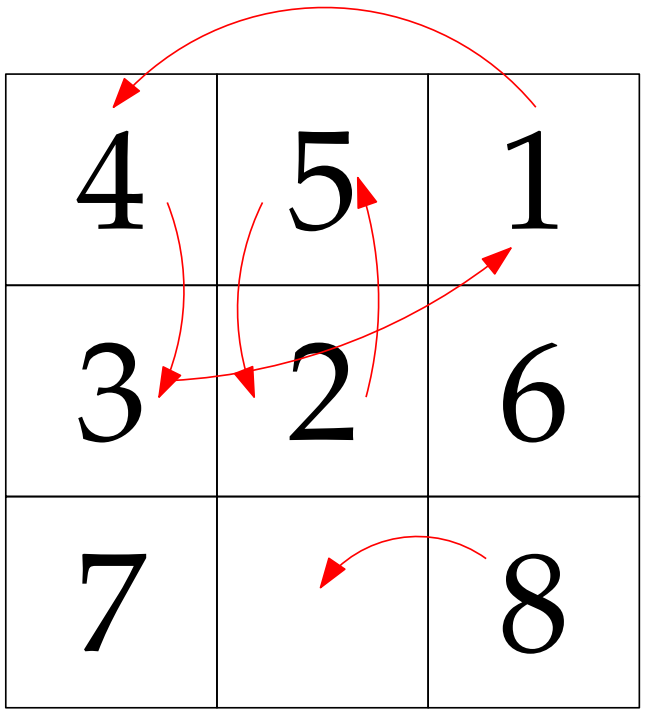
- number of misplaced tiles
- $h_1(s) = |\{x \mid \texttt{ON}(x,y), x \neq y\}|$

  Here, s is the whole state (represented by the currently valid predicates). Note that x is implicitly a tile, and y a cell, via the predicate.

Heuristic h_1

- number of misplaced tiles
- $h_1(s) = |\{x \mid \texttt{ON}(x,y), x \neq y\}|$

  Here, s is the whole state (represented by the currently valid predicates). Note that x is implicitly a tile, and y a cell, via the predicate.



Modification: $\texttt{MOVE}(x, y, z)$

Precondition: $\texttt{ON}(x, y), \texttt{CLEAR}(z), \texttt{ADJ}(y, z)$

Add List: $\texttt{ON}(x, z), \texttt{CLEAR}(y)$

Delete List: $\texttt{ON}(x, y), \texttt{CLEAR}(z)$

Modification: $\texttt{MOVE}''(x, y, z)$

Precondition: $\texttt{ON}(x, y), \sout{\texttt{CLEAR}(z)}, \texttt{ADJ}(y, z)$

Add List: $\texttt{ON}(x, z), \texttt{CLEAR}(y)$

Delete List: $\texttt{ON}(x, y), \texttt{CLEAR}(z)$

Relaxation

- delete only $\texttt{CLEAR}(z)$ from precondition
- i.e. a move requires a cell to be adjacent, but not empty
- successively shift tiles to neighbouring positions, closer to target, but "on top" of other tiles

Heuristic h_2

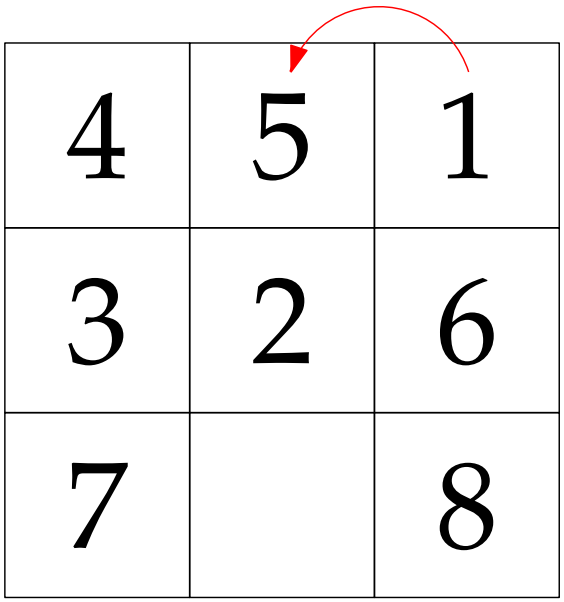
- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.

Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

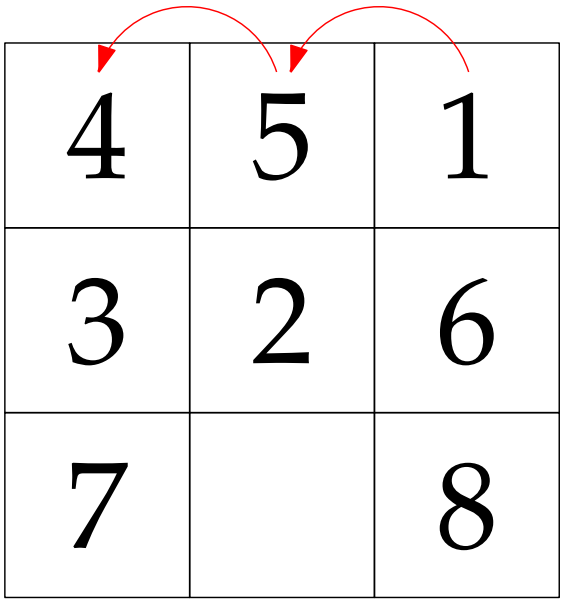
pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

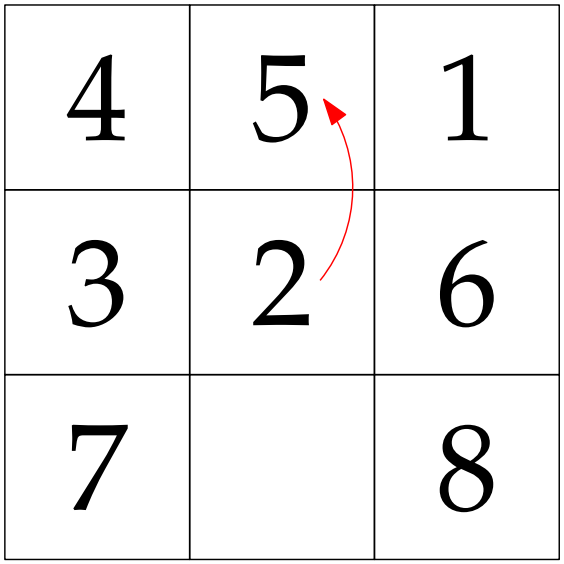
pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

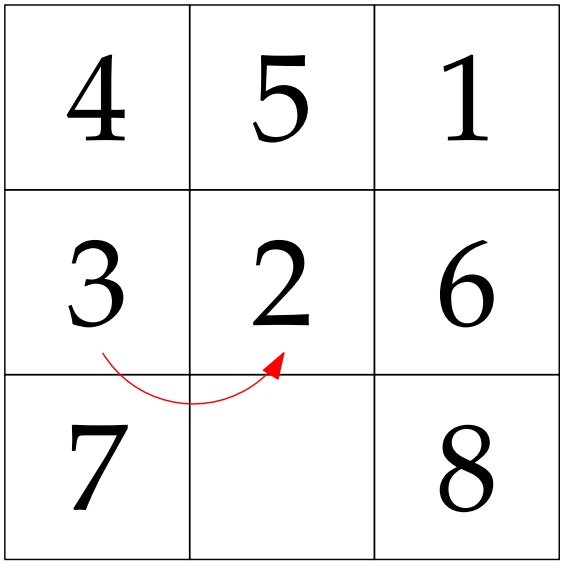
pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

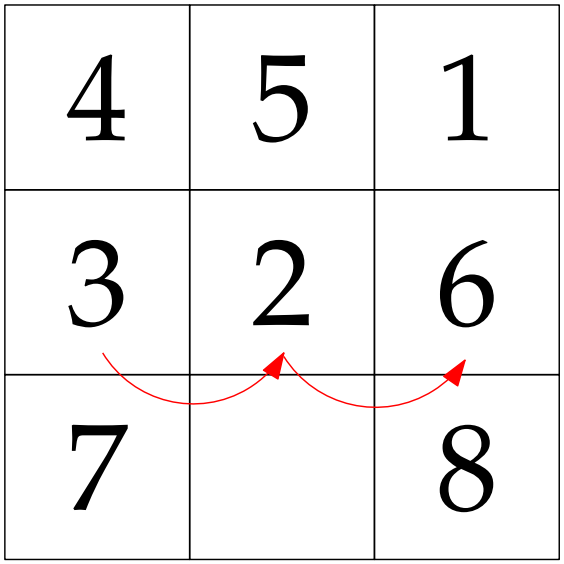
pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

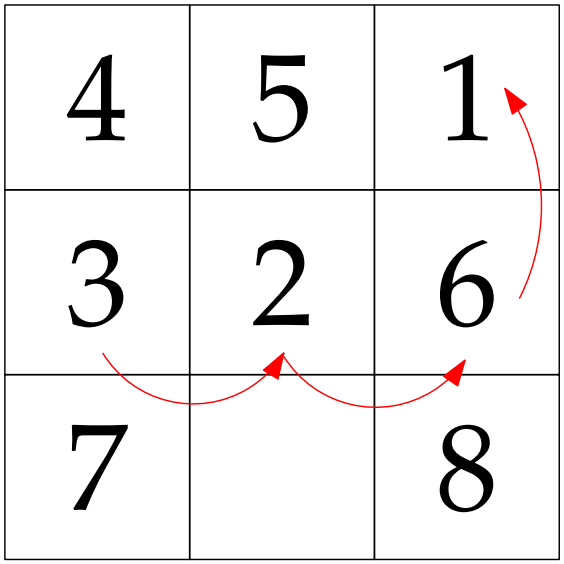
pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

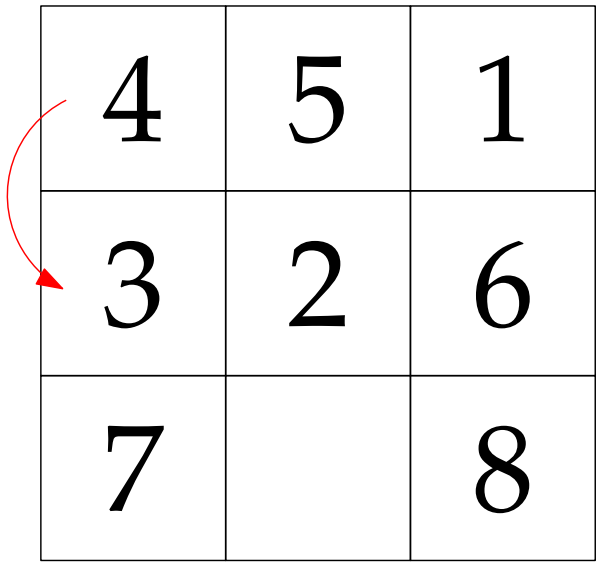
pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

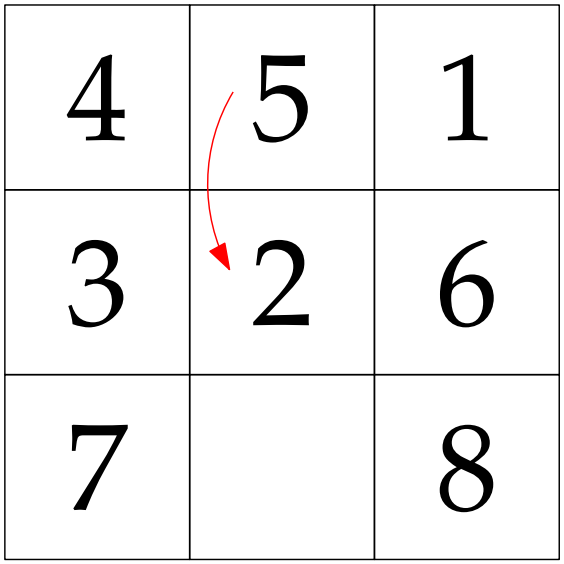
pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

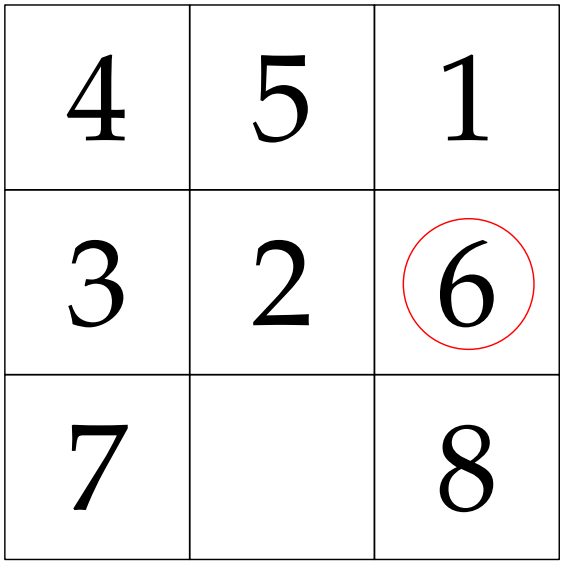
pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

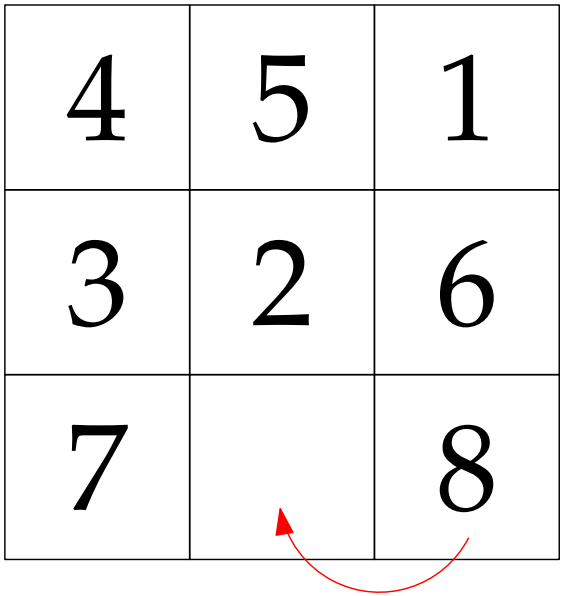
pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.

4	5	1
3	2	6
7		8

Heuristic h_2

- sum of number of steps to goal for each tile
- $h_2(s) = \sum_x ||\texttt{pos}(x) - \texttt{goal}(x)||_1$

pos is the position of the tile x and goal is where it should be. We interpret them as vector positions on a grid, so we can take the difference. $||.||_1$ is the *Manhattan metric*, i.e. the sum of vertical and horizontal steps to be taken for each tile to reach their goal.



Modification: $\texttt{MOVE}(x, y, z)$

Precondition: $\texttt{ON}(x, y), \texttt{CLEAR}(z), \texttt{ADJ}(y, z)$

Add List: $\texttt{ON}(x, z), \texttt{CLEAR}(y)$

Delete List: $\texttt{ON}(x, y), \texttt{CLEAR}(z)$

4	5	1
3	2	6
7		8

(Pearl 1984)

Modification: $\texttt{MOVE}'''(x,y,z)$

Precondition: $\texttt{ON}(x,y), \texttt{CLEAR}(z), \cancel{\texttt{ADJ}(y,z)}$

Add List: $\texttt{ON}(x,z), \texttt{CLEAR}(y)$

Delete List: $\texttt{ON}(x,y), \texttt{CLEAR}(z)$

4	5	1
3	2	6
7		8

Relaxation

- delete only $\texttt{ADJ}(y,z)$ from precondition
- i.e. a move requires a cell to be empty, but not adjacent
- successively jump tiles to empty position if target. If empty position itself is not a target (i.e. should remain empty), jump random tile

Modification: $\texttt{MOVE}'''(x,y,z)$

Precondition: $\texttt{ON}(x,y), \texttt{CLEAR}(z), \cancel{\texttt{ADJ}(y,z)}$

Add List: $\texttt{ON}(x,z), \texttt{CLEAR}(y)$

Delete List: $\texttt{ON}(x,y), \texttt{CLEAR}(z)$

4	5	1
3	2	6
7		8

Relaxation

- delete only $\texttt{ADJ}(y,z)$ from precondition
- i.e. a move requires a cell to be empty, but not adjacent
- successively jump tiles to empty position if target. If empty position itself is not a target (i.e. should remain empty), jump random tile

This is not an obvious strategy and was discovered much later than the others! (Gaschnig 1979)

Optimization of Existing Heuristics

- given admissible heuristics $h_1, h_2, h_3, \ldots, h_n$:
- is it possible to construct a heuristic at least as good as any of $h_1, h_2, h_3, \ldots, h_n$?

Heuristic Construction XXVII

Construction of Heuristics: Given admissible heuristics $h_1, h_2, h_3, \ldots, h_n$, is it possible to construct a heuristic that is as least as good as any of the above?

Consideration: first, what is a good heuristics?

Note: clearly a maximally admissible $h = 0$ is a bad heuristics, not helping at all

Ergo: a heuristic is most expressive/helpful/good the *larger* it is!

Heuristic Construction XXVIII

Construction of Heuristics: Given admissible heuristics $h_1, h_2, h_3, \ldots, h_n$, is it possible to construct a heuristic that is as least as good as any of the above?

Consideration: first, what is a good heuristics?

Note: clearly a maximally admissible $h = 0$ is a bad heuristics, not helping at all

Ergo: a heuristic is most expressive/helpful/good the *larger* it is!

Bottom Line: the heuristics

$$h_{\max} := \max(h_1, h_2, h_3, \ldots, h_n)$$

is at least as good as any of the h_i.

Heuristic Construction XXIX

Construction of Heuristics: Given admissible heuristics $h_1, h_2, h_3, \ldots, h_n$, is it possible to construct a heuristic that is as least as good as any of the above?

Consideration: first, what is a good heuristics?

Note: clearly a maximally admissible $h = 0$ is a bad heuristics, not helping at all

Ergo: a heuristic is most expressive/helpful/good the *larger* it is!

Bottom Line: the heuristics

$$h_{\max} := \max(h_1, h_2, h_3, \ldots, h_n)$$

is at least as good as any of the h_i.

Note: the best possible heuristic were the true value of the cost to a goal if it were known (which, in general, is difficult).