# CSE 473: Artificial Intelligence
Spring 2012

## Heuristics & Pattern Databases for Search

Dan Weld

With many slides from
Dan Klein, Richard Korf, Stuart Russell, Andrew Moore, & UW Faculty

## Recap: Search Problem

- States
  - configurations of the world
- Successor function:
  - function from states to lists of (state, action, cost) triples
- Start state
- Goal test

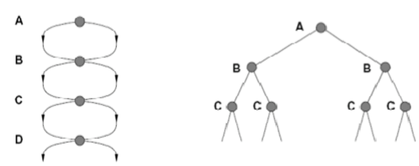
## General Tree Search Paradigm

```
function tree-search(root-node)
  fringe ← successors(root-node)
  while ( notempty(fringe) )
      {node ← remove-first(fringe)
         state ← state(node)
         if goal-test(state) return solution(node)
         fringe ← insert-all(successors(node),fringe) }
  return failure
end tree-search
```

3

## Extra Work?

- Failure to detect repeated states can cause exponentially more work (why?)




## General Graph Search Paradigm

```
function tree-search(root-node)
  fringe ← successors(root-node)
  explored ← empty
  while ( notempty(fringe) )
      {node ← remove-first(fringe)
         state ← state(node)
         if goal-test(state) return solution(node)
         explored ← insert(node,explored)
         fringe ← insert-all(successors(node),fringe, if node not in explored)
      }
  return failure
end tree-search
```

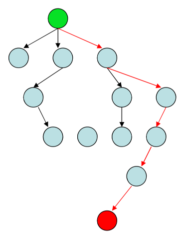
5

## Some Hints

- Graph search is almost always better than tree search (when not?)
- Implement your closed list as a dict or set!
- Nodes are conceptually paths, but better to represent with a state, cost, last action, and reference to the parent node

## Informed (Heuristic) Search

Idea: be **smart** about what paths to try.



7

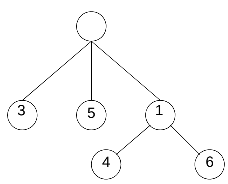
## Blind Search vs. Informed Search

- What's the difference?
- How do we formally specify this?
  A node is selected for expansion based on an evaluation function that estimates cost to goal.

8

## Best-First Search

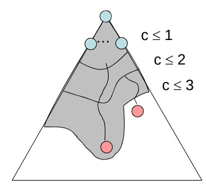
- Use an evaluation function f(n) for node n.
- Always choose the node from fringe that has the lowest f value.
  - Fringe = priority queue

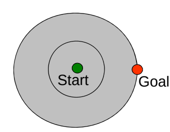



9

## Uniform Cost Search

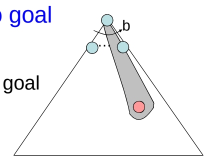
- f(n) = cost from root
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location



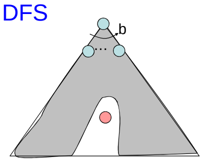




## Greedy Search

- f(n) = estimate of cost from n to goal
- A common case:
  - Takes you straight to the (wrong) goal




- Worst-case: like a badly-guided DFS




## A* search

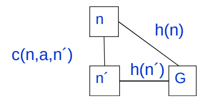
- *f(n)* = estimated total cost of path thru *n* to goal
- *f(n)* = *g(n)* + *h(n)*
  - *g(n)* = cost so far to reach *n*
  - *h(n)* = estimated cost from *n* to goal
    (satisfying some important conditions)

## Admissible heuristics

- A heuristic *h(n)* is admissible if for every node *n*, *h(n)* ≤ *h*(n)*, where *h*(n)* is the true cost to reach the goal state from *n*.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If *h(n)* is admissible, A* using `TREE-SEARCH` is optimal

## Consistent Heuristics

- h(n) is consistent if
  - for every node n
  - for every successor n´ due to legal action a
  - h(n) <= c(n,a,n´) + h(n´)

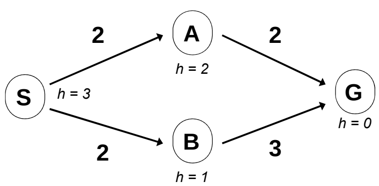



- Every consistent heuristic is also admissible.
- Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal

14

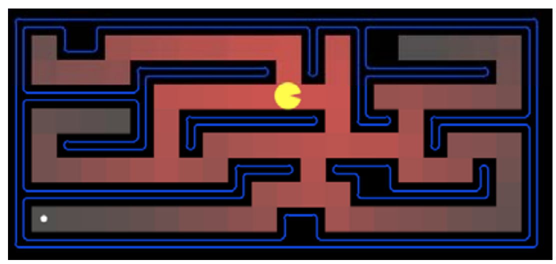
## When should A* terminate?

- Should we stop when we enqueue a goal?



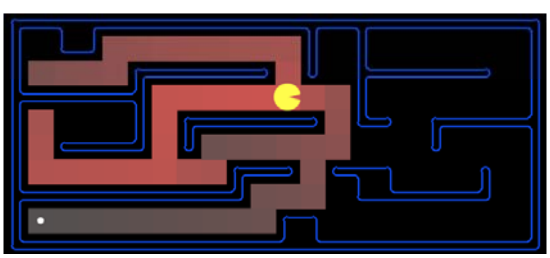

- No: only stop when we dequeue a goal

## Which Algorithm?

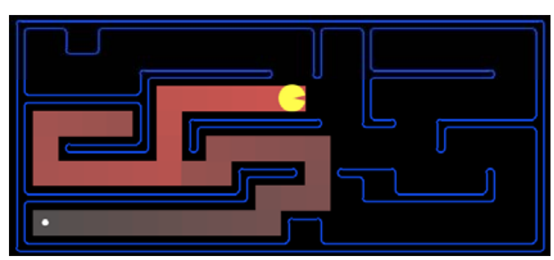


18

## Which Algorithm?

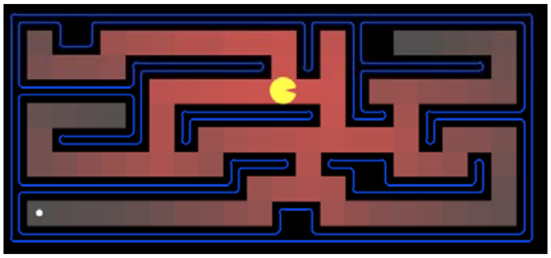


## Which Algorithm?



## Which Algorithm?

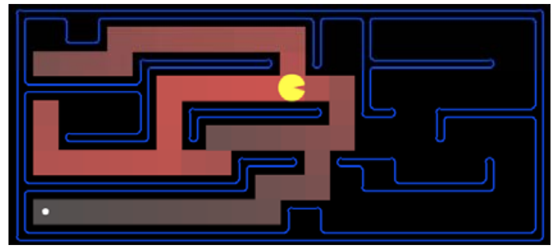
- Uniform cost search (UCS):



21

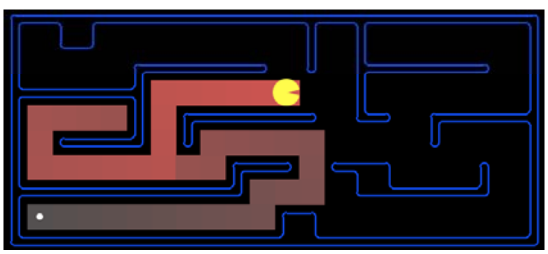
## Which Algorithm?

- A*, Manhattan Heuristic:



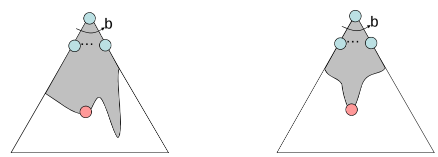
## Which Algorithm?

- Best First / Greedy, Manhattan Heuristic:



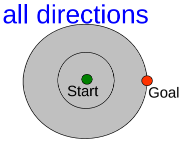
## Properties of A*

Uniform-Cost A*

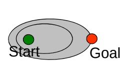



## UCS vs A* Contours

- Uniform-cost expanded in all directions




- A* expands mainly toward the goal, but does hedge its bets to ensure optimality




# Heuristics

It's what makes search actually work

## Admissable Heuristics

- f(x) = g(x) + h(x)
- g: cost so far
- h: underestimate of remaining costs

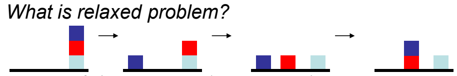
**Where do heuristics come from?**



37

## Relaxed Problems

- Derive admissible heuristic from exact cost of a solution to a relaxed version of problem
  - For transportation planning, relax requirement that car has to stay on road → Euclidean dist
  - For blocks world, distance = # move operations heuristic = number of misplaced blocks
  - *What is relaxed problem?*

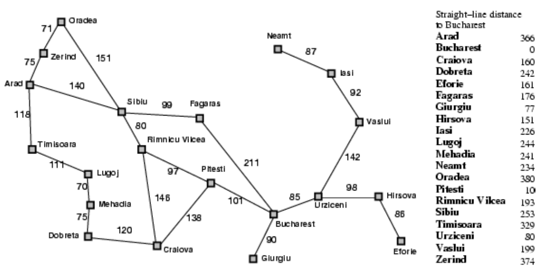


\# out of place = 2, true distance to goal = 3

- Cost of optimal soln to relaxed problem ≤ cost of optimal soln for real problem

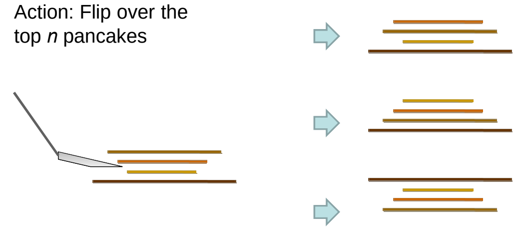
38

## What's being relaxed?




## Example: Pancake Problem

Action: Flip over the top *n* pancakes



Cost: Number of pancakes flipped

## Example: Pancake Problem

**BOUNDS FOR SORTING BY PREFIX REVERSAL**

William H. GATES
*Microsoft, Albuquerque, New Mexico*

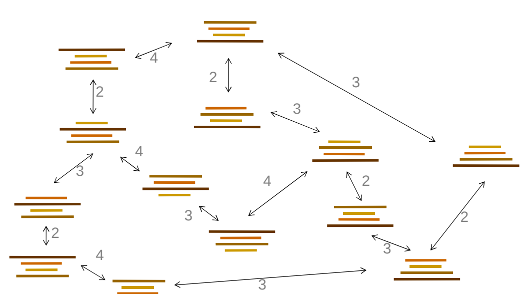
Christos H. PAPADIMITRIOU*†
*Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.*

Received 18 January 1978
Revised 28 August 1978

For a permutation $\sigma$ of the integers from 1 to $n$, let $f(\sigma)$ be the smallest number of prefix reversals that will transform $\sigma$ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all $\sigma$ in (the symmetric group) $S_n$. We show that $f(n) \leq (5n+5)/3$, and that $f(n) \geq 17n/16$ for $n$ a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n + 3$.

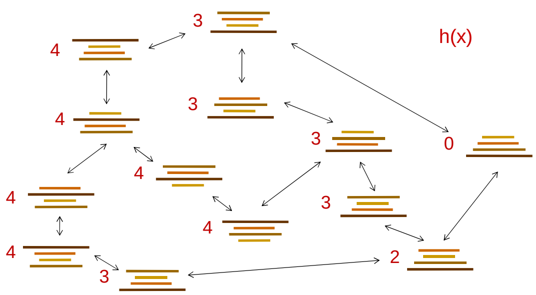
## Example: Pancake Problem

State space graph with costs as weights

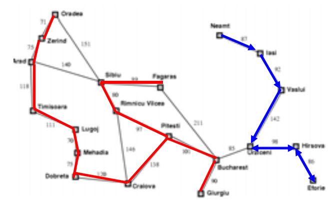


## Example: Heuristic Function

Heuristic: the largest pancake that is still out of place



## Traveling Salesman Problem



What can be Relaxed?

44

## Heuristics for eight puzzle

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

→

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

start goal

- What can we relax?

45

## Importance of Heuristics

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

h1 = number of tiles in wrong place

| D | IDS | A*(h1) |
|---|---|---|
| 2 | 10 | 6 |
| 4 | 112 | 13 |
| 6 | 680 | 20 |
| 8 | 6384 | 39 |
| 10 | 47127 | 93 |
| 12 | 364404 | 227 |
| 14 | 3473941 | 539 |
| 18 | | 3056 |
| 24 | | 39135 |

46

## Importance of Heuristics

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

h1 = number of tiles in wrong place

h2 = $\Sigma$ distances of tiles from correct loc

| D | IDS | A*(h1) | A*(h2) |
|---|---|---|---|
| 2 | 10 | 6 | 6 |
| 4 | 112 | 13 | 12 |
| 6 | 680 | 20 | 18 |
| 8 | 6384 | 39 | 25 |
| 10 | 47127 | 93 | 39 |
| 12 | 364404 | 227 | 73 |
| 14 | 3473941 | 539 | 113 |
| 18 | | 3056 | 363 |
| 24 | | 39135 | 1641 |

Decrease effective branching factor

47

## Combining Admissible Heuristics

- Can always take max
- Adding does not preserve admissibility in general

48

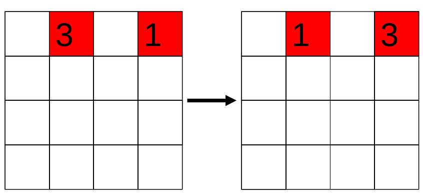
## Performance of IDA* on 15 Puzzle

- 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.

## Limitation of Manhattan Distance

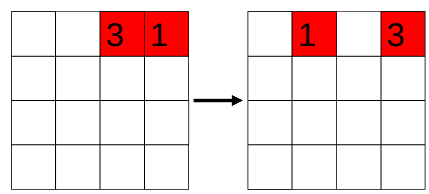
- To solve a 24-Puzzle instance, IDA* with Manhattan distance would take about 65,000 years on average.
- Assumes that each tile moves independently
- In fact, tiles interfere with each other.
- Accounting for these interactions is the key to more accurate heuristic functions.

## Example: Linear Conflict



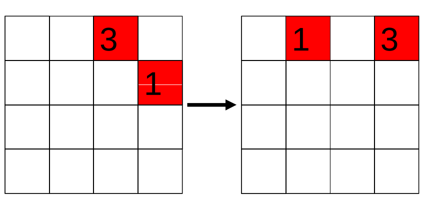

Manhattan distance is 2+2=4 moves

## Example: Linear Conflict



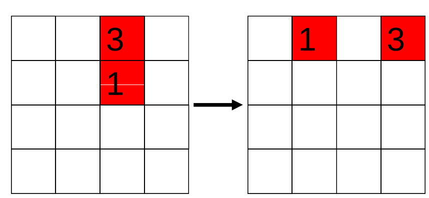

Manhattan distance is 2+2=4 moves

## Example: Linear Conflict



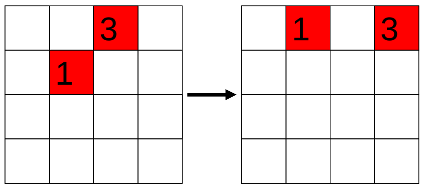

Manhattan distance is 2+2=4 moves

## Example: Linear Conflict



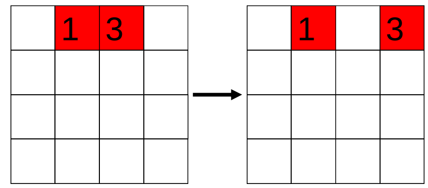

Manhattan distance is 2+2=4 moves

## Example: Linear Conflict



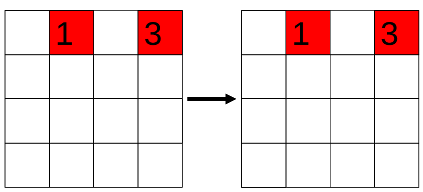

Manhattan distance is 2+2=4 moves

## Example: Linear Conflict




Manhattan distance is 2+2=4 moves

## Example: Linear Conflict




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

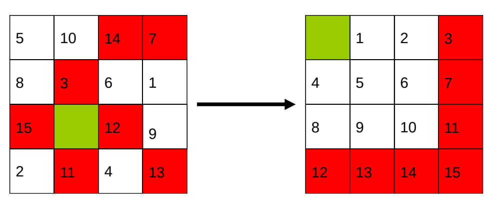
## Linear Conflict Heuristic

- 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.

## Pattern Database Heuristics

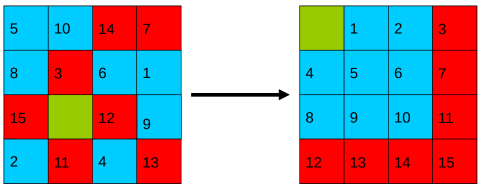
- Culberson and Schaeffer, 1996
- A pattern database is a complete set of such positions, with associated number of moves.
- e.g. a 7-tile pattern database for the Fifteen Puzzle contains 519 million entries.

## Heuristics from Pattern Databases




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

## Combining Multiple Databases




31 moves needed to solve red tiles

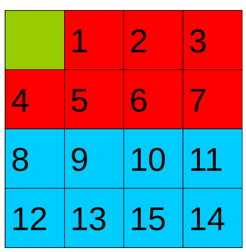
22 moves need to solve blue tiles

Overall heuristic is maximum of 31 moves

## Additive Pattern Databases

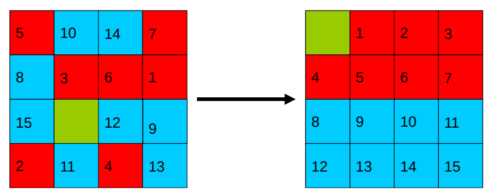
- Culberson and Schaeffer counted all moves needed to correctly position the pattern tiles.
- In contrast, we count only moves of the pattern tiles, ignoring non-pattern moves.
- If no tile belongs to more than one pattern, then we can add their heuristic values.
- Manhattan distance is a special case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

## 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

## Drawbacks of Standard Pattern DBs

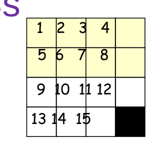
- Since we can only take *max*
  - Diminishing returns on additional DBs
- Would like to be able to *add* values

© Daniel S. Weld

Adapted from Richard Korf presentation



## Disjoint Pattern DBs




- Partition tiles into disjoint sets
  - For each set, precompute table
    - E.g. 8 tile DB has 519 million entries
    - And 7 tile DB has 58 million
- During search
  - Look up heuristic values for each set
  - *Can add values without overestimating!*
  - Manhattan distance is a special case of this idea where each set is a single tile

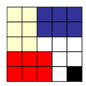
© Daniel S. Weld

Adapted from Richard Korf presentation

# Performance

- 15 Puzzle:  2000x speedup *vs* Manhattan dist
  - IDA* with the two DBs shown previously solves 15 Puzzles optimally in 30 milliseconds

- 24 Puzzle: 12 million x speedup *vs* Manhattan
  - IDA* can solve random instances in 2 days.
  - Requires 4 DBs as shown
    - Each DB has 128 million entries
  - Without PDBs: 65,000 years





Adapted from Richard Korf presentation

68