CSE 573: Artificial Intelligence
Autumn2012

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 Graph Search Paradigm

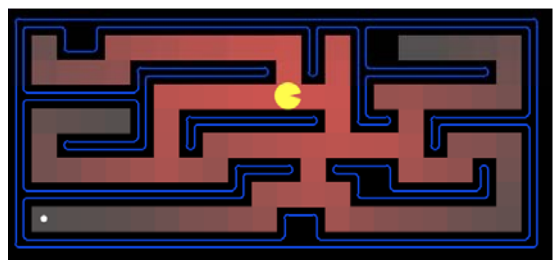
```
function tree-search(root-node)
  fringe ← successors(root-node)
  explored ← empty
  while ( notempty(fringe) )
      {node ← remove-first(fringe)
        state ← state(node)
        if goal-test(state) return solution(node)
        explored ←explored ∪ {node}
        fringe ← fringe ∪ (successors(node)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

Fringe = priority queue, ranked by heuristic
Often: $f(x) = g(x) + h(x)$

3

Which Algorithm?

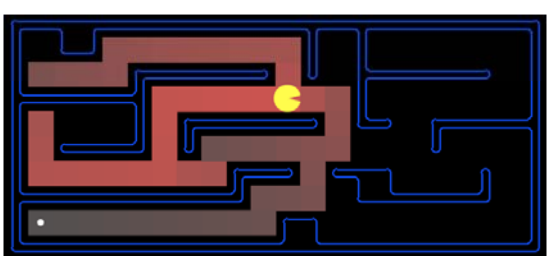
Uniform cost search



4

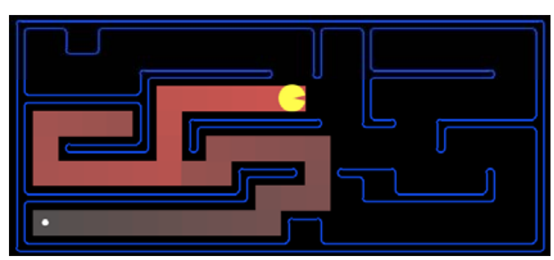
Which Algorithm?

A* using Manhattan



Which Algorithm?

Best-first search using Manhattan



Heuristics

It's what makes search actually work

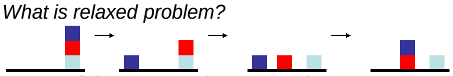
Admissable Heuristics

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

Where do heuristics come from?

Relaxed Problems

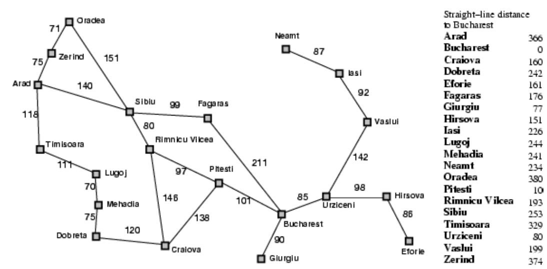
- Derive admissible heuristic from exact cost of a solution to a relaxed version of problem
  - For transportation planning, relax requirement that car has to stay on road → Euclidean dist
  - For blocks world, distance = # move operations heuristic = number of misplaced blocks
  - *What is relaxed problem?*



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

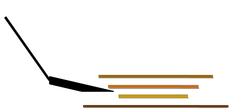
- Cost of optimal soln to relaxed problem ≤ cost of optimal soln for real problem

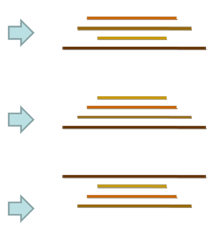
What's being relaxed?



Example: Pancake Problem

Action: Flip over the top *n* pancakes





Cost: Number of pancakes flipped
Goal: Pancakes in size order

Example: Pancake Problem

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES
Microsoft, Albuquerque, New Mexico

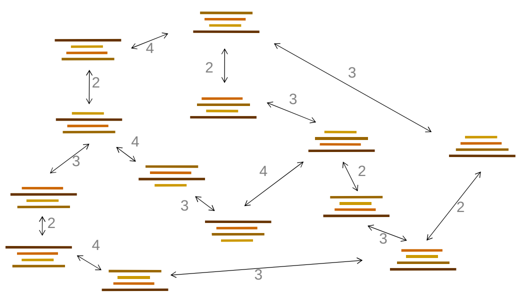
Christos H. PAPADIMITRIOU*†
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978
Revised 28 August 1978

For a permutation σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ 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

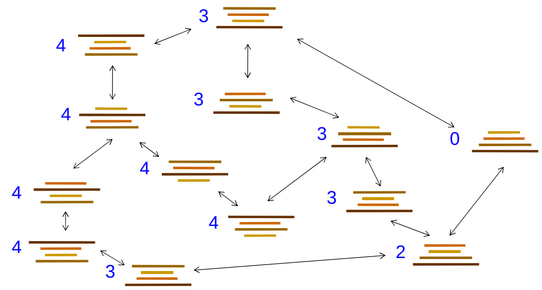
State space graph with costs as weights



Example: Heuristic Function

Heuristic: h(x) = the largest pancake that is still out of place

What is being relaxed?



Counterfeit Coin Problem

- Twelve coins
- One is counterfeit: maybe heavier, maybe light
- Objective:
  - Which is phony & is it heavier or lighter?
  - Max three weighings

15

Coins

- State = coin possibilities
- Action = weighing two subsets of coins
- Heuristic?
  - What is being relaxed?

16

Traveling Salesman Problem

What can be relaxed?

Path =

1) Graph
2) Degree 2 (except ends, degree 1)
3) Connected

Kruskal's Algo:
(Greedily add cheapest useful edges)

17

Traveling Salesman Problem

What can be relaxed?

Relax degree constraint
Assume can teleport to past nodes on path
→
Minimum spanning tree

Kruskal's Algorithm:
O(n^2)
(Greedily add cheapest useful edges)

18

Traveling Salesman Problem

What can be relaxed?

Relax connected constraint
→
Cheapest degree 2 graph

Optimal assignment
O(n^3)

Automated Generation of Relaxed Problems

- Need to reason about search problems
- Represent search problems in formal language

Planning

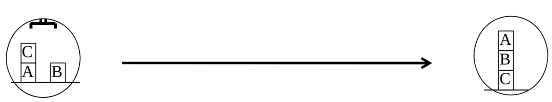
I have a plan - a plan that cannot possibly fail.

- Inspector Clousseau

Classical Planning

- Given
  - a logical description of the **initial situation**,
  - a logical description of the **goal conditions**, and
  - a logical description of a set of **possible actions**,
- Find
  - a **sequence of actions** (a **plan of actions**) that brings us from the initial situation to a situation in which the goal conditions hold.

Example: BlocksWorld



Planning Input: State Variables/Propositions

- **Types: block --- a, b, c**
- **(on-table a) ~~(on-table b) (on-table c)~~** on-table(a)
- **(clear a) (clear b) (clear c)**
- **(arm-empty)**
- **(holding a) (holding b) (holding c)**
- **(on a b) (on a c) (on b a) (on b c) (on c a) (on c b)**

No. of state variables =16
No. of states = 2^{16}
No. of reachable states = ?

Planning Input: Actions

- **pickup a b, pickup a c, …**
- **place a b, place a c, …**
- **pickup-table a, pickup-table b, …**
- **place-table a, place-table b, …**

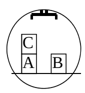
Total: 6 + 6 + 3 + 3 = 18 "ground" actions
Total: 4 action schemata

Planning Input: Actions (contd)

- **:action pickup ?b1 ?b2**
  :precondition
  (on ?b1 ?b2)
  (clear ?b1)
  (arm-empty)
  :effect
  (holding ?b1)
  (not (on ?b1 ?b2))
  (clear ?b2)
  (not (arm-empty))

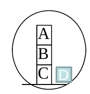
- :action pickup-table ?b
  :precondition
  (on-table ?b)
  (clear ?b)
  (arm-empty)
  :effect
  (holding ?b)
  (not (on-table ?b))
  (not (arm-empty))

Planning Input: Initial State



- **(on-table a) (on-table b)**
- **(arm-empty)**
- **(clear c) (clear b)**
- **(on c a)**
- **All other propositions false**
  - **not mentioned → assumed false**
  - **"Closed world assumption"**

Planning Input: Goal



- **(on-table c) AND (on b c) AND (on a b)**
- **Is this a state?**
- **In planning a goal is a *set of states***
  - **Like the goal test in problem solving search**
  - **But specified declaratively (in logic) rather than with code**

Specifying a Planning Problem

- Description of initial state of world
  - Set of propositions
- Description of goal:
  - E.g., Logical conjunction
  - Any world satisfying conjunction is a goal
- Description of available actions

Forward State-Space Search

- Initial state: set of positive ground literals
  - CWA: literals not appearing are false
- Actions:
  - applicable if preconditions satisfied
  - add positive effect literals
  - remove negative effect literals
- Goal test: does state logically satisfy goal?
- Step cost: typically 1

Heuristics for State-Space Search

- Count number of false goal propositions in current state
  - Admissible?
  - NO
- Subgoal independence assumption:
  - Cost of solving conjunction is sum of cost of solving each subgoal independently
  - Optimistic: ignores negative interactions
  - Pessimistic: ignores redundancy
  - Admissible? No
  - Can you make this admissible?

 31

Heuristics for State Space Search (contd)

- Delete all preconditions from actions, solve easy relaxed problem, use length
  - Admissible?
  - YES

- :action pickup-table ?b
  :precondition (a~~nd (on-table ?b)~~
  ~~(clear ?b)~~
  ~~(arm-empty))~~
  :effect (and (holding ?b)
  (not (on-table ?b))
  (not (arm-empty)))

32

CSE 573

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?

33

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

34

Importance of Heuristics

7	2	3
4	1	6
8	5	

h1 = number of tiles in wrong place

h2 = Σ 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

35

Combining Admissible Heuristics

- Can always take max
- Could add several heuristic values
  - Doesn't preserve admissibility in general

36

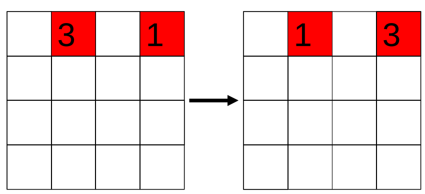
Performance of IDA* on 15 Puzzle

- 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

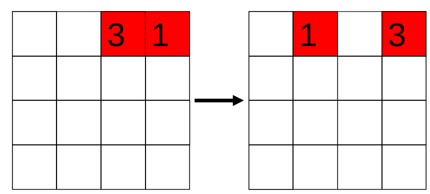
- Solving a 24-Puzzle instance,
  - IDA* with Manhattan distance …
  - 65,000 years on average.
- Assumes that each tile moves independently
- In fact, tiles interfere with each other.
- Accounting for these interactions is the key to more accurate heuristic functions.

Example: Linear Conflict



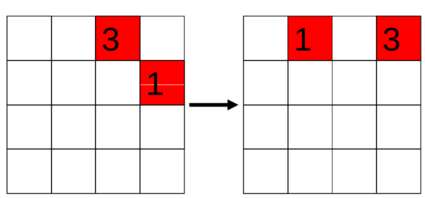
Manhattan distance is 2+2=4 moves

Example: Linear Conflict



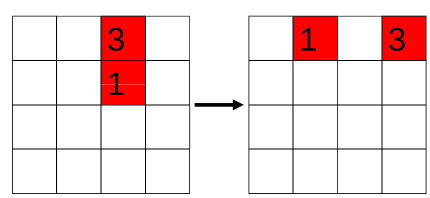
Manhattan distance is 2+2=4 moves

Example: Linear Conflict



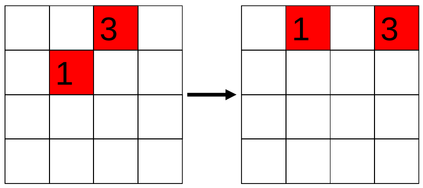
Manhattan distance is 2+2=4 moves

Example: Linear Conflict



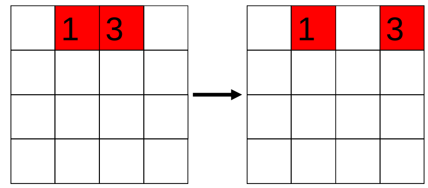
Manhattan distance is 2+2=4 moves

Example: Linear Conflict



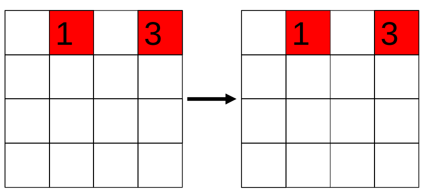
Manhattan distance is 2+2=4 moves

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.

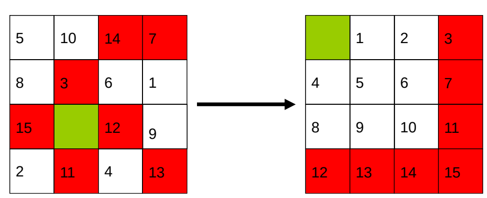
Linear Conflict Heuristic

- 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

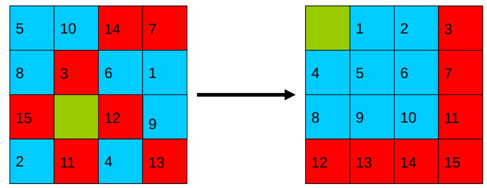


31 moves is a lower bound on the total number of moves needed to solve this particular state.

Precomputing Pattern Databases

- Entire database is computed with one backward breadth-first search from goal.
- All non-pattern tiles are indistinguishable,
  - But all tile moves are counted.
- The first time each state is encountered, the total number of moves made so far is stored.
- Once computed, the same table is used for all problems with the same goal 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

Drawbacks of Standard Pattern DBs

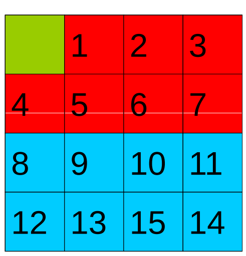
- Since we can only take *max*
  - Diminishing returns on additional DBs
- Would like to be able to *add* values

 Adapted from Richard Korf presentation 51

Additive Pattern Databases

- Culberson and Schaeffer counted all moves needed to correctly position the pattern tiles.
  - In contrast, we could 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.

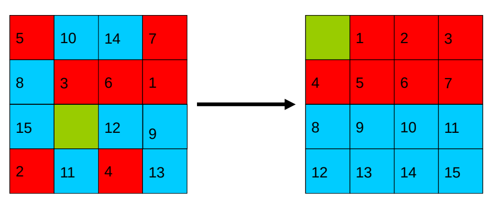
Example Additive Databases



The 7-tile database contains 58 million entries.

The 8-tile database contains 519 million entries.

Computing the Heuristic



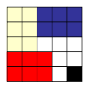
20 moves needed to solve red tiles

25 moves needed to solve blue tiles

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

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

55