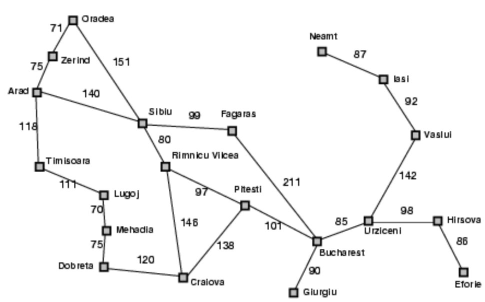
CSE 473

Chapter 4

Heuristics & Local Search

Recall: Admissable Heuristics

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



e.g., h_{SLD}

Where do heuristics come from?

Relaxed Problems

- Derive admissible heuristic from **exact** cost of a solution to a **relaxed** version of problem

For route planning, what is a relaxed problem?

Relax requirement that car has to stay on road
Straight Line Distance becomes optimal cost

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

Heuristics for eight puzzle

7	2	3
5	1	6
8	4	

start

1	2	3
4	5	6
7	8	

goal

- What can we relax?

Heuristics for eight puzzle

7	2	3
5	1	6
8	4	

1	2	3
4	5	6
7	8	

Original: Tile can move from location A to B if A is horizontally or vertically next to B *and* B is blank

Relaxed 1: Tile can move from any A to any B

Cost = h_1 = number of misplaced tiles

Relaxed 2: Tile can move from A to B if A is horizontally or vertically next to B

Cost = h_2 = total Manhattan distance

5

Importance of Heuristics

7	2	3
4	1	6
8	5	

Avg number of nodes generated

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

Recall from last time: h_2 dominates h_1

6

Need for Better Heuristics

Performance of h_2 (Manhattan Distance Heuristic)

8 Puzzle	< 1 second
15 Puzzle	1 minute
24 Puzzle	65000 years

Can we do better?

Adapted from Richard Korf presentation

Creating New Heuristics

- Given admissible heuristics $h_1, h_2, \ldots, h_m$, none of them dominating any other, how to choose the best?
- Answer: No need to choose only one! Use:
  $h(n) = \max \{h_1(n), h_2(n), \ldots, h_n(n)\}$
- h is admissible (why?)
- h dominates all h_i (by construction)
- Can we do better with:
  $h'(n) = h_1(n) + h_2(n) + \ldots + h_n(n)$?

Pattern Databases [Culberson & Schaeffer 1996]

- Idea: Use solution cost of a subproblem as heuristic. For 8-puzzle: pick any subset of tiles
  - E.g., 3, 7, 11, 12
- Precompute a table

  Compute optimal cost of solving just these tiles
  - This is a lower bound on actual cost with all tiles

  For all possible configurations of these tiles
  - Could be several million

  Use breadth first search back from goal state
  - State = position of just these tiles (& blank)

  Admissible heuristic h_{DB} for complete state = cost of corresponding sub-problem state in database

Adapted from Richard Korf presentation 9

Combining Multiple Databases

- Can choose another set of tiles

  Precompute multiple tables
- How to combine table values?

  Use the *max* trick!

- E.g. Optimal solutions to Rubik's cube

  First found w/ IDA* using pattern DB heuristics

  Multiple DBs were used (diff subsets of cubies)

  Most problems solved optimally in 1 day

  Compare with ***574,000 years*** for IDDFS

Adapted from Richard Korf presentation 10

Drawbacks of Standard Pattern DBs

- Since we can only take ***max***
  Diminishing returns on additional DBs

- Would like to be able to ***add*** values
  - But not exceed the actual solution cost (admissible)
  - How?

Adapted from Richard Korf presentation 11

Disjoint Pattern DBs

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- Partition tiles into disjoint sets
  For each set, precompute table
  Don't count moves of tiles not in set
  - This makes sure costs are disjoint
  - Can be added without overestimating!
  - E.g. 8 tile DB has 519 million entries
  - And 7 tile DB has 58 million
- During search
  Look up costs for each set in DB
  Add values to get heuristic function value

  Manhattan distance is a special case of this idea where each set is a single tile

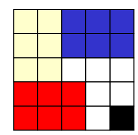
Adapted from Richard Korf presentation 12

Performance

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

- 24 Puzzle: 12 millionx speedup *vs* Manhattan
  IDA* can solve random instances in 2 days.
  Requires 4 DBs as shown
  - Each DB has 128 million entries

  Without PDBs: 65000 years



Adapted from Richard Korf presentation 13

14

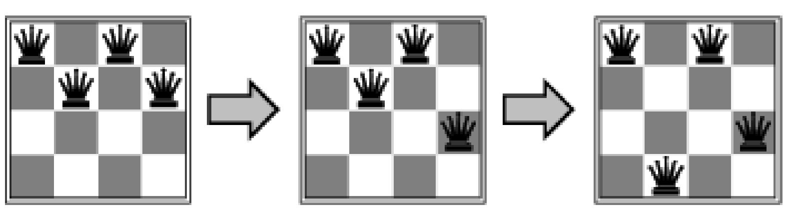
Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
- Keep a single "current" state, try to improve it

15

Example: *n*-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



16

Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

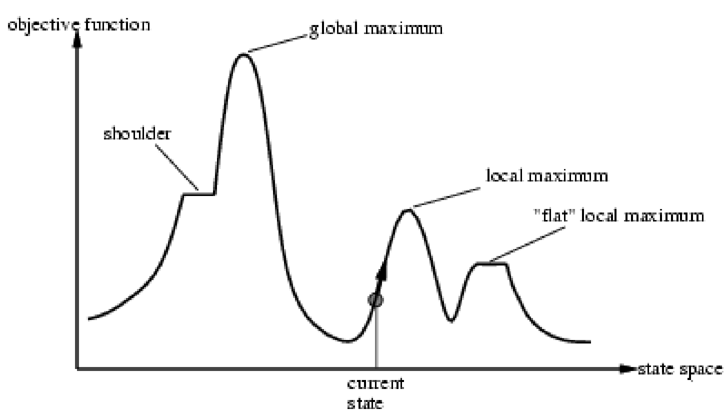
```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
       neighbor ← a highest-valued successor of current
       if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
       current ← neighbor
```

17

Hill-climbing search

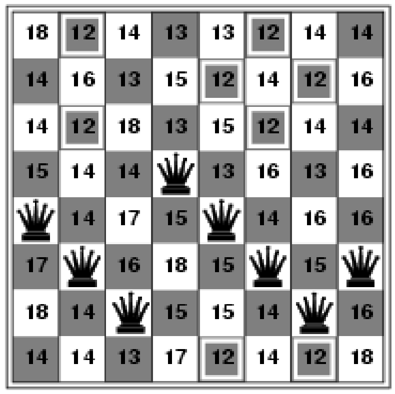
- Problem: depending on initial state, can get stuck in local maxima



18

Example: 8-queens problem

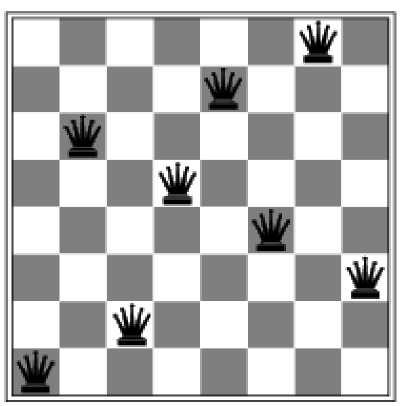
Heuristic?



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state

19

Example: 8-queens problem



- A local minimum with *h = 1*. Need *h = 0*
- *How to find global minimum/maximum?*

20

Simulated Annealing

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

21

Properties of simulated annealing

- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc

22

Local Beam Search

- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

23

Next Time

- Gaming search and searching for Games
- Homework #1 due

24