Local Search

2/2/18
Consider the following problems

**N-Queens**: place $n$ queens on an $n \times n$ chessboard so that none share a row, column, or diagonal.

**Min-Coloring**: assign as few colors as possible to graph nodes (map regions) so that no adjacent nodes share a color.
N-Queens as state space search

Search space size: $N^N$

$6^6 \approx 47,000$

$8^8 \approx 17,000,000$

$25^{25} \approx 8.9 \times 10^{34}$
Min-Coloring as state space search

Search space size: $|G|^K$

$48^3 = 110,592$

$10,000^8 = 10^{32}$
A different approach

**State Space Search**
- Search for paths to goals.
- Given a start state, goal state, operators.
- Apply operators to states to generate new states.

Examples: BFS, DFS, UCS, A*, iterative deepening

**Local Search**
- Search for solutions.
- Given candidate solutions.
- Make small adjustments to candidates to generate new, potentially better candidates.

Examples: hill climbing, simulated annealing, beam search, genetic algorithms
Requirements for local search

• We can generate and evaluate candidate solutions.

• We can generate neighbor candidates by small modifications.

• Similar candidates have similar values.
N-Queens Local Search Representation

Candidate solution: an assignment of 1 queen per row. can represent as a tuple: (0,1,2,5,6,7,2,3)

Neighbors: boards differing by the placement of 1 ♕. each state has N*(N-1) neighbors

Evaluation: minimize number of attacked queens.

\[ v(0,1,2,5,6,7,2,3) = 8 \]
\[ v(0,4,2,5,6,7,2,3) = 7 \]
Hill Climbing

Key idea:
- Start with an arbitrary candidate.
- Iteratively move to a neighbor with higher value.
  - Climb up the value hill.

Basic pseudocode:

```plaintext
neighbor ← random_candidate()
do{
    state ← neighbor
    neighbor ← best_neighbor(state)
}while value(state) < value(neighbor)
```

\[ \text{cost} = -\text{value} \]
Trace of successful run

state cost: 8 neighbor cost: 6
state cost: 6 neighbor cost: 5
state cost: 5 neighbor cost: 4
state cost: 4 neighbor cost: 3
state cost: 3 neighbor cost: 2
state cost: 2 neighbor cost: 0
state cost: 0 neighbor cost: 1

Goal found in 7 steps
Trace of unsuccessful run

state cost: 6 neighbor cost: 5
state cost: 5 neighbor cost: 4
state cost: 4 neighbor cost: 3
state cost: 3 neighbor cost: 2
state cost: 2 neighbor cost: 2

-------------
| |Q| | | | | | |
| | | |Q| | | | |
| | | | |Q| | | |
| | | | | |Q| | |
| | | | | | |Q| |
| | | | | | | |Q|
|Q| | | | | | | |
-------------
Goal NOT found in 5 steps
Getting stuck in local optima

- Random steps would help
- Random restarts would help
Branching factor
(search dimension)

• If states have lots of neighbors, searching through all of them to find the best can be slow.

• Instead, we can pick a random neighbor, and keep it if it’s better. Otherwise we try again.
Improved hill climbing pseudocode

for $i=1$:NUM_RESTARTS:
    state $\leftarrow$ random_candidate()
for $i$ in 1:MAX_ITERs:
    neighbor $\leftarrow$ random_neighbor(state)
    if random_move_decision():
        state $\leftarrow$ neighbor
    else:
        if cost(neighbor) < cost(state):
            state $\leftarrow$ neighbor

return best state found
Trace of improved pseudocode

| | | | | Q | | |
| | | Q | | | | |
| | | | Q | | | | |
| Q | | | | | | |
| | Q | | | | | |
| | Q | | | | | |

Hill climbing restarted (1)...

Goal found in 18 steps
This version can solve 25-queens
Exercise: map coloring

• How would you represent states?
• How big is the state space?
• What is the objective function?
• What is the successor function?
• How many successors would there be?
Simulated Annealing

• Always select moves randomly.
• Accept if improving.
• Accept bad moves with some probability.

Key idea: gradually reduce the probability of accepting bad moves.

parameters:
• initial temperature
• decay rate

\[
\text{prob} = e^{\frac{\Delta}{T}}
\]
\[
\Delta = \text{cost(state)} - \text{cost(neighbor)}
\]
\[
T = \text{initial\_temp} \times (\text{decay\_rate})^{\text{rounds}}
\]
Simulated annealing pseudocode

state ← random_candidate()

temp ← INIT_TEMP

for round in 1:MAX_ITERS:
    neighbor ← random_neighbor(state)
    if accept(state, neighbor, temp):
        state ← neighbor
        temp ← temp * DECAY

return best state found

function accept(state, neighbor, temp):
    delta ← cost(state) - cost(neighbor)
    r ~ U[0,1]
    return r < e^(delta / temp)
Effect of temperature