Local Search

Chapter 4

Local Search: Outline

We consider next *local* search, where we maintain a single current state.

- Iterative improvement algorithms
- Hill-climbing
- Very briefly:
  - Simulated annealing
  - Local beam search

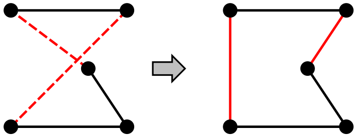
Iterative improvement algorithms

- Idea: In many optimization problems, the *path* to the goal is irrelevant.
  - The goal state itself is the solution
  - E.g. the n-queens problem
- So we may formulate a problem so that:

  state space = set of "complete" configurations
- Examples:
  - find *optimal* configuration, e.g., TSP
  - find configuration satisfying constraints, e.g., timetable
  - also, e.g. propositional satisfiability (SAT)
- In such cases, we can use *iterative improvement* algorithms
  - Keep a single "current" state; try to improve it
  - Uses constant space; suitable for online as well as offline search

Example: Travelling Salesperson Problem

- Start with any complete tour, perform pairwise exchanges



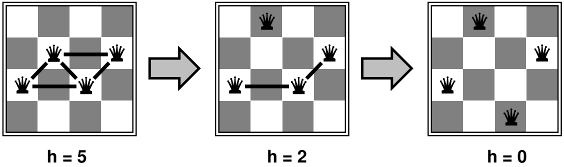
- Variants of this approach get within 1% of optimal very quickly with thousands of cities.

Example: n-queens

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

Example: n-queens

- Goal: Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.
- Move a queen to reduce number of conflicts.



- Almost always solves n-queens problems almost instantaneously for very large n, e.g., $n = 1,000,000$

Hill-climbing (or gradient ascent/descent)

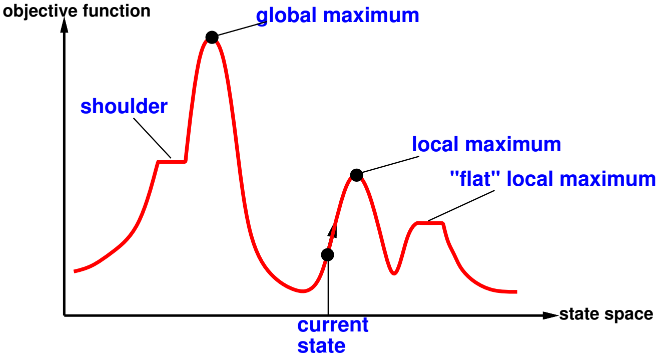
- Idea: Take the best move from a given position
- Aka *greedy local search*.
- "Like climbing a mountain in thick fog with amnesia"

Hill-climbing

```
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
  end
```

Hill-climbing contd.

Useful to consider *state-space landscape*



Hill-climbing contd.

- Hill climbing often gets stuck:

  Local Maxima: I.e. local "peaks".
  E.g. 8-queens gets stuck 86% of the time.

  Ridges: Essentially give a series of local maxima.
  Difficult for hill-climbing to navigate

  Plateaux: A plateau is a flat area in the search space.
  Search degenerates to exhaustive search, or may loop.

Hill-climbing: Strategies if stuck

- *Random-restart hill climbing*: Overcomes local maxima
  - Trivially complete *if* a goal is known to exist.
- *Random sideways moves*: Escape from shoulders but may loop on flat maxima
  - Can also define a hill-climbing version of depth-first search. (But then no longer a *local* search.)

Another Example: Propositional Satisfiabilty

- Goal: Find a *satisfying assignment* for a set of clauses in CNF.
- E.g.

$$(p \vee q \vee \neg r) \wedge (\neg p \vee r) \wedge (\neg p \vee \neg q)$$

is satisfied by setting: $p = \mathit{true}$, $q = \mathit{false}$, $r = \mathit{true}$.

Propositional Satisfiabilty

- Outline of an algorithm:

```
Function Sat(problem) returns a solution or failure
   Assign truth values arbitrarily to the set of propositional variables
   loop do {
      if the truth assignment satisfies problem
                  then return the assignment
      if timeout then return failure
      Find l such that l̄ gives the largest increase in clauses satisfied
      Change the truth value of l to l̄.
   }
```

☞ If l is p then $\bar{l}$ is $\neg p$;
if l is $\neg p$ then $\bar{l}$ is p.

Propositional Satisfiabilty

- This algorithm, when proposed in the 1990's, worked *very* well.
- The algorithm also featured random restarts. (I.e. after a while reassign all variable and start over).
  - It handily beat all previous algorithms (notably DPLL).
- Subsequent work in satisfiability has led to huge improvements over the naive greedy algorithm.

- Aside: Another thing that this work pointed out was the importance of choice of test instances.
  - DPLL (and other algorithms) appeared to work well because it turned out they were often tested on easy instances.

Simulated annealing

- Goal: Avoid local maxima
  - Local maxima is the biggest problem with local search.
- Idea: Take a step in a direction other than the best, from time to time.
  - Try to escape local maxima by allowing some “bad” moves *but gradually decrease their size and frequency*
  - These steps are designed to get the solver out of a possible local maximum
- The step size varies.
  - As time passes the step size and probabilty of a non-best step decreases.
- Simulated annealing has proven very effective in a wide range of problems, including VLSI layout, airline scheduling, etc.

Local beam search

Idea:

- Begin with k randomly-generated states.
- Keep k states instead of 1; choose top k of all their successors
- Not the same as k searches run in parallel!
- Searches that find good states recruit other searches to join them

Problem:

Quite often, all k states end up on same local hill

Variant: Stochastic beam search:

Choose k successors randomly, biased towards good ones

- Observe the analogy to natural selection!