Artificial Intelligence

5. Local Search

Prof Sara Bernardini
bernardini@diag.uniroma1.it
www.sara-bernardini.com

SAPIENZA
UNIVERSITÀ DI ROMA

Autumn Term

Agenda

Our Agenda for This Chapter

- **Local Search:** Overview of methods forsaking completeness/optimality, taking decisions based only on the local surroundings.

  $\rightarrow$ How to exploit the knowledge in a greedy way?

- **Hill Climbing:** Can we find a solution by looking only at the h values of the successors of the current state?

  $\rightarrow$ Is a purely local greedy search a good idea?

- **Simulated Annealing:** Can we escape local minima by injecting some randomness into the hill-climbing algorithm? Can we combine hill climbing with random walks?

  $\rightarrow$ Sometime, you can take inspiration from metallurgy for AI algorithms!

- **Beam Search / Genetic Algorithm:** Keeping in memory only one node seems rather extreme. Can we keep track of k states instead of one? Even better, can we combine two states to generate a successor state that is better than both of them?

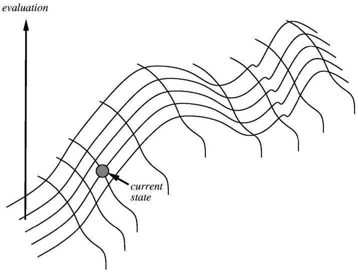
  $\rightarrow$ From metallurgy to genetics, sources of inspiration are endless!

Local Search

Do *you* "think through all possible options" before choosing your path to the cafeteria?

→ Sometimes, "going where your nose leads you" works quite well.

What is the computer's "nose"?



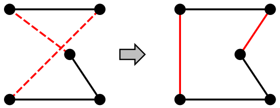
→ Local search takes decisions based on the h values of immediate neighbor states.

Local Search

In many optimization problems, path is irrelevant; the goal state itself is the solution.
→ E.g: Find a configuration satisfying some constraints, e.g., timetable.

In these cases, the state space is set of "complete" configurations and the task is to find the optimal configuration.

In such cases, we can use iterative improvement algorithms; keep a single "current" state and try to improve it.

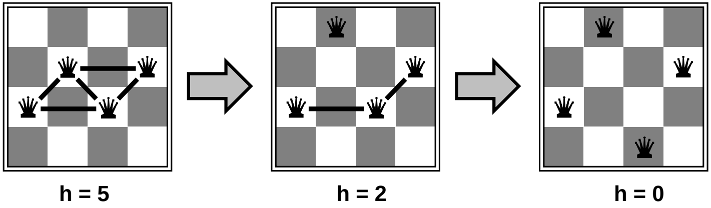


TSP: Start with any complete tour, perform pairwise exchanges.
→ Variants of this approach get within 1% of optimal very quickly with thousands of cities.

Example: n-Queens

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

Strategy: Move a queen at a time to reduce number of conflicts.



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

Local Search: Advantages and Disadvantages

Local search algorithms are not systematic.

Two key advantages:

- Use very little memory - typically constant space.
- Find reasonable solutions in large or infinite state spaces where systematic search is unsuitable.

Useful for solving pure optimization problems: find best state according to an objective function.

Hill-Climbing Algorithm

```
function Hill-Climbing(problem)
  $n \leftarrow$ a node $n$ with $n$.state=problem.InitialState
  loop do
      $n' \leftarrow$ among child nodes $n'$ of $n$ with minimal $h(n')$,
              randomly pick one
      if $h(n') \geq h(n)$ then return the path to $n$
      $n \leftarrow n'$
```

$\rightarrow$ Hill Climbing keeps choosing actions leading to a direct successor state with best heuristic value. It stops when no more immediate improvements can be made.

- Alternative names: Greedy Local Search and Gradient-Descent.
- Often used in optimization problems where all "states" are feasible solutions, and we can choose the search neighborhood ("child nodes") freely. (Return just n.State, rather than the path to n)

Hill Climbing: Guarantees and Complexity

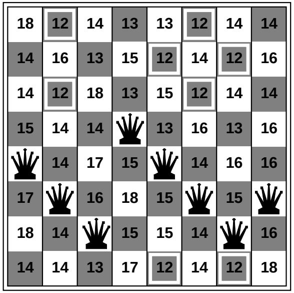
Guarantees:

- Completeness: No. Search ends when no more immediate improvements can be made (= local minimum, up next). This is not guaranteed to be a solution.
- Optimality: No, for the same reason.

Complexity:

- Time: We stop once the value doesn't strictly increase, so the state space size has a bound.
  $\rightarrow$ Note: This bound is (a) huge, and (b) applies to a single run of Hill-Climbing, which typically does not find a solution.
- Memory: Basically no consumption: $O(b)$ states at any moment in time.

Hill Climbing: Example 8-Queens Problem



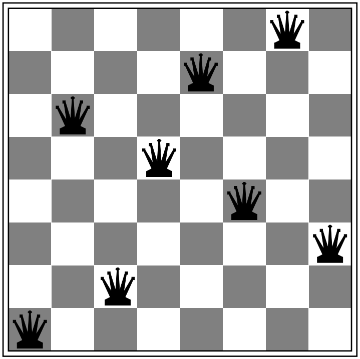
Problem: Place the queens so that they don't attack each other.

Heuristic: Number of pairs attacking each other. E.g. $h = 17$.

Neighborhood: Move any queen within its column. Best successors: states with $h = 12$.

→ Starting from random initialization, solves only 14% of cases.

A Local Minimum in the 8-Queens Problem



→ Current h value is? 1. To get $h = 0$, we must move either of the 2 queens involved in the conflict. But every such move results in at least 2 new conflicts, so this is a local minimum.

Hill Climbing: Difficulties

Difficulties:

- Local minima: All neighbors look worse (have a worse h value) than the current state (e.g.: previous slide).

  $\rightarrow$ If we stop, the solution may be sub-optimal (or not even feasible). If we don't stop, where to go next?
- Plateaus: All neighbors have the same h value as the current state.

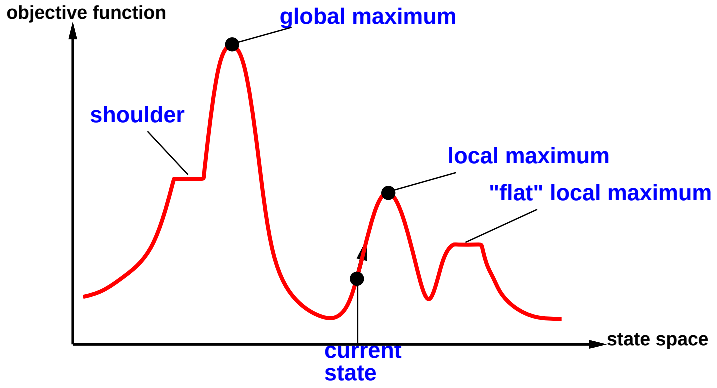
  $\rightarrow$ Moves will be chosen completely at random.

Strategies addressing these:

- Re-start when reaching a local minimum, or when we have spent a certain amount of time without "making progress".
- Do random walks (moving to a successor chosen uniformly at random from the set of successors) in the hope that these will lead out of the local minimum/plateau.

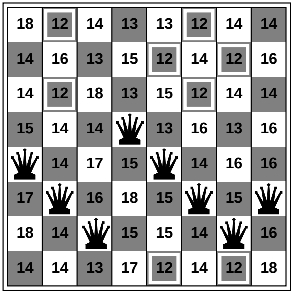
$\rightarrow$ Configuring these strategies requires lots of algorithm parameters. Selecting good values is a big issue in practice. (Cross your fingers . . .)

State Space Landscape



Random sideways moves: escape from shoulders, but loop on flat local maxima

Hill Climbing With Sideways Moves: Example 8-Queens



Heuristic: Number of pairs attacking each other.

Neighborhood: Move any queen within its column.

Sideways Moves: In contrast to slide 13, allow up to 100 consecutive moves in which the h value does not get better.

$\rightarrow$ Starting from random initialization, solves 94% of cases!

$\rightarrow$ Successful local search procedures often combine randomness (exploration) with following the heuristic (exploitation).

Simulated Annealing

- Hill-climbing that never moves downhill is incomplete as it can get stuck on local minima.
- Random walk (move to a neighbor chosen uniformly at random) is complete but very slow.
- Can we combine the two?

Simulated Annealing: escape local maxima by allowing some "bad" moves but gradually decrease their size and frequency.

- In metallurgy, annealing is process of hardening a metal by heating it to a high temperature and then gradually cooling it down.
- **Idea:** Taking inspiration from processes in physics (objects cooling down), inject "noise" systematically: first a lot, then gradually less.

Simulated Annealing: Algorithm

```
function SIMULATED-ANNEALING($problem$, $schedule$) returns a solution state
  inputs: $problem$, a problem
          $schedule$, a mapping from time to “temperature”

  $current \leftarrow$ MAKE-NODE($problem$.INITIAL-STATE)
  for $t = 1$ to $\infty$ do
     $T \leftarrow schedule(t)$
     if $T = 0$ then return $current$
     $next \leftarrow$ a randomly selected successor of $current$
     $\Delta E \leftarrow next$.VALUE – $current$.VALUE
     if $\Delta E > 0$ then $current \leftarrow next$
     else $current \leftarrow next$ only with probability $e^{\Delta E/T}$
```

Probability of accepting a "“bad” move decreases exponentially with the “badness” of the move and the temperature going down.

$\rightarrow$ Used since early 80s for VSLI Layout and other optimization problems.

Questionnaire

Question!

Can local minima occur in route planning with h :=straight line distance?

(A): Yes. (B): No.

$\rightarrow$ Yes, namely if, from the current position, all roads lead away from the target. For example, following straight line distance from Rome to Egypt will lead you to? A beach in southern Italy . . . (similar for the dead-end street with Manhattan Distance in the path planning "bad case").

Questionnaire, ctd.

Question!

What is the maximum size of plateaus in the 15-puzzle with $h :=$Manhattan distance?

(A): 0 (B): 1

(C): 2 (D): ∞

$\rightarrow$ 0: Any move in the 15-puzzle changes the position of exactly one tile x. So the Manhattan distance changes for x and remains the same for all other tiles, thus the overall Manhattan distance changes. So the h value of every neighbor is different from the current one, and there aren't any plateaus.

Local Beam Search

Idea: Keep k states in memory instead of 1.

- At each step, all successors of k states are generated.
  $\rightarrow$ If any is a goal, stop.
  $\rightarrow$ If not, choose top k of the successors and repeat.
- Not the same as k searches run in parallel! Searches that find good states recruit other searches to join them.
  $\rightarrow$ Useful information is passed among parallel search threads.
- **Problem**: Often, all k states end up on same local hill.
- **Idea**: Stochastic Beam Search: Choose k successors randomly, biased towards good ones. $\rightarrow$ Prob. of choosing a successor is an increasing function of its value.
- Observe the close analogy to natural selection!

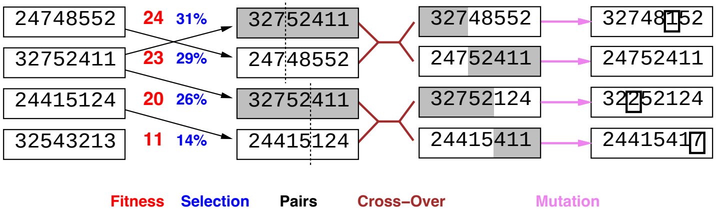
Genetic Algorithms

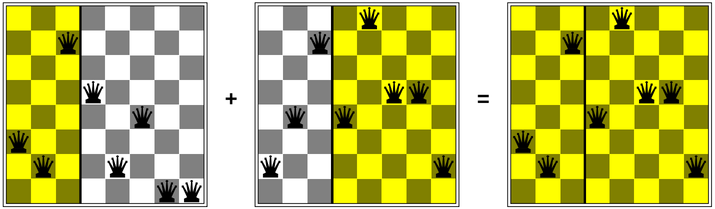
- Can we take inspiration from natural selection?
  $\rightarrow$ Organisms evolve: those adaptable to the environment survive and reproduce, the others die (Darwin).
- How? Variant of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state.
- Ingredients of genetic algorithms:
  - Initial population: individuals or chromosomes
  - Selection: by fitness function
  - Reproduction: crossover
  - Mutation
- Search in the space of individuals.
  $\rightarrow$ Steepest ascent hill-climbing, since little genetic alterations are performed on selected individuals.

Genetic Algorithms: Basic Principles

- Start with population: set of k randomly generated states
- Each state (individual) represented via a string on finite alphabet (e.g. 8-queens state represented via a string with 8 digits: positions of the queens in the columns).
- Next generation of states:
  - Each state ranked by fitness function (objective function).
    $\rightarrow$ It should return higher values for better states (e.g. number of non-attacking pairs of queens).
  - Pairs selected at random for reproduction.
    $\rightarrow$ Probability of being chosen directly proportional to fitness score.
    $\rightarrow$ For each pair, crossover point chosen randomly.
  - Offspring production by crossing over the parent strings at the crossover point.
  - Random mutation of some locations with small probability.

Genetic Algorithms: 8-Queens





The yellow shaded columns are retained in the crossover step and the unshaded columns are lost.

Genetic Algorithms: Algorithm

```
function GENETIC-ALGORITHM($population$, FITNESS-FN) returns an individual
  inputs: $population$, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     $new\_population$ ← empty set
     for $i$ = 1 to SIZE($population$) do
        $x$ ← RANDOM-SELECTION($population$, FITNESS-FN)
        $y$ ← RANDOM-SELECTION($population$, FITNESS-FN)
        $child$ ← REPRODUCE($x, y$)
        if (small random probability) then $child$ ← MUTATE($child$)
        add $child$ to $new\_population$
     $population$ ← $new\_population$
  until some individual is fit enough, or enough time has elapsed
  return the best individual in $population$, according to FITNESS-FN
```

```
function REPRODUCE($x, y$) returns an individual
  inputs: $x, y$, parent individuals

  $n$ ← LENGTH($x$); $c$ ← random number from 1 to $n$
  return APPEND(SUBSTRING($x, 1, c$), SUBSTRING($y, c + 1, n$))
```

Genetic Algorithms: Do They Work?

- Genetic algorithms combine an uphill tendency with random exploration and exchange of information among parallel search threads.
- Their primary advantage comes from the crossover operation.
- Crossover helps iff substrings are meaningful components: we combine large blocks of letters that have evolved independently to perform useful functions.
- Much work remains to be done to identify the conditions under which genetic algorithms perform well.

Summary

- Local search takes decisions based on its direct neighborhood. It is neither complete nor optimal, and suffers from local minima and plateaus. Nevertheless, it is often successful in practice.
- Local search methods such as hill climbing operate on complete-state formulations, keeping only a small number of nodes in memory.
- Several stochastic algorithms have been developed, including simulated annealing, which returns optimal solutions when given an appropriate cooling schedule.
- A genetic algorithm is a stochastic hill-climbing search in which a large population of states is maintained. New states are generated by mutation and by crossover, which combines pairs of states from the population.

Reading

Chapter 4: Beyond Classical Search, Section 4.1 [Russell and Norvig (2010)].

Content: Similar to what I cover in "Local Search Strategies", except it mistakenly acts as if local search could not be used to solve classical search problems. Discusses also Genetic Algorithms, and is nice as additional background reading.

References I

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Third Edition)*. Prentice-Hall, Englewood Cliffs, NJ, 2010.