CS 161 Fundamentals of Artificial Intelligence
Lecture 5

Local search algorithms

Quanquan Gu

Department of Computer Science
UCLA

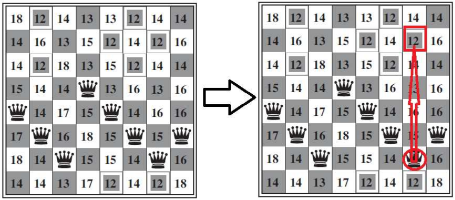
January 24, 2023

Outline

- Hill-climbing
- Simulated annealing
- Local beam search
- Genetic algorithms
- Local search in continuous spaces

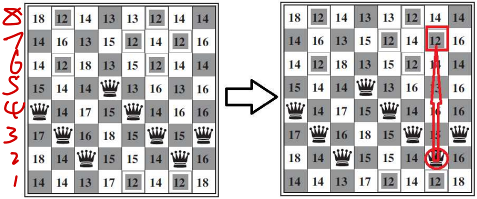
Iterative improvement algorithms

- In many optimization problems, **path** is irrelevant; the goal state itself is the solution



Iterative improvement algorithms

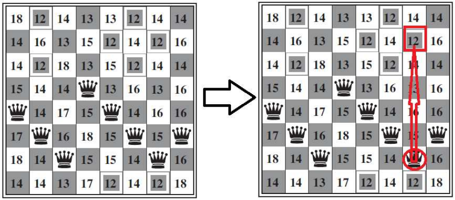
- In many optimization problems, **path** is irrelevant; the goal state itself is the solution
- For example, in the 8-queens problem, what matters is the final configuration of queens, not the order in which they are added.



43254323

Iterative improvement algorithms

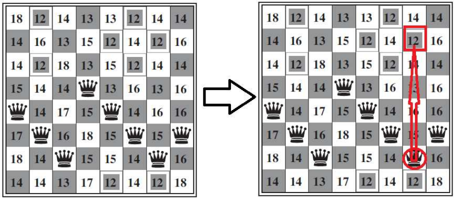
- In many optimization problems, **path** is irrelevant; the goal state itself is the solution
- For example, in the 8-queens problem, what matters is the final configuration of queens, not the order in which they are added.



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

Iterative improvement algorithms

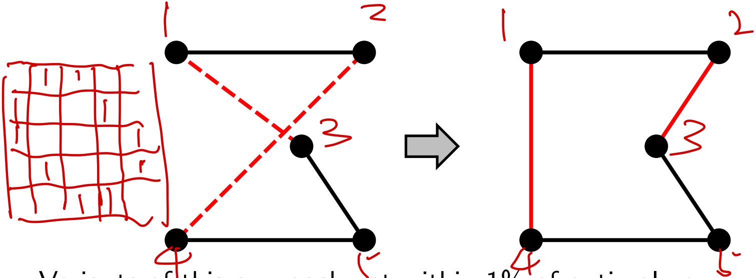
- In many optimization problems, **path** is irrelevant; the goal state itself is the solution
- For example, in the 8-queens problem, what matters is the final configuration of queens, not the order in which they are added.



- In such cases, can use **iterative improvement** algorithms; keep a single "current" state, try to improve it
- Constant space, suitable for online as well as offline search

Example: Travelling Salesperson Problem

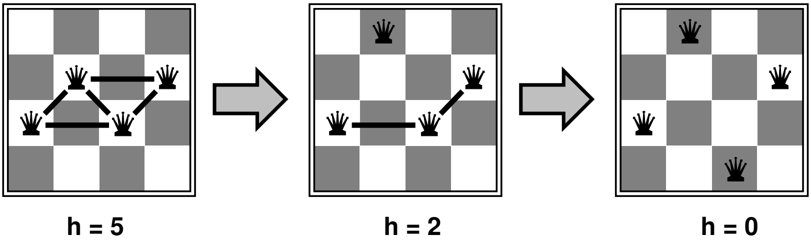
- Goal: to find the shortest path that visits each city and returns to the origin city
- Start with any complete tour (may have cross path, not optimal)
- Perform pairwise exchanges, each iteration reduces length of path



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

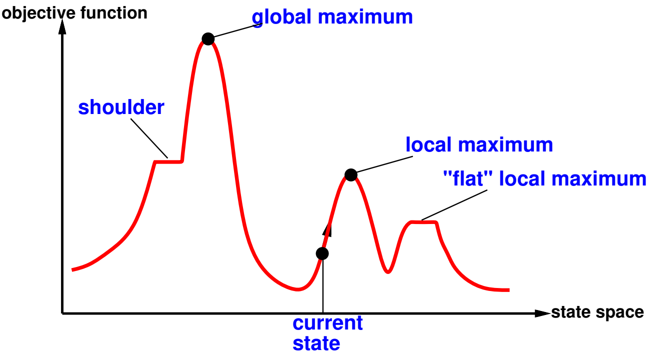
Example: n-queens

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



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

State space landscape



- Goal: to find global maximum
- Complete: finds a goal if one exists;
- Optimal: finds a global minimum/maximum
-

Hill-climbing (or gradient ascent/descent)

"Like climbing Everest in thick fog with amnesia"

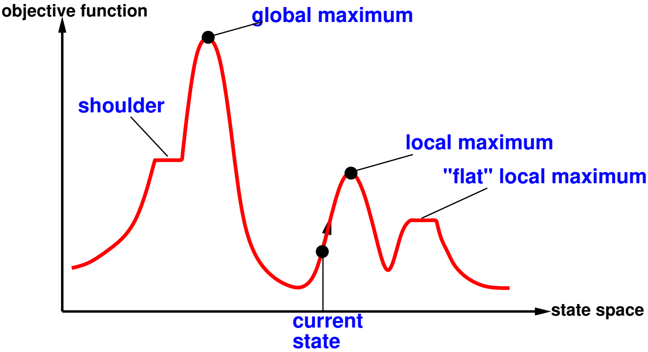
- Moves in the direction of increasing value

```
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
```

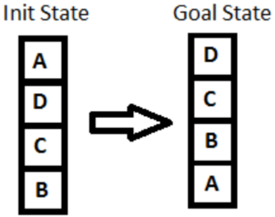
Hill-climbing contd.

Useful to consider **state space landscape**



- Escape from shoulders: **Random sideways moves**, maybe loop on flat maxima
- Escape from local maxima: **Random-restart hill climbing**, trivially complete

Hill-climbing example[1]

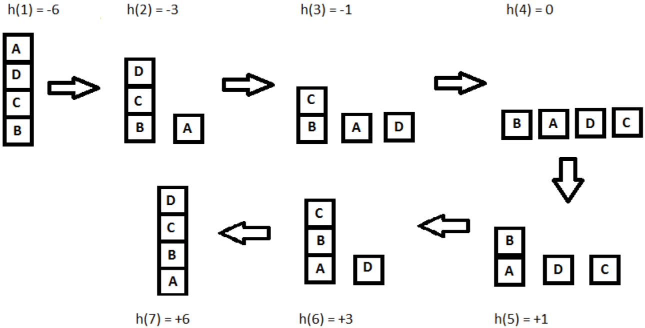


h(x) = +1 for all the blocks in the support structure if the block is correctly positioned otherwise -1 for all the blocks in the support structure.
h(1): A is incorrectly positioned with 3 support blocks (-3), B is incorrectly positioned with 0 support blocks (-0), C is incorrectly positioned with 1 support blocks (-1), D is incorrectly positioned with 2 support blocks (-2).

[1]Reference: https://www.baeldung.com/java-hill-climbing-algorithm

Hill-climbing example

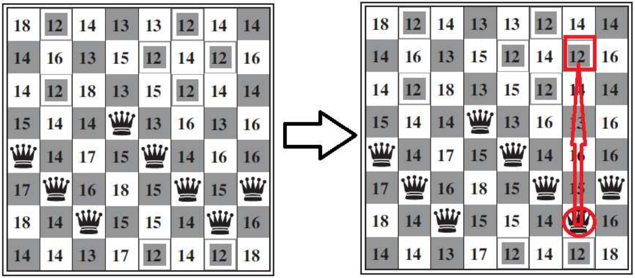
what about $\boxed{\begin{array}{c} C \\ B \end{array}}$ $\boxed{\begin{array}{c} D \\ A \end{array}}$? $h = +0 - 0 - 1 - 1 = -2$



h(x) = +1 for all the blocks in the support structure if the block is correctly positioned otherwise -1 for all the blocks in the support structure.

- h(1) = (-3) + (-0) + (-1) + (-2) = -6
- h(2) = (+0) + (-0) + (-1) + (-2) = -3
- h(3) = (+0) + (-0) + (-1) + (-0) = -1
- ...

Hill-climbing example: 8 queen



- Each state has 8 queens on the board, one per column
- Successors: move a single queen to another square in the same column
- heuristic cost function h: the number of pairs of queens that are attacking each other
- Before: $h = 0(\text{column}) + 5(\text{row}) + 12(\text{diagonal}) = 17$
- After: $h = 0(\text{column}) + 4(\text{row}) + 8(\text{diagonal}) = 12$

Simulated annealing

$e^{\Delta E / T}$

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and 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*
Δ*E* ← VALUE[*next*] − VALUE[*current*]
if Δ*E* > 0 **then** *current* ← *next*
else *current* ← *next* only with probability $e^{\Delta E / T}$

$\Delta E \le 0$

$T = O(1)$

① $\Delta E = -1$

② $\Delta E = -100$

$e^{-\frac{1}{T}}$

$e^{-\frac{100}{T}} \ll e^{-\frac{1}{T}}$

Properties of simulated annealing

$e^{\Delta E/T}$

- If the move improves the situation, it is always accepted. Otherwise, the algorithm accepts the move with some probability less than 1.
- The probability decreases exponentially with the "badness" of the move—the amount ΔE by which the evaluation is worsened.
- The probability also decreases as the "temperature" T goes down: "bad" moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases.
- If the schedule lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1

$\Delta E = -1$

① $T = 100$ $e^{-\frac{1}{100}}$

② $T = 1$ $e^{-1} \le e^{-\frac{1}{100}}$

Properties of simulated annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T decreased slowly enough $\implies$ always reach best state x^* $(E(x^*) = \max_x E(x))$

- $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T
- Thus, very likely to choose x^*!

Local beam search

each state has 100 successors

$k = 10$

• Idea: keep k states instead of 1; At each step, all the successors of all k states are generated. If any one is a goal, the algorithm halts. Otherwise, it selects the best k successors from the complete list and repeats.

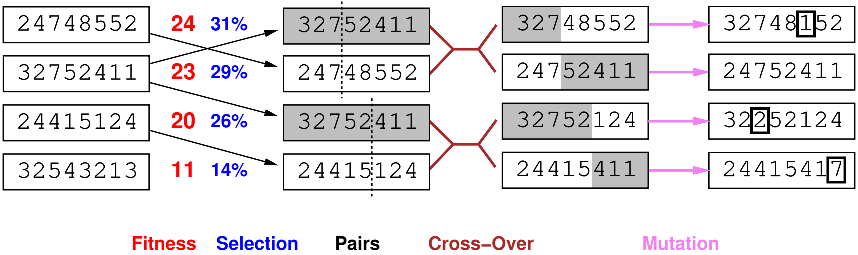
• Not the same as k searches run in parallel!
Searches that find good states recruit other searches to join them

• Problem: quite often, all k states end up on same local hill

• **Stochastic beam search**: choose k successors randomly, biased towards good ones
Observe the close analogy to natural selection!

Evolutionary algorithms/Genetic algorithms

• stochastic local beam search + generate successors from **pairs** of states

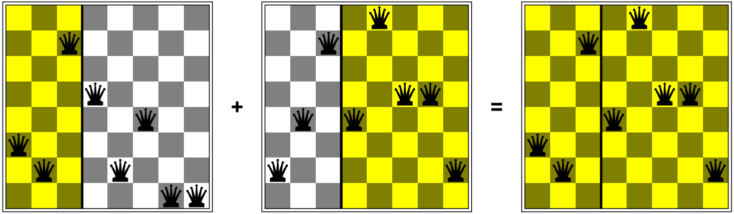


- Fitness function: higher score, higher chance to be selected
- Cross-over: crossover point is chosen randomly
- Mutation: small probability

Genetic algorithms contd.

GAs require states encoded as strings

Crossover helps **iff substrings are meaningful components**



Local search in continuous state spaces

Suppose we want to site three airports in Romania:
– 6-D state space defined by (x_1, y_1), (x_2, y_2), (x_3, y_3)
– objective function $f(x_1, y_1, x_2, y_2, x_3, y_3) =$
sum of squared distances from each city to nearest airport

Discretization methods turn continuous space into discrete space, e.g., **empirical gradient** considers $\pm\delta$ change in each coordinate

$f(x, y) = x^2 + 2xy + \frac{1}{2}y^2$

$x, y \in \mathbb{R}$ $\quad \min_{x,y} f(x, y)$

Gradient methods

$$\frac{\partial f}{\partial x} = 2x + 2y$$

$$\frac{\partial f}{\partial y} = 2x + y$$

Gradient methods compute

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

to increase/reduce f, e.g., by $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - \alpha \begin{bmatrix} 2x_0 + 2y_0 \\ 2x_0 + y_0 \end{bmatrix}$$

$$= \begin{bmatrix} (1-2\alpha)x_0 - 2\alpha y_0 \\ -2\alpha x_0 - (1-\alpha)y_0 \end{bmatrix}$$

Newton-Raphson Method

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}\left[\frac{\partial f}{\partial x}\right] = 2 \qquad \frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right) = 2$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial x}\right) = 2 \qquad \frac{\partial^2 f}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial y}\right) = 1$$

Sometimes can solve for $\nabla f(\mathbf{x}) = 0$ exactly.
Newton–Raphson (1664, 1690) iterates $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x})\nabla f(\mathbf{x})$ to solve $\nabla f(\mathbf{x}) = 0$, where $\mathbf{H}_{ij} = \partial^2 f/\partial x_i \partial x_j$

$$H_f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 2 & 2 \\ 2 & 1 \end{bmatrix}$$

$$H_f^{-1}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 2 & 2 \\ 2 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} -\frac{1}{2} & 1 \\ 1 & -1 \end{bmatrix}$$

Acknowledgment

The slides are adapted from Stuart Russell et al.