Chapter 4 (Section 4.3, …) 2nd Edition
or
Chapter 4 (3rd Edition)
Local Search and Optimization

Outline

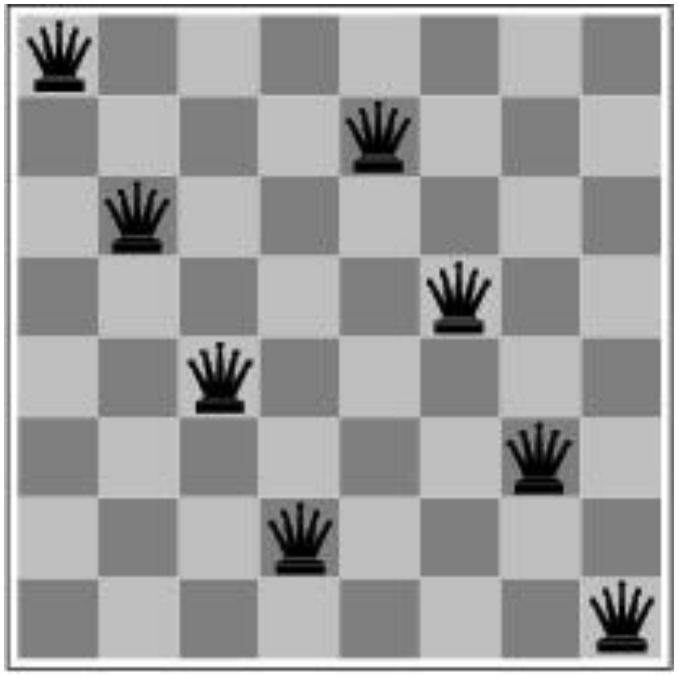
- Local search techniques and optimization
  - Hill-climbing
  - Gradient methods
  - Simulated annealing
  - Genetic algorithms
  - Issues with local search

Local search and optimization

- Previously: systematic exploration of search space.
  - Path to goal is solution to problem

- YET, for some problems path is irrelevant.
  - E.g 8-queens

- Different algorithms can be used
  - Local search



Local search and optimization

- Local search
  - Keep track of single current state
  - Move only to neighboring states
  - Ignore paths

→ শুধুমাত্র একটা current state কে maintain করবো। previous যে state visit করেছিলাম তার কোনো info রাখিনা। problem হলো আগের visited state গুলো আবার visit করতে পারি। যদিও state টা ২ বার visit করছি trajectory টা change হবে। আগে যেভাবে execute করেছি not necessarily same কাজ আবার করবো। contour টা different.
যেহেতু past visit info রাখিনা → so memory কম লাগবে।

- Advantages:
  - Use very little memory
  - Can often find reasonable solutions in large or infinite (continuous) state spaces.

memory করাতে চাই কেন?
→ সব node memory তে রাখা মানে looks like bfs. $O(b^d)$ chess এ branching factor 35 on avg. যদি depth 100 হয় $O(35^{100})$

- "Pure optimization" problems
  - All states have an objective function
  - Goal is to find state with max (or min) objective value
  - Does not quite fit into path-cost/goal-state formulation
  - Local search can do quite well on these problems.

local search → prescribed philosophy নাই।
↳ খুঁজি neighbour গুলো search করে।
↳ যাদের hamming distance, euclidean distance কম or correlated যারা
neighbour যাবার faulty → current state এর performance and neighbour state এর performance কাছাকাছি হবে।
improve
deteriorate
improve
deteriorate

12.5, 8.5 ...
কিন্তু 20 হবে না।

so neighbour state choose করা হলে → stable state টা maintained হয়

limited amount of computational budget এ reasonable soln চাই

↳ কত iteration এ execution শেষ করে

শুরুর দিকে performance খারাপ থাকে। As time progress → stable হয়।

Total searching time (computational budget) এর শেষ দিকে local search apply করবো।

2 টা algo expect করছি।

(i) শুরুর দিকে jump করে এমন algo → যে জায়গাতে soln hidden আছে সেটা পেতে চাই। তা যদি শুধু neighbouring region check করি তাহলে chance থাকে যে আমরা promising region এ পৌঁছালামই না।

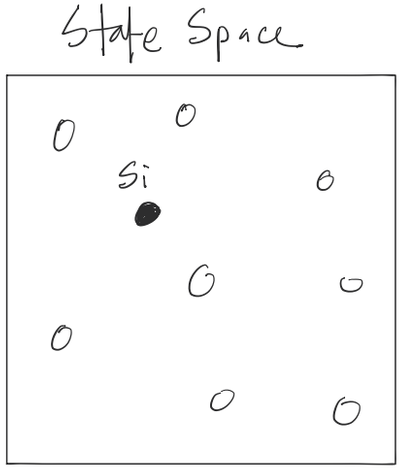
(ii) শেষ দিকে fine tunning

$$S_{i+1} = S_i + \eta_i \times d_i$$

- S_{i+1} ↑ next state
- S_i ↑ current state
- η_i → step size → current থেকে কত দূরে যেতে চাই → fixed না, iteration dependent parameter
- d_i ↳ direction of i-th iteration

d_i → each iteration এ direction change করতে পারবো।

hamming distance
correlation
euclidean distance



এক state থেকে আরেক state এ যেতে →

① কোন direction এ যাচ্ছি

② সেই direction এ কতটুকু যাচ্ছি।

শুরুর দিকে η_i বেশি নিলে → দূরে jump করতে পারবো।

So একটা function লিখবো যেটা শুরুতে η_i বড় রাখবে শেষের দিকে ছোট।

Let computational budget 200 iterations

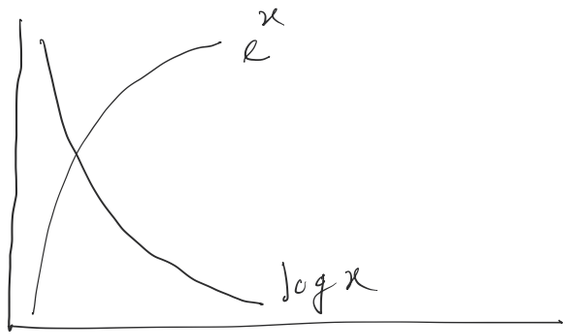
— শুরুর দিকে long jump

— শেষের দিকে short jump

$i = 200$

η_i = long jump

η_i = short jump



e^x, $\log x$ → smooth transition

η_i এর জন্য adaptive function লিখবো e^x, $\log x$ দিয়ে যেন transition smooth হয়।

methods

(i) adaptive local search algorithm → n_j, d_j change করে।

(ii) Local search in conjunction with a global search algo

শেষ দিকে — প্রথম দিকে

যদি বলে দেই $n_j = (rand) \times \frac{1}{t}$

0 – 1 এর মধ্যে

let 0.5

শুরুর দিকে হবে 0.5×1

শেষের দিকে $0.5 \times \frac{1}{200}$

$f = (1-s)(1-v)D$

$= f_1 \times f_2 \times f_3$ → এটা factor ই বড় হলে f বড় হবে।

$f = w_1 f_1 + w_2 f_2 + \dots + w_n f_n$

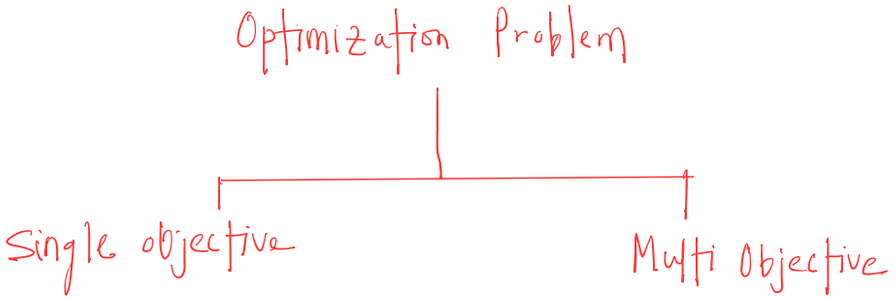
$f = f_1 f_2 f_3$

objective function গুলোকে equation করছি

f_1, f_2, f_3 } objectives. এটা combine করে f বানাচ্ছি।

কোনো কোনো situation এ যোগ করা যেতে পারে।

$w_1, w_2 \dots$ → কোন objective কে কতটুকু priority দিচ্ছি।



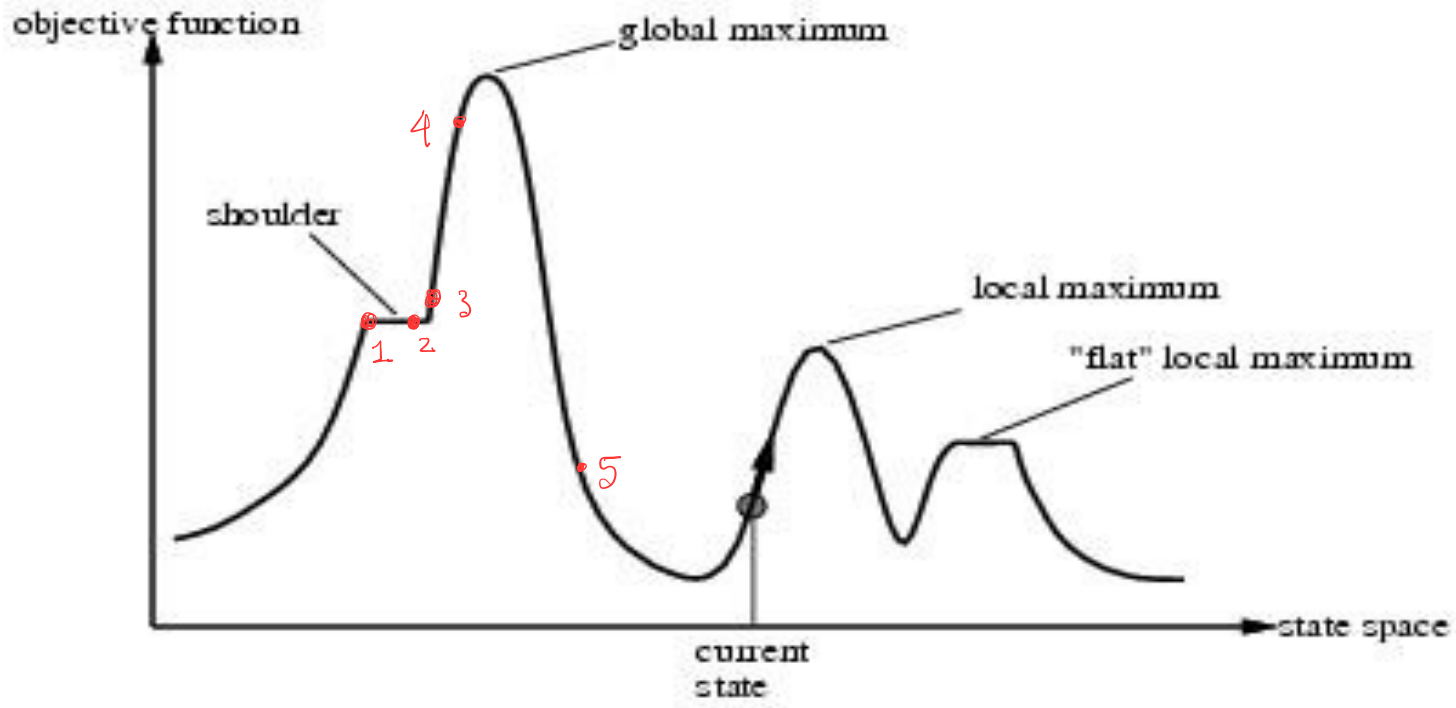
Conversion of Multiobjective to Single objective → কোন objective কে কতটুকু priority দিচ্ছি।

মন মাকি যোগ আকারে লিখতে চাই তাদা লাগবে।

Optimization problem: problem query তেই superlative degree থাকবে।

↳ obv. objective functions (1 or more) থাকবে।

"Landscape" of search



1,2,3 → performance কাছাকাছি

4 এ আসার পর big jump দিলে 5 এ চলে যেতে পারি।

Q. appropriate algorithm কি এবং সেটার component গুলো কি কি?

Hill-climbing search

function HILL-CLIMBING(*problem*) **return** a state that is a local maximum
input: *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*

↳ minimization problem

Hill-climbing search

- "a loop that continuously moves in the direction of increasing value"
  - terminates when a peak is reached
  - Aka greedy local search

- Value can be either
  - Objective function value
  - Heuristic function value (minimized)

- Hill climbing does not look ahead of the immediate neighbors of the current state.

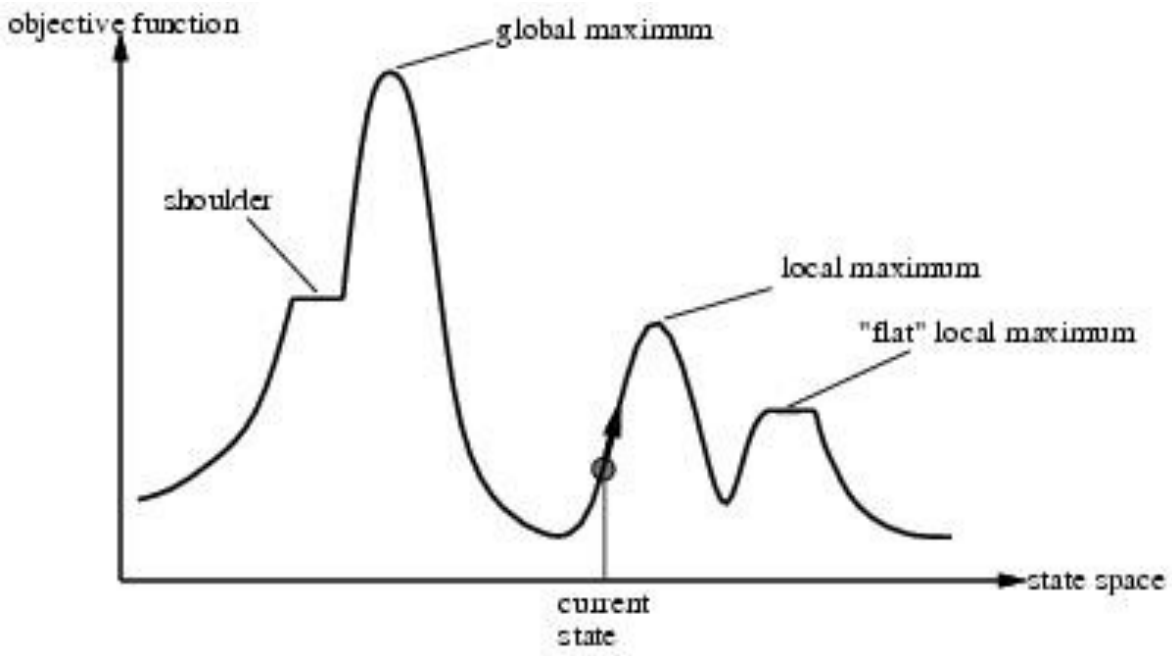
- Can randomly choose among the set of best successors, if multiple have the best value

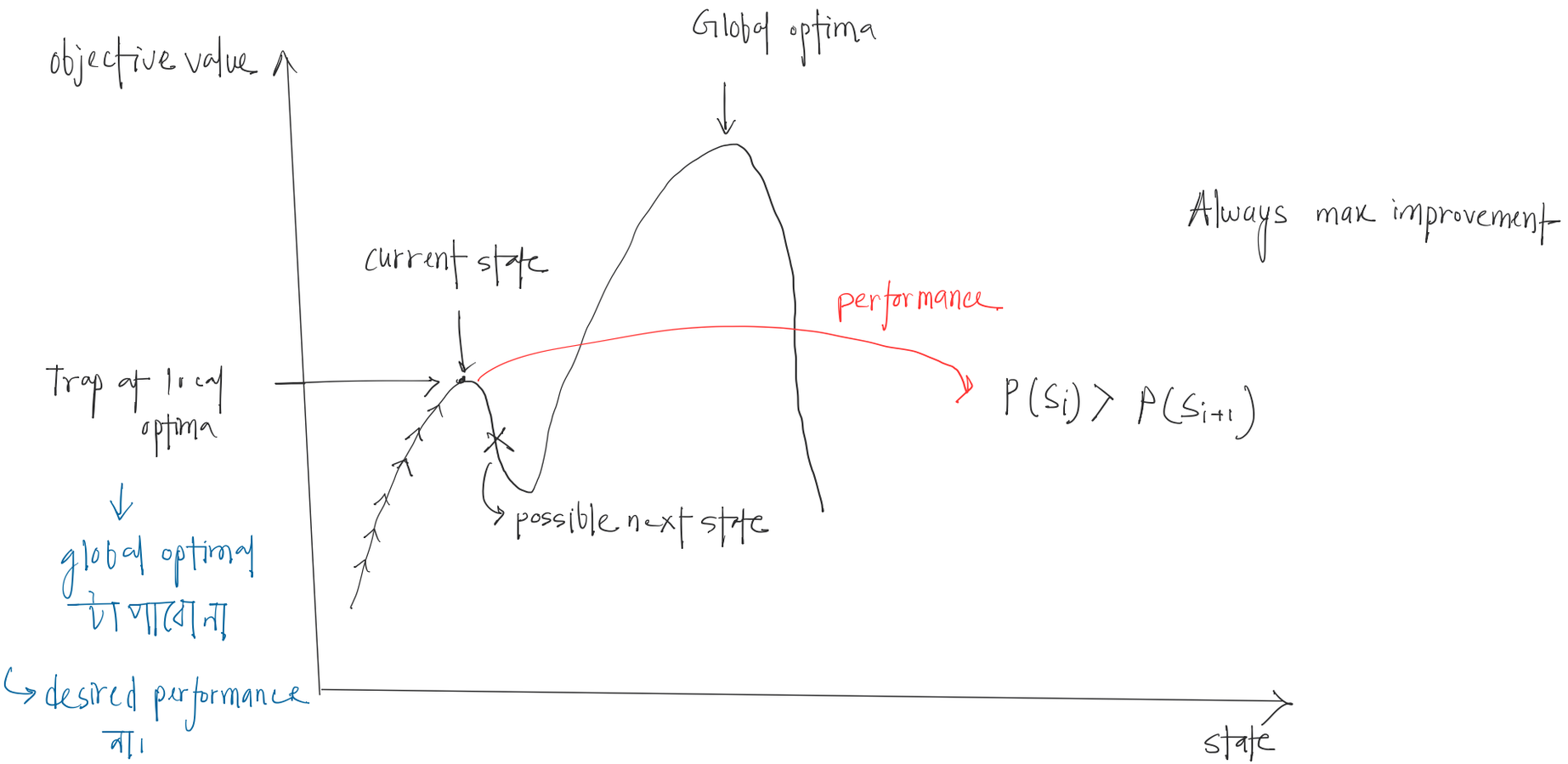
- Characterized as "trying to find the top of Mount Everest while in a thick fog"

Hill climbing and local maxima

- When local maxima exist, hill climbing is suboptimal

- Simple (often effective) solution
  - Multiple random restarts





Always max improvement

$P(S_i) > P(S_{i+1})$

↓

global optimal টা পাবো না

↳ desired performance না।

soln: ① আবার algorithm start, initial state random

আগের initial state টা নাও হতে পারে।

multiple random start করে দেখবো better soln পাই কি না

Hill-climbing example

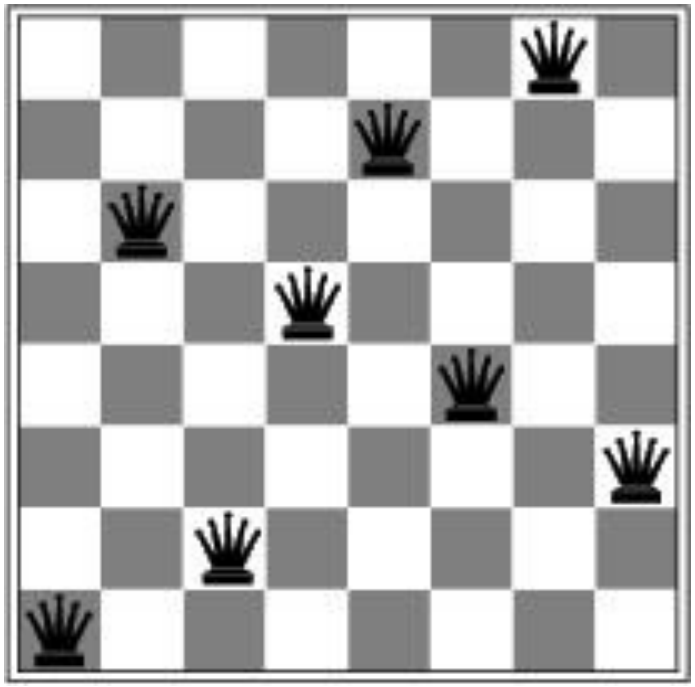
- 8-queens problem, complete-state formulation
  - All 8 queens on the board in some configuration
- Successor function:
  - move a single queen to another square in the same column.
- Example of a heuristic function *h(n)*:
  - the number of pairs of queens that are attacking each other (directly or indirectly)
  - (so we want to minimize this)

Hill-climbing example

Current state: h=17

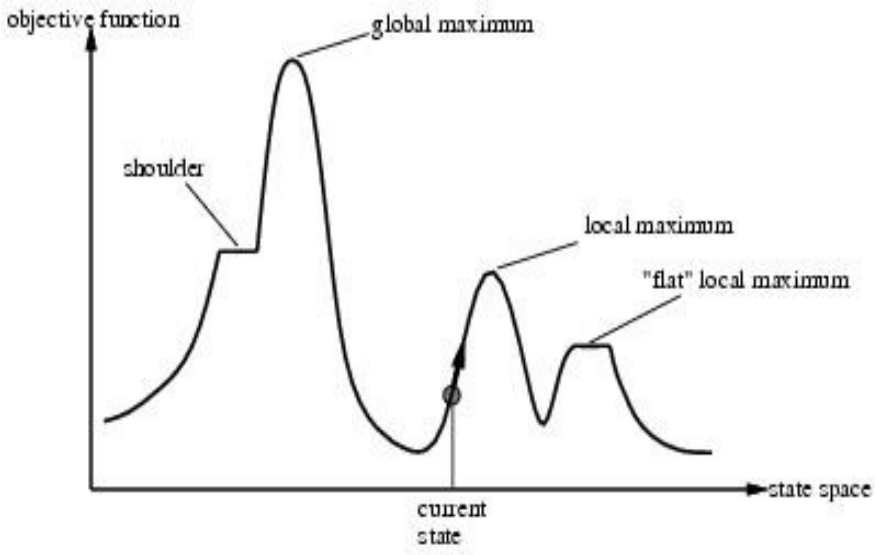
Shown is the h-value for each possible successor in each column

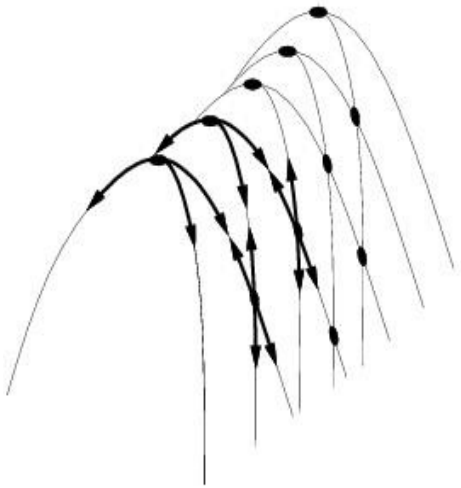
A local minimum for 8-queens



A local minimum in the 8-queens state space (h=1)

Other drawbacks





- Ridge = sequence of local maxima difficult for greedy algorithms to navigate

- Plateau = an area of the state space where the evaluation function is flat. –practically দেখা যাবে decimal point এর পরে ৮ 5 ঘর পর mismatch হলে ও flat

greedy algo তে যদি multiple local optima পাশাপাশি থাকে তখন একটা multiple restart

Performance of hill-climbing on 8-queens

- Randomly generated 8-queens starting states…
- 14% the time it solves the problem
- 86% of the time it get stuck at a local minimum
- However…
  - Takes only 4 steps on average when it succeeds
  - And 3 on average when it gets stuck
  - (for a state space with ~17 million states)

Possible solution…sideways moves

- If no downhill (uphill) moves, allow sideways moves in hope that algorithm can escape
  - Need to place a limit on the possible number of sideways moves to avoid infinite loops

- For 8-queens
  - Now allow sideways moves with a limit of 100
  - Raises percentage of problem instances solved from 14 to 94%

  - However….
    - 21 steps for every successful solution
    - 64 for each failure

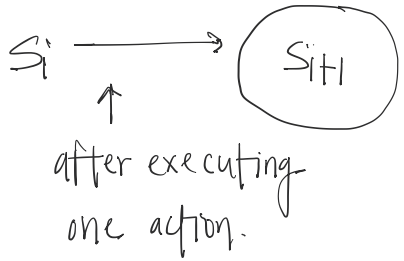
Hill-climbing variations

- Stochastic hill-climbing → একাধিক next state এর মধ্য থেকে একটাকে randomly pick করবো।
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move.

- First-choice hill-climbing
  - stochastic hill climbing by generating successors randomly until a better one is found
  - Useful when there are a very large number of successors

- Random-restart hill-climbing
  - Tries to avoid getting stuck in local maxima.

First-choice hill climbing



multiple next state তৈরি করবো যেটা current state থেকে ভালো সেটাকে select. যেটা performance এর respect এ first সেটা select

Hill-climbing with random restarts

- Different variations
  - For each restart: run until termination v. run for a fixed time
  - Run a fixed number of restarts or run indefinitely

- Analysis
  - Say each search has probability p of success
    - E.g., for 8-queens, p = 0.14 with no sideways moves
  - Expected number of restarts?
  - Expected number of steps taken?

Expected number of restarts

- Probability of Success = p
- Number of restarts = 1 / p
- This means 1 successful iteration after (1/p – 1) failed iterations
- Let avg. number of steps in a failure iteration = f
  and avg. number of steps in a successful iteration = s

Therefore, expected number of steps in random-restart hill climbing = 1 * s + (1/p – 1) f

So for 8-queens, p = 14%, s = 4, f = 3,

Expected no of moves = 1 * 4 + (1/0.14 -1) * 3 = 22

With sideways moves, p = 94%, s = 21, f = 64

Expected no of moves = 1 * 21 + (1/0.94 -1) * 64 = 25

Local beam search

- Keep track of *k* states instead of one
  - Initially: *k* randomly selected states
  - Next: determine all successors of *k* states
  - If any of successors is goal → finished
  - Else select *k* best from successors and repeat.

- Major difference with random-restart search
  - Information is shared among *k* search threads.

- Can suffer from lack of diversity.
  - Stochastic beam search
    - choose k successors proportional to state quality.

extreme:

① কোনো কিছু memory তে রাখছি না।

② যা কিছু generate করছি সবকিছু memory তে রাখছি

এদের মাঝামাঝি হলো beam search.

DFS এর মতো একটা part memory থেকে বাইরে রাখছি।

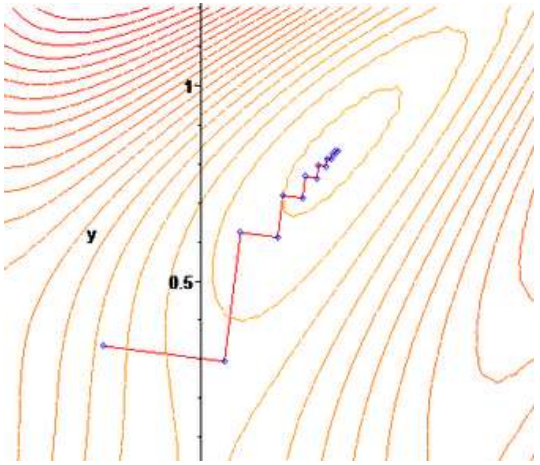
একটার পরিবর্তে k সংখ্যক state কে memory তে রাখবো।

k সংখ্যক next state রাখতে পারবো। multiple direction এ move unlike pure local search (একদিকে move)

objective function এ multiple dependent variable থাকে।

Gradient Descent

Assume we have some cost-function: $C(x_1,\ldots,x_n)$
and we want minimize over continuous variables X1,X2,..,Xn



1. Compute the *gradient* : $\frac{\partial}{\partial x_i} C(x_1,\ldots,x_n) \quad \forall i$

2. Take a small step downhill in the direction of the gradient:

$$x_i \rightarrow x'_i = x_i - \lambda \frac{\partial}{\partial x_i} C(x_1,\ldots,x_n) \quad \forall i$$

step size (fixed) — আগেই ধরে নিচ্ছি

cost function

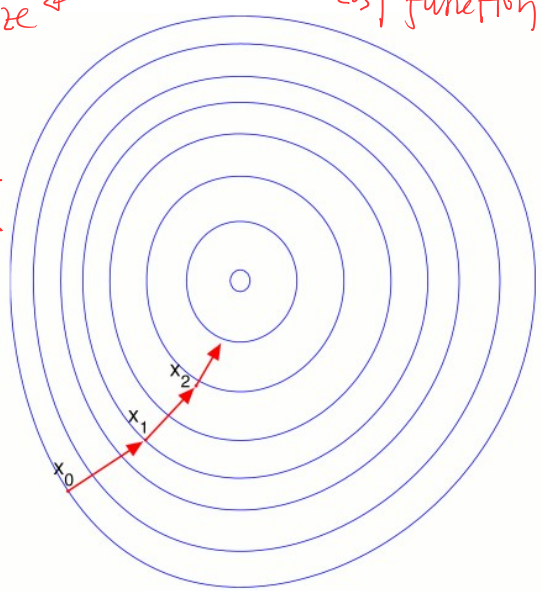
3. Check if $C(x_1,..,x'_i,..,x_n) < C(x_1,..,x_i,..,x_n)$

4. If true then accept move, if not reject.

5. Repeat.

$s_{i+1} = s_i + \eta_i \times d_i$
↳ direction নাচ্ছে।

direction
① down hill
② up hill



Learning as optimization

- Many machine learning problems can be cast as optimization

- Example:
  - Training data D = $\{(\underline{x}_1, c_1), \ldots\ldots (\underline{x}_n, c_n)\}$
    where $\underline{x}_i$ = feature or attribute vector
    and c_i = class label (say binary-valued)

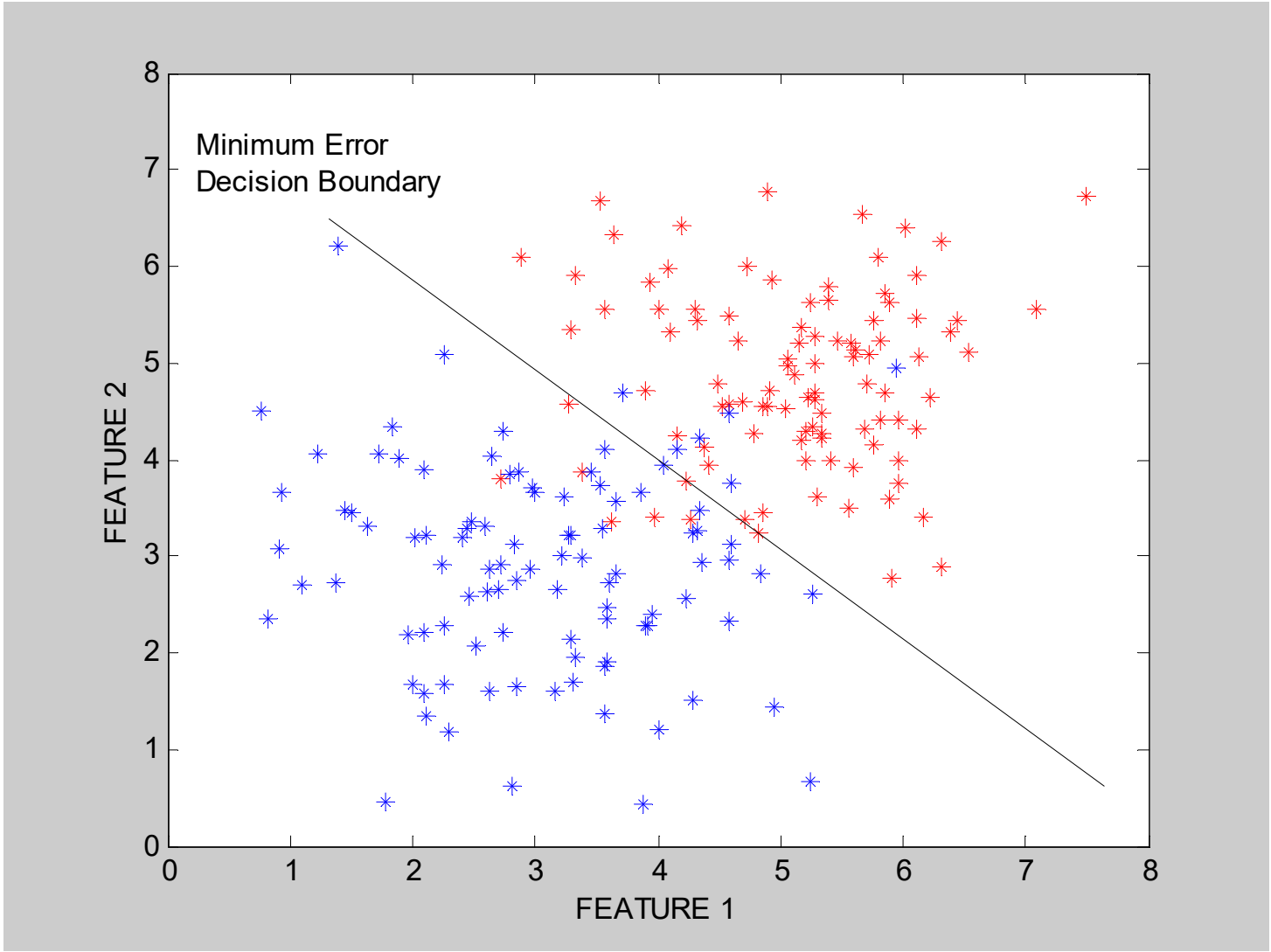
  - We have a model (a function or classifier) that maps from x to c
    e.g., sign($\underline{w}.\ \underline{x}'$) = {-1, +1}

  - We can measure the error E($\underline{w}$) for any settig of the weights $\underline{w}$, and given a training data set D

  - Optimization problem: find the weight vector that minimizes $\underline{E(w)}$

  (general idea is "empirical error minimization")

Learning a minimum error decision boundary



deterministic → best soln/path কে accept করি with the hope that এটা goal কে reach করবে। এজন্যই worst সুলো নিবো না।

Search using Simulated Annealing

- Simulated Annealing = hill-climbing with non-deterministic search

better soln পেলে definately সেটা receive করবো but যদি S_{i+1}, S_i থেকে খারাপ হয় এটাকেও receive করবো।

- Basic ideas:
  - like hill-climbing identify the quality of the local improvements
  - instead of picking the best move, pick one randomly
  - say the change in objective function is δ
  - if δ is positive, then move to that state
  - otherwise:
    - move to this state with probability proportional to δ
    - thus: worse moves (very large negative δ) are executed less often
  - however, there is always a chance of escaping from local maxima
  - over time, make it less likely to accept locally bad moves
  - (Can also make the size of the move random as well, i.e., allow "large" steps in state space)

better soln পেলাম না।

analogy: উপর থেকে বল ছেড়ে দিলাম। নিচ অবধি গেলে slowly roll করতে করতে থেমে যাবে। সেখানে যদি shake করার ব্যবস্থা করি আবার move করা start করতে পারে।

Physical Interpretation of Simulated Annealing

- A Physical Analogy:
  - imagine letting a ball roll downhill on the function surface
    - this is like hill-climbing (for minimization) → S_{i+1} is always better than S_i
  - now imagine shaking the surface, while the ball rolls, gradually reducing the amount of shaking
    - this is like simulated annealing

- Annealing = physical process of cooling a liquid or metal until particles achieve a certain frozen crystal state
  - simulated annealing:
    - free variables are like particles
    - seek "low energy" (high quality) configuration
    - get this by slowly reducing temperature T, which particles move around randomly

analogy (লোহার পাতকে heat bath(1000°C) এ insert করে। লাল হলে বের করে এনে expected shape দেয় then পানির বালতায় ডুবিয়ে দেয়।

excessive temp এ heat → property reverse হয়ে যায় (hardness → soft)

পানিতে ডুবিয়ে নিলে → property regain

Simulated annealing

function SIMULATED-ANNEALING(*problem, schedule*) **return** a solution state

input: *problem*, a problem

schedule, a mapping from time to temperature

local variables: *current*, a node.

next, a node.

T, a "temperature" controlling the probability 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* // greedy algo-এর property follow করে

else *current* ← *next* only with probability $e^{\Delta E/T}$

↳ greedy তে এই probability হবে 0

greedy	1	0	probability
simulated annealing	1	small	"

আগে $S_{i+1} > S_i$ হলে accept করতাম

এখন $S_{i+1} < S_i$ হলেও accept করবো। → property reverse

↳ (as we are not getting the better solution)

analogy: কত সময় পর্যন্ত worst soln কে accept করবো?

Simulated annealing: → idea metallurgy থেকে আসে

→ better solution accept এর probability = 1

better ছাড়া worst কে accept এর certain probability আছে $e^{\Delta E/T}$

$\Delta E > 0$

2টা soln এর difference if objective values

if $f(S_{i+1}) > f(S_i)$

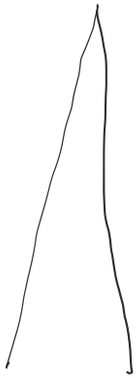
current ← S_{i+1}

else

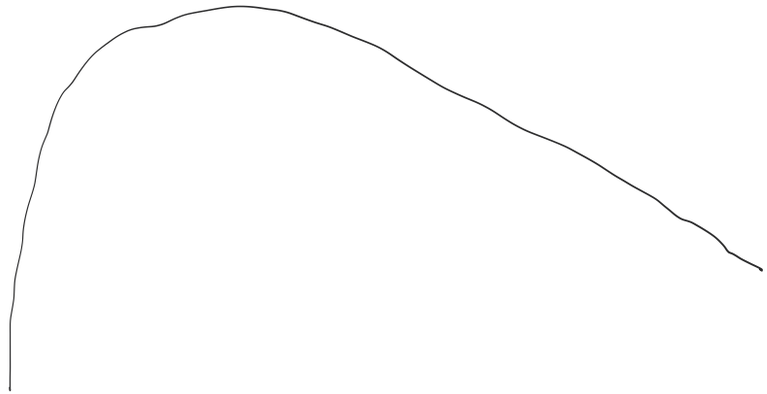
current ← S_{i+1} with probability $e^{\Delta E/T*(K_B)}$

- objective function এর value similar to energy

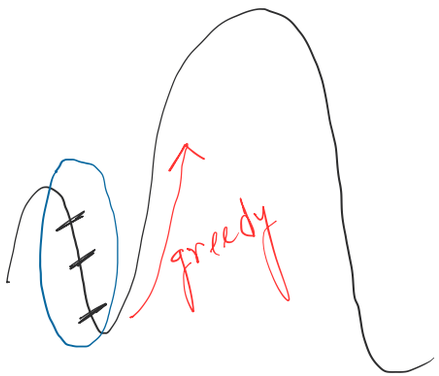
→ Boltzmann constant



step function



আমরা এরকম smooth transition চাই



3step এ

আমাদের worst soln কে accept করা লাগবে। তারপর আবার greedy follow

More Details on Simulated Annealing

- Lets say there are 3 moves available, with changes in the objective function of d1 = -0.1, d2 = 0.5, d3 = -5. (Let T = 1).
- pick a move randomly:
  - if d2 is picked, move there.
  - if d1 or d3 are picked, probability of move = exp(d/T)
  - move 1: prob1 = exp(-0.1) = 0.9,
    - i.e., 90% of the time we will accept this move
  - move 3: prob3 = exp(-5) = 0.05
    - i.e., 5% of the time we will accept this move

- T = "temperature" parameter
  - high T => probability of "locally bad" move is higher
  - low T => probability of "locally bad" move is lower
  - typically, T is decreased as the algorithm runs longer
    - i.e., there is a "temperature schedule"

Simulated Annealing in Practice

- method proposed in 1983 by IBM researchers for solving VLSI layout problems (Kirkpatrick et al, *Science*, 220:671-680, 1983).
  - theoretically will always find the global optimum (the best solution)

- useful for some problems, but can be very slow
    - slowness comes about because T must be decreased very gradually to retain optimality
  - In practice how do we decide the rate at which to decrease T? (this is a practical problem with this method)

Genetic algorithms

- Different approach to other search algorithms
  - A successor state is generated by combining two parent states
  -
- A state is represented as a string over a finite alphabet (e.g. binary)
  - 8-queens
    - State = position of 8 queens each in a column
      => 8 x log(8) bits = 24 bits (for binary representation)
- Start with *k* randomly generated states (population)
-
- Evaluation function (fitness function).
  - Higher values for better states.
  -
  - Opposite to heuristic function, e.g., # non-attacking pairs in 8-queens
- Produce the next generation of states by "simulated evolution"
  - Random selection
  - Crossover
  - Random mutation
  -

Genetic algorithms

- 23/(24+23+20+11) = 29% etc
-

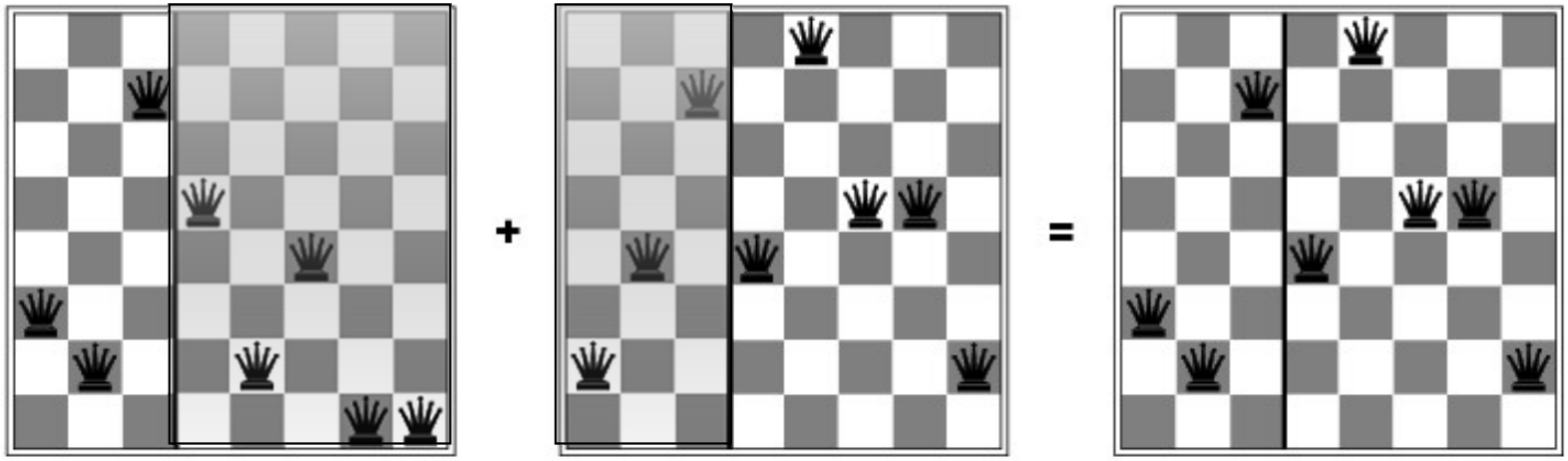
4 states for 8-queens problem

2 pairs of 2 states randomly selected based on fitness. Random crossover points selected

New states after crossover

Random mutation applied

Genetic algorithms



Has the effect of "jumping" to a completely different new part of the search space (quite non-local)

Genetic algorithm pseudocode

function GENETIC_ALGORITHM(*population,* FITNESS-FN) **return** an individual
    input: *population*, a set of individuals
        FITNESS-FN, a function which determines the quality of the individual
    repeat
        new_population ← empty set
        loop for i **from** 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

Comments on genetic algorithms

- Positive points
  - Random exploration can find solutions that local search can't
    - (via crossover primarily)
  - Appealing connection to human evolution
    - E.g., see related area of genetic programming

- Negative points
  - Large number of "tunable" parameters
    - Difficult to replicate performance from one problem to another
  - Lack of good empirical studies comparing to simpler methods
  - Useful on some (small?) set of problems but no convincing evidence that GAs are better than hill-climbing w/random restarts in general

Summary

- Local search techniques and optimization
  - Hill-climbing
  - Gradient methods
  - Simulated annealing
  - Genetic algorithms
  - Issues with local search