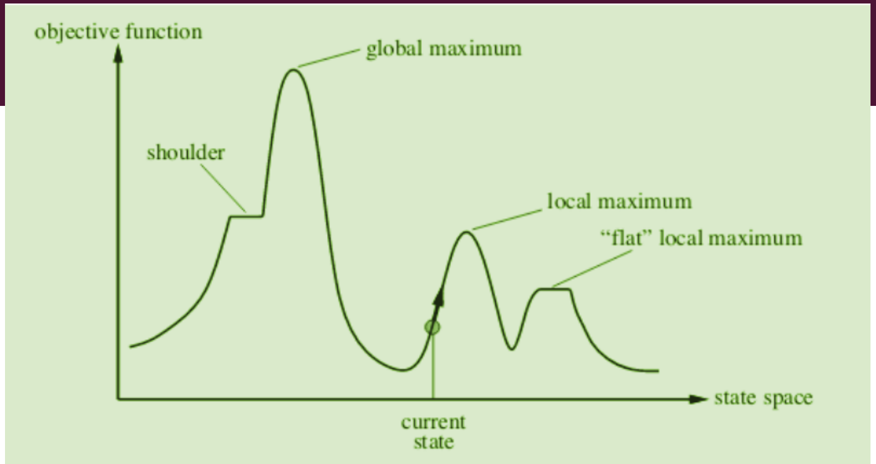
ARTIFICIAL INTELLIGENCE (AI)
STATE SPACE AND SEARCH ALGORITHMS

Shreya Ghosh
Assistant Professor, Computer Science and Engineering
IIT Bhubaneswar

HILL-CLIMBING (GREEDY LOCAL SEARCH) MAX VERSION



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*])

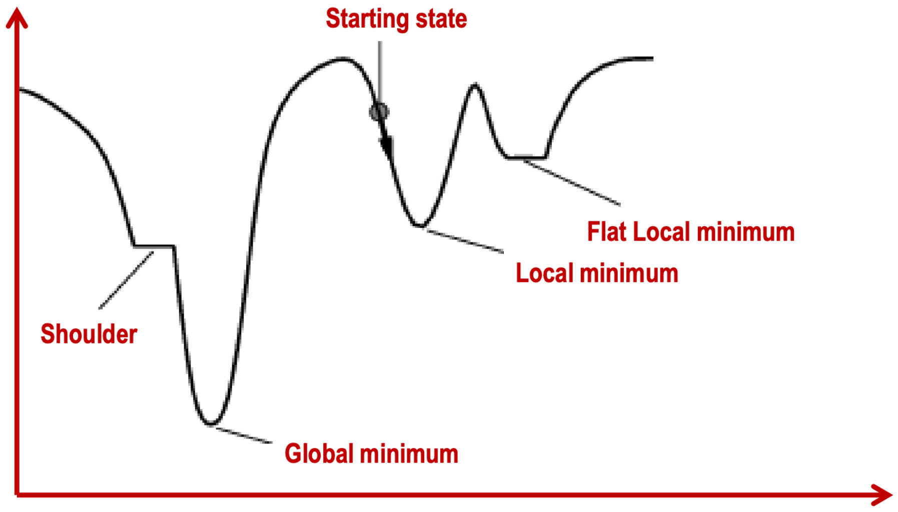
loop do

neighbor ← a highest valued successor of *current*

if VALUE [*neighbor*] ≤ VALUE[*current*] **then return** STATE[*current*]

current ← *neighbor*

min version will reverse inequalities and look for lowest valued successor



OPTIMIZATION OF CONTINUOUS FUNCTIONS

- Discretization
  - use hill-climbing

- Gradient descent
  - make a move in the direction of the gradient
    - gradients: closed form or empirical

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 prob. of downward steps

current $\leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])

for t $\leftarrow$ **1 to** ∞ **do**

T $\leftarrow$ *schedule*[*t*]

if *T = 0* **then return** *current*

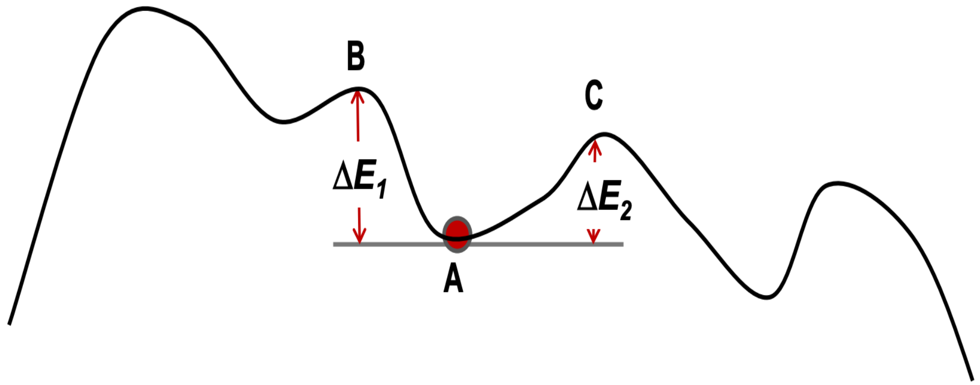
next $\leftarrow$ a randomly selected successor of *current*

$\Delta E \leftarrow$ VALUE[*next*] - VALUE[*current*]

if $\Delta E > 0$ **then** *current* $\leftarrow$ *next*

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

Probability of making a bad move = $e^{-\Delta E/T} = \dfrac{1}{e^{\Delta E/T}}$



Since $\Delta E_1 > \Delta E_2$ moving from A to C is exponentially more probable than moving from A to B

TEMPERATURE T

- 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"

LOCAL BEAM SEARCH

- Idea: Keeping only one node in memory is an extreme reaction to memory problems.

- 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 $\rightarrow$ finished
  - Else select *k* best from successors and repeat

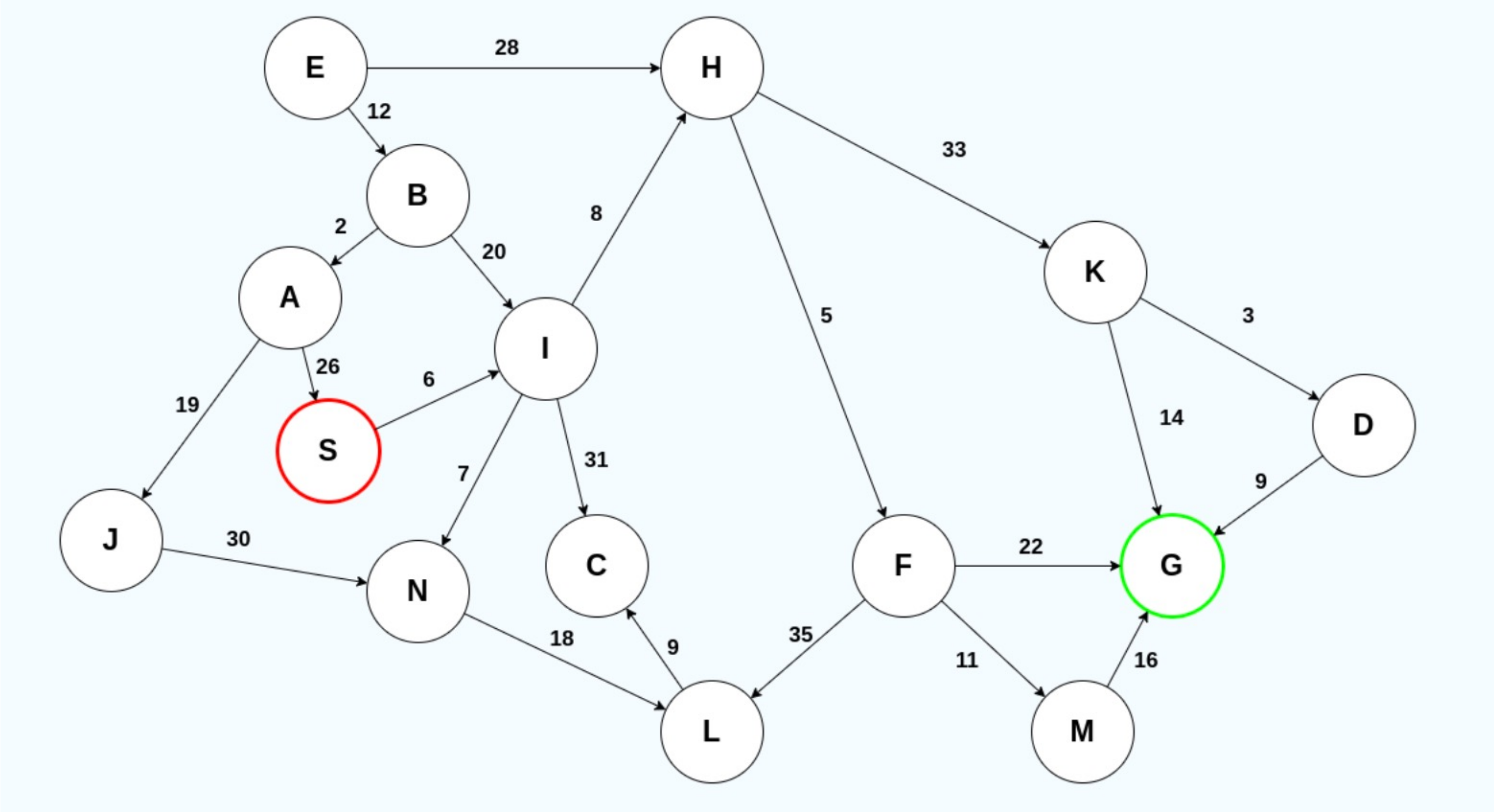
LOCAL BEAM SEARCH (CONTD)

- Not the same as *k random-start 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*
- Idea: Stochastic beam search
  - Choose *k successors randomly, biased towards good ones*

- Observe the close analogy to natural selection!

EXAMPLE



E
H
B
A
I
K
S
D
J
N
C
F
G
L
M
28
12
33
2
8
20
5
3
26
6
19
14
7
31
9
30
22
18
9
35
11
16

GENETIC ALGORITHMS

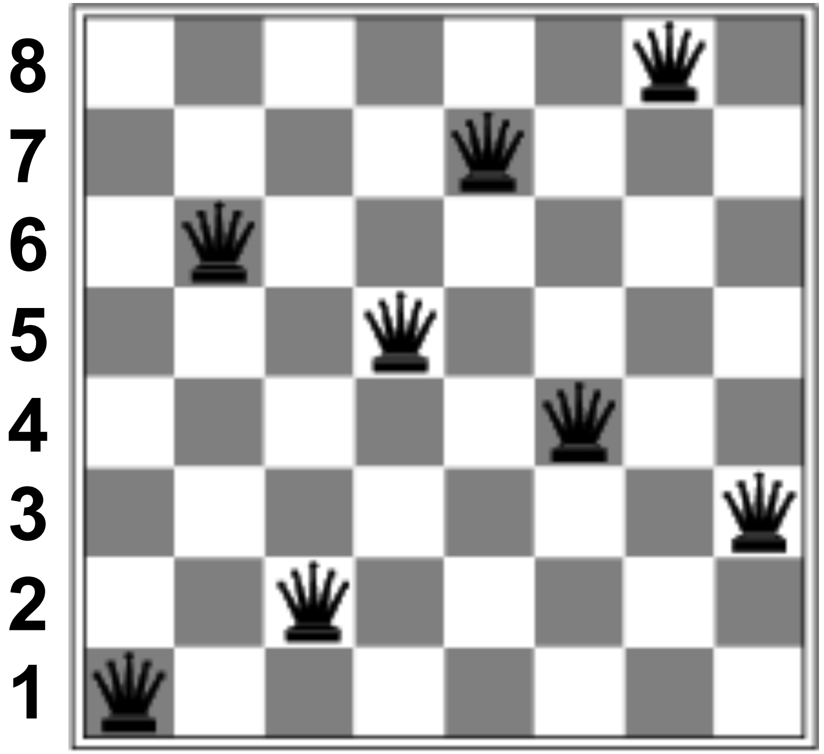
- Twist on Local Search: successor is generated by combining two parent states

- 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

- 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

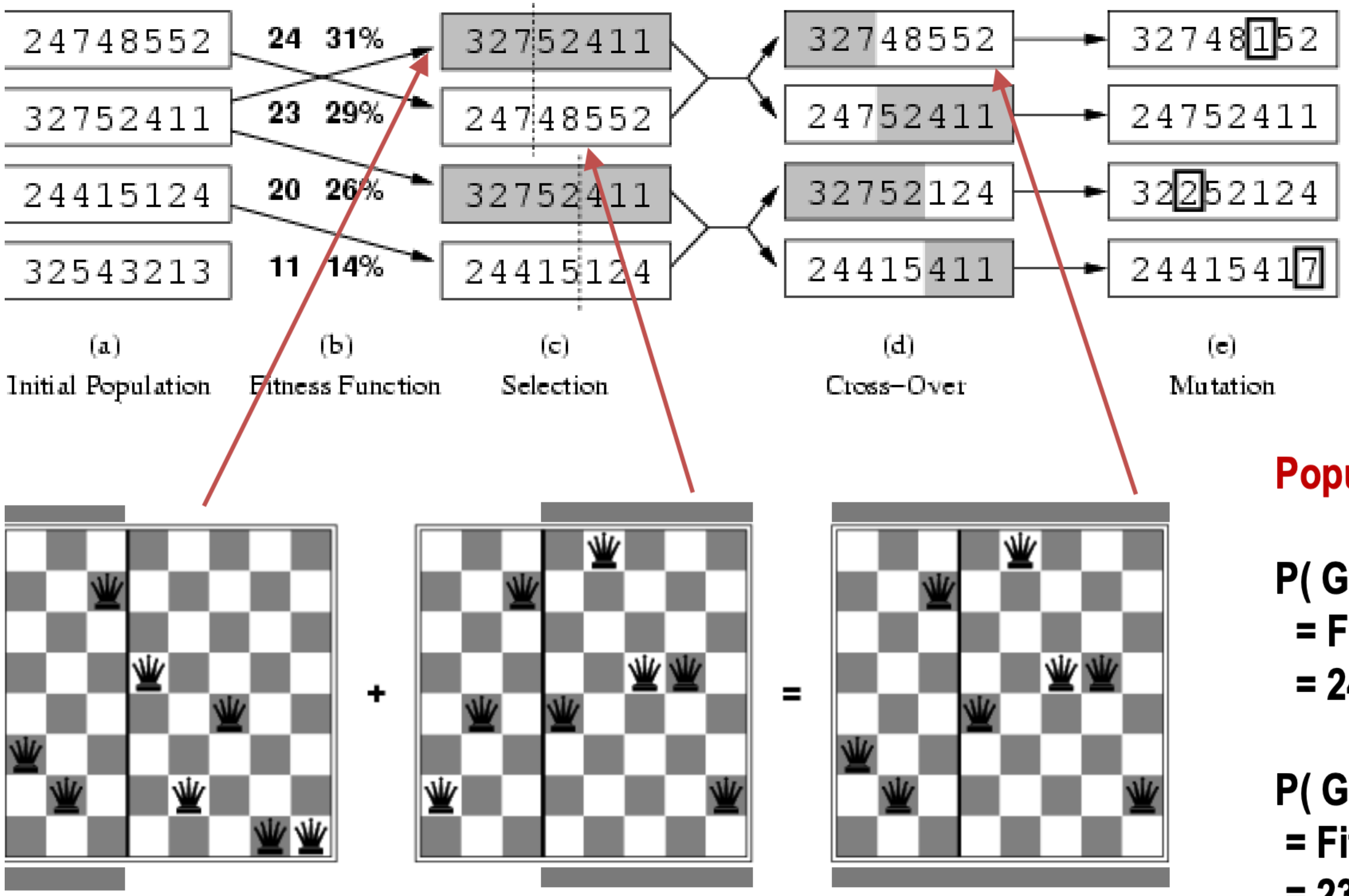


String representation
16257483

Can we evolve 8-queens through genetic algorithms?

Genetic Algorithm for 8 Queens

Fitness function: # non-attacking pairs
(min = 0, max = 8×7 / 2 = 28)



Population fitness = 24+23+20+11 = 78

P(Gene-1 is chosen)
= Fitness of Gene-1 / Population fitness
= 24 / 78 = 31%

P(Gene-2 is chosen)
= Fitness of Gene-2 / Population fitness
= 23 / 78 = 29%

GENETIC ALGORITHMS

4 states for 8-queens problem

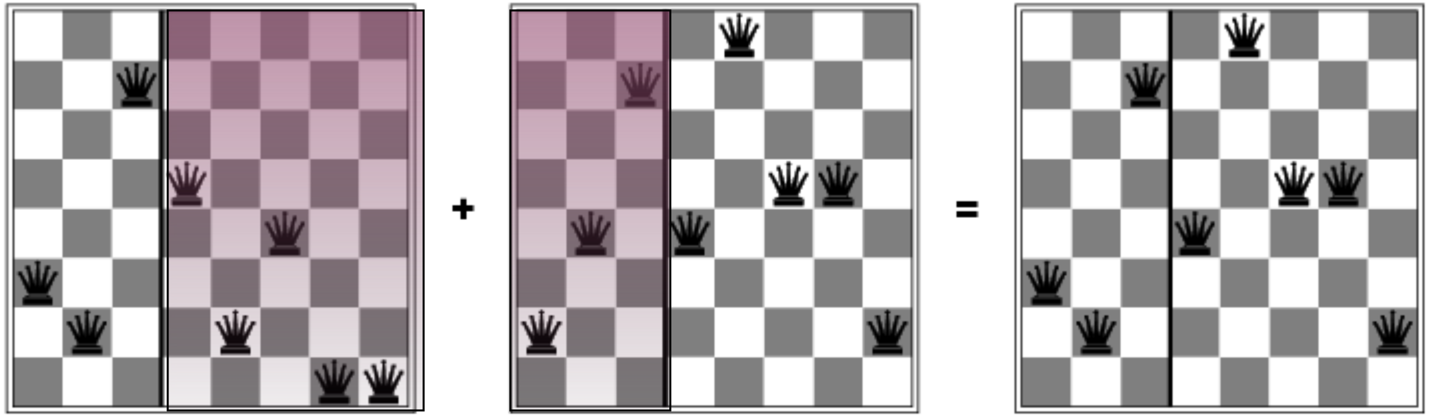
2 pairs of 2 states randomly selected based on fitness. Random crossover points selected

New states after crossover

Random mutation applied

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

GENETIC ALGORITHMS



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

COMMENTS ON GENETIC ALGORITHMS

- Positive points
  - Random exploration can find solutions that local search can't
    - (via crossover primarily)
  - Appealing connection to human evolution
    - "neural" networks, and "genetic" algorithms are **metaphors!**
- 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

IMPORTANT POINTS…

- Memory usage is one of the determining factors for choosing a search algorithm
  - For large state spaces, local search is an attractive practical option
- For local search:
  - It is important to understand the tradeoff between time and solution quality
  - It is important to understand the shape of the state space to decide things like temperature schedule in simulated annealing, durations of locking of moves in tabu search, and number of random restarts in gradient descent