

innovate

achieve

lead

BITS Pilani
Pilani Campus

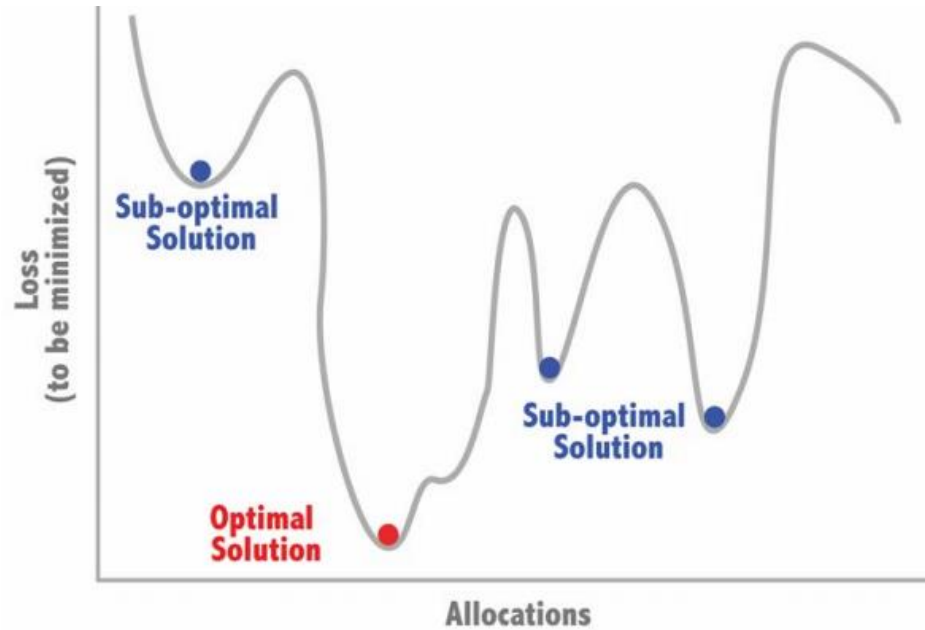
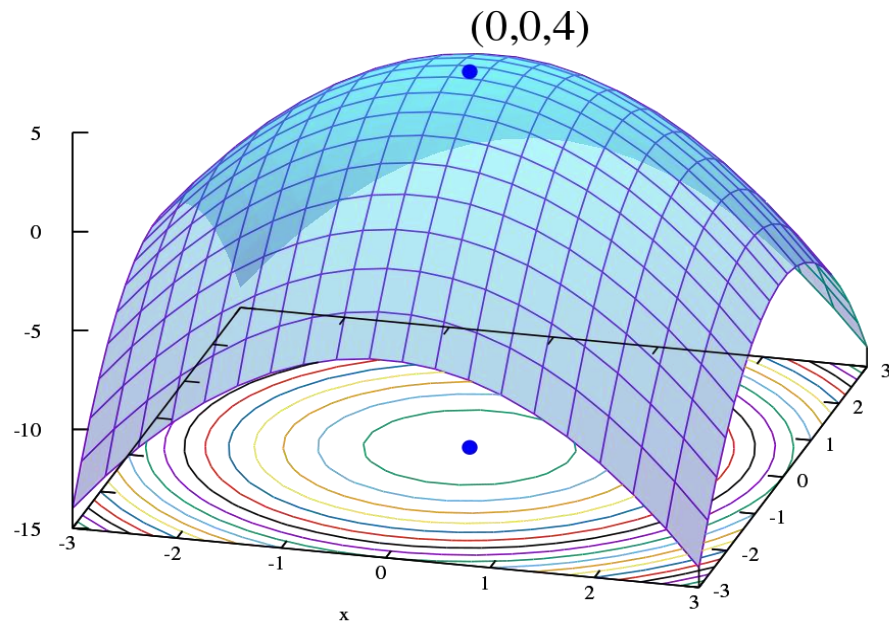


ACI: 17TH OCT

Optimization

Let $y = x^2 - 4x$, at what value of x does y will be maximum.

a. 2 b. 4 c. 6 d. 8



Path vs. State Optimization

Previous lecture: path to goal is solution to problem

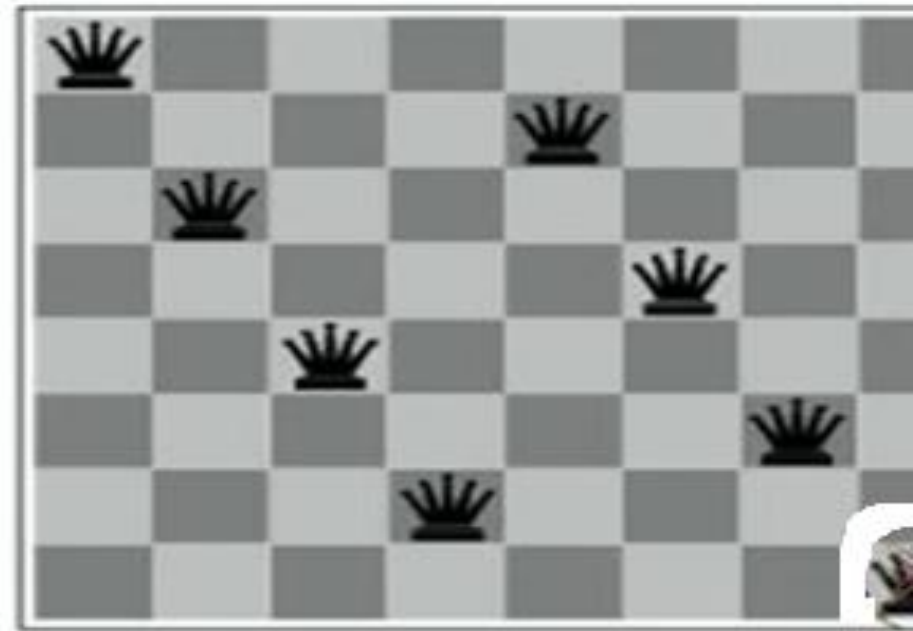
- systematic exploration of search space.

This lecture: a state is solution to problem

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

Different algorithms can be used

- Local search

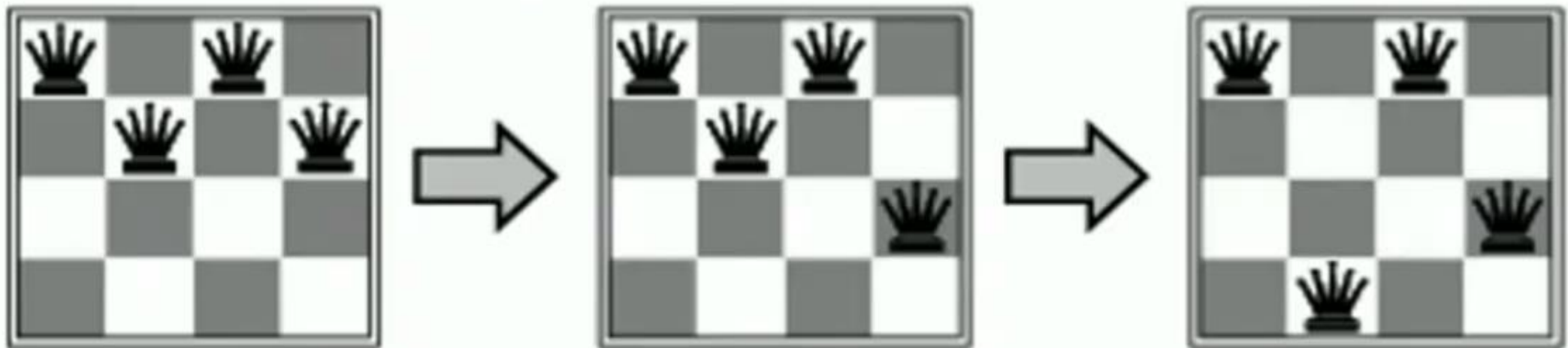


Local search and optimization

- Local search
 - Keep track of single current state
 - Move only to neighboring states
 - Ignore paths
- Advantages:
 - Use very little memory
 - Can often find reasonable solutions in large or infinite (continuous) state spaces.
- “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.

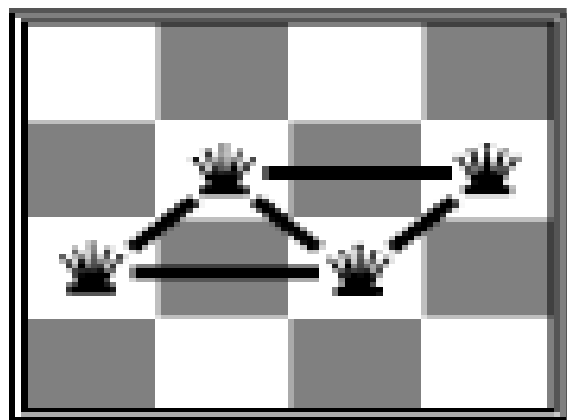
Example: n -queens

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

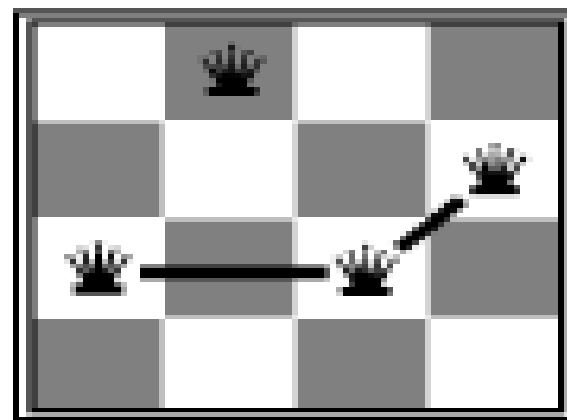
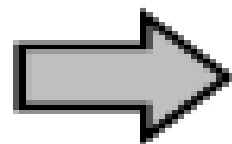


Example: n -queens problem

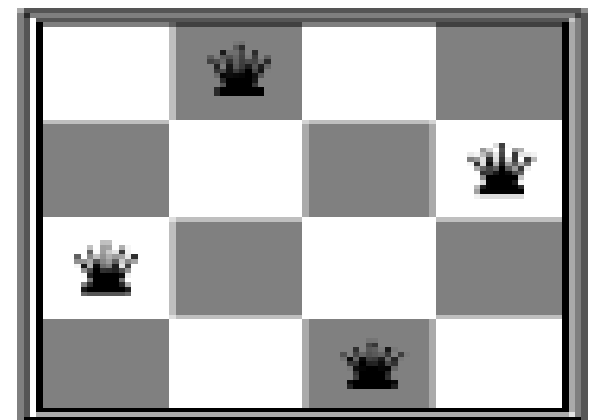
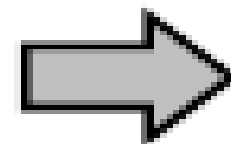
- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- **State space:** all possible n -queen configurations
- **Objective function:** number of pairwise conflicts
- What's a possible local improvement strategy?
 - Move one queen within its column to reduce conflicts



$h = 5$



$h = 2$



$h = 0$

Convert N queen problem to optimization problem



It can be treated as maximizing problem. In that case you just maximize the number of non attacking pairs of queens.

Or

It can be treated as minimizing problem. For this case you just minimize the number of attacking pair of queens.

Search Space

- State
 - 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
 - (so we want to minimize this)

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

loop do

neighbor \leftarrow a highest valued successor of *current*

if VALUE [*neighbor*] \leq VALUE[*current*] **then return** STATE[*current*]

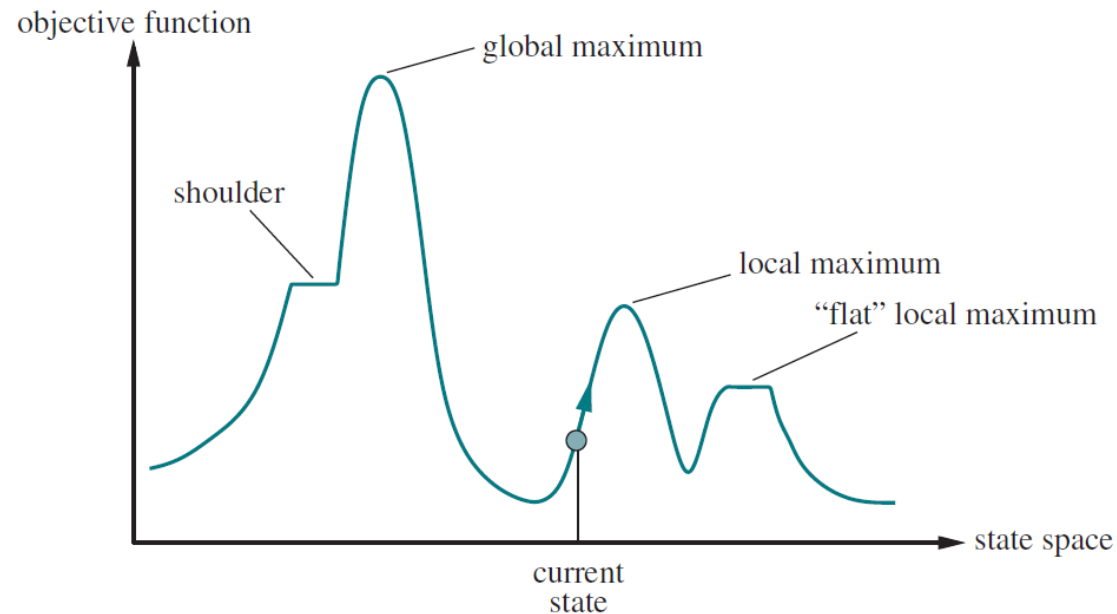
current \leftarrow *neighbor*

Hill-climbing search

- “a loop that continuously moves towards 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
- Can randomly choose among the set of best successors
 - if multiple have the best value

“Landscape” of search

Hill Climbing gets stuck in local maximum



Local maxima: a local maximum, as opposed to a global maximum, is a peak that is lower than the highest peak in the state space.

Plateaux: a plateau is an area of the state space where the evaluation function is essentially flat. The search will conduct a random walk.

Ridges: a ridge may have steeply sloping sides, so that the search reaches the top of the ridge with ease, but the top may slope only very gently toward a peak. Unless there happen to be operators that move directly along the top of the ridge, the search may oscillate from side to side, making little progress.

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 $8^8 \approx 17$ million states)

Escaping Shoulders: Sideways Move

- If no downhill (uphill) moves, allow sideways moves in hope that algo ^{min} can escape ^{next}
 - 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

Trivial Algorithms

- Random Sampling
 - Generate a state randomly
- Random Walk
 - Randomly pick a neighbor of the current state
- Both algorithms asymptotically complete.

Hill-climbing: stochastic variations

- Stochastic hill-climbing
 - Random selection among the uphill moves.
 - The selection probability can vary with the steepness of the uphill move.
- To avoid getting stuck in local minima
 - Random-walk hill-climbing
 - Random-restart hill-climbing
 - Hill-climbing with both

Hill Climbing with random walk

- When the state-space landscape has local minima, any search that moves only in the greedy direction cannot be complete
- Random walk, on the other hand, is asymptotically complete

Idea: Put random walk into greedy hill-climbing

- At each step do one of the two
 - Greedy: With prob p move to the neighbor with largest
 - Random: With prob $1-p$ move to a random neighbor

Hill-climbing with random restarts

- If at first you don't succeed, try, try again!
- Different variations
 - For each restart: run until termination vs. 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-queen $p = 0.14$ with no sideways moves
 - Expected number of restarts? $\rightarrow 1/p =$
 - Expected number of steps taken? \rightarrow Please calculate

Simulated Annealing

Simulated Annealing = physics inspired twist on random walk

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

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  $\infty$  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  $e^{\Delta E / T}$ 
```

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”

Simulated Annealing



Observations:

If bad neighbor then $\Delta E < 0$


1. If very bad neighbor, then ΔE will approach towards large negative number

Then $\exp(\Delta E/T)$ will approach towards 0.

It means the probability of accepting bad neighbor is 0.

2. As time advances Temperature T reduces to small value. Eventually T will tend to 0. In that case $\exp(\Delta E/T)$ will approach 0. That means, when t is very large and $T=0$, then probability of accepting the bad neighbor becomes less. So as time passes algorithm is less likely to accept bad moves

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
- Other applications: Traveling salesman, Graph partitioning, Graph coloring, Scheduling, Facility Layout, Image Processing, ...
- useful for some problems, but can be very slow
 - slowness comes about because T must be decreased very gradually to retain optimality 

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

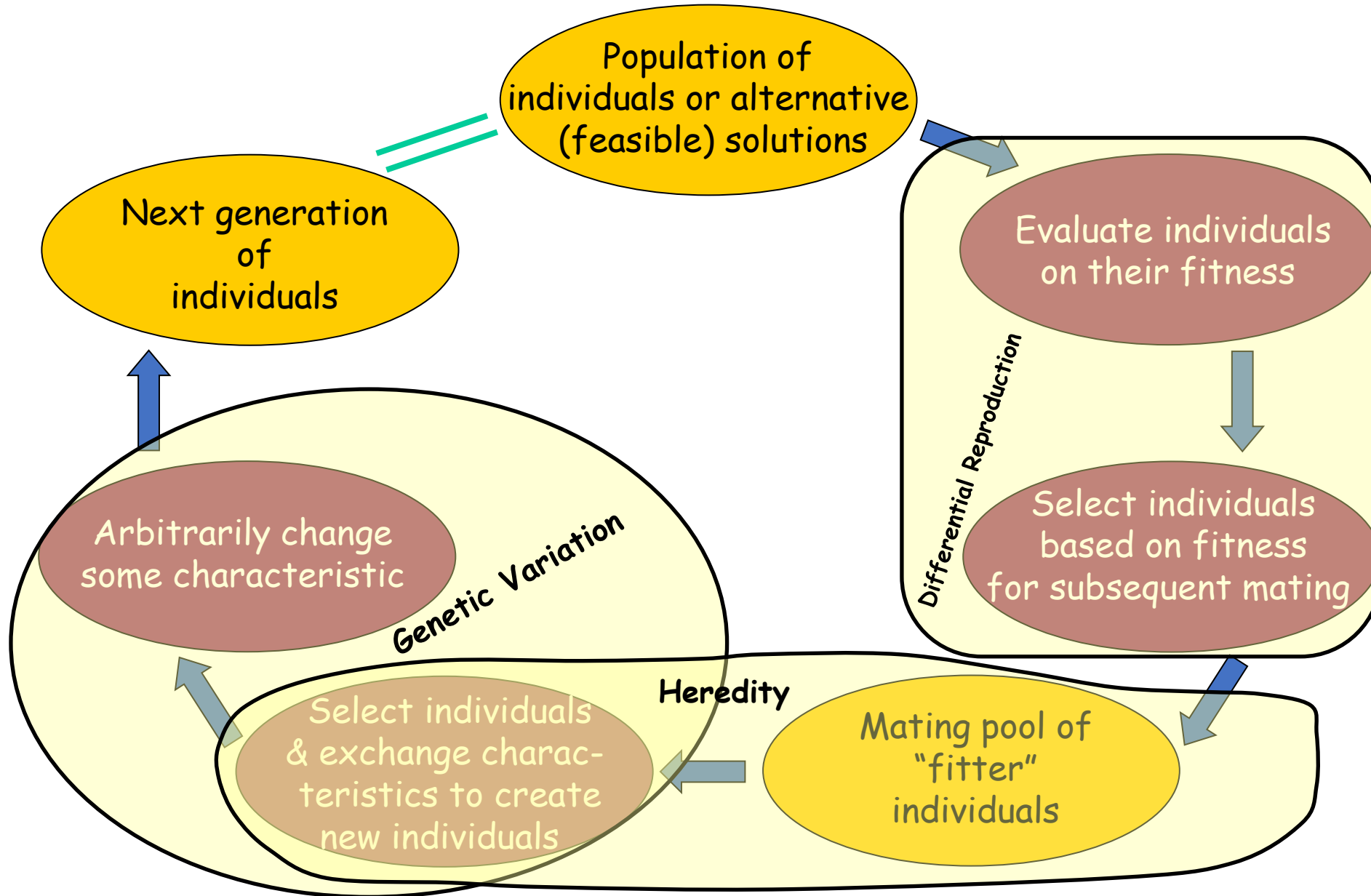
Local Beam Search (contd)

- 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*
- Observe the close analogy to natural selection

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

Genetic Algorithms



Genetic Algorithms

Basic Tasks

Generation of initial population

Evaluation

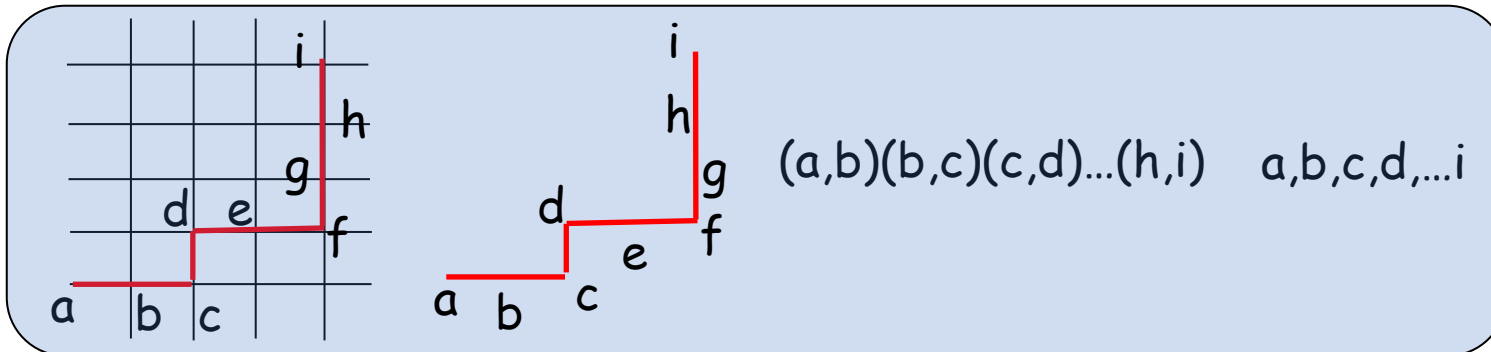
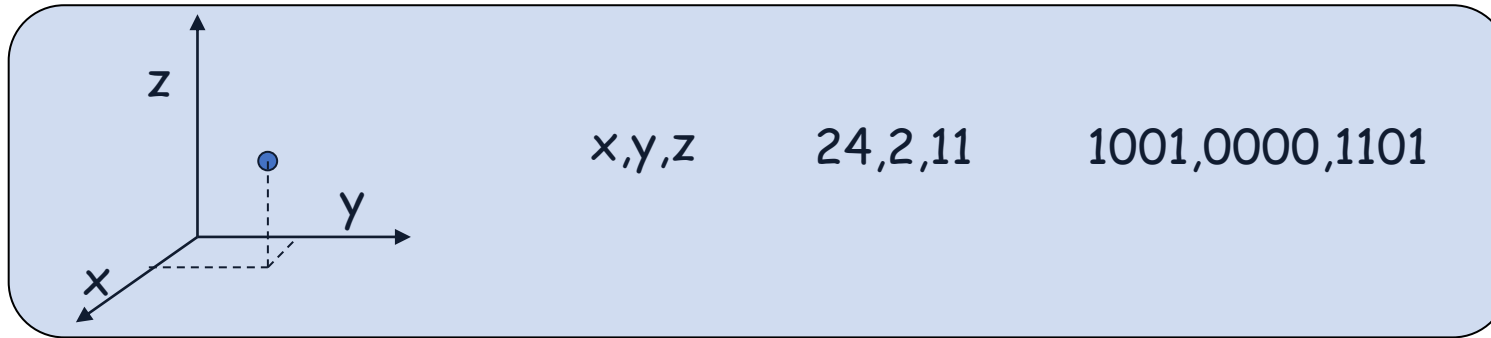
Selection (Reproduction operation)

Exchange characteristics to develop new individuals (Crossover operation)

Arbitrarily modify characteristics in new individuals (Mutation operation)

Genetic Algorithms

What is an individual?



Genetic Algorithms

Basic Tasks

Generation of initial population

Evaluation

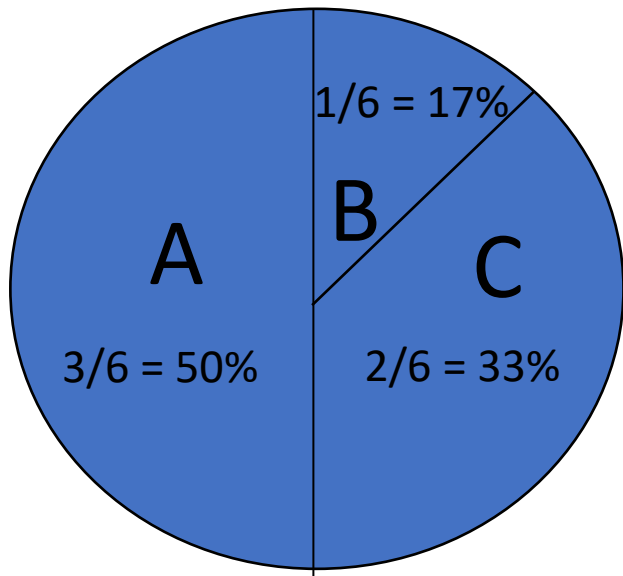
Selection (Reproduction operation)

Exchange characteristics to develop new individuals (Crossover operation)

Arbitrarily modify characteristics in new individuals (Mutation operation)

GA operators: Selection

- Main idea: better individuals get higher chance
 - Chances proportional to fitness
 - Implementation: roulette wheel technique
 - Assign to each individual a part of the roulette wheel
 - Spin the wheel n times to select n individuals

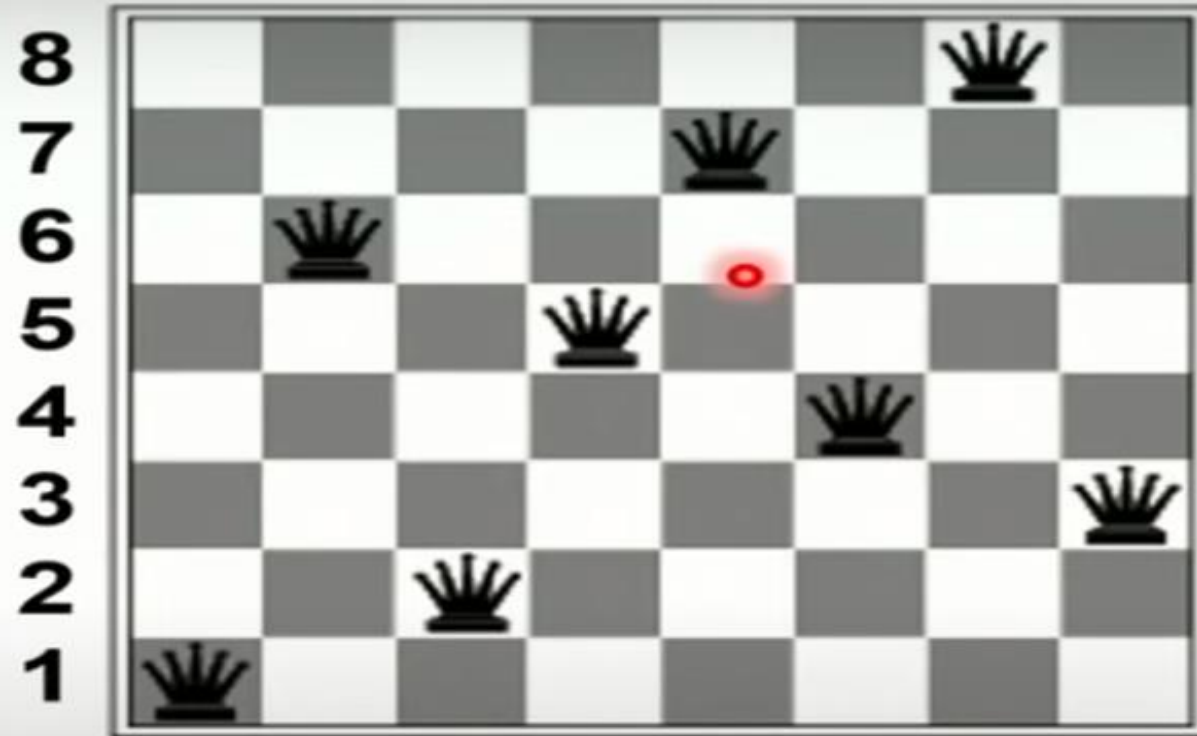


$\text{fitness}(\text{A}) = 3$

$\text{fitness}(\text{B}) = 1$

$\text{fitness}(\text{C}) = 2$

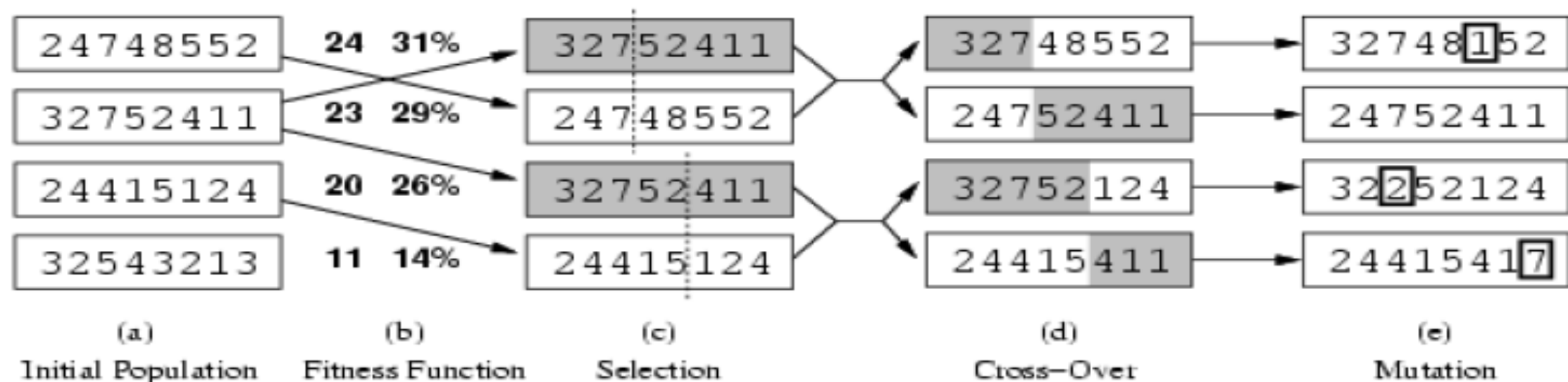
Can we solve n-queen using Genetic Algo?



String represent
16257483

Can we evolve 8-queens through genetic algorithms?

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 = $8 \times 7/2 = 28$)
- $24/(24+23+20+11) = 31\%$
- $23/(24+23+20+11) = 29\%$ etc

Genetic Algorithms

Issues

Representation

Generation of initial population

Evaluation

Reproduction operation

Crossover and Mutation operations and
feasibility issues

Some Applications

Decision making / decision support systems

Engineering component / equipment design

Engineering process optimization

Portfolio optimization

Route optimization; optimal layout; optimal packing

Schedule optimization

Protein structure analysis