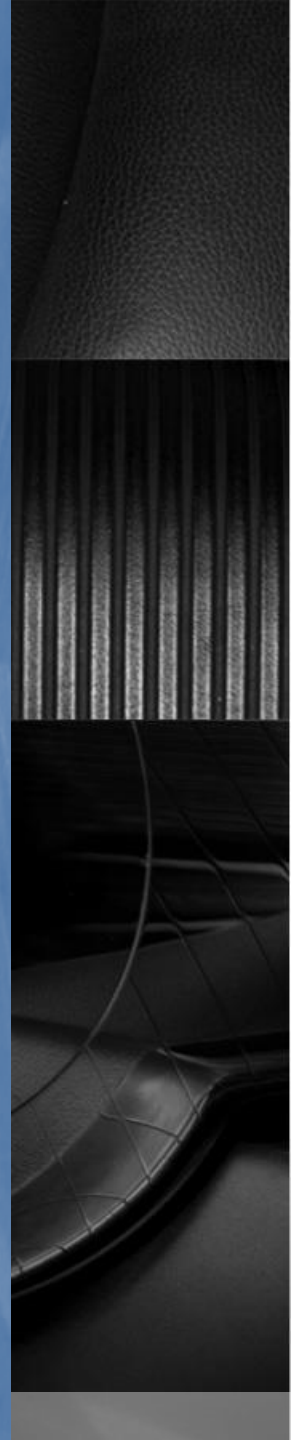
COMP4431 Artificial Intelligence

Simulated Annealing and Genetic Algorithm

Raymond Pang
Department of Computing
The Hong Kong Polytechnic University

Largely based on Mausam, Stuart Russell's Slides



Quiz Arrangement (20 Feb)

- Quiz is open book and notes
- Use of mobile phone, Internet and other electronic devices are NOT ALLOWED!!
- Quiz will be held at the end of the lecture session
- Quiz time is **15 mins**
- Lecture will end at 7:45pm, Quiz starts at **7:55pm**

Agenda

- Local search techniques and optimization
- Hill-climbing
- Simulated annealing
- Genetic algorithms

PATH Search v.s. Local Search

- The algorithms discussed so far are designed to find a goal state from a start state: the path to the goal constitutes a solution to the search problem
- In many problems the path doesn't matter: the goal state itself is the solution
- Local search algorithms:
  - at each step consider a single "current" state, and try to improve it by moving to one of its neighbors
    ➔ Iterative improvement algorithms

Optimization and Local search

- "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 is a kind of strategy that do quite well on these problems
  - Many varieties that can solve the optimization problem
- Local search
  - Keep track of single current state
  - Moves continuously in the direction of increasing objective
  - No search tree is used
- Advantages:
  - Use very little memory
  - Can often find reasonable solutions in large or infinite (continuous) state spaces.

Trivial Algorithms

- Random Sampling
  - Generate a state randomly
- Random Walk
  - Randomly pick a neighbor of the current state

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*

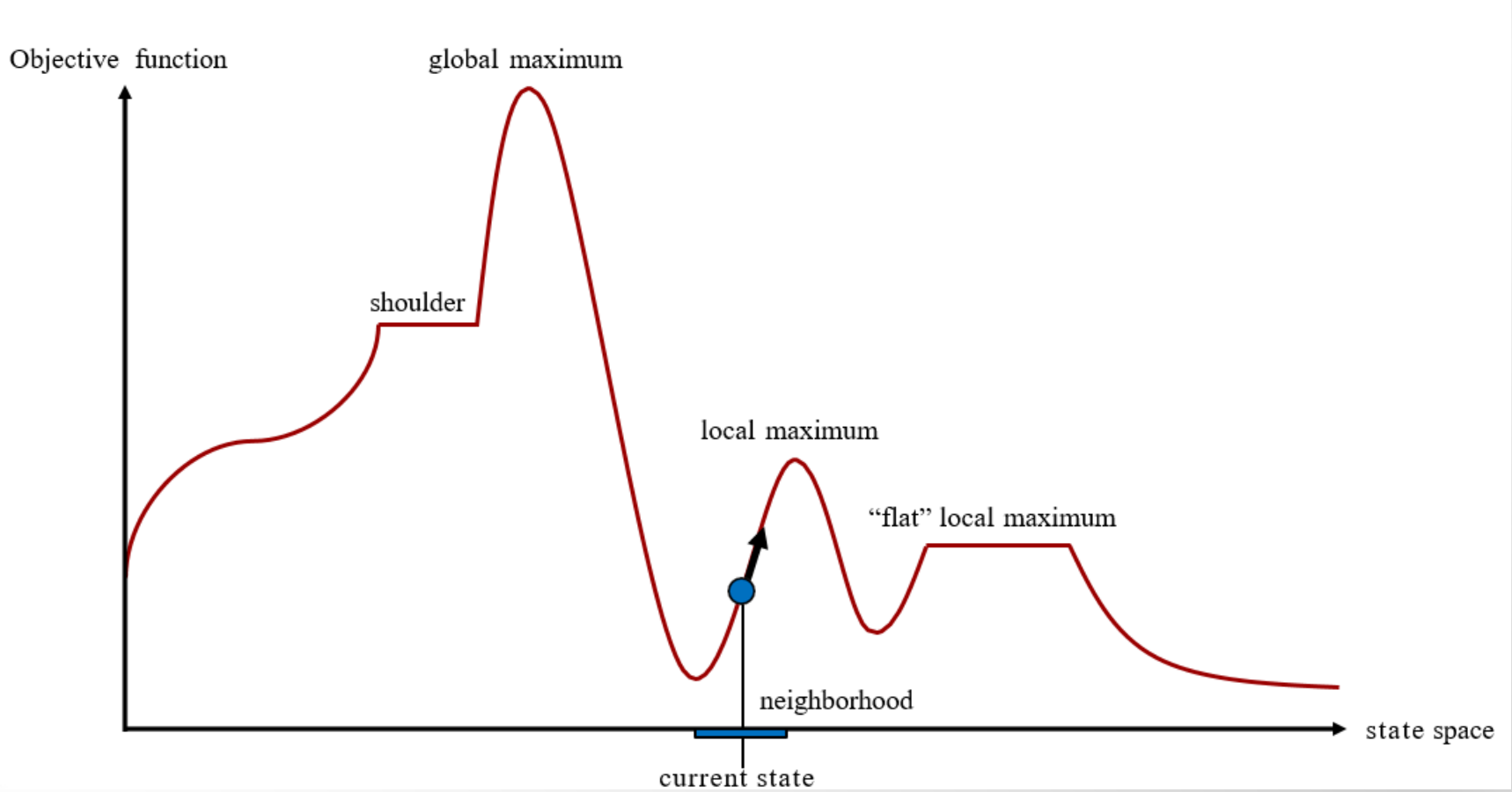
min version will reverse inequalities and look for lowest valued successor

Hill-climbing search

- “a loop that continuously moves towards increasing value”
  - terminates when a peak is reached
  - Also called 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

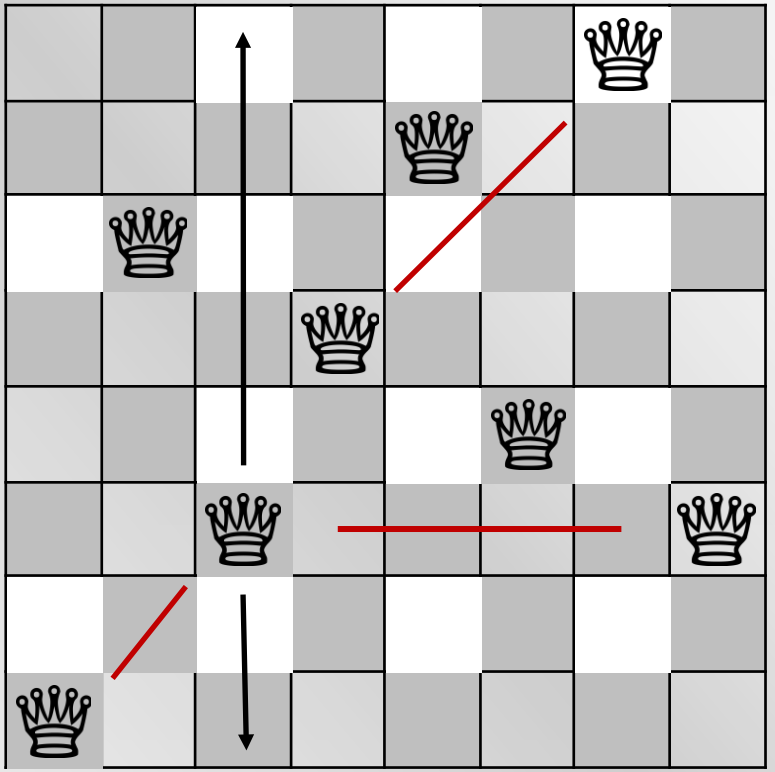
“Landscape” of search



Hill Climbing gets stuck in local minima depending on?

Example: *n*-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
  - *State:* Position of the n queens, one per column (or row)
  - *Successor states:* generated by moving a single queen to another square in its column *(n(n-1))*
  - *Cost of a state:* the number of constraint violations

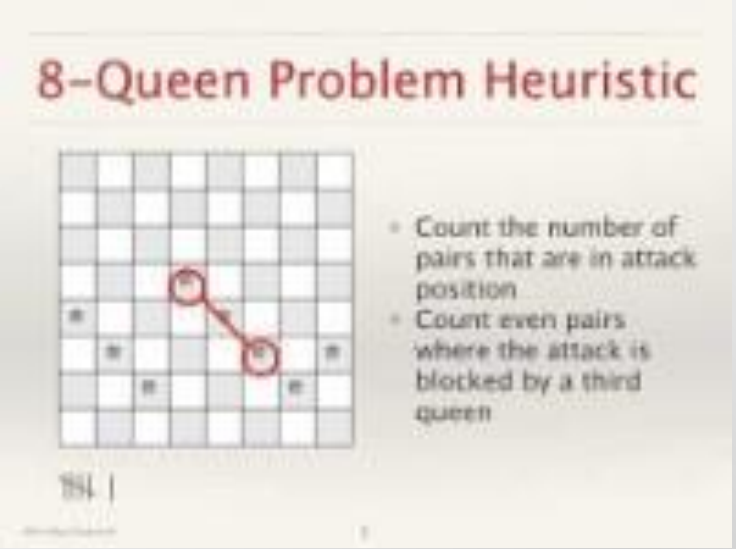


Search Space of 8-queens

- 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 search: 8-queens problem

- Need to convert to an optimization problem
- *h* = number of pairs of queens that are attacking each other
- *h = 17* for the below state



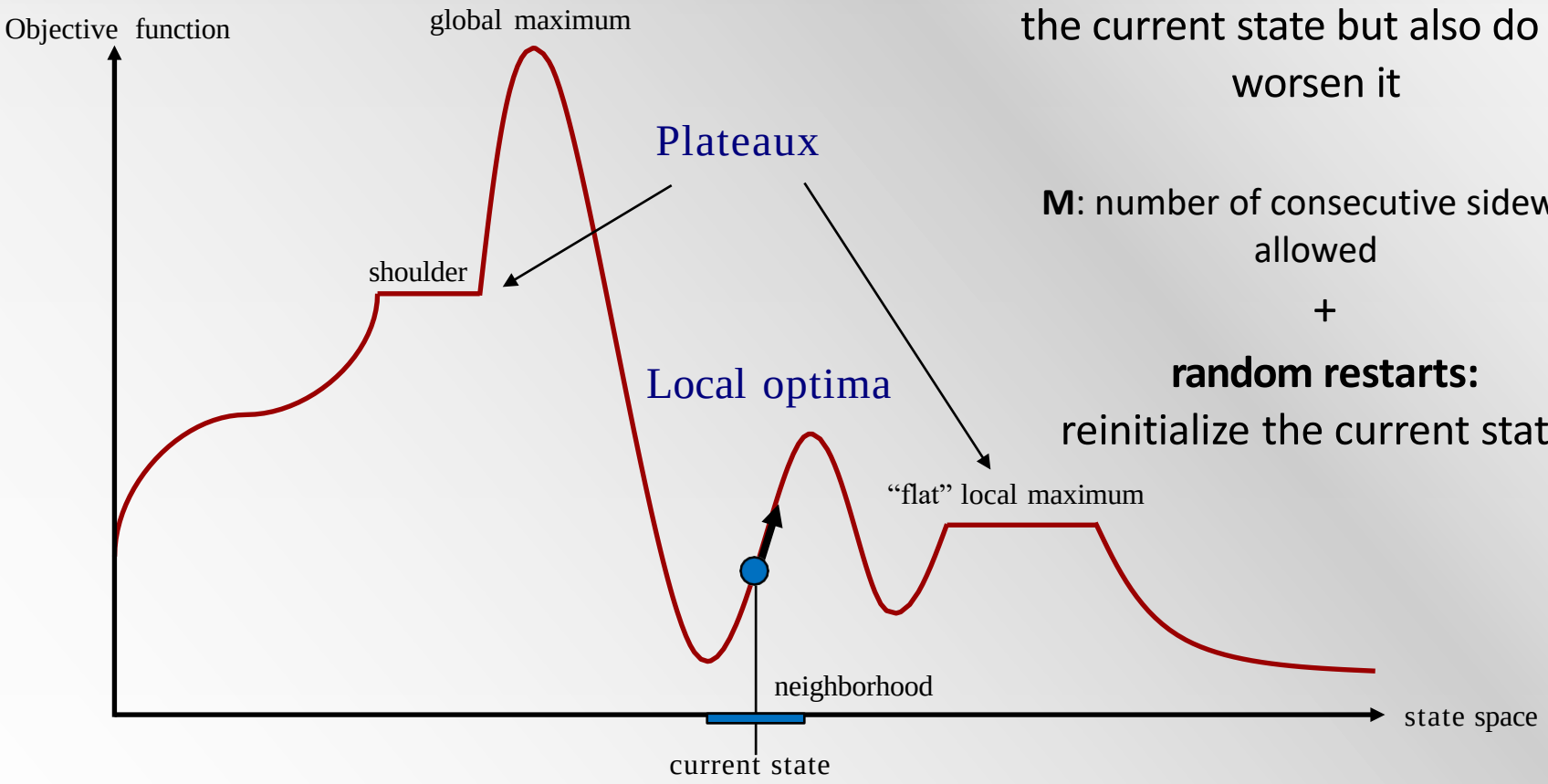
https://www.youtube.com/watch?v=qMUIvlxnoHo

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 =~17 million states)

HILL-CLIMBING CAN GET STUCK!



sideways moves (*M*):

Allow moves that do not improve the current state but also do not worsen it

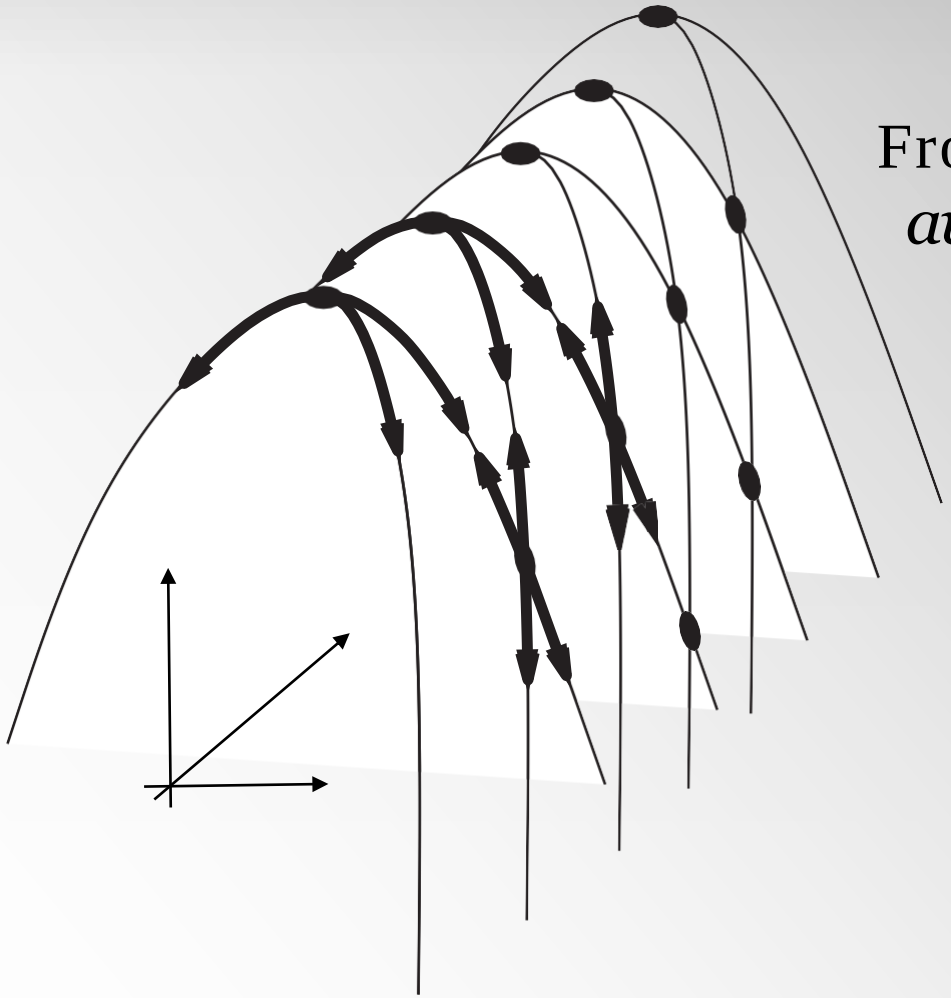
M: number of consecutive sideways allowed

+

random restarts:

reinitialize the current state

HILL-CLIMBING CAN GET STUCK!



Diagonal ridges:

From each local maximum all the *available* actions point downhill, but there is an uphill path!

Zig-zag motion,
very long ascent time!

Variants of Hill-climbing

- Sideways moves:
  - if no uphill moves, allow moving to a state with the same value as the current one (escape shoulders)
- Stochastic hill-climbing:
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move (i.e. to be “less” greedy)

Variants of Hill-climbing

To avoid getting stuck in local minima:

- Random-walk hill-climbing:
  - move to a random neighbor
  - Choose a random walk with a certain probability
- Random-restart hill-climbing
  - If at first you don't succeed, try, try again with a new start
  - Different variations

    For each restart: run until termination vs. run for a fixed time

    Run a fixed number of restarts or run indefinitely

Simulated Annealing (SA)& Genetic Algorithm (GA)

Simulated Annealing

- Simulated Annealing = physics inspired twist on random walk
- Escape from local optima by accepting, with a probability that decreases during the search, also moves that are worse than the current solution (going downhill!)
- Stochastic, solution-improvement metaheuristic for global optimization

Simulated Annealing

- 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)

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)

  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

  slowly reducing temp. T with particles moving around randomly

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 ← 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*]

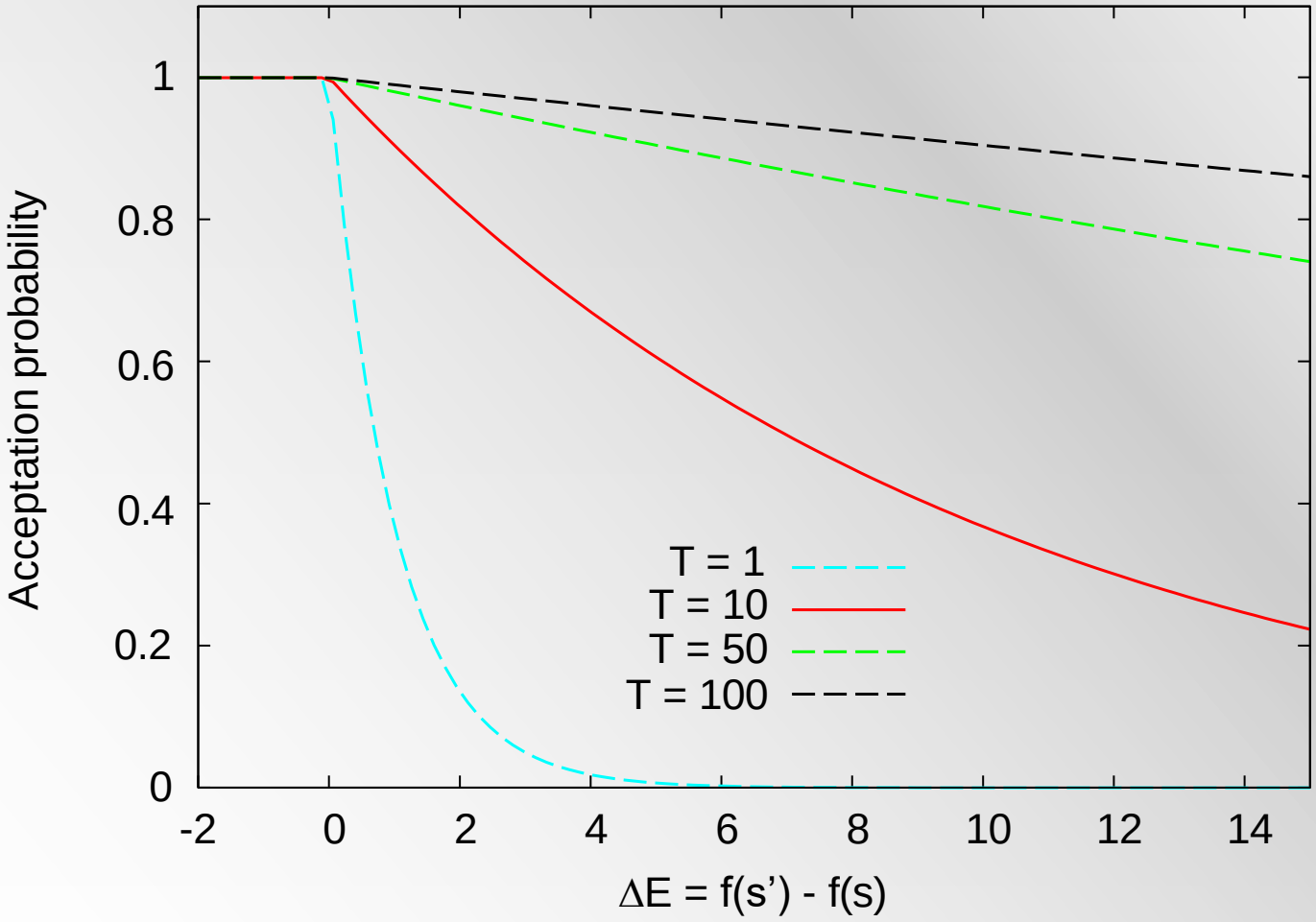
if *ΔE* > 0 **then** *current* ← *next*

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

Temperature T

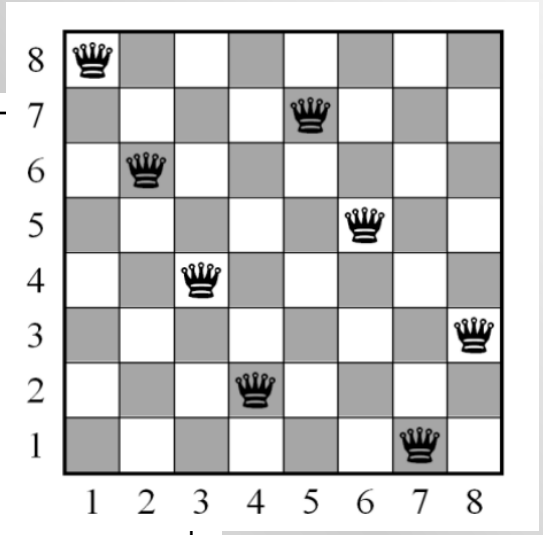
- high T: probability of “locally bad” move is higher
- low T: probability of “locally bad” move is lower
- typically, T is decreased as the algorithm runs longer
- i.e., there is a “temperature schedule”

Effect of Temperature



8-queen problem using SA

- Each state is defined with an objective function f
- randomly check which move can have better f
- Accept the move with a probability



$T = T_{max}$; /* Initializing the temperature */
$s = s_0$; /* initial solution */
Best solution = s
repeat
  Generation of a random neighbor: s'
  If $\Delta E = f(s') - f(s) \leq 0$
    Then $s = s'$; /* Accept the neighbor */
    Else Accept s' with a probability of $e^{-(\Delta E/T)}$ and set $s = s'$
  Best solution = s
  $T = g(T)$; /* Temperature update according to the cooling schedule */
Until Stopping criteria satisfied
Return Best solution

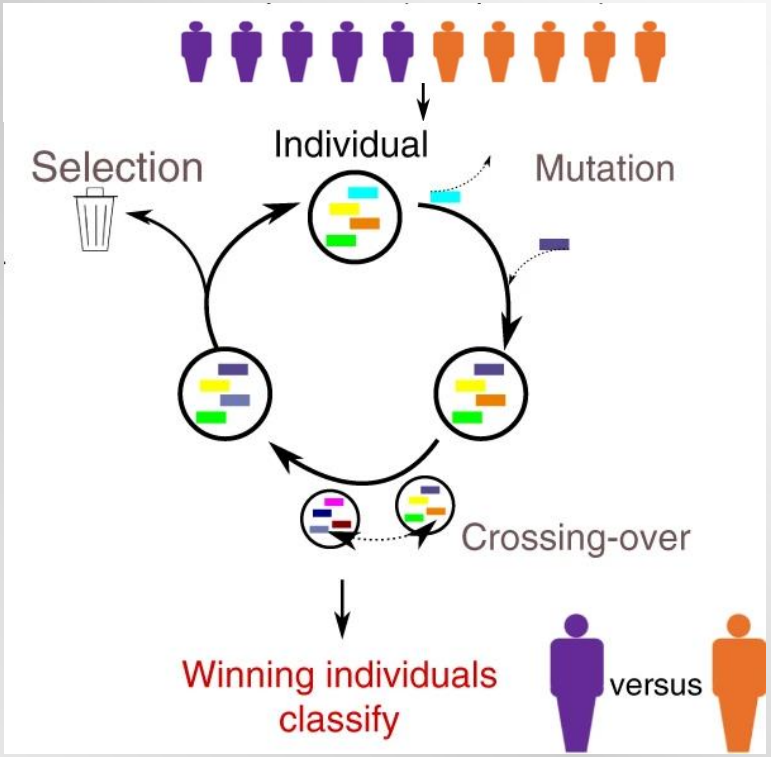
Genetic Algorithm

- A particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).
- The evolution usually starts from a population of randomly generated individuals and happens in generations.
- In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on their fitness), and modified (recombined and possibly mutated) to form a new population.

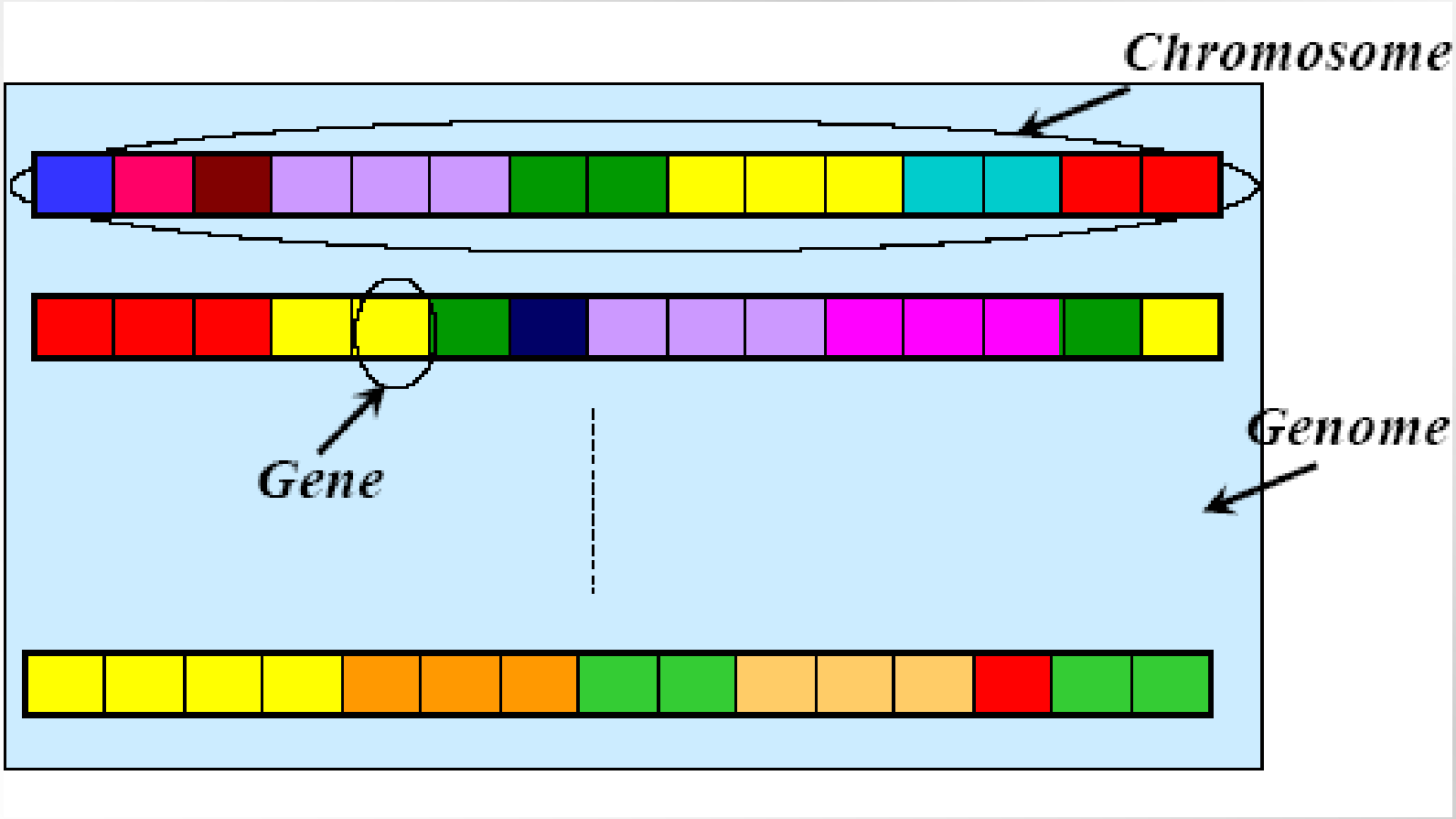
From: GECKO is a genetic algorithm to classify and explore high throughput sequencing data

Key terms

- **Individual** - Any possible solution
- **Population** - Group of all *individuals*
- **Search Space** - All possible solutions to the problem
- **Chromosome** - Blueprint for an *individual*
- **Trait** - Possible aspect (*features)* of an *individual*
- **Locus** - The position of a *gene* on the *chromosome*
- **Genome** - Collection of all *chromosomes* for an *individual*



Chromosome, Genes and Genomes



Representation

Chromosomes could be:

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of element (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)
- ... any data structure ...

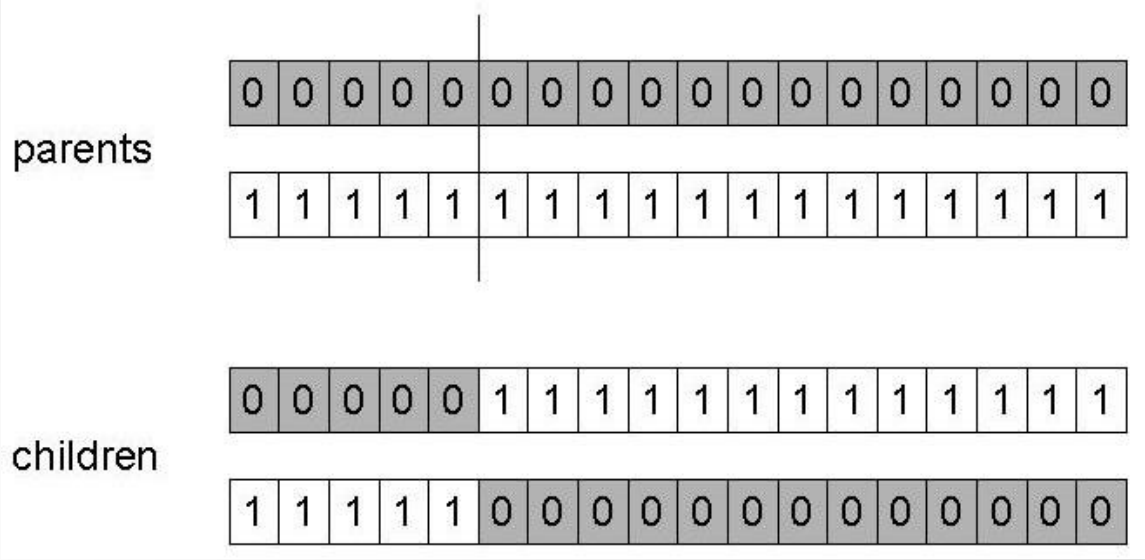
Genetic algorithms

- 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

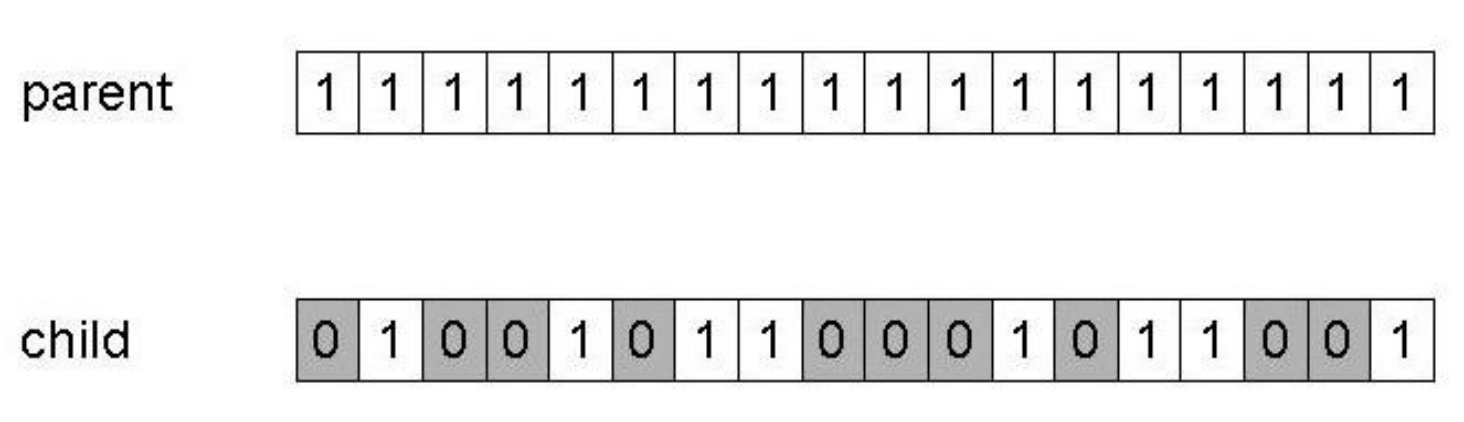
GA operators: 1-point crossover

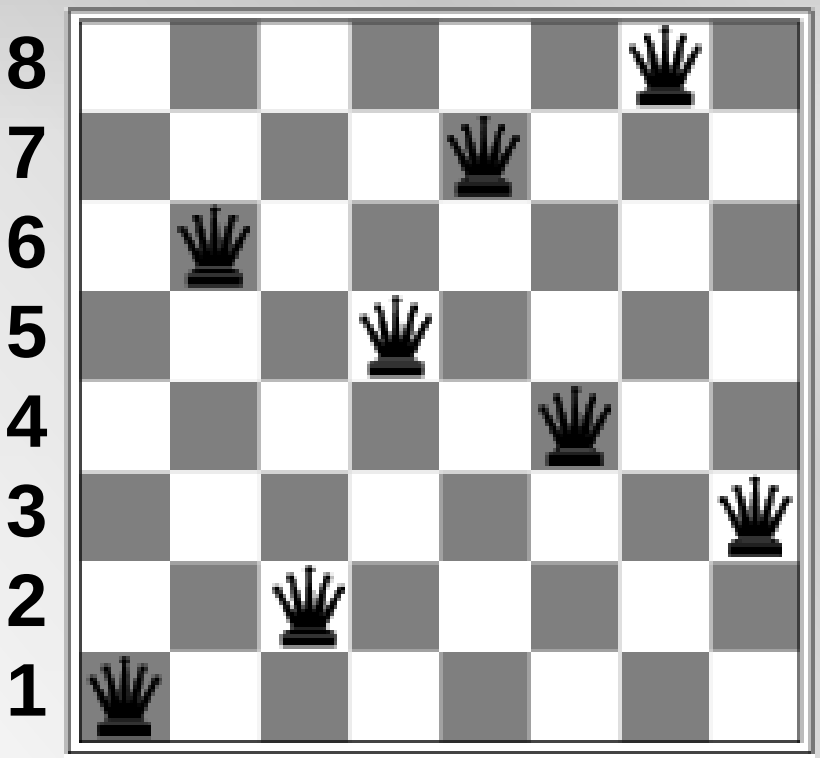
- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



GA operators: mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/ chromosome_length

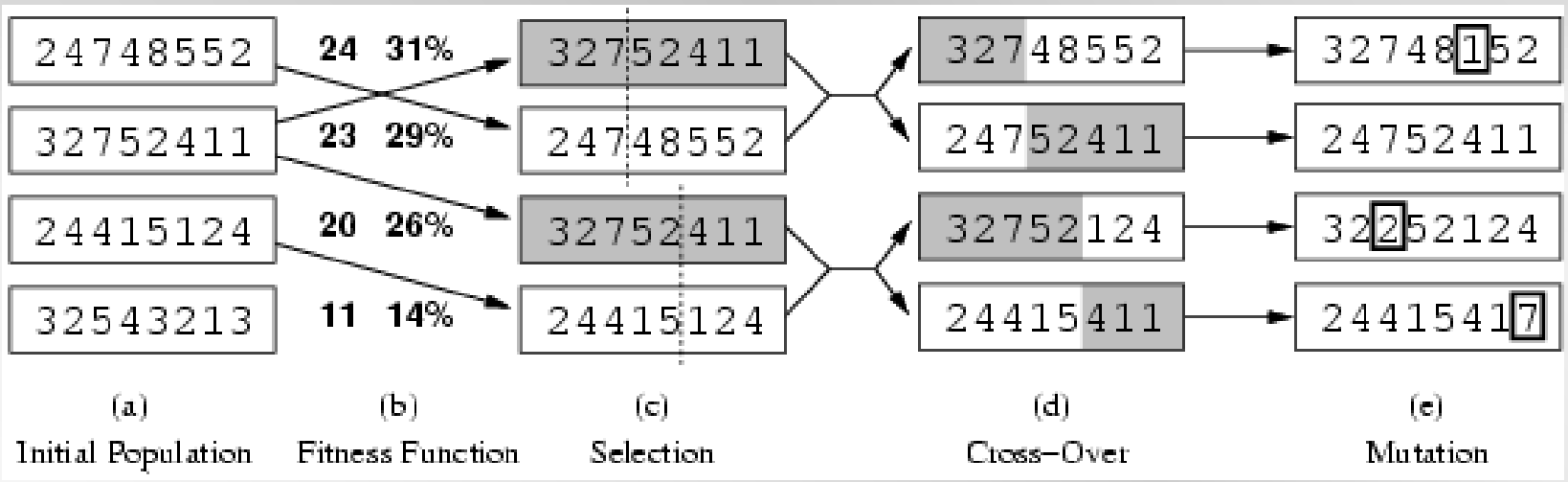




String representation
16257483

Can we evolve 8-queens through genetic algorithms?

Genetic algorithms



4 states for 8-queens problem

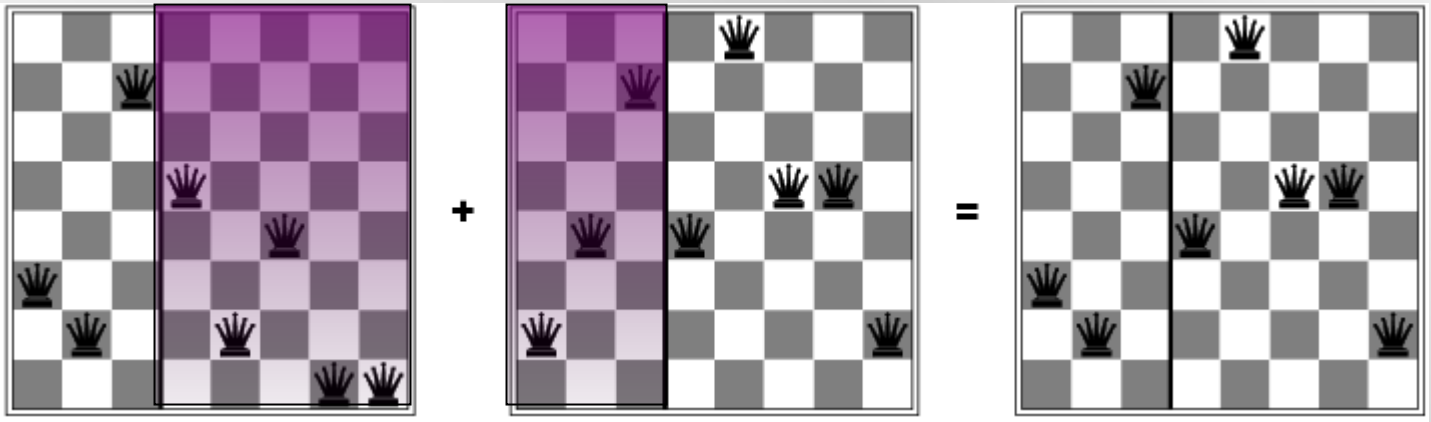
2 pairs of 2 states randomly selected based on fitness. Random crossover points selected

New states after crossover

Random mutation applied

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 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

- Pros
  - Random exploration can find solutions that local search can't

    (via crossover primarily)
  - Appealing connection to human evolution

    "neural" networks, and "genetic" algorithms are good examples!
- Cons
  - 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
- Simulated annealing
- Genetic algorithms