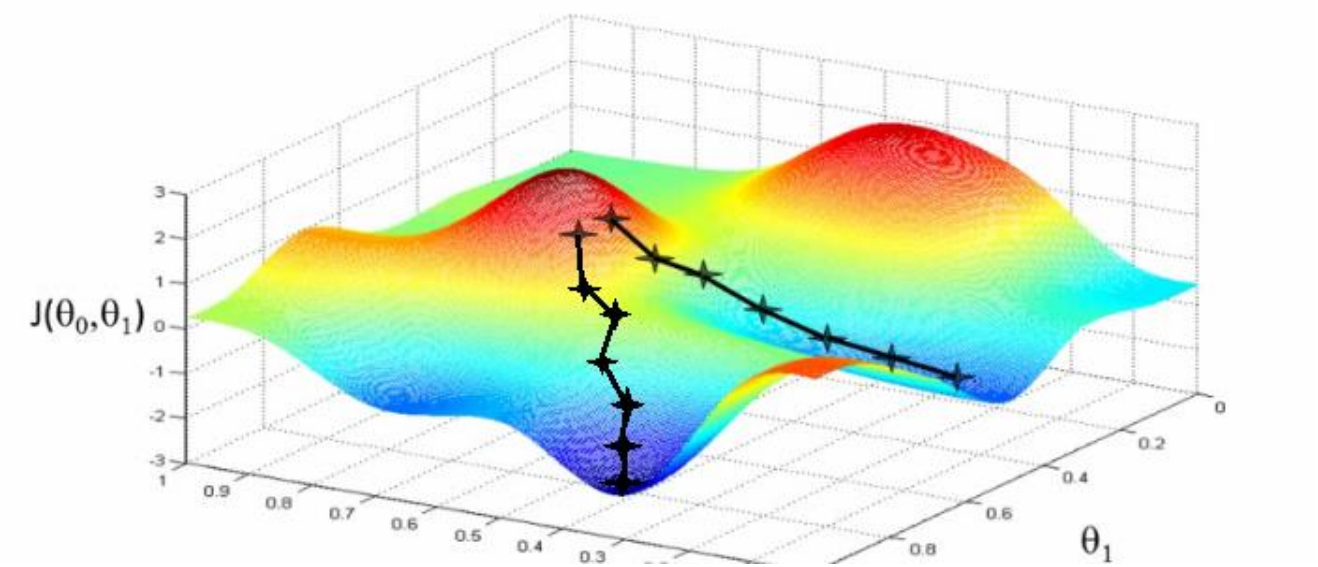
Artificial Intelligence

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

Outline

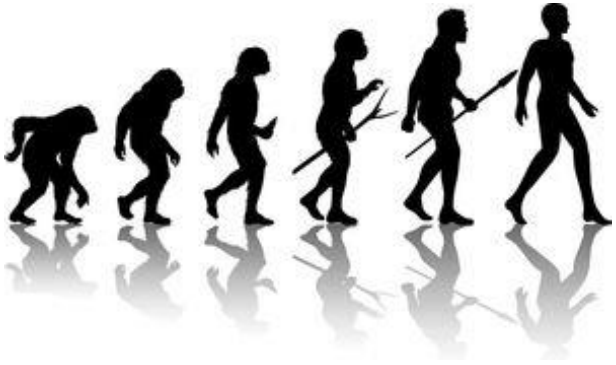
- Optimization problems
- Hill-climbing search
- Simulated annealing
- Local beam search
- Genetic algorithm

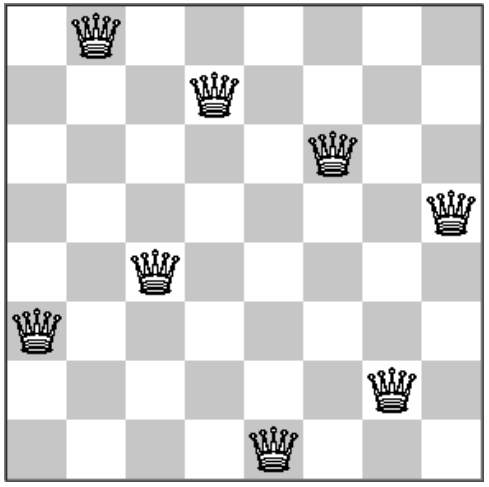


Optimization problems

Global search algorithms

- Global search: explore search spaces systematically
  - Keep one or more paths in memory and record which alternatives have been explored at each point along the path
- Many problems do not fit the “standard” search model.
  - The final configuration matters, not the order in which it is formed.
  - No “goal test” and no “path cost”, e.g., Darwinian evolution with “natural” reproductive fitness

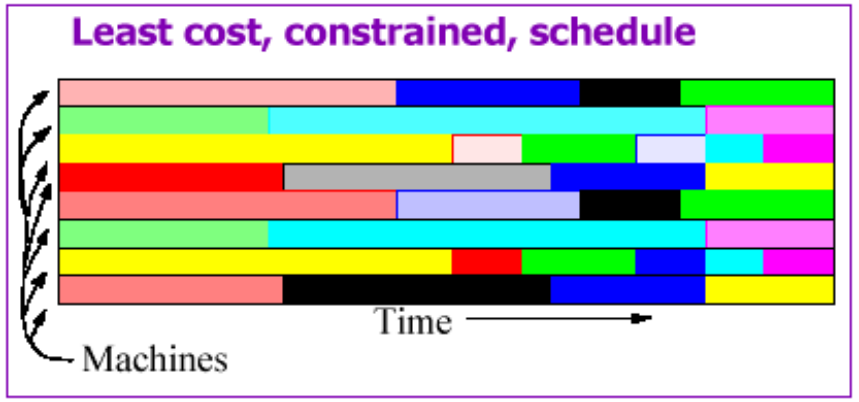


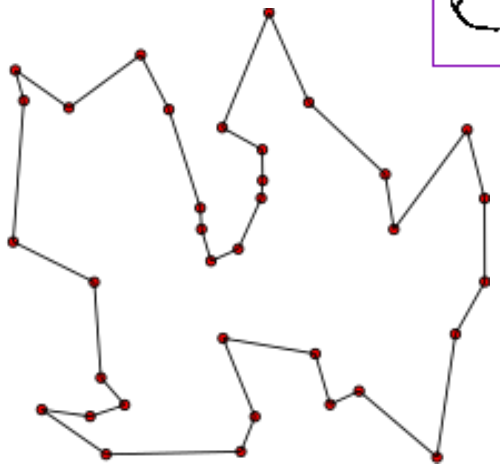


Optimization problems

- Find the best state according to an objective function.
  - E.g., the Knight's tour, TSP, scheduling, integrated-circuit design, factory-floor layout, automatic programming, vehicle routing, etc.

35	40	47	44	61	08	15	12
46	43	36	41	14	11	62	09
39	34	45	48	07	60	13	16
50	55	42	37	22	17	10	63
33	38	49	54	59	06	23	18
56	51	28	31	26	21	♞	03
29	32	53	58	05	02	19	24
52	57	30	27	20	25	04	01





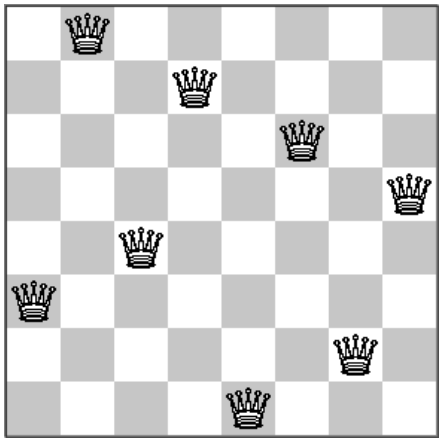
Local search algorithms

- Operate using a single current node and generally move only to neighbors of that node
- Not systematic
  - The paths followed by the search are typically not retained.
- Use very little memory, usually a constant amount
- Find reasonable solutions in large or infinite (continuous) state spaces

Global search: $n = 200$

vs.

Local search: $n = 1{,}000{,}000$



Local search and Optimization

“Pure optimization” problems

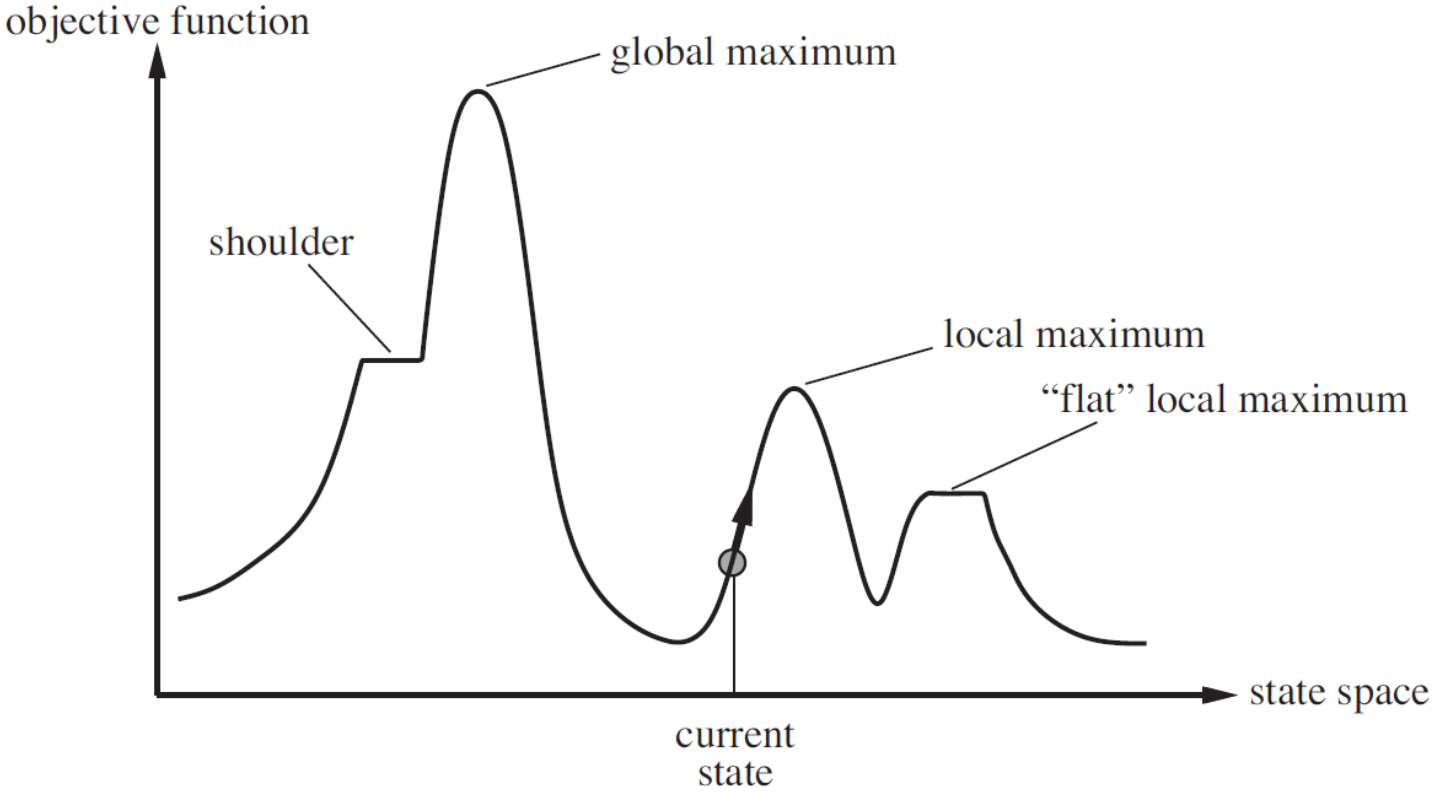
All states have an objective function

Goal is to find state with max (or min) objective value

Local search can do quite well on these problems.

State-space landscape

- A landscape has both "location" and "elevation"
  - Location ← state, elevation ← heuristic cost or objective function



A 1D state-space landscape in which elevation corresponds to the objective function.

State-space landscape

Global minimum	Global maximum
Elevation corresponds to cost	Elevation corresponds to an objective function
Find the lowest valley	Find the highest peak

- A complete local search algorithm finds a goal if one exists.
- An optimal local search algorithm finds a global extremum.

Hill-climbing search

Hill-climbing search

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

A version of HILL-CLIMBING that finds local maximum.

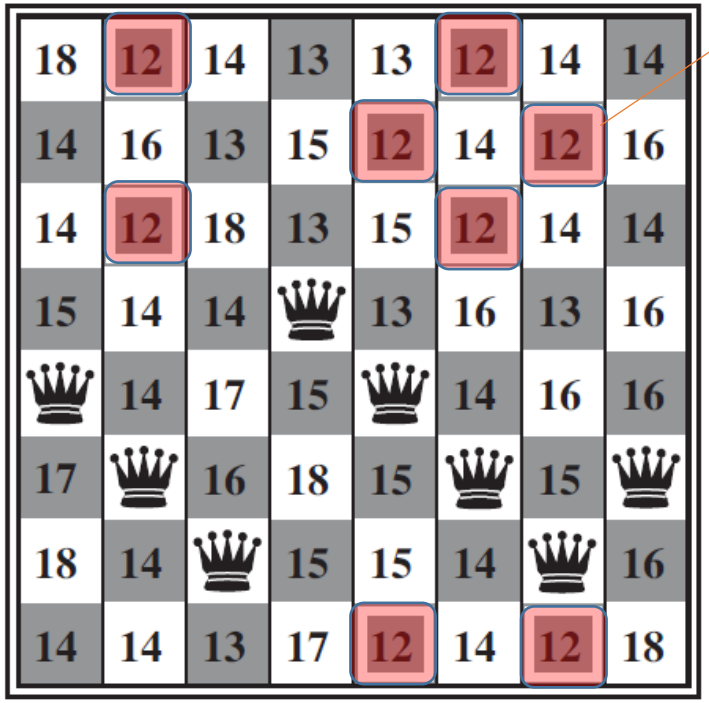
Hill-climbing search

- A loop that continually moves in the direction of increasing value and terminates when it reaches a “peak”.
  - Peaks are where no neighbor has a higher value.
- Not look ahead beyond the immediate neighbors of the current state
- No search tree maintained, only the state and the objective function’s value for the current node recorded
- Sometimes called **greedy local search**
  - Grab a good neighbor without thinking ahead about where to go next

Hill-climbing: 8-queens problem

- Complete-state formulation
  - All 8 queens on the board, one per column
- Successor function
  - Move a queen to another square in the same column $\rightarrow 8 \times 7 = 56$ successors
- Heuristic cost function $h(n)$
  - The number of pairs of queens that are **ATTACKING** each other, either directly or indirectly $\rightarrow$ global minimum has $h(n) = 0$
- Make rapid progress toward a solution
  - 8-queens ($8^8 \approx 17$ million states): 4 steps on average when succeeds and 3 when get stuck

Hill-climbing: 8-queens problem

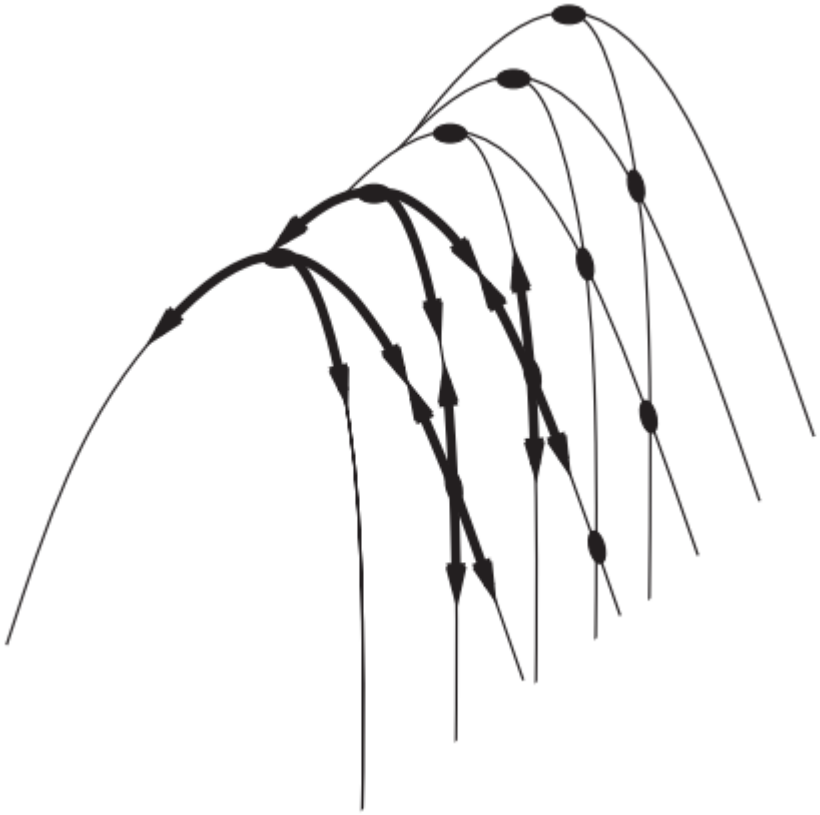


The best moves

- Current state $(c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8) = (5\ 6\ 7\ 4\ 5\ 6\ 7\ 6)$ $h(n) = 17$
- The best successors has $h = 12 \rightarrow$ choose randomly among the set of best successors if there is more than one

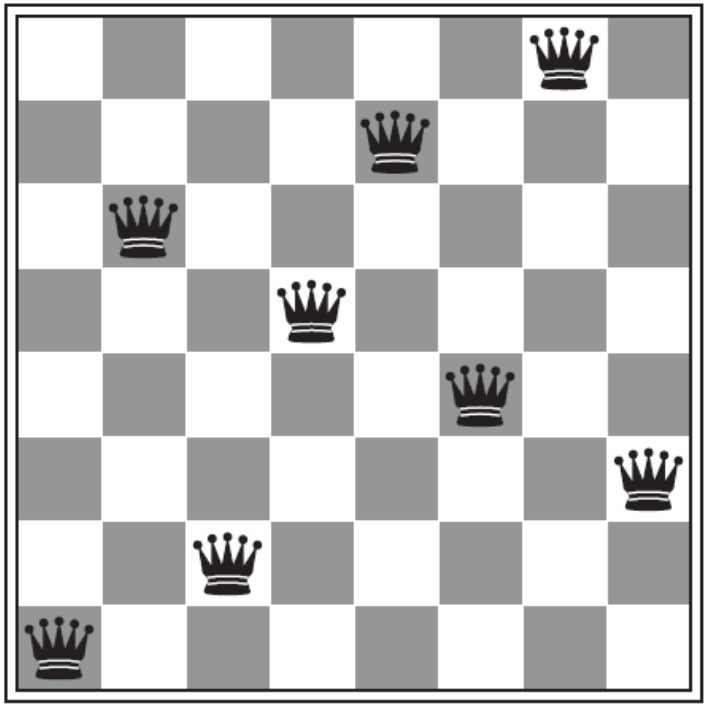
Hill climbing and local maxima

- Often **suboptimal**, due to local maxima, ridges and plateau
  - E.g., 8-queens: steepest-ascent hill climbing gets stuck 86% of the time, solving only 14% of problem instances



The grid of states (dark circles) is superimposed on a ridge rising from left to right, creating a sequence of local maxima that are not directly connected to each other. From each local maximum, all the available actions point downhill.

Hill climbing and local maxima



- Current state (8 3 7 4 2 5 1 6) $h(n) = 1$
- Every successor has a higher cost $\rightarrow$ local minimum

Solutions: Sideways moves

- If no downhill (uphill) moves, the algorithm can escape with **sideways moves**.
  - A limit on the possible number of sideways moves required to avoid infinite loops
- For example, 8-queens problem
  - Allow sideways moves with a limit of 100 $\rightarrow$ percentage of problem instances solved raises from 14 to 94%
  - Success comes at a cost: the algorithm averages roughly 21 steps for each successful instance and 64 for each failure.

Solutions: Hill-climbing variants

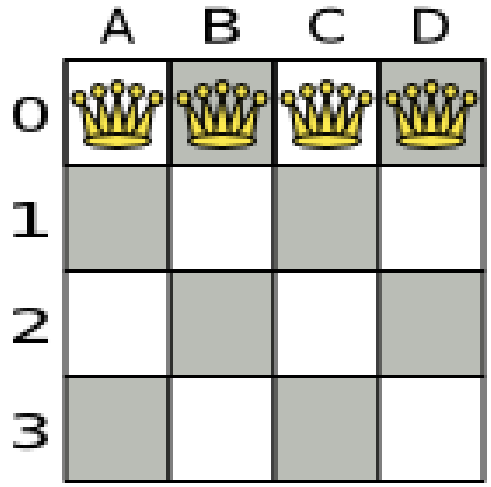
- **Stochastic hill climbing**
  - Choose at random from among the uphill moves with a probability of selection varied with the moves' steepness
  - Usually converge more slowly than steepest ascent, but find better solutions in some cases
- **First-choice hill climbing**
  - Generate successors randomly until one is generated that is better than the current state
  - Good strategy when a state has many successors (e.g., thousands)

Solutions: Random-restart hill climbing

- **Random-restart hill climbing** can find a good solution quickly after a small number of restarts.
- That is a series of hill-climbing searches from randomly generated initial states until a goal is found.
  - For each restart: run until termination vs run for a fixed time
  - Run a fixed number of restarts or run indefinitely
- If each search has a probability p of success, the expected number of restarts required is $1/p$
- Very effective indeed for 8-queens

Quiz 01: 4-queens problem

- Consider the following 4-queen problem

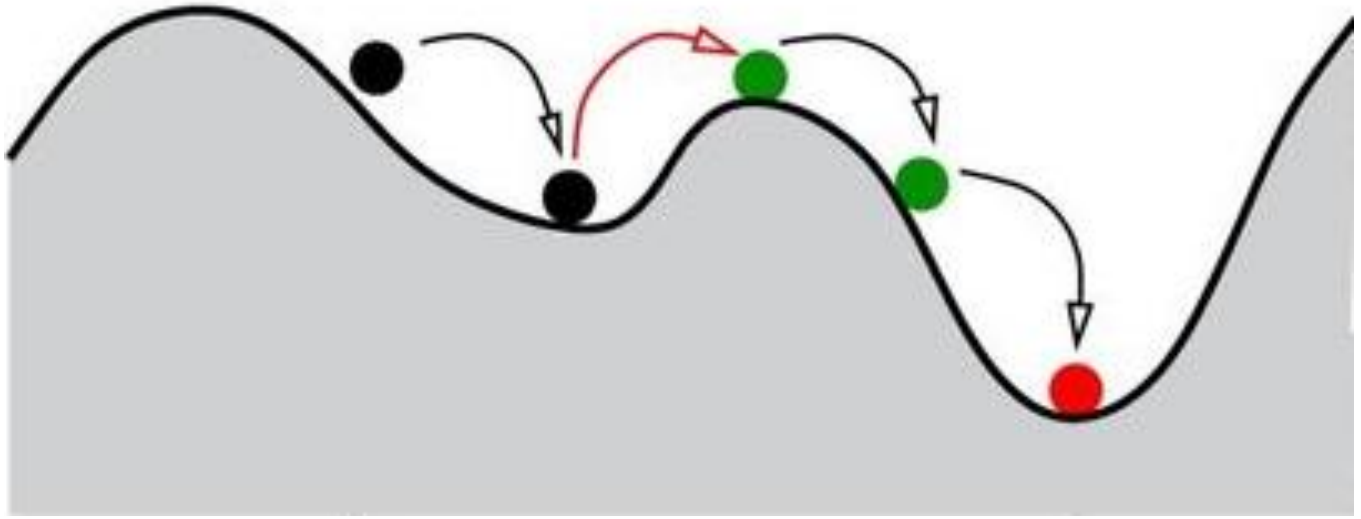


- Apply (first-choice) hill-climbing to find a solution, using the heuristic "*The number of pairs of queens ATTACKING each other.*"

Simulated annealing

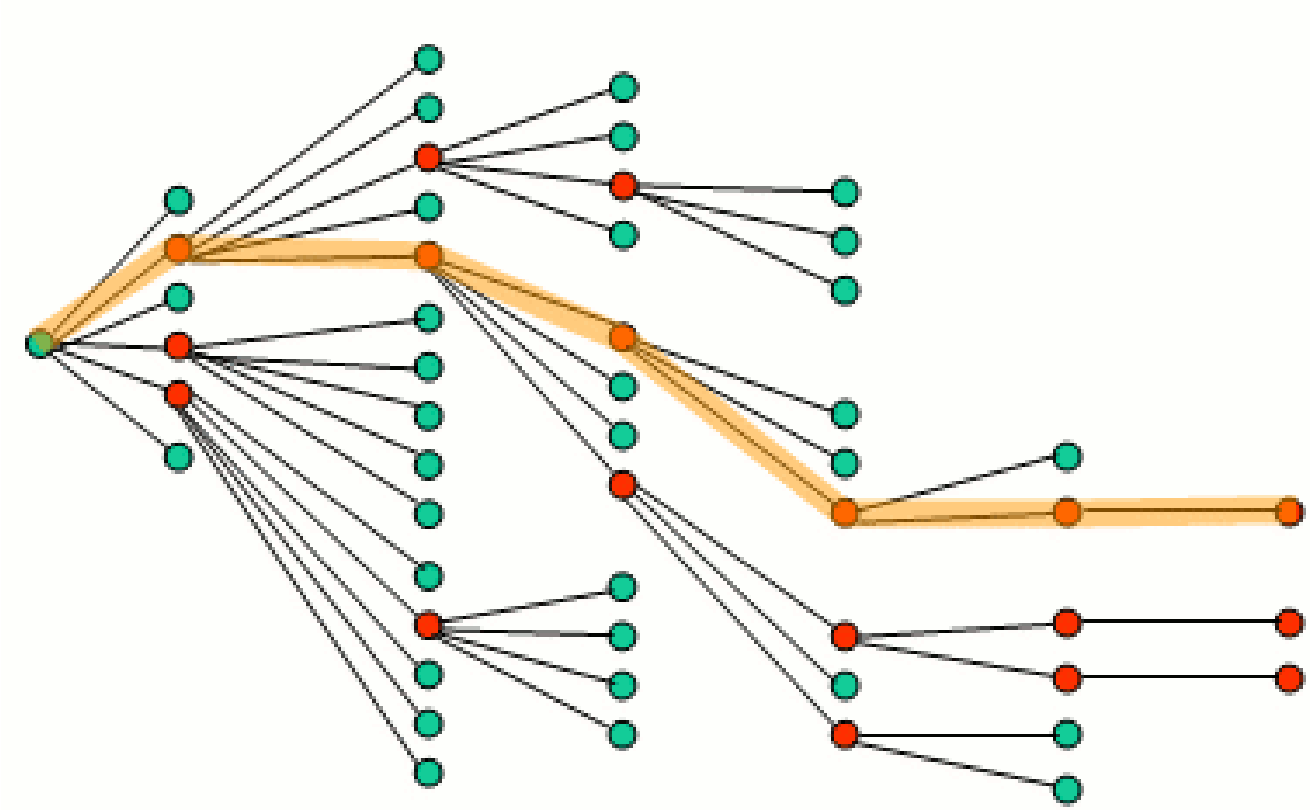
Simulated annealing

- Combine hill climbing with a random walk in some way that yields both efficiency and completeness
- Shake hard (i.e., at a high temperature) and then gradually reduce the intensity of shaking (i.e., lower the temperature)



Simulated annealing

function SIMULATED-ANNEALING(*problem, schedule*)
  returns a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** current
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ *next*.VALUE – current.VALUE
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$



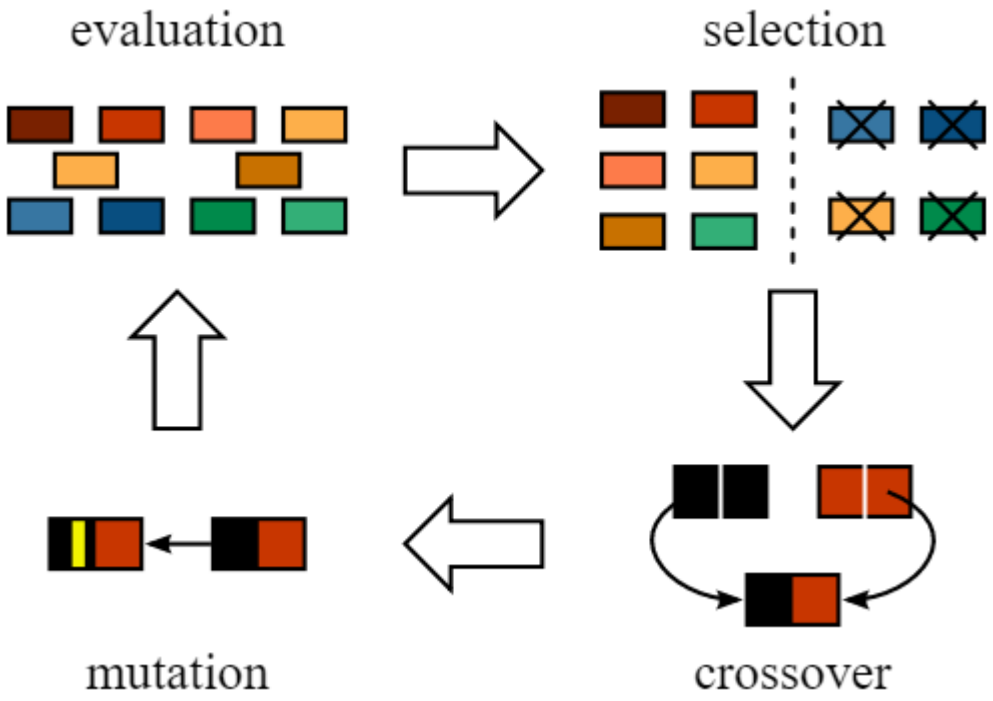
Local beam search

Local beam search

- Keep track of k states rather than just one
- Begin with k randomly generated states
- At each step, all successors of all k states are generated
- If the goal is not found, select the k best successors from the complete list and repeat

Local beam search

- Useful information is passed among the parallel search threads $\rightarrow$ major difference from random-restart search
- Possibly suffer from a lack of diversity among the k states
  - Quickly concentrate in a small region of the state space $\rightarrow$ an expensive version of hill climbing
- **Stochastic beam search**
  - Choose k successors at random, with the probability of choosing a given successor being an increasing function of its value

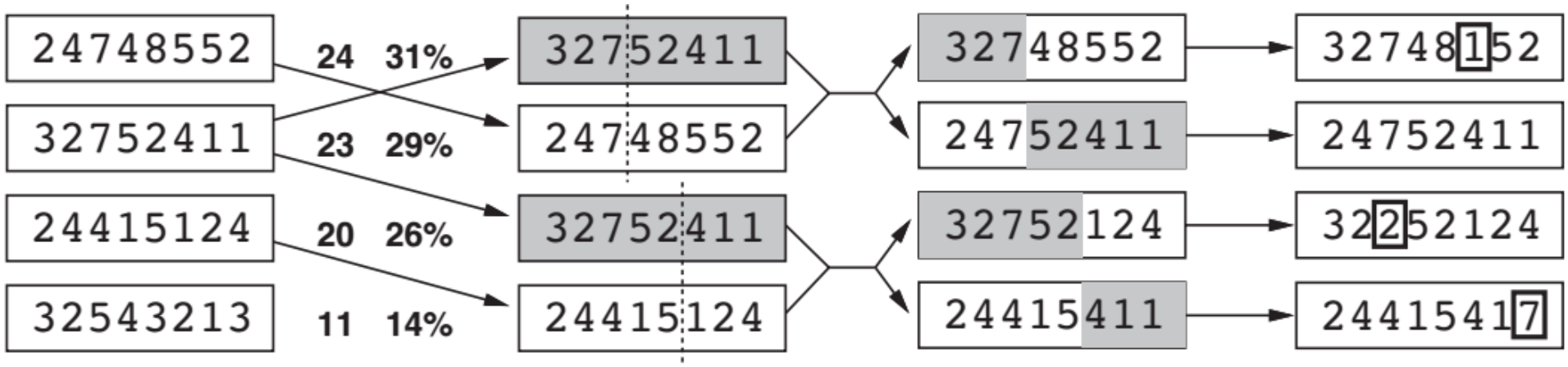


Genetic algorithms

Genetic algorithms

- A variant of stochastic beam search
- Successor states are generated by combining two parent states rather than by modifying a single state
- The reproduction are sexual rather than asexual

Genetic algorithms: 8-queens



4 states for 8-queens problem

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

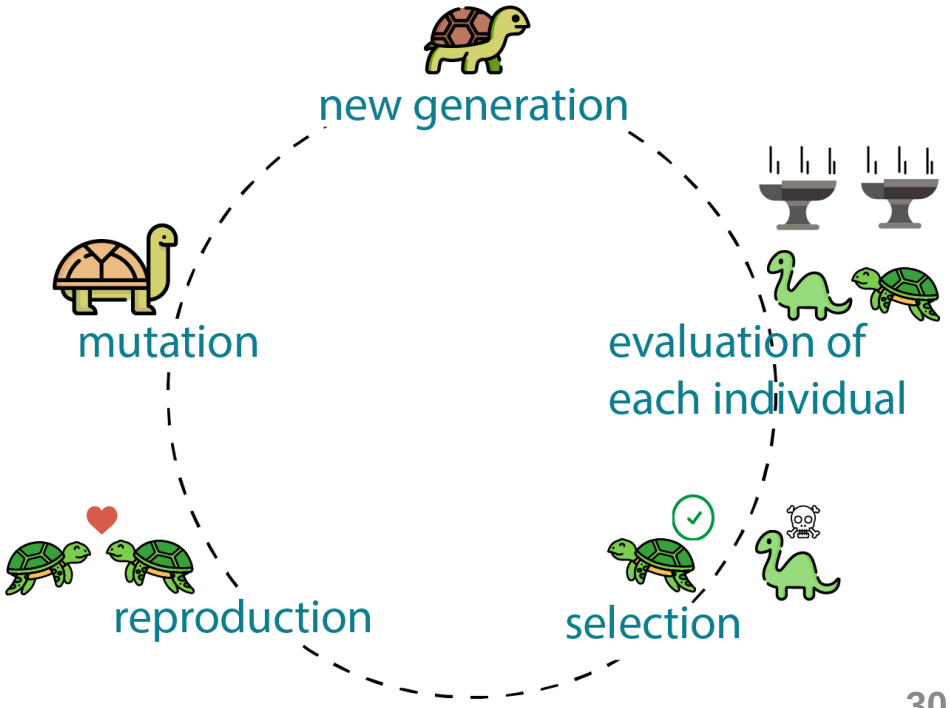
New states after crossover

Random mutation applied

Genetic algorithms

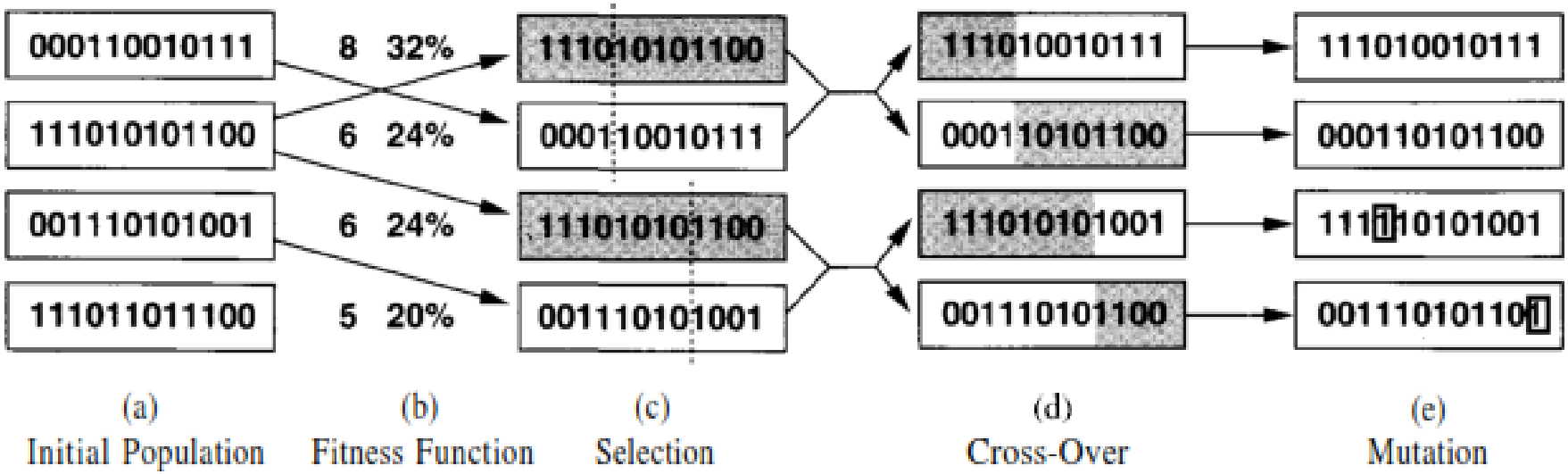
- Population: a set of k randomly generated states to begin with
- Fitness function: an objective function that rates each state
  - Higher values for better state
  - E.g., 8-queens: *the number of nonattacking pairs of queens* (min = 0, max = 8×7/2 = 28)

- Produce the next generati
  by "simulated evolution"
  - Random selection
  - Crossover
  - Random mutation



Representation of Individuals

- Each state, or individual, is represented as a string over a finite alphabet – most commonly, a string of 0s and 1s.
  - E.g., 8-queens: $8 \times \log_2 8 = 24$ bits

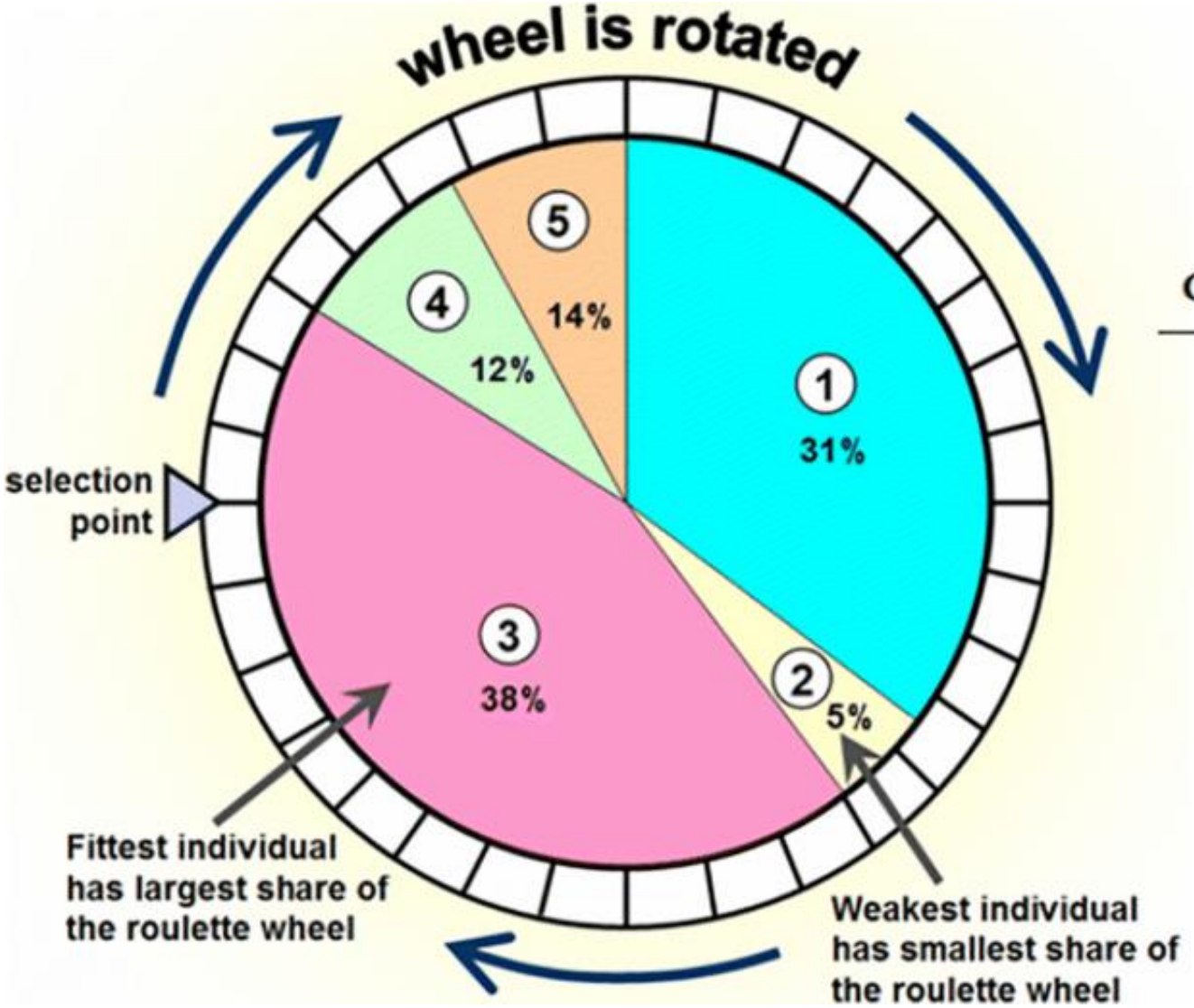


- Alternatively, the state could be represented as 8 digits, each in the range from 1 to 8.

Reproduction

- Pairs are selected at random for reproduction.
- The probability of being chosen for reproducing is directly proportional to the fitness score.
  - E.g., 24/(24+23+20+11) = 31%, 23/(24+23+20+11) = 29%, etc.
- One individual may be selected several times or not at all.

Reproduction: Roulette wheel



Chromosome	Fitness value
C1	31%
C2	5%
C3	38%
C4	12%
C5	14%

Quiz 02: Calculate fitness scores

- The current generation include 4 states, S1, S2, S3, and S4.
- Their evaluation functions' values are:

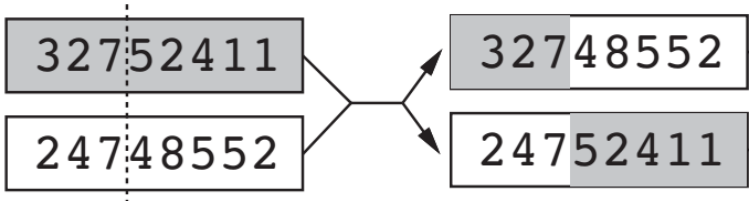
  Eval(S1) = 7, Eval(S2) = 15,

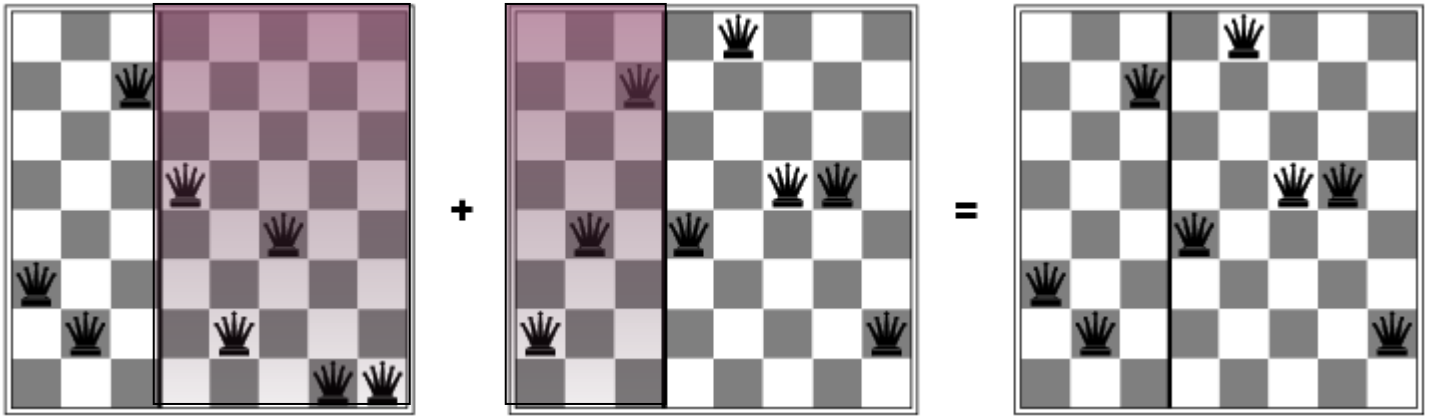
  Eval(S3) = 10, Eval(S4) = 18

- Calculate the probability (the fitness that) each of them will be chosen in the "selection" step.
- Which two of four states have the highest possibilities to be selected for the next generation?

Crossover operation

- For each pair to be mated, a crossover point is chosen randomly from the positions in the string.

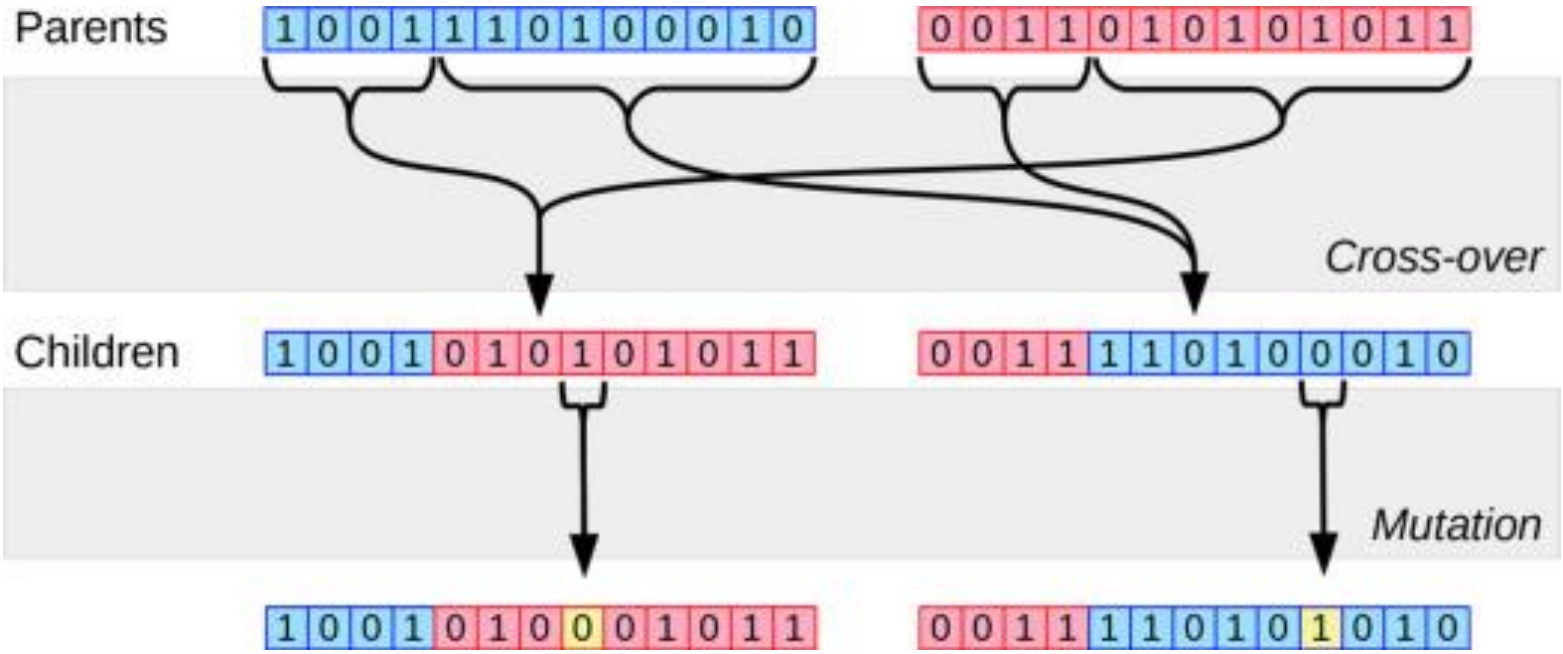




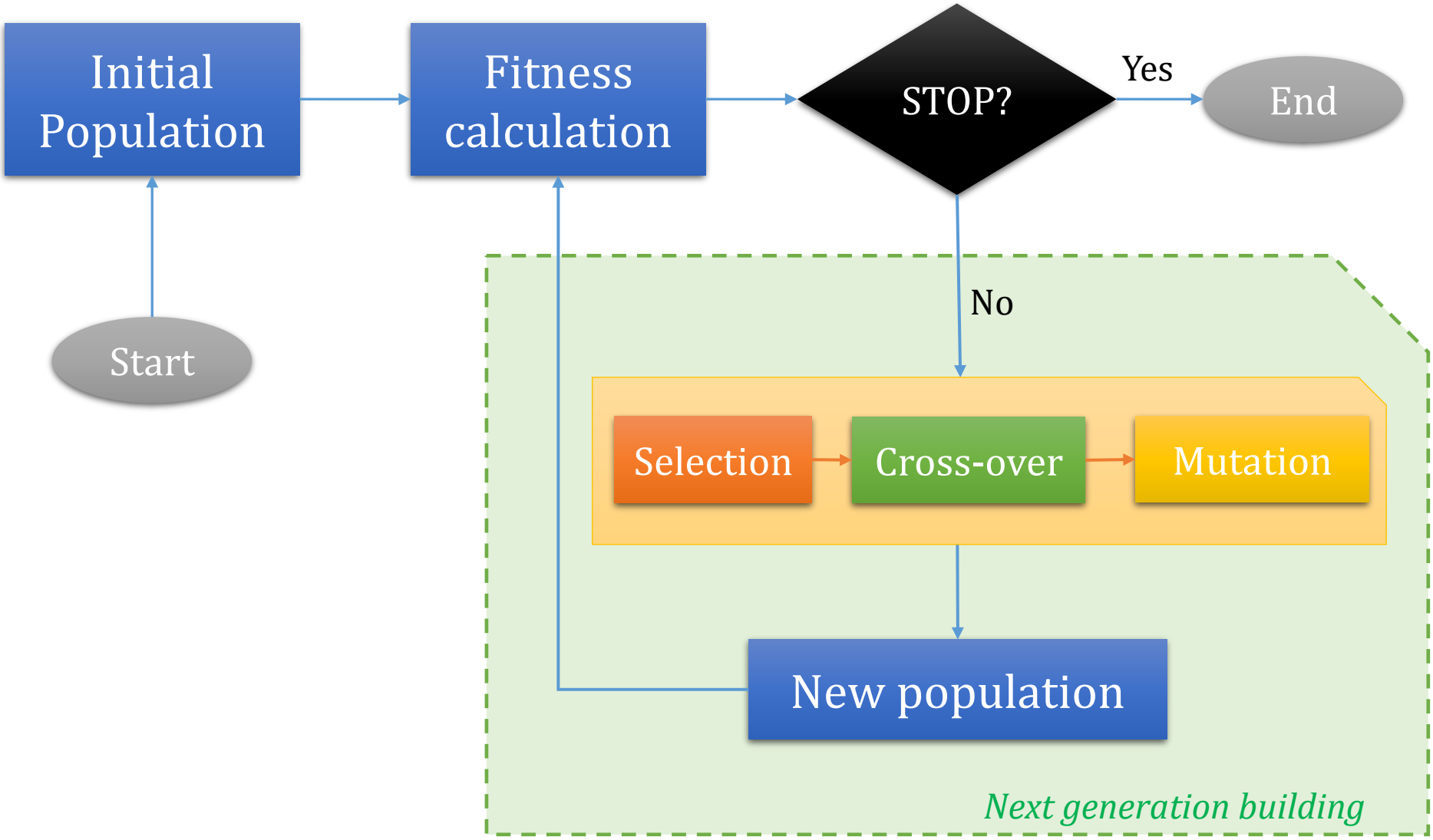
- Effectively “jump” to a completely different new part of the search space (quite non-local)

Mutation operation

- Each location is subject to random mutation with a small independent probability.
  - E.g., 8-queens: choosing a queen at random and moving it to a random square in its column



A workflow of Genetic algorithms



Genetic algorithms

function GENETIC-ALGORITHM(*population*, FITNESS-FN)
returns an individual
inputs: *population*, a set of individuals
FITNESS-FN, a function that measures the fitness of an individual
repeat
new_population ← empty set
for i = 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 in *population*, according to FITNESS-FN

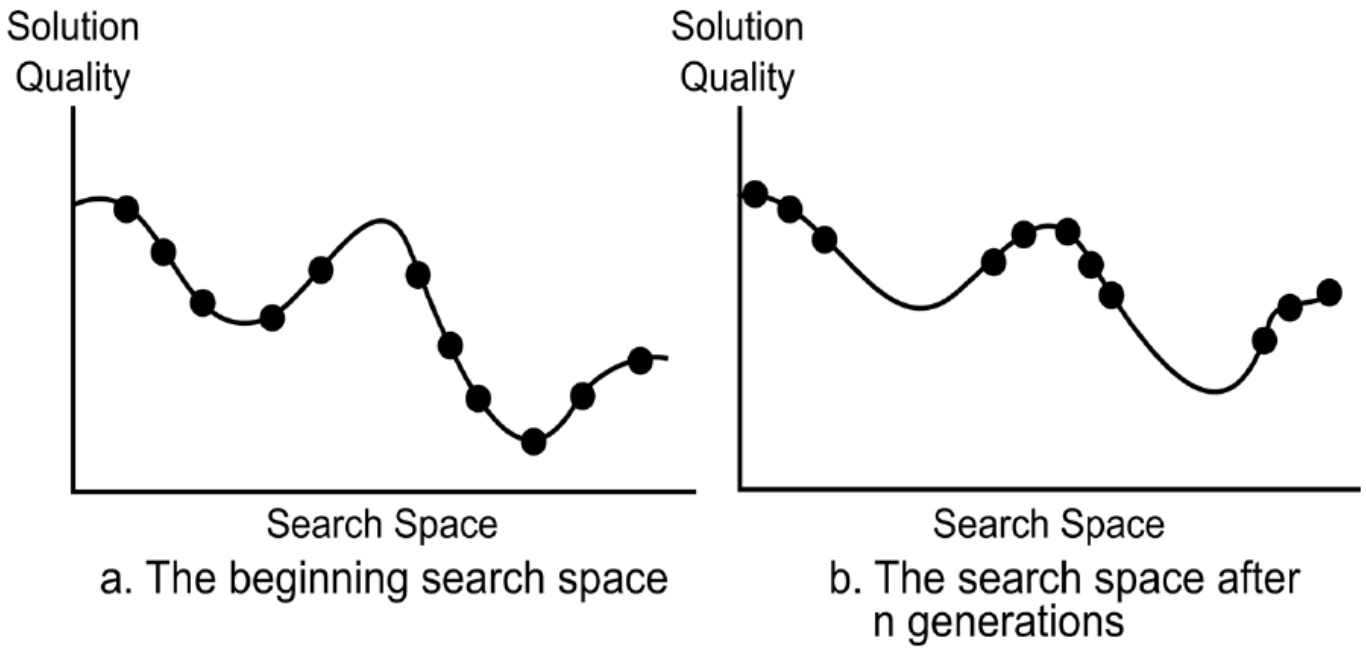
Genetic algorithms

function REPRODUCE(x , y) **returns** an individual

inputs: x , y, parent individuals

$n \leftarrow$ LENGTH(x); $c \leftarrow$ random number from 1 to n

return APPEND(SUBSTRING(x , 1, c), SUBSTRING(y, c + 1, n))



a. The beginning search space

b. The search space after n generations

Comments on Genetic algorithms

- Random exploration find solutions that local search does not
  - Via crossover primarily
  - Can solve “hard” problem
- Rely on very little domain knowledge
- Appealing connection to human evolution

Comments on Genetic algorithms

- 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?) sets of problem, yet no convincing evidence that GAs are better than hill-climbing w/random restarts in general.
- Require careful engineering of the representation

- Application: Genetic Programming!

THE END