Last class

- Greedy best-first search
- A* search
- Recursive best-first search
- Heuristic generation
- Heuristic goodness

Informed (heuristic) search

Uses problem-specific knowledge beyond the problem definition

NJUAI 南京大学 人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE, NANJING UNIVERSITY

南京大學

Heuristic Search and Evolutionary Algorithms

Lecture 4: Local Search and Evolutionary Algorithms

Chao Qian （钱超）

Associate Professor, Nanjing University, China

Email: qianc@nju.edu.cn
Homepage: http://www.lamda.nju.edu.cn/qianc/

Classical search

A search problem can be defined formally by five components:

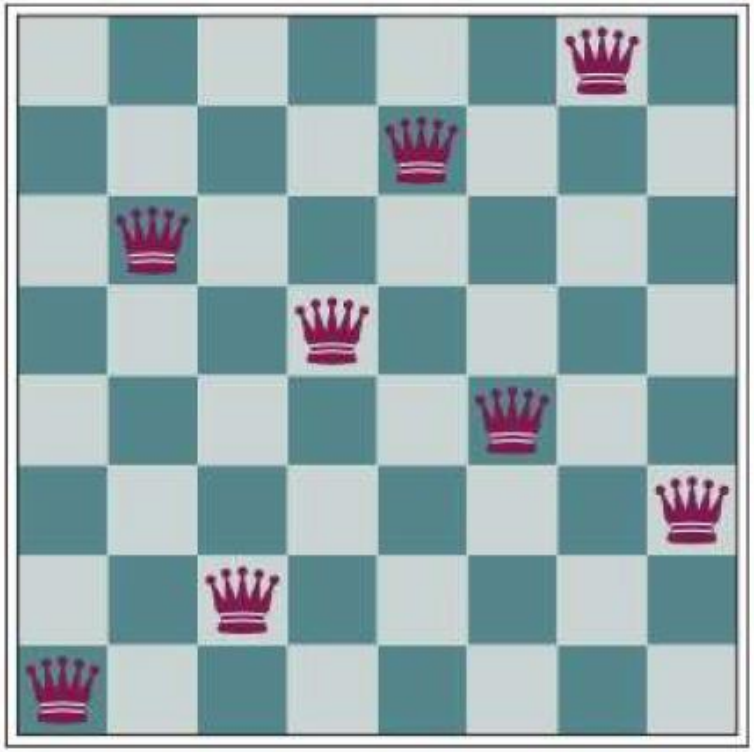
- Initial state
- Actions
- Transition model
- Goal test
- Path cost

Solution: a path (i.e., an action sequence) from the initial state to a goal state

Optimal solution: a path with the lowest cost

Search example: Path is irrelevant

8-queens problem: to place eight queens on a chessboard such that no queen attacks any other



Heuristic function h: number of pairs of queens that are attacking each other

What is a goal state, i.e., a state with $h = 0$?

The path to the goal state is irrelevant

Search and optimization

General Search: to find a goal state, i.e., a state with $h = 0$

Optimization: to find an optimal solution

$$\arg\min_x h(x) \quad \text{or} \quad \arg\max_x f(x)$$

Note that: classical search can be transformed into this form by treating an action sequence as a solution and the cost as the objective to be minimized

Hill-climbing search

Hill-climbing search: maintain only the current state

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  current ← *problem*.INITIAL
  while *true* **do**
    neighbor ← a highest-valued successor state of *current*
    if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
    current ← *neighbor*

Select the best neighbor state

Stop until no neighbor has a higher objective value

Need to define a neighbor space

Hill-climbing search – example

8-queens problem: to place eight queens on a chessboard such that no queen attacks any other

Heuristic function *h*: number of pairs of queens that are attacking each other

The current *h* value: 17

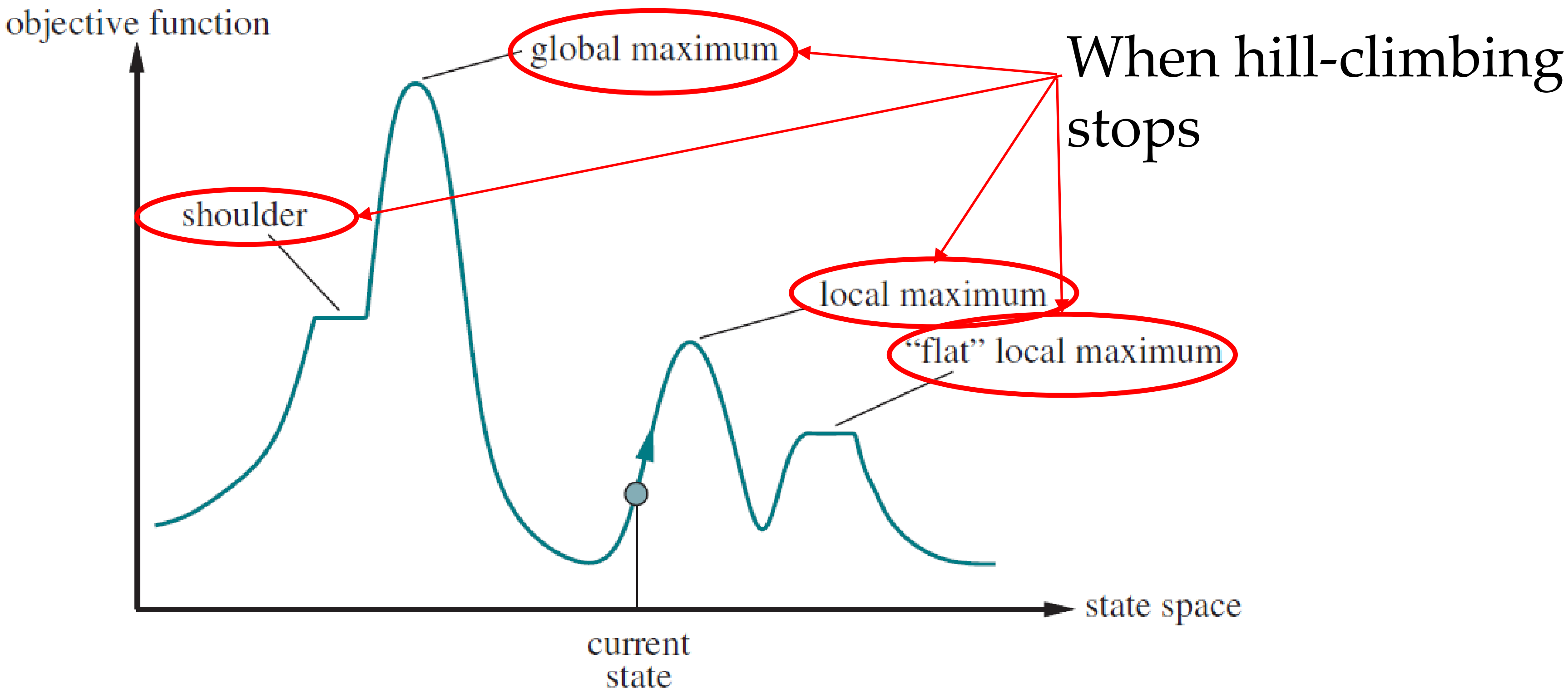
Neighbor space: states generated by moving a single queen to another square in the same column

The number of neighbors: 56

Move to the best neighbor with *h* value 12

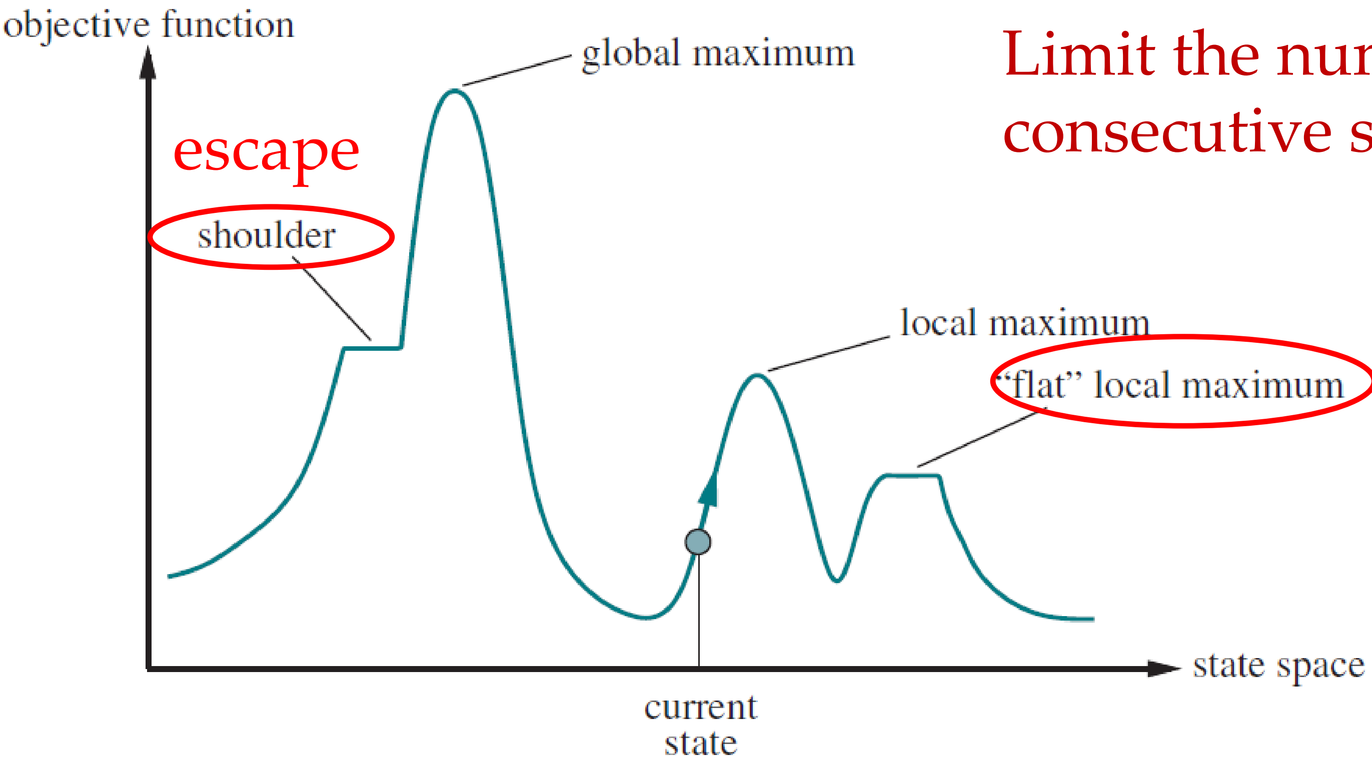
Hill-climbing search

An example of one-dimensional state-space landscape



Hill-climbing search

Hill-climbing search with sideways move: accept the best neighbor if it has the same value as the current state



Limit the number of consecutive sideways moves

infinite loop

Hill-climbing search

8-queens problem: to place eight queens on a chessboard such that no queen attacks any other

Heuristic function *h*: number of pairs of queens that are attacking each other

Neighbor space: states generated by moving a single queen to another square in the same column

Hill-climbing	Without sideways move	With sideways move
Success rate	14%	94%
Average steps for a success	4 steps	21 steps

Random-restart hill-climbing search

Random-restart hill-climbing search: conduct a series of hill-climbing searches from randomly generated initial states

Given unlimited time, it will eventually find a goal state

The success probability of each hill-climbing search: p

⇓ geometric distribution with parameter p

The expected number of restarts: $1/p$

Variants of hill-climbing search

hill-climbing: move to the best neighbor state

Stochastic hill-climbing: find all better neighbor states, and select one as the next state with probability related to its objective value

First-choice hill-climbing: repeatedly generate neighbor states randomly, and select the first better neighbor as the next state

Can be applied to continuous spaces

Simulated annealing

Hill-climbing search: efficient, but may get trapped in local optima

Random search: find global optima, but inefficient

Simulated annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow Value(next) - Value(current)$
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

randomly generate a neighbor

if the neighbor is better, move to it

Otherwise, move to the worse state with some probability

Simulated annealing

Simulated annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for t = 1 **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow Value(next) - Value(current)$
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

randomly generate a neighbor

if the neighbor is better, move to it

Otherwise, move to the worse state with some probability

Can be applied to both discrete and continuous spaces

Simulated annealing

Simulated annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for t = 1 **to** ∞ **do**
    T ← *schedule*(t)
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow Value(next) - Value(current)$
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

randomly generate a neighbor

if the neighbor is better, move to it

Otherwise, move to the worse state with some probability

The probability $e^{\Delta E/T}$ of accepting the worse state

- Increase with ΔE
- Increase with the temperature parameter T

Simulated annealing

Simulated annealing

The probability $e^{\Delta E/T}$ of accepting the worse state

- Increase with ΔE
- Increase with the temperature parameter T

T is initially set to a large value, and gradually decreased to 0

⇩

The probability of accepting worse states gradually decreases

Inspired from the annealing process in metallurgy

Local beam search

Local beam search: maintain k states

- The initial k states are generated randomly
- In each iteration, generate all neighbors of the current k states, and select the best k ones

Different from hill-climbing search with k random-restarts

Can be applied to discrete spaces

Local search for continuous spaces

Gradient descent: for minimization

$$\boldsymbol{x} = \boldsymbol{x} - \alpha \cdot \nabla f(\boldsymbol{x})$$

Gradient ascent: for maximization

$$\boldsymbol{x} = \boldsymbol{x} + \alpha \cdot \nabla f(\boldsymbol{x})$$

Converge to $\nabla f(\boldsymbol{x}) = 0$: local optimum or saddle point

There are many variants of gradient descent/ascent, as well as methods using the Hessian matrix, e.g., Newton-Raphson

$$\boldsymbol{x} = \boldsymbol{x} + \mathbf{H}_f^{-1}(\boldsymbol{x}) \cdot \nabla f(\boldsymbol{x})$$

The theory of evolution

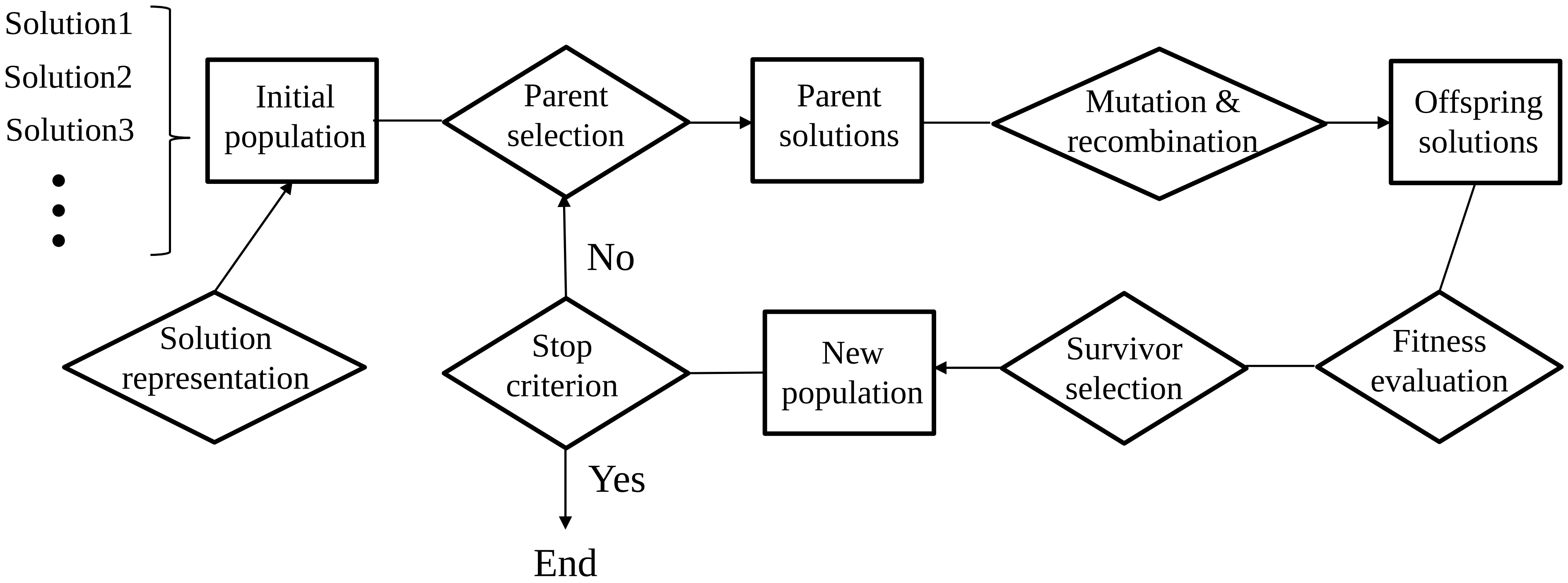
Central idea of Darwinism: reproduction with variation and natural selection based on the fitness

Core components of Darwinian evolutionary system:

- *One or more populations of individuals competing for limited resources*
- *The notion of dynamically changing populations due to the birth and death of individuals*
- *A concept of fitness which reflects the ability of an individual to survive and reproduce*
- *A concept of variational inheritance: offspring closely resemble their parents, but are not identical*

Evolutionary algorithms

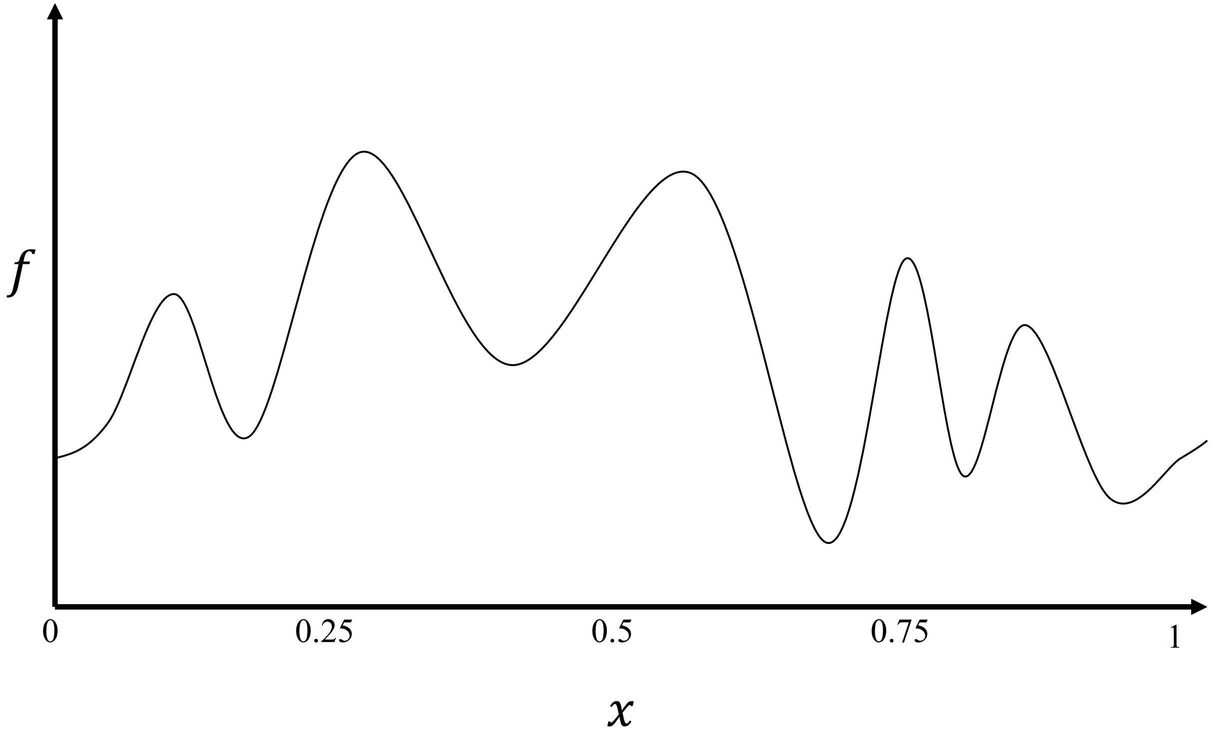
General structure of evolutionary algorithms for $\arg\max_{x} f(x)$



Can be applied to both discrete and continuous spaces

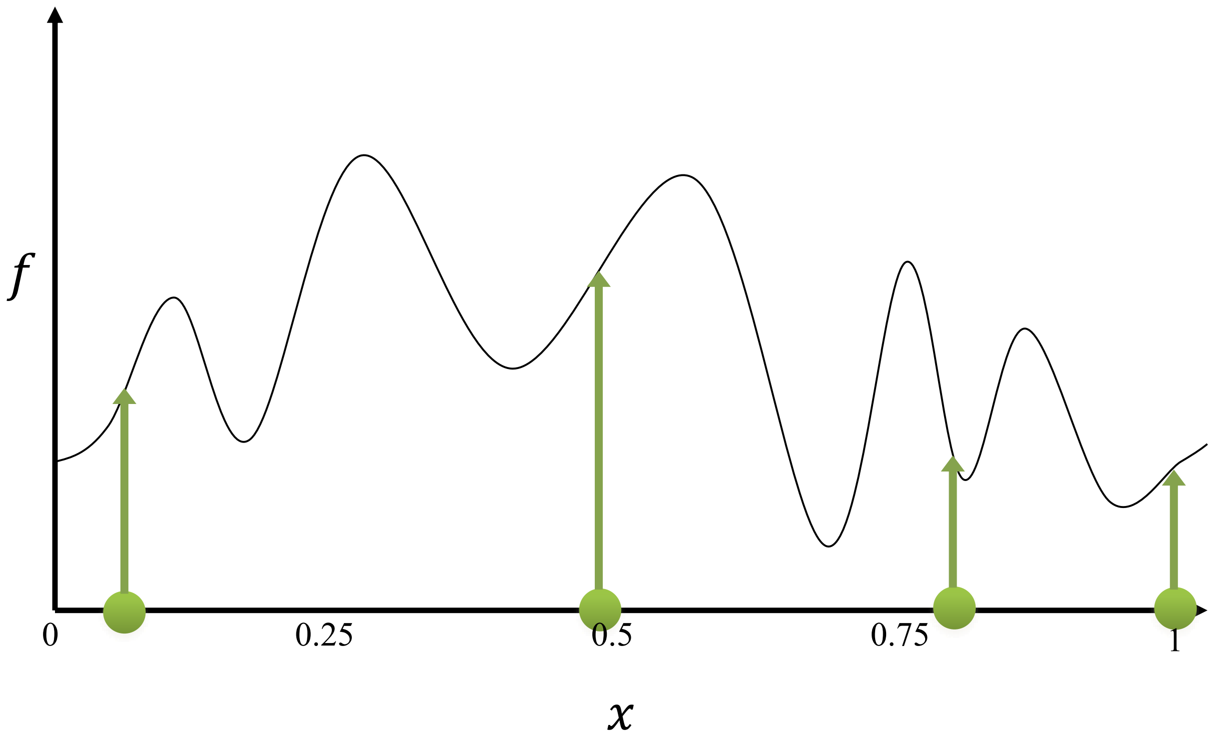
An illustration of running

$\mathcal{X} = [0, 1]$



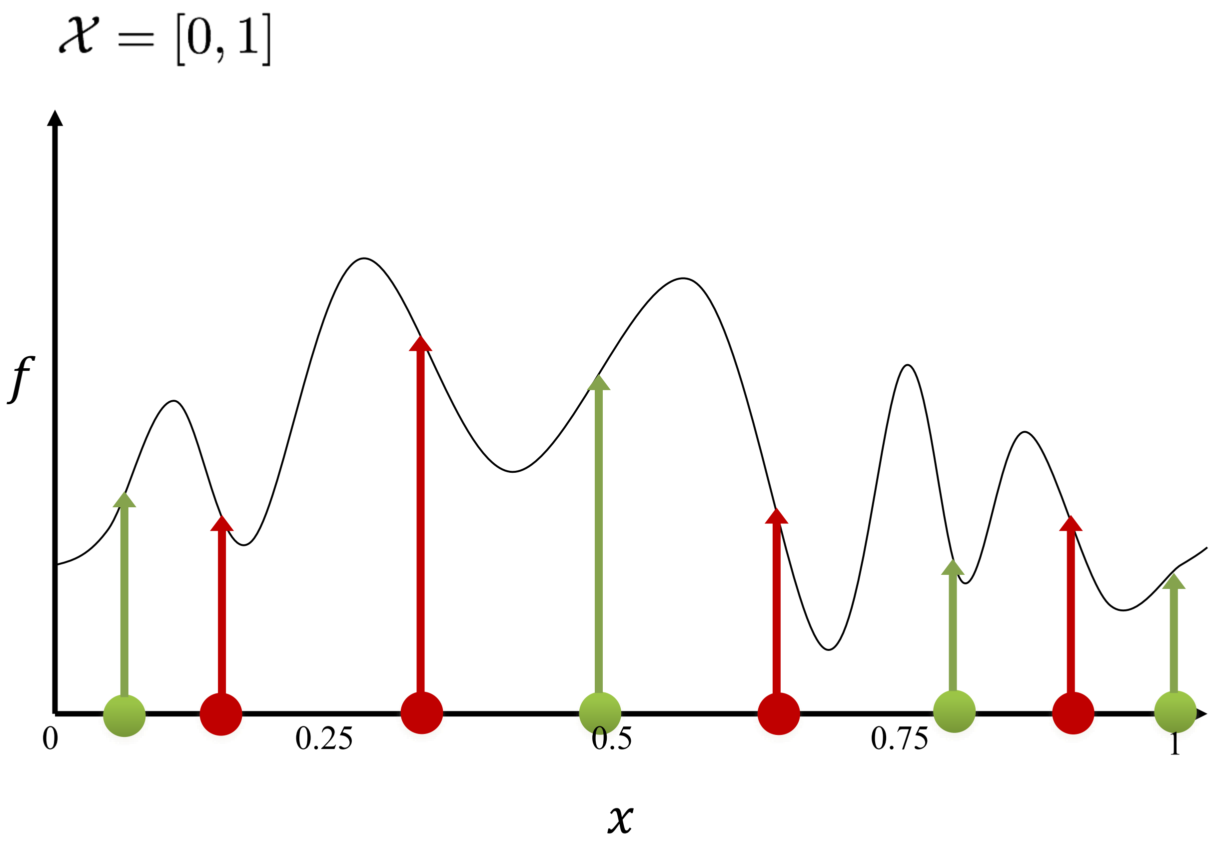
An illustration of running

$\mathcal{X} = [0, 1]$



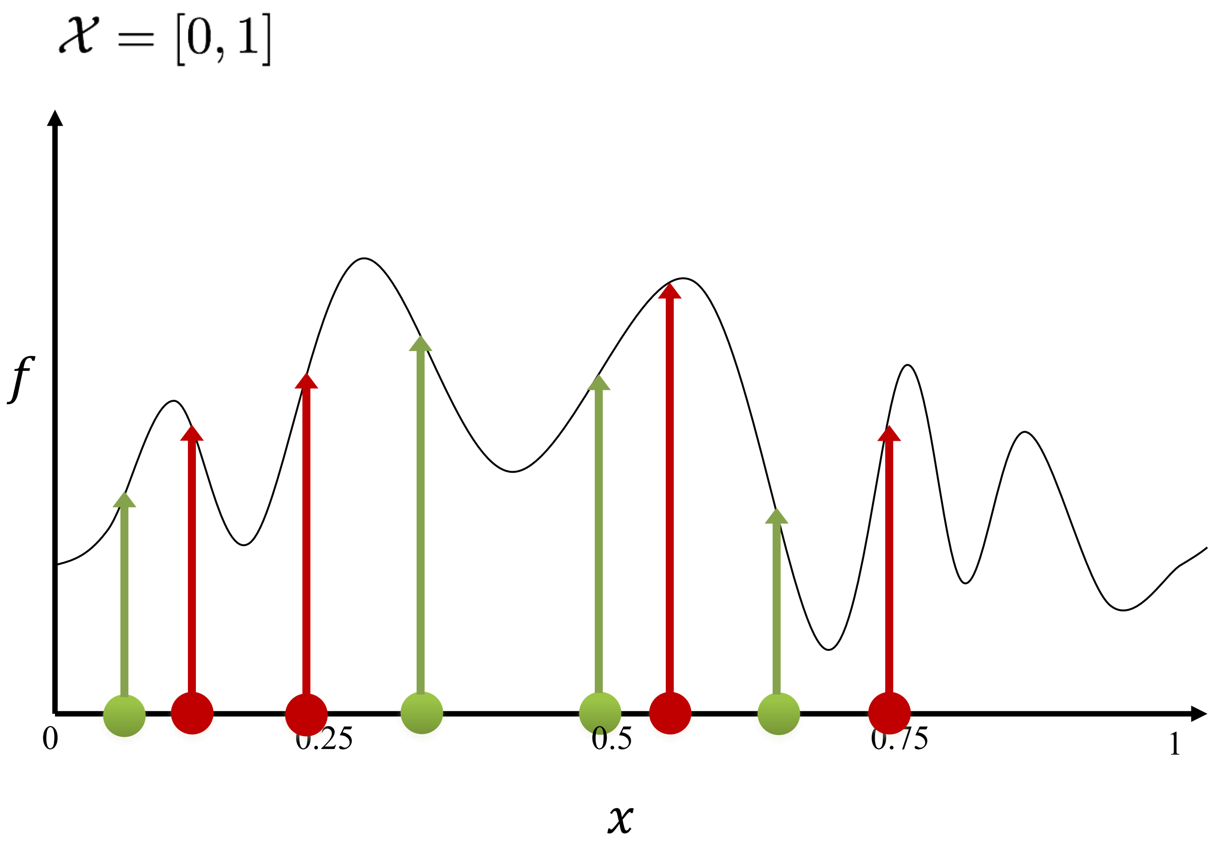
initialization
evaluation

An illustration of running



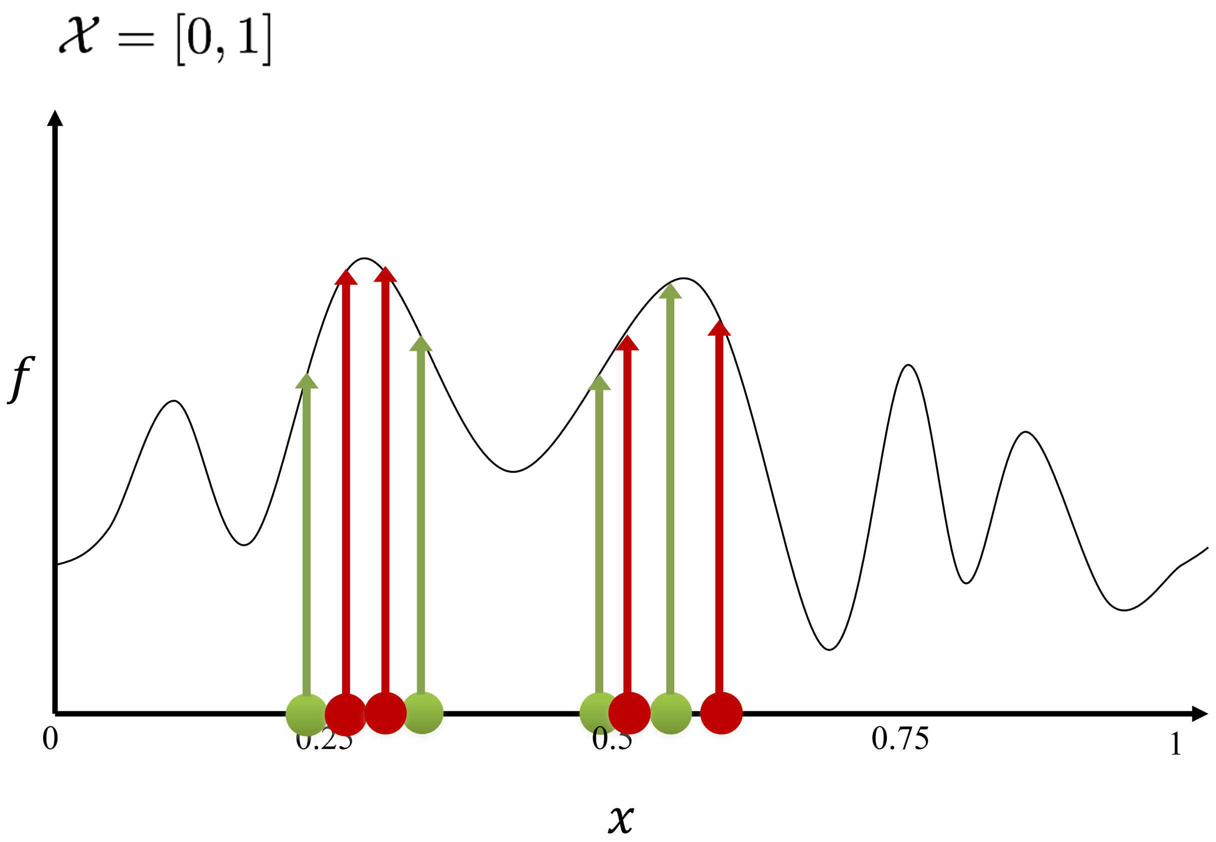
initialization
evaluation
reproduction
evaluation

An illustration of running



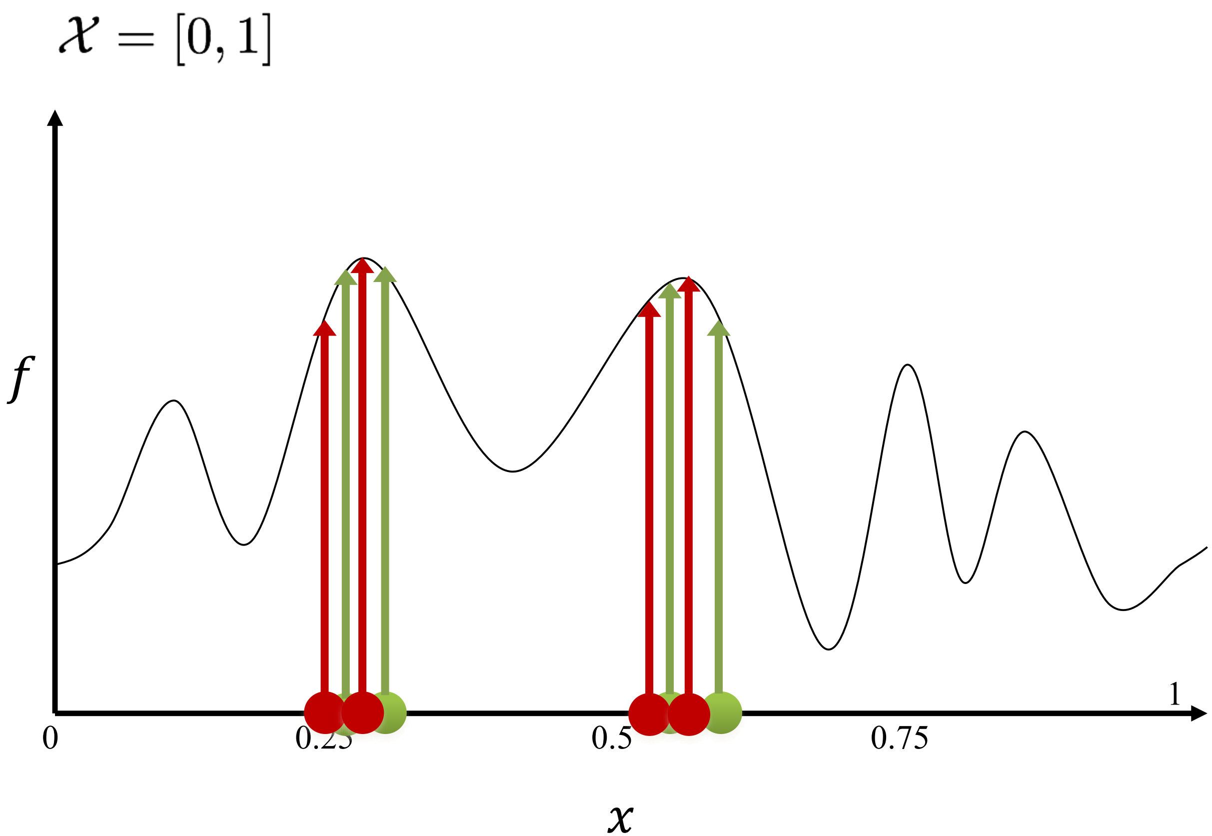
initialization
evaluation
reproduction
evaluation
selection
reproduction
evaluation

An illustration of running



initialization
evaluation
reproduction
evaluation
selection
reproduction
evaluation
selection
reproduction
evaluation

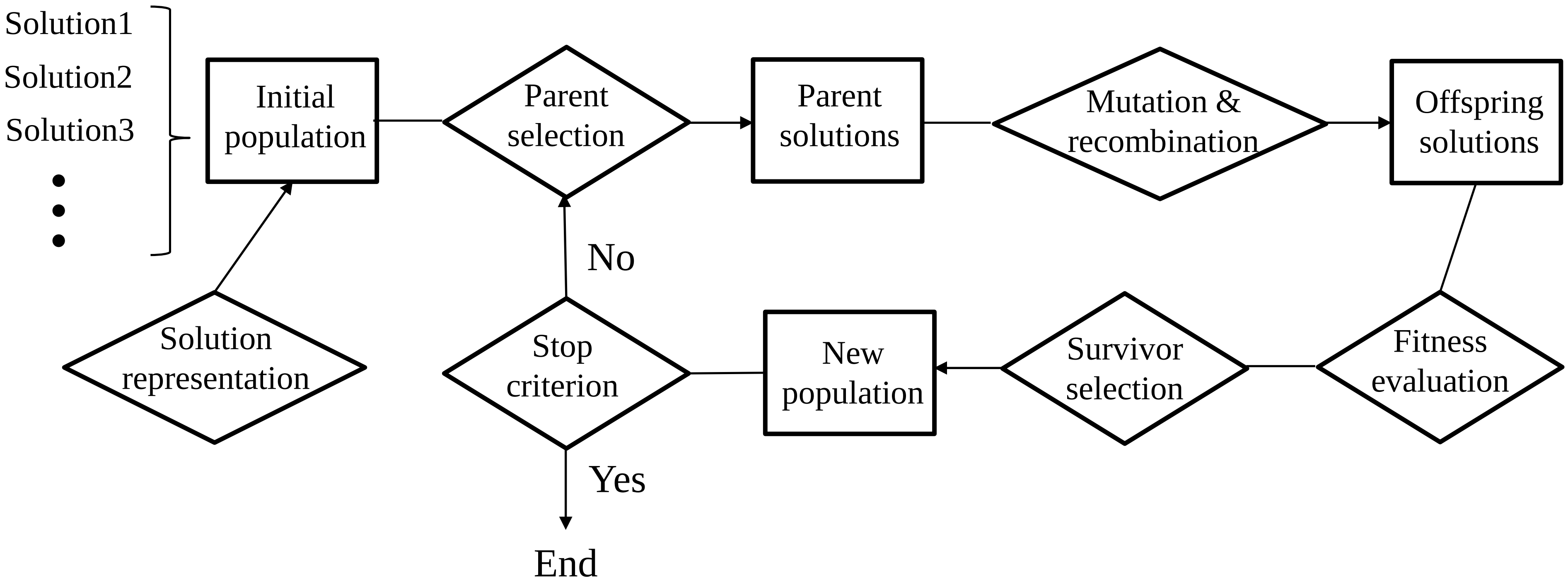
An illustration of running



initialization
evaluation
reproduction
evaluation
selection
reproduction
evaluation
selection
reproduction
evaluation
selection
reproduction
evaluation
...

Evolutionary algorithms

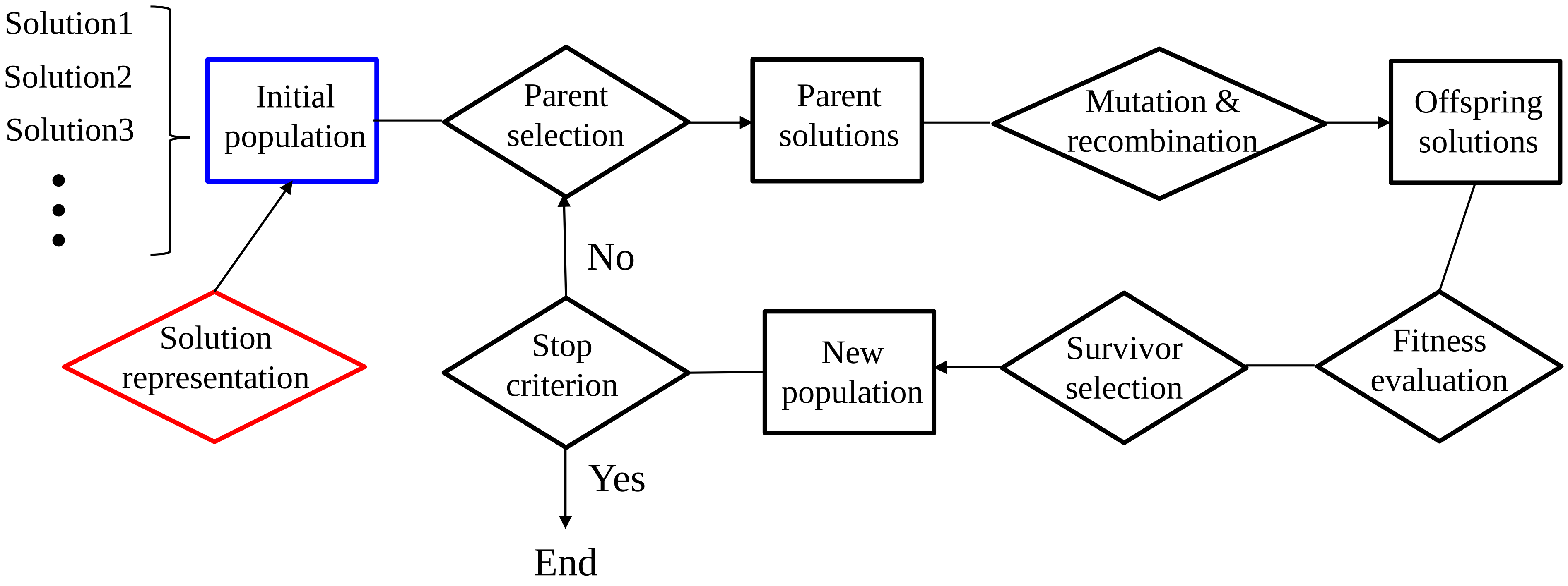
General structure of evolutionary algorithms



Need to design each component of evolutionary algorithms

Evolutionary algorithms

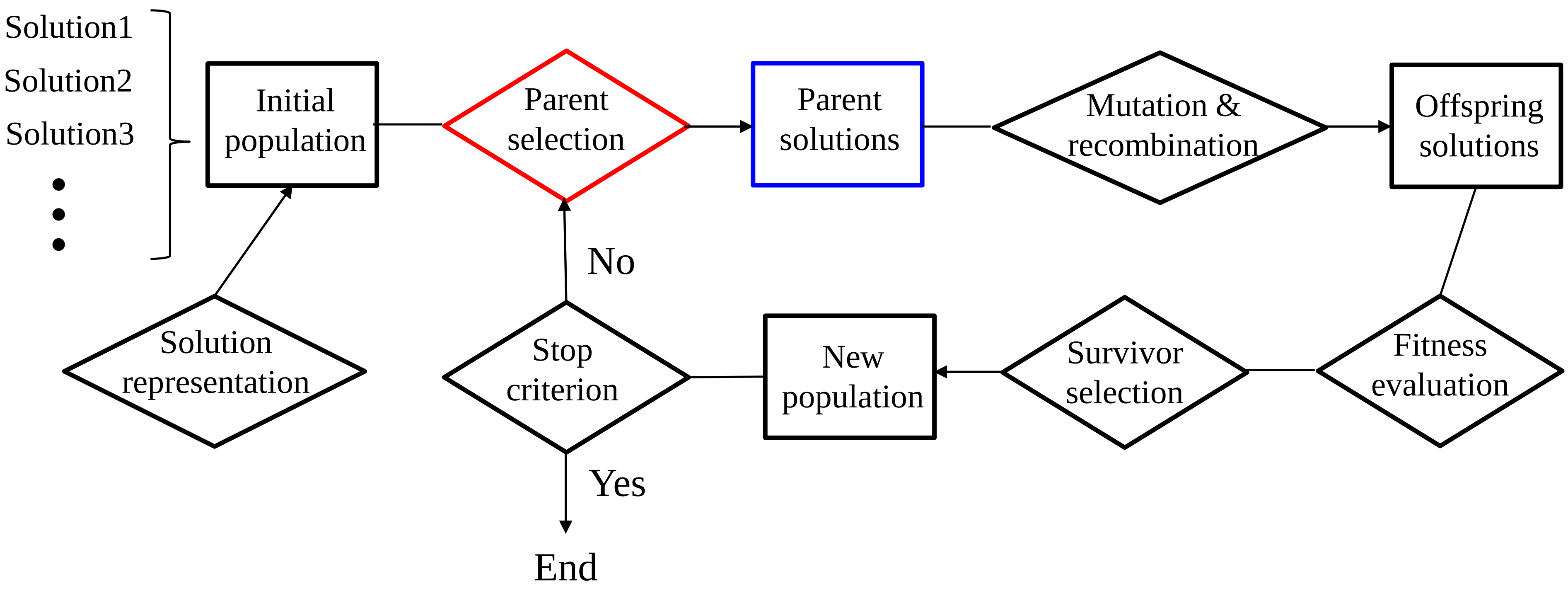
General structure of evolutionary algorithms



Need to design each component of evolutionary algorithms

Evolutionary algorithms

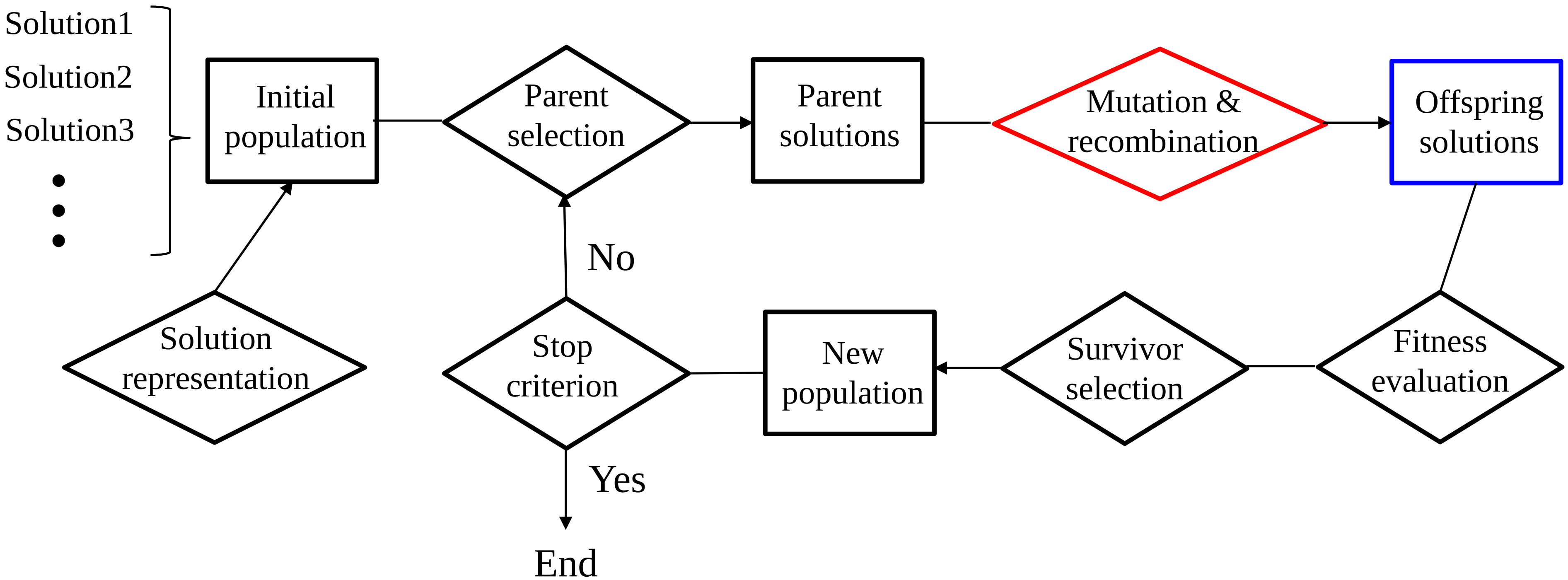
General structure of evolutionary algorithms



Need to design each component of evolutionary algorithms

Evolutionary algorithms

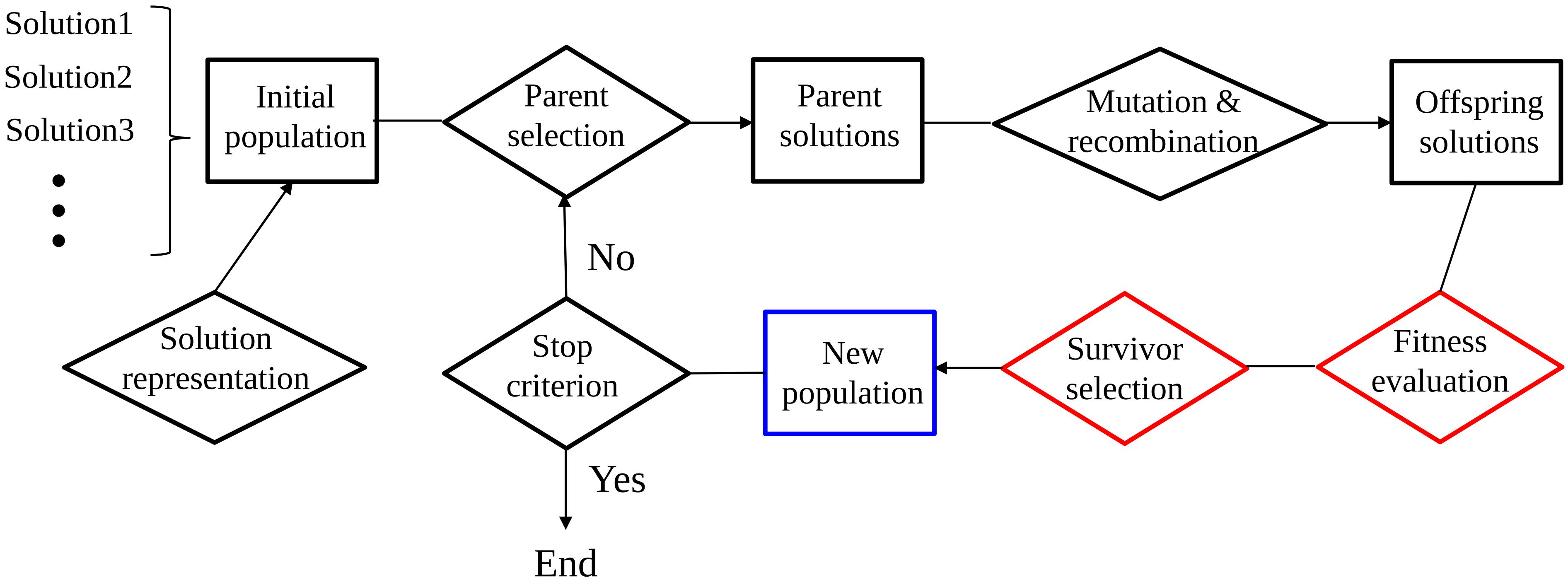
General structure of evolutionary algorithms



Need to design each component of evolutionary algorithms

Evolutionary algorithms

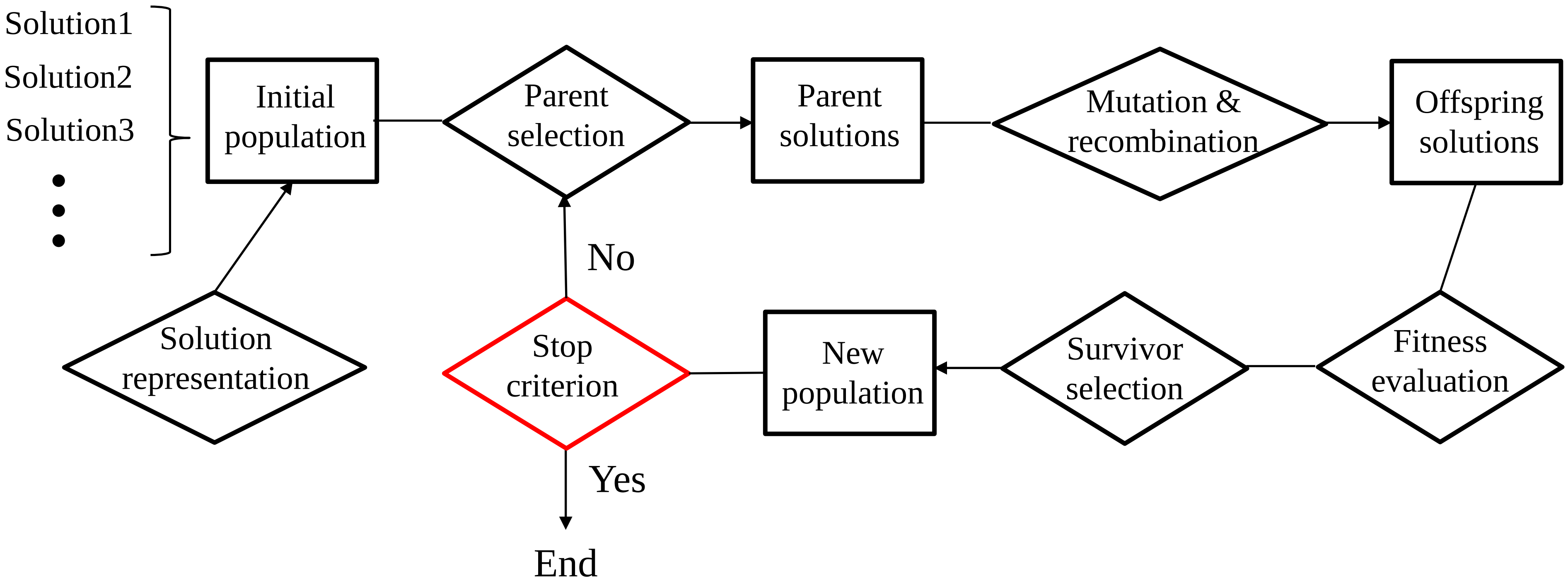
General structure of evolutionary algorithms



Need to design each component of evolutionary algorithms

Evolutionary algorithms

General structure of evolutionary algorithms

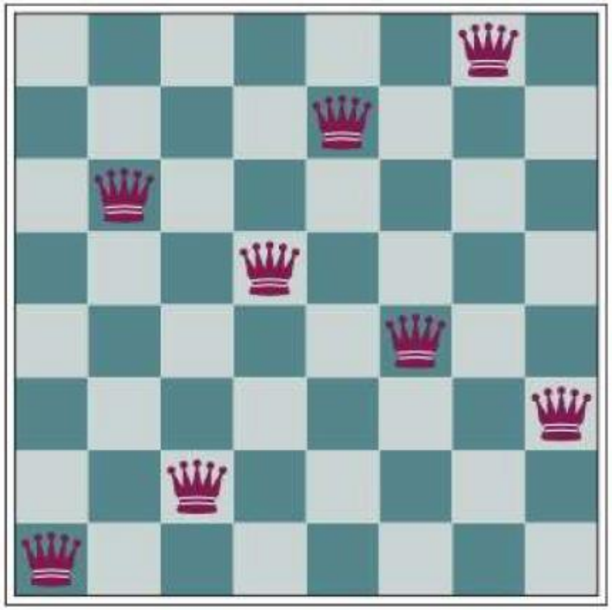


Need to design each component of evolutionary algorithms

An application to 8-queens problem

8-queens problem: to place eight queens on a chessboard such that no queen attacks any other

Objective function f: number of nonattacking pairs of queens



Solution representation

Integer vector

1	6	2	5	7	4	8	3

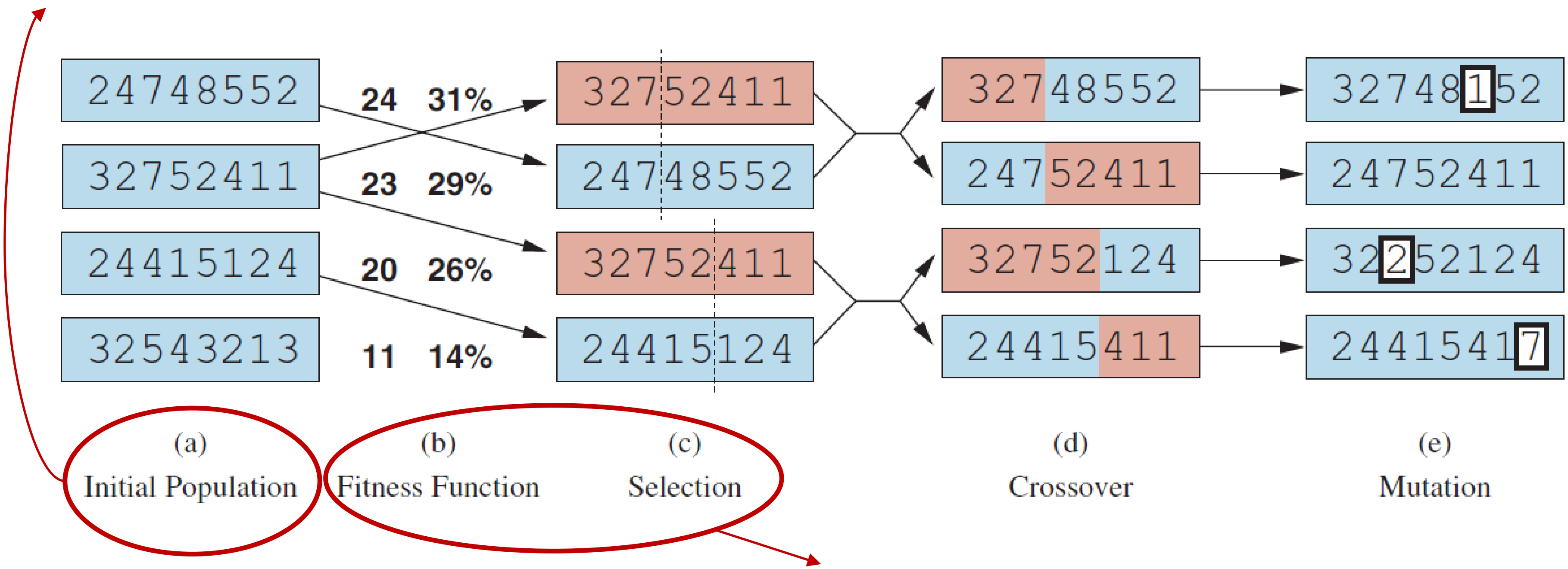
position of the queen on each column

Binary vector

000101001100110011111010

An application to 8-queens problem

Initialization: four randomly generated solutions

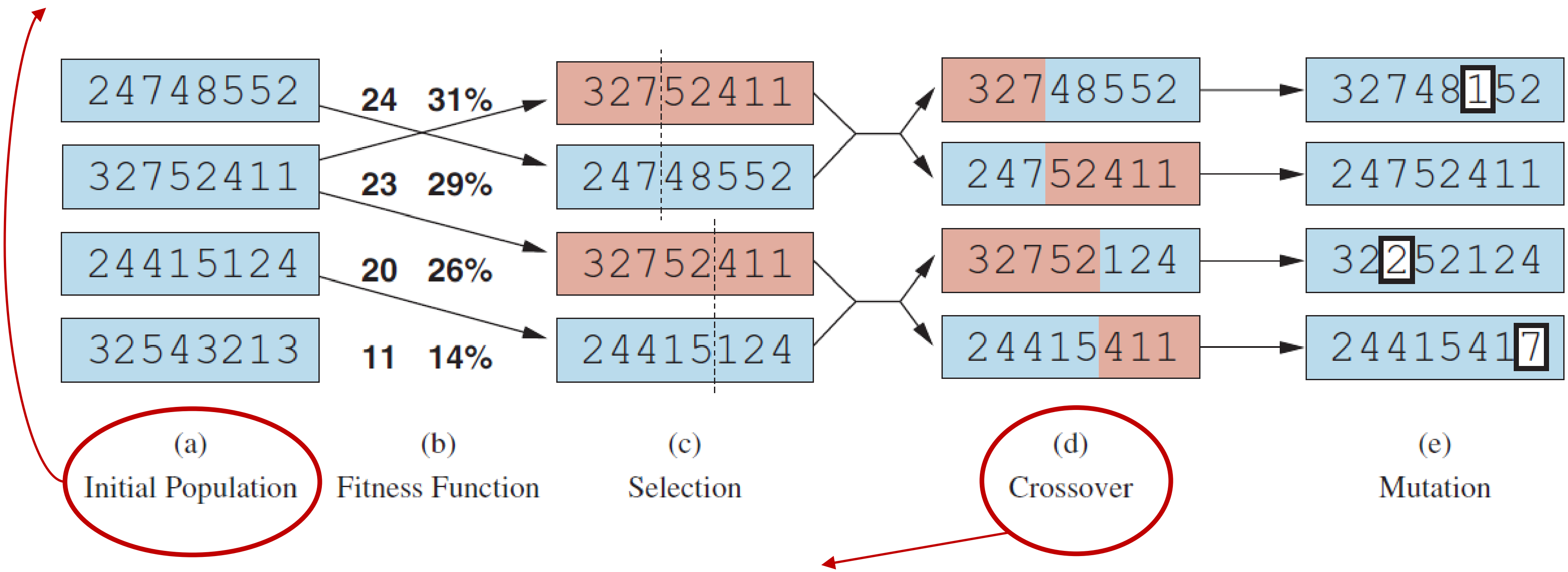


Parent selection: fitness proportional selection

Probability of selecting the i-th solution ← $p_i = \frac{f_i}{\sum_{j=1}^{\mu} f_j}$ → Fitness (objective) value of the i-th solution

An application to 8-queens problem

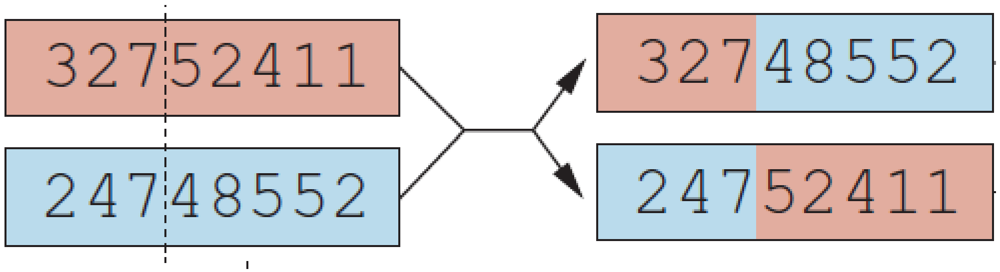
Initialization: four randomly generated solutions

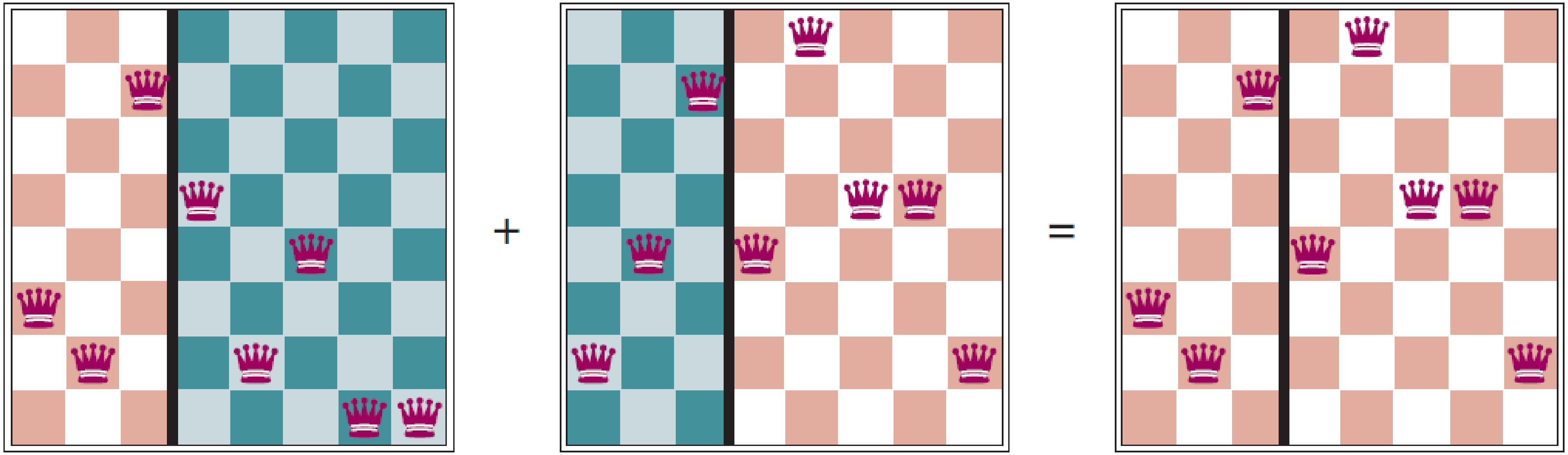


Recombination: one-point crossover

Select one crossover point randomly, and exchange the parts of the two solutions after the point

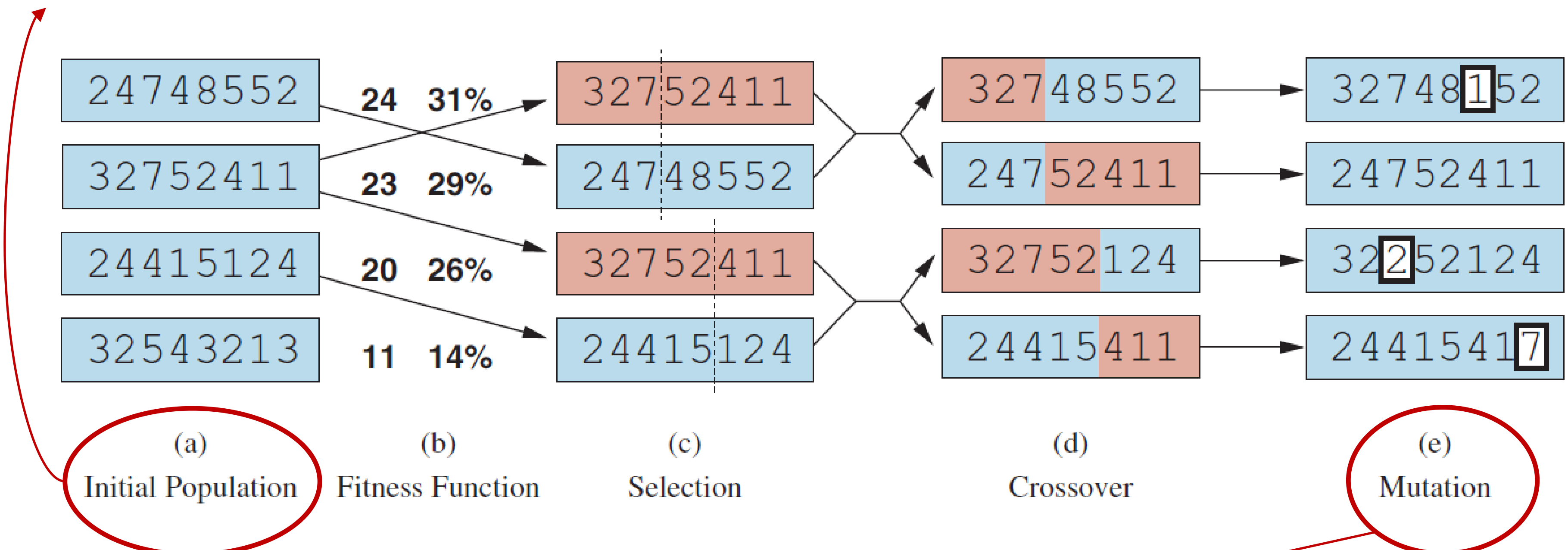
An application to 8-queens problem





An application to 8-queens problem

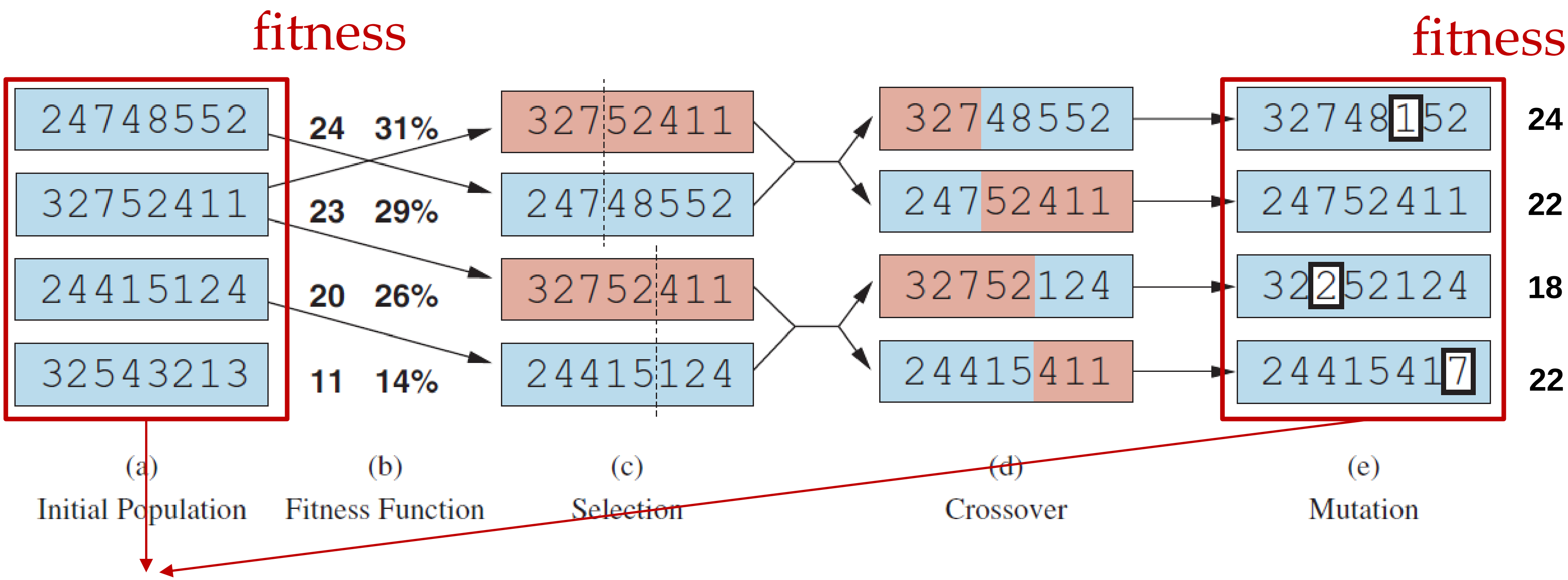
Initialization: four randomly generated solutions

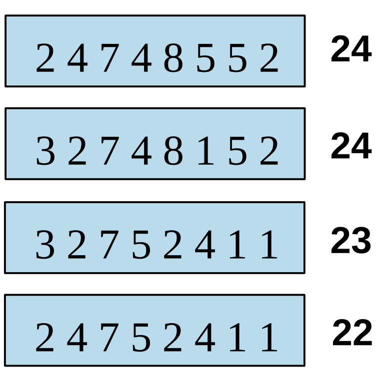


Mutation:

For each element of a solution, change it to a randomly chosen different value with probability 1/8

An application to 8-queens problem





Survivor selection:

Select the best four solutions from the current population and offspring solutions to generate the next population

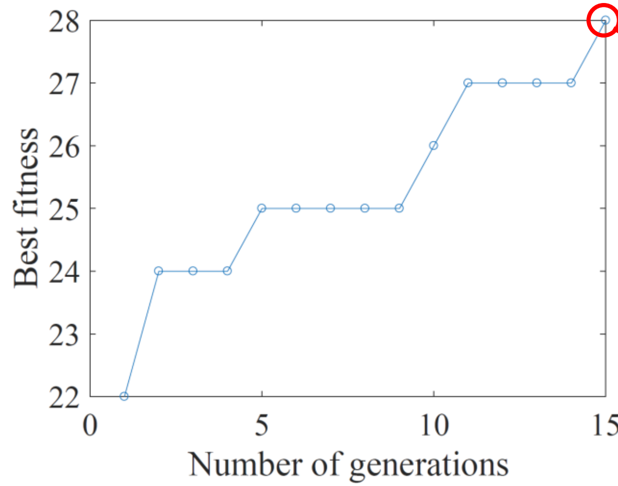
An application to 8-queens problem

Run 1:

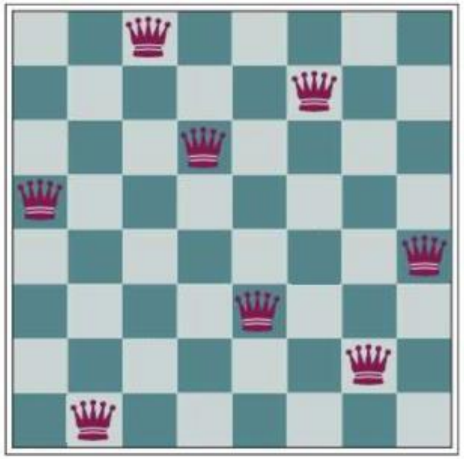
Initial population	fitness
4 7 8 7 7 2 2 2	**18**
1 1 8 6 3 5 5 3	**20**
6 6 7 4 4 5 6 2	**18**
2 4 1 3 1 6 6 1	**22**

Final population	fitness
5 1 8 6 3 7 2 4	**28**
5 1 8 6 3 7 2 8	**27**
5 1 8 6 3 7 2 8	**27**
5 1 8 6 3 7 2 8	**27**

Curve change of the best fitness



15 generations



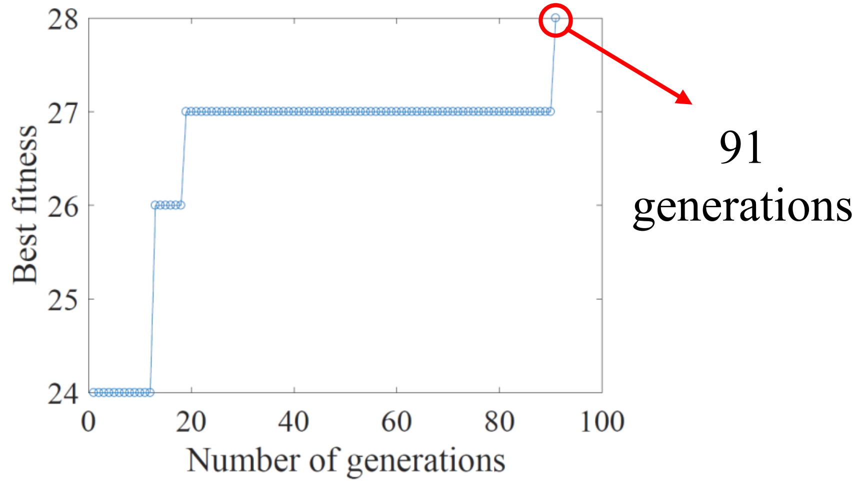
An application to 8-queens problem

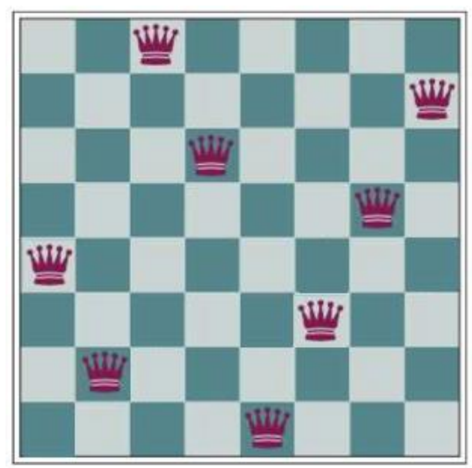
Run 2:

Initial population	fitness
3 8 8 1 4 3 2 7	**20**
6 1 4 6 1 3 5 2	**24**
6 7 1 3 7 4 5 6	**17**
7 7 8 8 6 2 4 5	**20**

Final population	fitness
4 2 8 6 1 3 5 7	**28**
4 6 8 6 1 3 5 7	**27**
4 6 8 6 1 3 5 7	**27**
4 6 8 6 1 3 5 7	**27**

Curve change of the best fitness





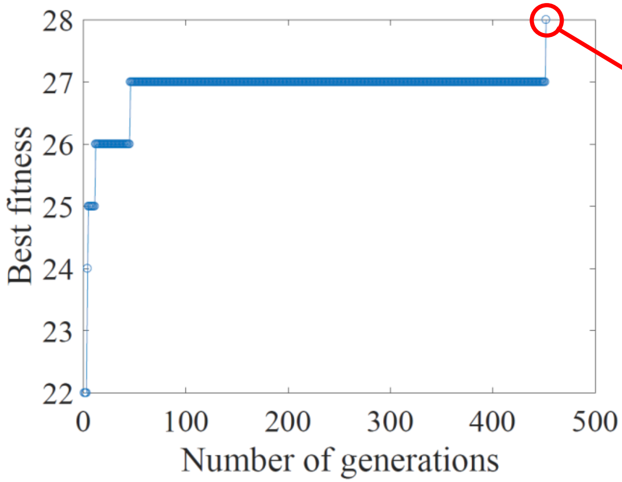
An application to 8-queens problem

Run 3:

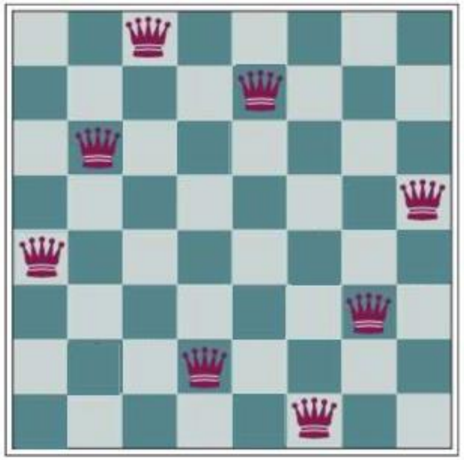
Initial population	fitness
4 6 5 7 2 5 1 2	20
2 5 7 6 4 3 3 6	22
5 8 7 4 3 5 4 7	20
4 6 2 1 4 4 6 7	15

Final population	fitness
4 6 8 2 7 1 3 5	28
4 1 8 2 7 6 3 5	27
4 1 8 2 7 6 3 5	27
4 1 8 2 7 6 3 5	27

Curve change of the best fitness



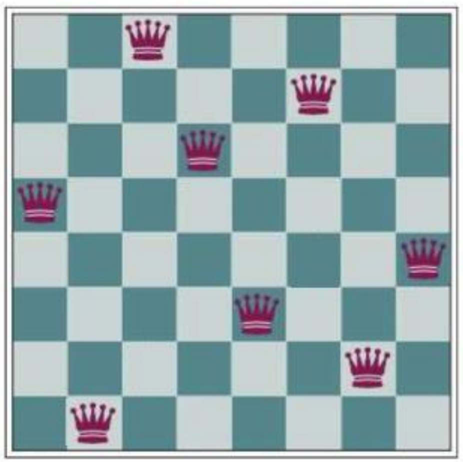
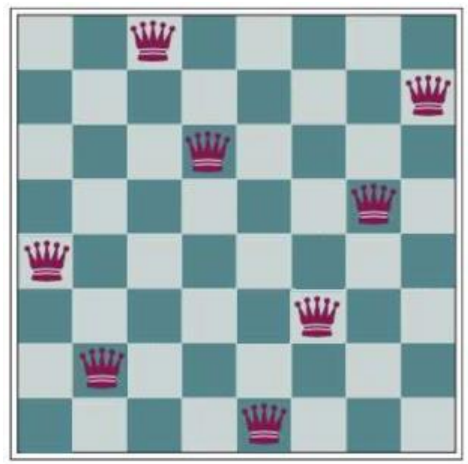
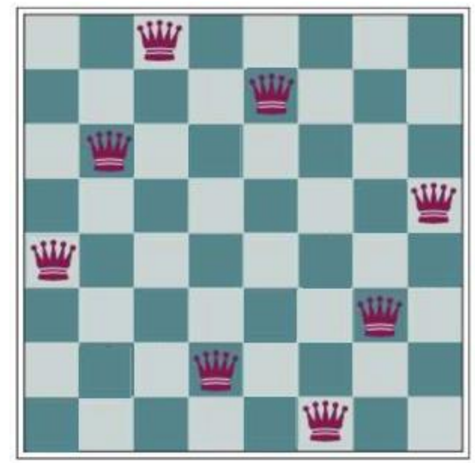
452 generations



An application to 8-queens problem

Run 1 | Run 2 | Run 3

The generated optimal solution

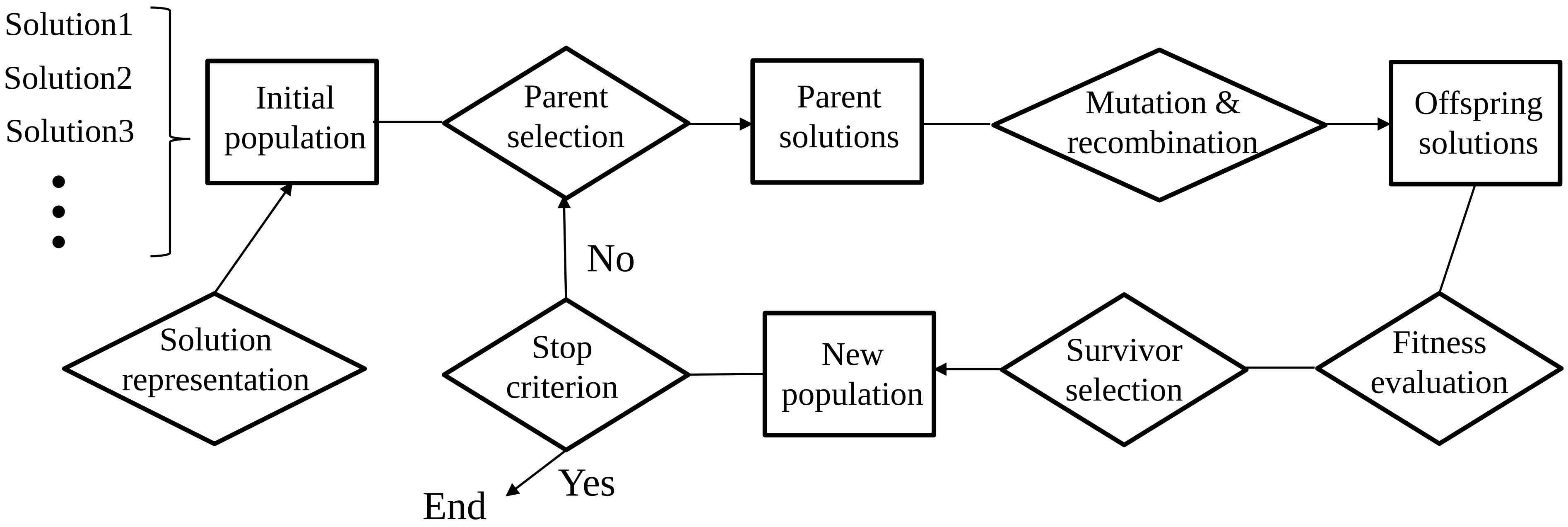




The required number of generations

15 | 91 | 452

Evolutionary algorithms are randomized algorithms

Local search vs. Evolutionary algorithms



Characteristics of evolutionary algorithms

- Population-based search
- Recombination
- Mutation, which can be a global search operator

Local search vs. Evolutionary algorithms

Advantages and disadvantages of evolutionary algorithms

- Easy to be parallelized
- Good ability of escaping from local optima
- Applicable to a wide range of problems, requiring only that the goodness of solutions can be evaluated
  - *non-differentiable problems*
  - *problems without explicit objective function formulation*
  - *problems with multiple objective functions*

Black-box

- Not very efficient, but can be accelerated by
  - *utilizing modern computer facilities*
  - *combining with local search*
  - *using the machine learning techniques*

Summary

- Hill-climbing search
- Simulated annealing
- Local beam search
- Local search for continuous spaces
- Evolutionary algorithms

(Hill-climbing search through Local search for continuous spaces: Local search)

References

- S. J. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Chapter 4.1-4.2, Third edition.
- K. A. De Jong. Evolutionary Computation – A Unified Approach. Chapter 1.