Finite Markov Chain Results in Evolutionary Computation: A Tour d'Horizon

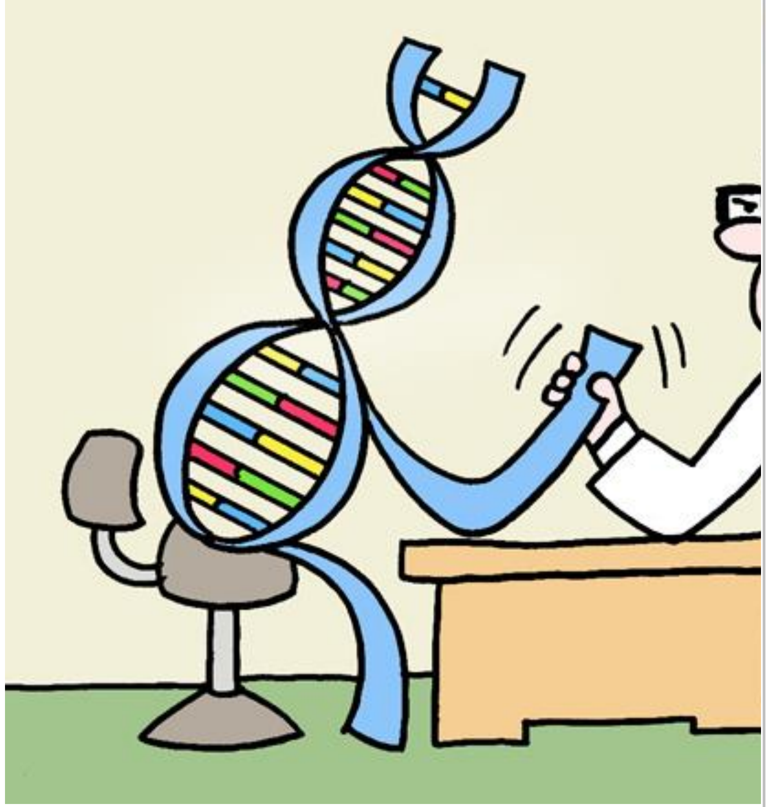
Siyeong Lee

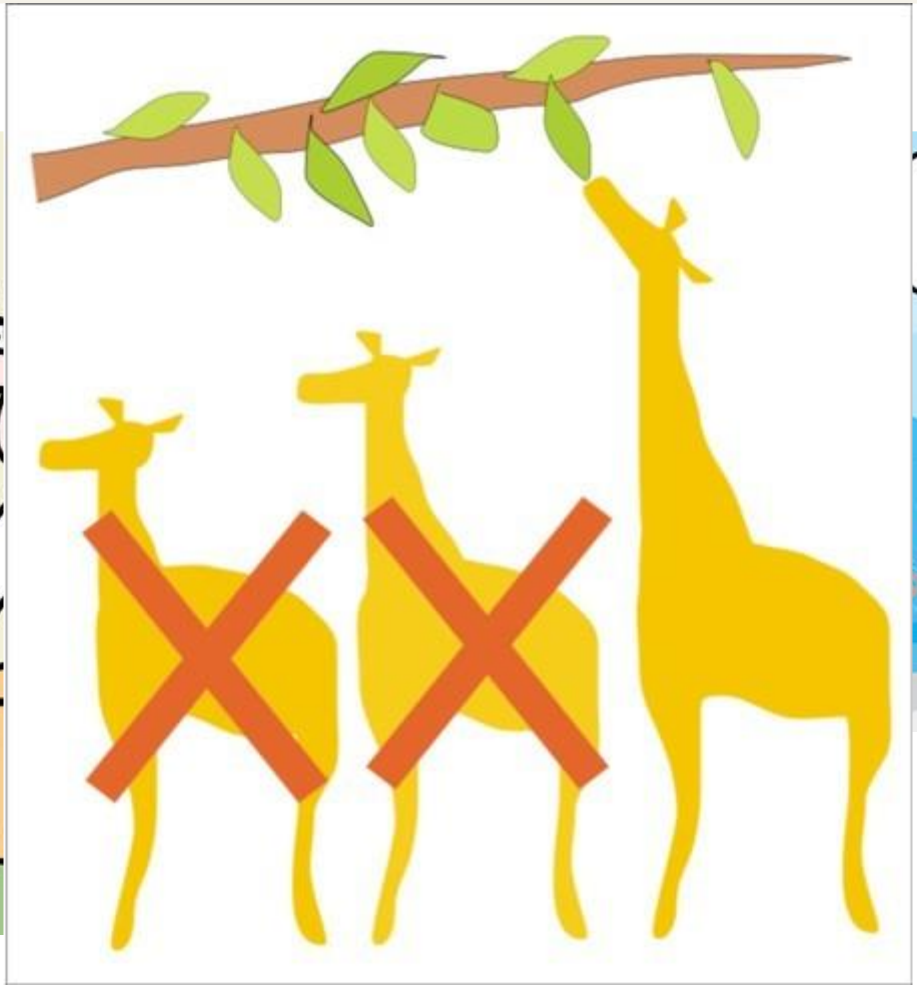
Dept. of Electronic Engineering

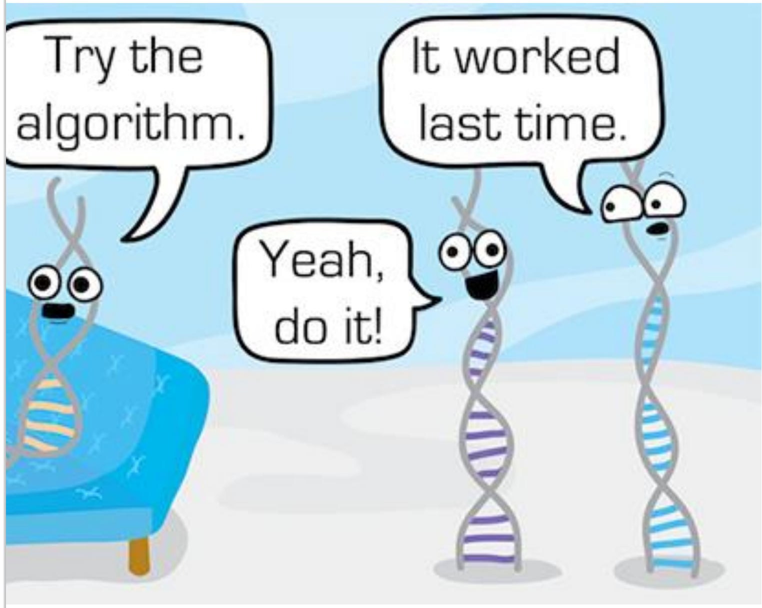
Sogang University

April 11, 2017

Introduction







Problem setting – Optimization problem

Definition 1. Optimization problem

$$Minimize\ f_0(x)$$

$$Subject\ to\ f_i(x) \leq b_i, i = 1, \ldots, m$$

$f_0 : \boldsymbol{R}^n \rightarrow \boldsymbol{R}$; Optimization variables
$f_i : \boldsymbol{R}^n \rightarrow \boldsymbol{R}, i = 1, \ldots m$: Constraint functions

$x = (x_1, \ldots, x_n) \in \boldsymbol{X}$: *Optimization variables*

Specially, solutions which satisfy all constraints is called *feasible solutions*

Combinatorial optimization

- **Combinatorial optimization** is a topic that consists of finding an optimal object from a finite of objects.
- It operates on the domain of those optimization problems, in which
  - the ***set of feasible solutions is discrete*** or
  - ***can be reduced to discrete***

Goal: to ***find the best solution*** in feasible solutions

Representation

- Teminology
  - A **individual** $X \times A$
    - where A is a possible empty set collecting additional search state information.

u_0	u_1							
1	0	0	⋯	1	0	1	1	0

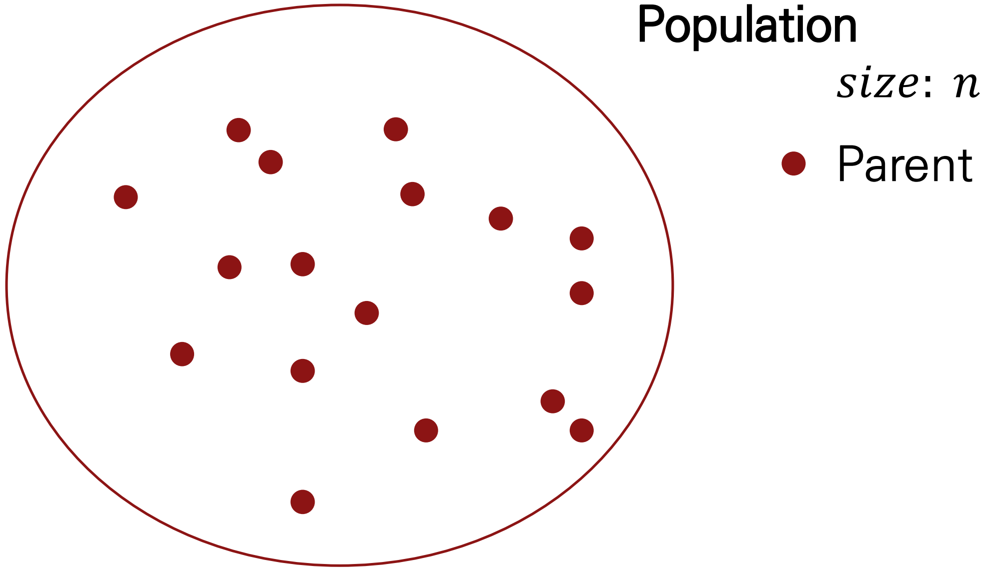
Figure 1: Solution representation of Combinatorial optimization

  - A **fitness of individual** $(x, a) \in X \times A$ *is given* $f(x); objective\ function$
  - A **population of** $\mathrm{n} < \infty$ **individual**
    - An element of the product space $(X \times A)^n$

Pseudocode [1]

- During each iteration of an evolutionary algorithm the population is modified by a number of successive probabilistic transformations.

(1) initialise population;
(2) evaluate population;
(3) **while** (*!stopCondition*) **do**
(4) select the best-fit individuals for reproduction;
(5) breed new individuals through crossover and mutation operations;
(6) evaluate the individual fitness of new individuals;
(7) replace least-fit population with new individuals;



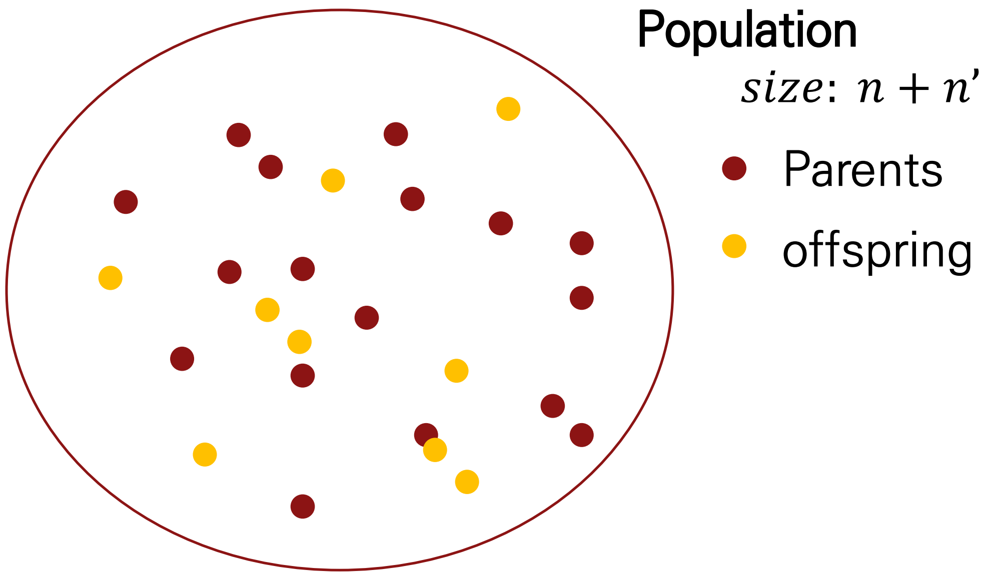
At the beginning of each iteration, the n members of the population are called the **parents** which produce $n' < \infty$ **offspring** by random variation

In this case,
The way to produce offspring: ***crossover, mutation***

Pseudocode [1]

- During each iteration of an evolutionary algorithm the population is modified by a number of successive probabilistic transformations.

(1) initialise population;
(2) evaluate population;
(3) **while** (*!stopCondition*) **do**
(4) select the best-fit individuals for reproduction;
(5) breed new individuals through crossover and mutation operations;
(6) evaluate the individual fitness of new individuals;
(7) replace least-fit population with new individuals;



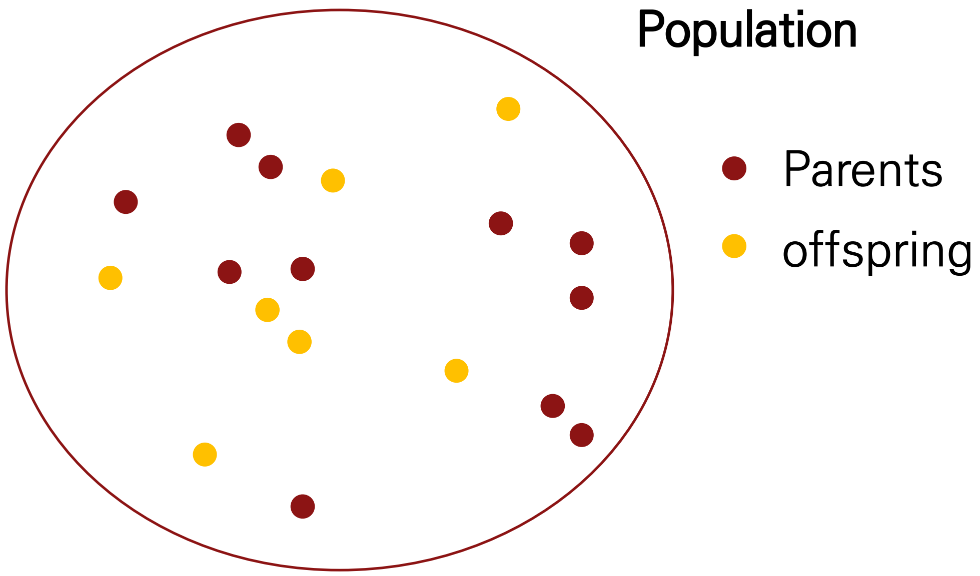
In this case,
The way to produce offspring: ***crossover, mutation***

At the beginning of each iteration,
the n members of the population are called the **parents** which produce $n' < \infty$ **offspring** by random variation

Pseudocode [1]

- During each iteration of an evolutionary algorithm the population is modified by a number of successive probabilistic transformations.

(1) initialise population;
(2) evaluate population;
(3) **while** (*!stopCondition*) **do**
(4) select the best-fit individuals for reproduction;
(5) breed new individuals through crossover and mutation operations;
(6) evaluate the individual fitness of new individuals;
(7) replace least-fit population with new individuals;

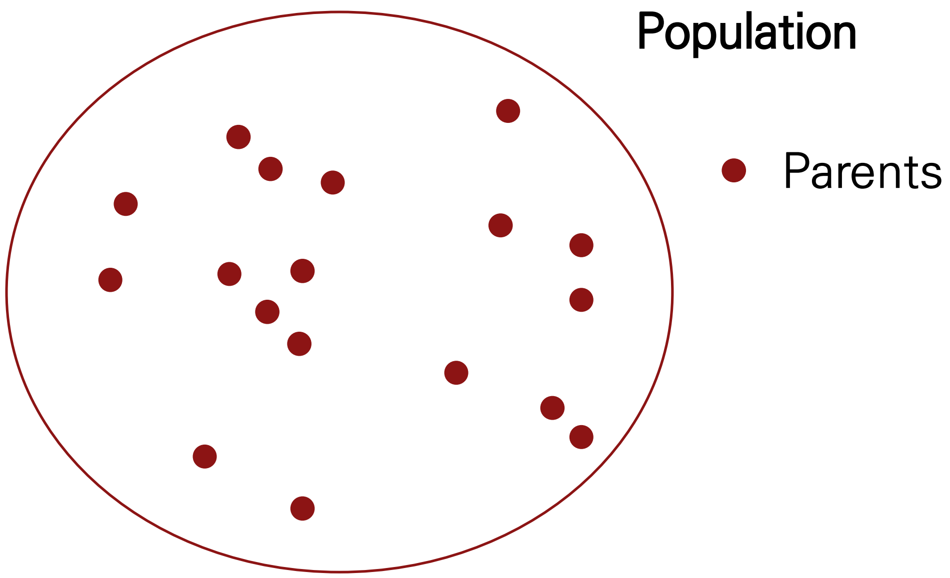


To keep the population at constant size n, a selection method decides which parents and or offspring will serve as parents in the next iteration.

Pseudocode [1]

- During each iteration of an evolutionary algorithm the population is modified by a number of successive probabilistic transformations.

```
(1) initialise population;
(2) evaluate population;
(3) while (!stopCondition) do
(4)     select the best-fit individuals for reproduction;
(5)     breed new individuals through crossover and mutation operations;
(6)     evaluate the individual fitness of new individuals;
(7)     replace least-fit population with new individuals;
```



To keep the population at constant size n, a selection method decides which parents and or offspring will serve as **parents in the next iteration**.

Toy problem

Individual

0	0	1	1	0	0	1	1	0	1

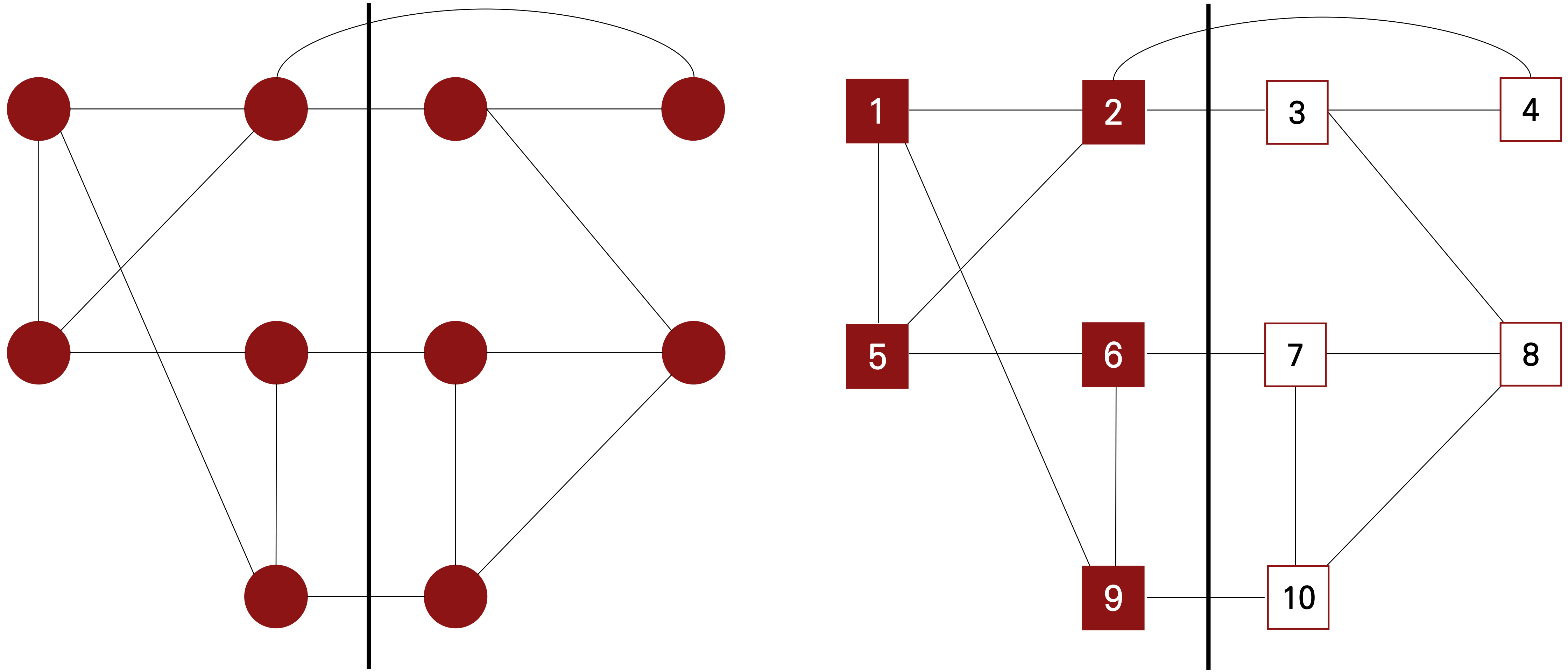


Figure 2: Bipartite graph problem

Operational description of evolutionary algorithms

- Let $(x_1, x_2, \ldots, x_n) \in \mathrm{X}^n$ denote the population of parents.
  - An offspring is produced as follows:

1 **Mat:** $\mathrm{X}^n \rightarrow \mathrm{X}^\rho$ where $2 < \rho \leq \mathrm{n}.$; selecting ρ parents

↓

2 **Reco:** $\mathrm{X}^\rho \rightarrow \mathrm{X}$; these individuals are recombined

↓

3 **Mut:** $\mathrm{X} \rightarrow \mathrm{X}$; Mutation

✓ **After all m** offspring have been produced, **the selection procedure** ➡ **sel:** $\mathbf{X}^k \rightarrow \mathbf{X}^\mathbf{n}$

; Decides which offspring and possibly parents $(k \geq n)$ will serve as the new parents in the next iteration

Operational description of evolutionary algorithms

- Let $(x_1, x_2, \ldots, x_n) \in \mathrm{X}^n$ denote the population of parents.
  - An offspring is produced as follows:

During a single iteration,

$$\forall i \in \{1, \ldots, m\} : x'_i = \mathrm{mut}(\mathrm{reco}(\mathrm{mat}(x_1, \ldots, x_n)))$$

$$(y_1, \ldots, y_n) = \begin{cases} \mathrm{sel}(x_{\pi(1)}, \ldots, x_{\pi(q)}, x'_1, \ldots, x'_m) & \text{(parents and offspring)} \\ \mathrm{sel}(x'_1, \ldots, x'_m) & \text{(only offspring)} \end{cases}$$

After all m offspring have been produced, the selection procedure → sel: $\mathbf{X}^k \rightarrow \mathbf{X}^n$

; Decides which offspring and possibly parents $(k \geq n)$ will serve as the new parents in the next iteration

Markov property in Population.

- Evidently, the resulting **new population** only depends on the state of **the current population** in a probabilistic manner.

✓ Therefore, we regard **the probabilistic behavior of evolutionary algorithms** as **Markov processes**.

Division of MP

- The state space may be finite, denumerable or not denumerable.
- The evolution may happen in discrete or continuous time.
- The transition probabilities may depend on the time parameter or not.

Markov property in Population

- Evidently, the resulting **new population** only depends on the state of **the current population** in a probabilistic manner.

Therefore, we regard **the probabilistic behavior of evolutionary algorithms** as **Markov processes**.

✓ Most results are available for evolutionary algorithms with time *homogeneous transitions* and
(1) $\boxed{\textit{finite search space in discrete time,}}$
(2) not denumerable search space R^l in discrete as well as continuous time.

Guarantee of Optimal Solution

Let $X_k = (X_{k,1}, X_{k,2}, \dots, X_{k,n})$ be the random population of size n, at step k and $F_k = \max\{f(X_{k,i}) : i = 1, \dots, \infty\}$ the best fitness value within the population at step $k \geq 0$.

- **Goal**
  - As soon as the random variable F_k attains the value of the global maximum f^*, it is ensured that the population contains an individual representing the global solution of the maximization problem.

We **always** want to **find optimal solution regardless of the initialization**.

Definition 2.1.

Let random variable $T = \min\{k \geq 0 : F_k = f^*\}$ denote the first hitting time of the global solution. An evolutionary algorithm is said to *visit the global optimum in finite time with probability one* if $P\{T < \infty\} = 1$ regardless of the initialization.

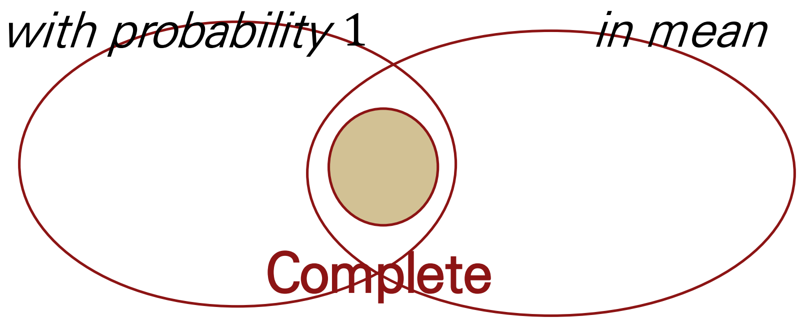
Stochastic Convergence

Definition 2.2.

Let $D_0, D_1, \ldots$ be non−negative random variables defined on a probability space (Ω, A, P).

The sequence $(D_k\colon k \geq 0)$ is said
to *converge completely to zero* if $\sum_{k=1}^{\infty} P\{D_k > \epsilon\} < \infty$ for any $\epsilon > 0$,
to *converge with probability* 1 *or almost surely to zero* if $P\left\{\lim_{k\to\infty} D_k = 0\right\} = 1$
to *converge in probability to zero* if $P\{D_k > \epsilon\} = o(1)\ as\ k \to \infty\ for\ any\ \epsilon > 0$,
and to *converge in mean to zero* if $E[D_k] = o(1)\ as\ k \to \infty$.

1



2

Suppose that a sequence D_k is bounded above i.e. there $exists\ M \in \boldsymbol{R}\ such\ that\ D_k < M$ for all k $\in \boldsymbol{N}$.

then D_k convergence **in probability** implies convergence **in mean**

the EA converges to the global optimum

Definition 2.3.

Let X_k be the sequence of populations generated by some evolutionary algorithm and let $F_k = \max(\mathrm{f}(\mathrm{X}_{\mathrm{k},1}), \ldots, \mathrm{f}(\mathrm{X}_{\mathrm{k,n}}))$ denote the best objective function value of the population of size $n < \infty$ at generation $k \geq 0$.

An evolutionary algorithm is said to converge completely with probability to the global maximum $f^* = \max\{f(x): x \in X\}$ of objective function $f : X \to \boldsymbol{R}$ if the nonnegative random sequence D_k with $D_k = f^* - F_k$ converges completely to zero.

Note that
the property of visiting the global solution with probability 1 is
a precondition for convergence but that the additional property of convergence **does not automatically indicate any advantage with respect to finding the global solution.**

the EA converges to the global optimum

- Main question
  - First Question
    - whether some evolutionary algorithm will visit the global optimum in finite time or not?
  - Second Question
    - whether it will converge in some mode to the optimum or not

Using finite Markov chain for some part!!

Operational description of evolutionary algorithms

- assumptions

  about **the properties of the variation and selection operators**.

  - $(A_1)\ \forall x \in (x_1, \dots, x_n)$: $P\left\{x \in reco\big(mat(x_1, \dots, x_n)\big)\right\} \geq \delta_r > 0.$
  - (A_2) $For\ every\ pair\ x, y \in \mathrm{X},$
    $there\ exists\ a\ finite\ path\ x_1, x_2, \dots, x_k\ of\ pairwise\ distinct\ points$
    $with\ x_1 = x, x_k = y\ such\ that\ P\{x_{i+1} = mut(x_i)\} \geq \delta_m > 0$
    $for\ all\ i = 1. \dots, k-1.$
  - $(A_2{}')$ $For\ every\ pair\ x, y \in \mathrm{X}\ holds\ P\{y = mut(x)\} \geq \delta_m > 0.$
  - $(A_3)\ \forall x \in (x_1, \dots, x_n)$: $P\{x \in sel(x_1, \dots, x_k)\} \geq \delta_s > 0.$
  - (A_4) $Let\ v_k^*(x_1, \dots, x_k) = \max\{f(x_i): i = 1, \dots, k\}$
    $denotes\ the\ best\ fitness\ value\ within\ a\ proulation\ of\ k\ individulas\ (k \geq n).$
    $the\ selection\ method\ fufills\ the\ condition$

$$P\big\{v_k^*\big(sel(x_1, \dots, x_k)\big) = v_k^*(x_1, \dots, x_k)\big\} = 1$$

the EA converges to the global optimum

Theorem 2.1.

If the assumptions $(A_1), (A_2)$, and (A_3) are valid then the evolutionary algorithm visits the global optimum after a finite number of iterations with probability one, regardless of the initialization.

If assumption (A_4) is valid additionally and the selection method chooses from parents as well as o spring then the evolutionary algorithm converges completely and in mean to the global optimum regardless of the initialization.

visit the global optimum in finite time with probability one.

$f^* - F_k$ *converges completely to zero.*

the EA converges to the global optimum

Corollary 2.1.

Theorem 2.1 remains valid if the assumptions $(A_1), (A_2), and\ (A_3)$ are replaced by assumption $(A_2{}')$

- Note that

1. Assumptions $(\boldsymbol{A_2})$ or $(\boldsymbol{A_2}{}')$ → the reachability of the optimum is guaranteed solely by the properties of the **mutation** operators.

2. The potential positive effects of **recombination are completely neglected.**

3. Notice that an EA without recombination will always visit the optimum whereas an EA without mutation but with a usual recombination operator does not have this guarantee.

the EA converges to the global optimum

Theorem 2.2.

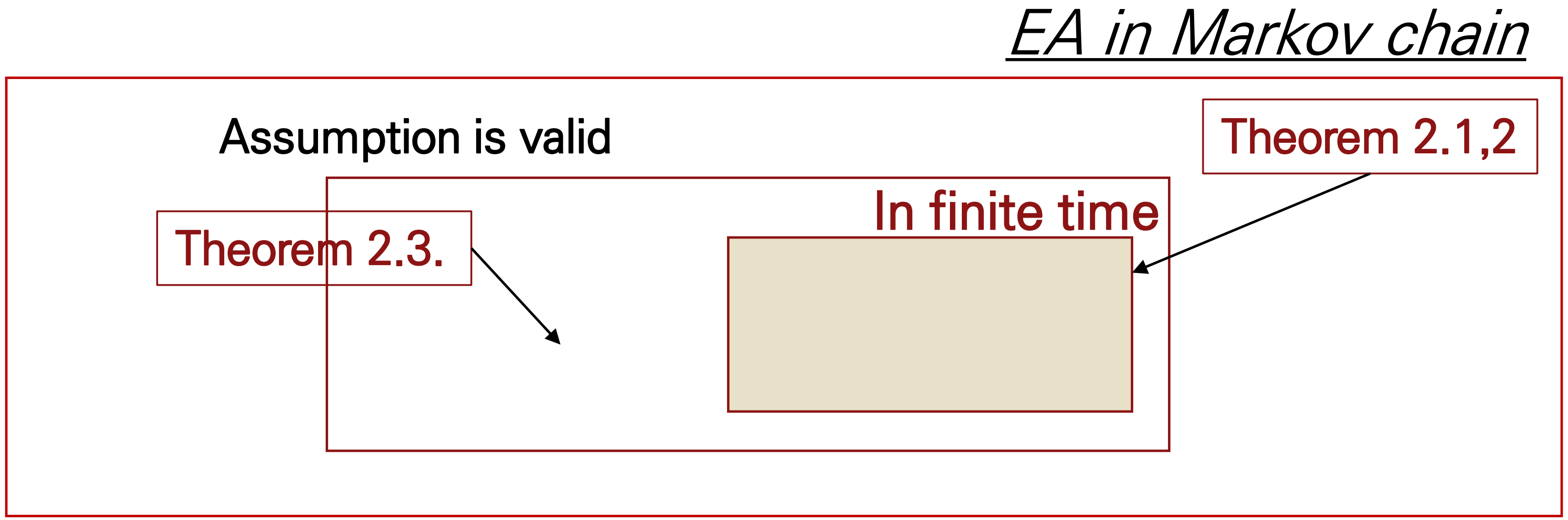
An evolutionary algorithm visits the global optimum infinitely often if assumption $(A_2{}')$ or the assumptions $(A_1), (A_2)$, and (A_3) are valid.

If the selection method only chooses from the offspring then the sequence $\{F_k : k \geq 0\}$ will not converge to the global optimum, even if assumption (A_4) is valid.

The assumptions and their implications presented so far are valid or the vast majority of evolutionary algorithms ***with finite search space and time homogeneous transitions.***

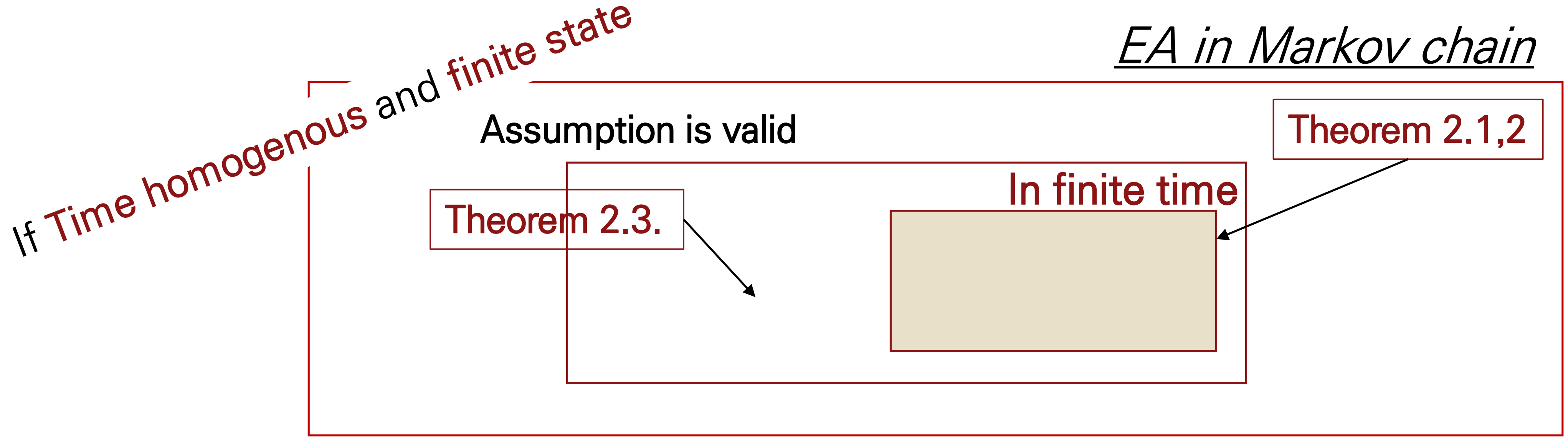
in summary

- If the transition operator for a Markov chain does not change across transitions, the Markov chain is called time homogenous
  - A nice property of time homogenous Markov chains is that as the chain runs for a long time and $t \to \infty$, **the chain will reach an equilibrium that is called the chain's stationary distribution:**
    - $p(x^{t+1}|x^{t}) = p(x^{t}|x^{t-1})\ as\ t \to \infty$



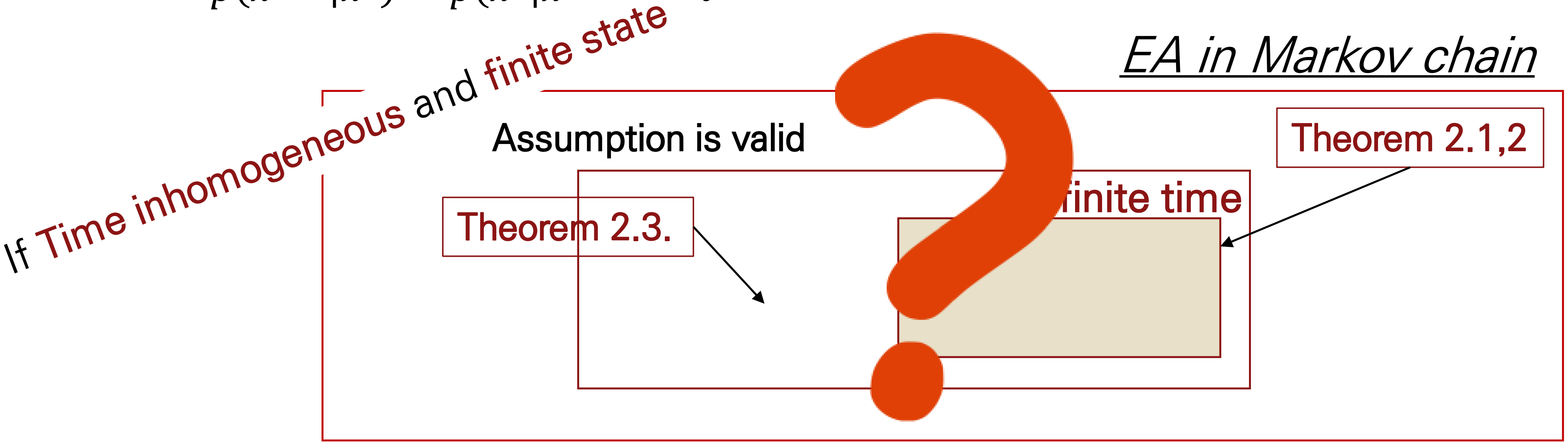
in summary

- If the transition operator for a Markov chain does not change across transitions, the Markov chain is called time homogenous
  - A nice property of time homogenous Markov chains is that as the chain runs for a long time and $t \to \infty$, **the chain will reach an equilibrium that is called the chain's stationary distribution:**
    - $p(x^{t+1}|x^{t}) = p(x^{t}|x^{t-1})$ *as* $t \to \infty$



What if?

- If the transition operator for a Markov chain does not change across transitions, the Markov chain is called time homogenous
  - A nice property of time homogenous Markov chains is that as the chain runs for a long time and $t \rightarrow \infty$, **the chain will reach an equilibrium that is called the chain's stationary distribution:**
    - $p(x^{t+1}|x^{t}) = p(x^{t}|x^{t-1})$ as $t \rightarrow \infty$



Stochastic convergence with time inhomogeneous transitions

- *Condition 1*
  - requires the precondition that the optimum **will be found in finite time with probability one.**
- *Condition 2*
  - if the optimum is not guaranteed to stay in the population, then it is necessary that it will be **found again**.
- *Condition 3*
  - In order to prevent everlasting oscillation of the sequence, it must be ensured that the event of finding the optimum happens infinitely often whereas the event of loosing the optimum happens only finitely often.

Stochastic convergence with time inhomogeneous transitions

- *Condition 1*
  - requires the precondition that the optimum **will be found in finite time with probability one.**
- *Condition 2*
  - if the optimum is not guaranteed to stay in the population, then it is necessary that it will be **found again.**

✓ **the decisive property**

- *Condition 3*
  - In order to prevent everlasting oscillation of the sequence, it must be ensured that the event of finding the optimum happens infinitely often whereas the event of loosing the optimum happens only finitely often.

Using Borel–Cantelli Lemma and its extension

* the Borel–Cantelli lemma is a theorem about sequences of events

Since EAs have the Markov property the condition is as follows:

Let $\boldsymbol{\alpha_k}$ be the probability of loosing the optimum and
$\boldsymbol{\beta_k}$ the probability of finding the optimum at step k

Lemma 1.8 (Borel-Cantelli). *(i) Let $A_i \in \mathcal{A}$, $i \in \mathbb{N}$. Then*

$$\sum_{i=1}^{\infty} P(A_i) < \infty \quad \Rightarrow \quad P(\limsup_{i\to\infty} A_i) = 0.$$

(ii) Assume that $A_i \in \mathcal{A}$, $i \in \mathbb{N}$, are independent. Then

$$\sum_{i=1}^{\infty} P(A_i) = \infty \quad \Rightarrow \quad P(\limsup_{i\to\infty} A_i) = 1.$$

If $\sum_k \boldsymbol{\alpha_k} < \infty$ and $\sum_k \boldsymbol{\beta_k} = \infty$,
then the event of loosing the optimum happens finitely often with probability 1
whereas the probability of visiting the optimum happens infinitely often with probability one.

✓ the probability of loosing the optimum must decrease
faster than the probability of finding the optimum.

Using Borel–Cantelli Lemma and its extension

* the Borel–Cantelli lemma is a theorem about sequences of

Since EAs have the Markov property the condition is as follows:

Let α_k be the probability of loosing the optimum ...

β_k the probability of finding the opt...

Lemma ...

... $\in \mathbb{N}$, *are independent. Then*

$$\sum_{i=1}^{\infty} P(A_i) = \infty \quad \Rightarrow \quad P\left(\limsup_{i \to \infty} A_i\right) = 1.$$

Since most of these results are specialized to certain combinations of variation and selection operators, it is refrained from reproducing all assumptions here

t... ...ing the optimum happens finitely often with probability 1

w... ...the probability of visiting the optimum happens infinitely often with probability one.

✓the probability of loosing the optimum must decrease faster than the probability of finding the optimum.

Finite Time Behavior in Finite Space and Discrete Time

- The examination of the finite time behavior of evolutionary algorithms cannot be treated in the same general manner as it is possible for the limit behavior.

Pseudo-Boolean optimization problem

Maximizing real-valued fitness functions with domain $\mathrm{X} = \mathbf{B}^l = \{0,1\}^l$.

Assumption

- the time of calculating the fitness value $f(x)$ for $x \in \mathbf{B}^l$ is bounded by a polynomial in l.

- Since the population size is finite, the number of fitness evaluations is an appropriate measure to assess the efficiency of an evolutionary algorithm

Evidently, needs a stopping rule
that indicates the termination of the stochastic process.

about terminal condition

- Let H_k contain the information available to the process until iteration k.

- **the stopping rule $\boldsymbol{\tau(H_k)}$**
  - termination of the process at step k if it evaluates to 1
  - continuation of the process if it evaluates to 0.

✓ **Note that,** a stopping rule induces a random stopping time $S = \min\{k \geq 0 : \tau(H_k) = 1\}$

Finite Time Behavior in Finite Space and Discrete Time

Definition 3.1.

Let $F_k^* = \max\{F_{k-1}^*, F_k\}\ for\ k \geq 1$ and $F_0^* = F_0$ denote the best fitness value found until iteration $k \geq 0$.

An evolutionary algorithm is said to be efficient for a problem class C if $E[S] \leq poly_1(l)$ and $P\{F_S^* = f^*\} \geq \frac{1}{poly_2(l)}$ for every instance of C,

where $poly_1(\cdot)$ and $poly_2(\cdot)$ are two polynomial functions of the problem dimension l.

The association of the term *"efficient"* with this criterion is justified by the algorithmic technique known as *probability amplification* or *probability boosting*

Finite Time Behavior in Finite Space and Discrete Time

Definition 3.1.

Let $F_k^* = \max\{F_{k-1}^*, F_k$ $1/\text{poly}_2(\ell) \le \mathsf{P}\{F_S^* = f^*\} < 1$ fitness value found until iteration $k \ge 0$.

An evolutio after r independent runs of the EA is at most $\left(1 - \frac{1}{poly_2(l)}\right)^r$

if $E[S] \le poly_1(l)$ and $P\{F_S^* = f^*\} \ge \frac{1}{poly_2(l)}$ for every instance of C,

where $poly_1($ Choose $r = k \cdot poly_2(l)$ with $k \in$ ne problem dimension l.

a total expected runtime "$r \cdot E(S)$"

whereas the probability of not finding the optimum in r **runs decreases exponentially in k.**

The association of the term *"efficient"* with the algorithmic technique known as *probab*

Again about terminal condition

(M$_1$) An individual $x \in \mathbb{B}^\ell$ is mutated by drawing an index uniformly at random and inverting the associated entry in x.

(M$_2$) An individual $x \in \mathbb{B}^\ell$ is mutated by inverting each entry in x independently with probability $p \in (0, 1)$.

+

The results are based on bounds on the expected first hitting time $E(T)$

Assume it can be shown that $E[T] \le poly_1(l)$ for every instance of a specific problem class C where the bound $poly_1(l)$ is explicitly known.

Since $P\{F_s^* = f^*\} =$

we obtain $P\{F_s^* \neq f^*\} = \mathrm{P}\{\mathrm{T} > \mathrm{s}\} \le \frac{E[T]}{s}$ ($\because$ Markov inequality)

If the stopping time is set to $s = c \cdot poly_1(l)$ with $c \ge 1 + 1/poly_2(l)$ for an arbitrary polynomial $poly_2(l)$

then $P\{F_s^* = f^*\} \ge 1 - \frac{1}{c} \ge \frac{1}{poly_2(l)+1}$ for every instance of problem class C

Again about terminal condition

(M$_1$) An individual $x \in \mathbb{B}^\ell$ is mutated by drawing an index uniformly at random and inverting the associated entry in x.

(M$_2$) An individual $x \in \mathbb{B}^\ell$ is mutated by inverting each entry in x independently with probability $p \in (0, 1)$.

the development of a polynomial upper bound for $E[T]$ is sufficient for proving the efficiency of the evolutionary algorithm for a specific problem class.

Since $P\{F_s^* = f^*\} = \mathrm{P}\{\mathrm{T} \le \mathrm{s}\}$ for every $s \ge 0$,

we obtain $P\{F_s^* \neq f^*\} = \mathrm{P}\{\mathrm{T} > \mathrm{s}\} \le \frac{E[T]}{s}$ ($\because$ Markov inequality)

If the stopping time is set to $s = c \cdot poly_1(l)$ with $c \ge 1 + 1/poly_2(l)$ for an arbitrary polynomial $poly_2(l)$

then $P\{F_s^* = f^*\} \ge 1 - \frac{1}{c} \ge \frac{1}{poly_2(l)+1}$ for every instance of problem class C

Finite Time Behavior in Finite Space and Discrete Time

Definition 3.2.

A function $f: \boldsymbol{B}^l \rightarrow \boldsymbol{R}$ is said to be modular if $f(x \wedge y) + f(x \vee y) = f(x) + f(y)$ for $all\ x, y \in \boldsymbol{B}^l$.

Theorem 3.1.

Let the fitness function $f: \boldsymbol{B}^l \rightarrow \boldsymbol{R}$ be modular.
If the evolutionary algorithm only uses mutation and elitist selection,
(a) $E[T] \geq l\ log\ l$ under (M_1)
(b) $E[T] = \Omega(l\ log\ l)$ under (M_2) with $p = \frac{1}{l}$ (c) $E[T] \leq l\ (log\ l + 1)$ under (M_1). (d) $E[T] = O(l\ log\ l)$ under (M_2) with $p = \frac{1}{l}$

Finite Time Behavior in Finite Space and Discrete Time

Proposition 3.1.
The expected first hitting time of the $(1+1)$–EA with fitness function (1) is bounded by
(a) $\mathsf{E}[\,T\,] \le \ell^2$ under (M_1),
(b) $\mathsf{E}[\,T\,] \le \ell^2\,(\exp(1)-1)$ under (M_2) with $p = 1/\ell$.

Proposition 3.2.
Let the unimax fitness function $f : \mathbb{B}^\ell \to \mathbb{R}$ be the long "Root2"-path problem given in [35]
The expected first hitting time of the $(1+1)$-EA can be bounded by
(a) $\mathsf{E}[\,T\,] \ge 3 \cdot \ell \cdot 2^{(\ell-1)/2} - 2\,\ell$ under (M_1),
(b) $\mathsf{E}[\,T\,] \le (\ell^3 - \ell)\,\exp(1)/2$ under (M_2) with $p = 1/\ell$,
if the EA starts at the bottom of the increasing path.

Finite Time Behavior in Finite Space and Discrete Time

Definition 3.5.
A function $f : \mathbb{B}^\ell \to \mathbb{R}$ is called *almost-positive* if the coefficients of all nonlinear terms are non-negative. □

An instance of this problem class is the pseudo-boolean function

$$f(x) = \ell - \sum_{i=1}^{\ell} x_i + (\ell + 1) \prod_{i=1}^{\ell} x_i \,. \tag{2}$$

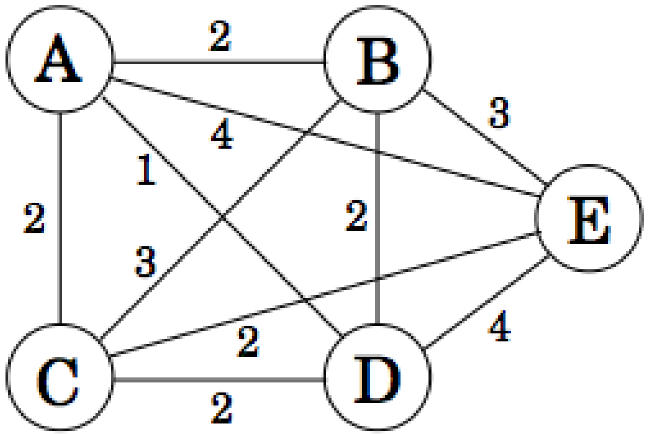
Proposition 3.3.
The expected first hitting time of the $(1+1)$-EA with fitness function (2) can be bounded by
(a) $\mathsf{E}[\,T\,] = \infty$ under (M_1), unless being started at the optimum;
(b) $\mathsf{E}[\,T\,] \geq \ell^\ell$ under (M_2) with $p = 1/\ell$ with worst starting point;
(c) $\mathsf{E}[\,T\,] \geq [\,(\ell+1)^\ell - 1]/2^\ell$ under (M_2) with $p = 1/\ell$ and random starting point.

Application of the algorithm for an NP–C/NP–hard example problem

- Traveling Salesman Problem
  - Finding a shortest closed tour
    - Visiting each node of a given graph with given edge length exactly once.

Figure 3: Traveling Salesman Problem and Ants build solutions, that is paths, from a source to a destination.

- Let $G = (X, E, W)$ be a complete weight graph,
  - $X = (x_1, x_2, \dots, x_n)\ (n \geq 3)$
  - $E = \{e_{ij} | x_i, x_j \in X\}$
  - $W = \{w_{ij} | w_{ij} \geq 0 \; and \; w_{ii} = 0, \; for \; all \; i, j \in \{1, 2, \dots, n\}\}$

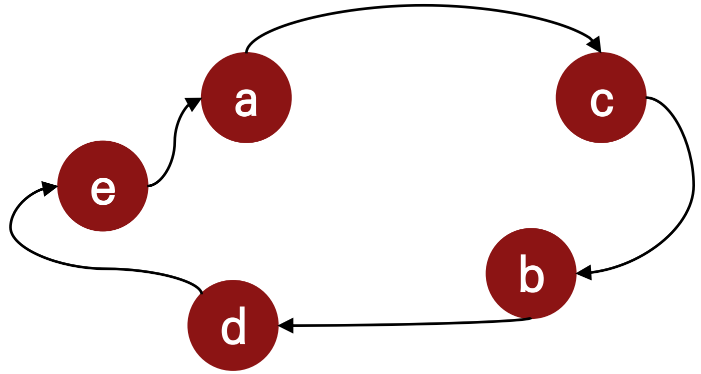


Application of the algorithm for an NP−C/NP−hard example problem

```
procedure EA;

  begin
    t := 0;
    initializePopulation(P(0));
    evaluate(P(0));
    repeat
      P' := selectForVariation(P(t));
      recombine(P');
      mutate(P');
      evaluate(P');
      P(t + 1) := selectForSurvival(P(t), P');
      t := t + 1;
    until terminate = true;
  end;
```

- Representation of feasible solution
  - Each chromosome is a pair of strings; (S, P)
    - S: the successors of the cities in a Hamiltonian cycle, which has n genes.
    - P; containing the predecessors of all the cities.
      - the cycle acbdea is represented by the (S, P) pair (cdbea, ecabd)



Application of the algorithm for an NP–C/NP–hard example problem

```
procedure EA;

  begin
    t := 0;
    initializePopulation(P(0));
    evaluate(P(0));
    repeat
      P′ := selectForVariation(P(t));
      recombine(P′);
      mutate(P′);
      evaluate(P′);
      P(t + 1) := selectForSurvival(P(t), P′);
      t := t + 1;
    until terminate = true;
  end;
```

- Initialization
  - Population size: about 50
  - p random permutation of size n
- Fitness of solution
  - for $i \in X$,
    $F_i = (C_w - C_i) + (C_w - C_b)/3$
    - C_w: *the path length of the worst solution in current population*
    - C_b: *the path length of the best solution in current population*
    - C_i: *the path length of* i

Application of the algorithm for an NP−C/NP−hard example problem

```
procedure EA;

  begin
    t := 0;
    initializePopulation(P(0));
    evaluate(P(0));
    repeat
      P′ := selectForVariation(P(t));
      recombine(P′);
      mutate(P′);
      evaluate(P′);
      P(t + 1) := selectForSurvival(P(t), P′);
      t := t + 1;
    until terminate = true;
  end;
```

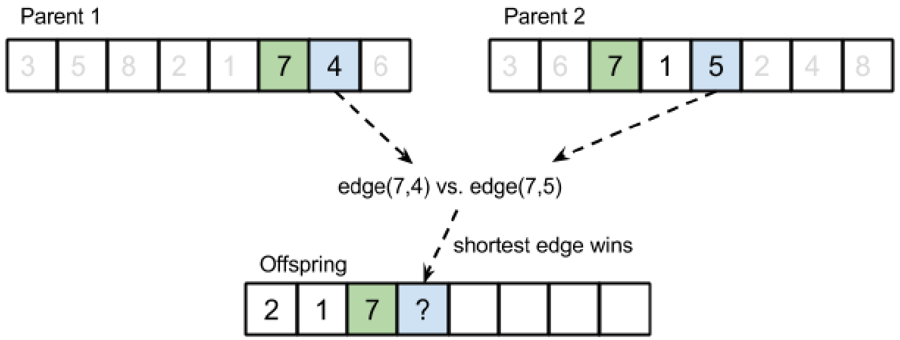
- Initialization
  - Population size: about 50
  - p random permutation of size n
- Fitness of solution
  - for $i \in X$,
    $F_i = (C_w - C_i) + (C_w - C_b)/3$
    - C_w: *the path length of the worst solution in current population*
    - C_b: *the path length of the best solution in current population*
    - C_i: *the path length of* i

Application of the algorithm for an NP–C/NP–hard example problem

```
procedure EA;

  begin
    t := 0;
    initializePopulation(P(0));
    evaluate(P(0));
    repeat
      P′ := selectForVariation(P(t));
      recombine(P′);
      mutate(P′);
      evaluate(P′);
      P(t + 1) := selectForSurvival(P(t), P′);
      t := t + 1;
    until terminate = true;
  end;
```

- Crossover Operator [3]



- Mutation Operator
  - for $i \in X$,

    $F_i = (C_w - C_i) + (C_w - C_b)/3$

    – C_w: *the path length of the worst solution in current population*

    – C_b: *the path length of the best solution in current population*

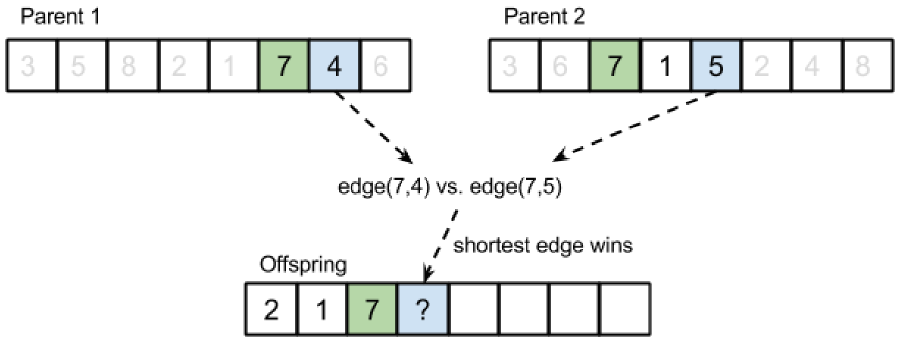
    – C_i: *the path length of* i

Application of the algorithm for an NP−C/NP−hard example problem

```
procedure EA;

  begin
    t := 0;
    initializePopulation(P(0));
    evaluate(P(0));
    repeat
      P′ := selectForVariation(P(t));
      recombine(P′);
      mutate(P′);
      evaluate(P′);
      P(t + 1) := selectForSurvival(P(t), P′);
      t := t + 1;
    until terminate = true;
  end;
```

- Crossover Operator [3]



- Mutation Operator
  - As mutation operator edge inversion is used with a mutation rate of 50%. In edge inversion, **two cities are selected randomly and the order of all cities between them is reversed.**

Application of the algorithm for an NP−C/NP−hard example problem

```
procedure EA;

  begin
    t := 0;
    initializePopulation(P(0));
    evaluate(P(0));
    repeat
      P' := selectForVariation(P(t));
      recombine(P');
      mutate(P');
      evaluate(P');
      P(t + 1) := selectForSurvival(P(t), P');
      t := t + 1;
    until terminate = true;
  end;
```

- Replacement
  - As replacement procedure, all individuals are replaced by the newly created ones apart from **the best individual, the elite**.
- Termination
  - when it fails to improve the best solution within 30 subsequent generations.

Application of the algorithm for an NP–C/NP–hard example problem

```
procedure EA;

  begin
    t := 0;
    initializePopulation(P(0));
    evaluate(P(0));
    repeat
      P' := selectForVariation(P(t));
      recombine(P');
      mutate(P');
      evaluate(P');
      P(t + 1) := selectForSurvival(P(t), P');
      t := t + 1;
    until terminate = true;
  end;
```

- Replacement
  - As replacement procedure, all individuals are replaced by the newly created ones apart from the best individual, the elite.
- Termination
  - when it fails to improve the best solution within 30 subsequent generations.

Application of the algorithm for an NP–C/NP–hard example problem

```
procedure EA;

  begin
    t := 0;
    initializePopulation(P(0));
    evaluate(P(0));
    repeat
      P′ := selectForVariation(P(t));
      recombine(P′);
      mutate(P′);
      evaluate(P′);
      P(t + 1) := selectForSurvival(P(t), P′);
      t := t + 1;
    until terminate = true;
  end;
```

- Comparison with exact solver[3]

Benchmark	Exact Solver		Genetic Algorithm (average over 10 runs)	
	Cycle Cost	Computation Time	Cycle Cost	Computation Time
att48	33523.71	10 sec	33546.76 (+0,07%)	1 sec
kroD100	21294.29	241 sec	21416.59 (+0,57%)	13 sec

References

[1] Rashid, Mahmood A., et al. "Mixing energy models in genetic algorithms for on-lattice protein structure prediction." *BioMed research international* 2013 (2013).

[2] Merz, Peter, and Bernd Freisleben. "Memetic algorithms for the traveling salesman problem." *Complex Systems* 13.4 (2001): 297–346.

[3] Ahmed, Zakir H. "Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator." *International Journal of Biometrics & Bioinformatics (IJBB)* 3.6 (2010): 96.

[4] 문병로, "쉽게 배우는 유전 알고리즘: 진화적 접근법" (2008)