

ФГБОУ ВПО  
«Сибирский государственный аэрокосмический университет  
имени академика М.Ф. Решетнева»

Сергиенко Антон Борисович

**Генетический алгоритм. Стандарт. v.3.5**

Красноярск – 2013

# Оглавление

<b>Условные обозначения</b>	<b>3</b>
<b>Введение</b>	<b>4</b>
<b>1 Постановка задачи оптимизации</b>	<b>5</b>
<b>2 Определения генетического алгоритма</b>	<b>7</b>
<b>3 Общая модель стандартного генетического алгоритма</b>	<b>9</b>
<b>4 Схема работы стандартного генетического алгоритма на бинарных строках</b>	<b>13</b>
<b>5 Описание стандартного генетического алгоритма на бинарных строках</b>	<b>15</b>
<b>6 Описание операторов</b>	<b>19</b>
6.1 Инициализация популяции . . . . .	19
6.2 Вычисление функции пригодности . . . . .	19
6.3 Селекция . . . . .	22
6.4 Скрещивание . . . . .	26
6.5 Мутация . . . . .	30
6.6 Формирование нового поколения . . . . .	31
<b>7 Решение задачи оптимизации на вещественных строках с помощью сГА</b>	<b>33</b>
7.1 Стандартное представление целого числа — номер узла в сетке дискретизации .	33
7.2 Стандартный рефлексивный Грей-код . . . . .	35
<b>8 Вектор параметров генетического алгоритма</b>	<b>39</b>
<b>Заключение</b>	<b>42</b>
<b>Литература</b>	<b>43</b>

# Условные обозначения

$a \in A$  — элемент  $a$  принадлежит множеству  $A$ .

$\bar{x}$  — обозначение вектора.

$\arg f(x)$  — возвращает аргумент  $x$ , при котором функция принимает значение  $f(x)$ .

$\text{Random}(X)$  — случайный выбор элемента из множества  $X$  с равной вероятностью.

$\text{Random}(\{x^i \mid p^i\})$  — случайный выбор элемента  $x^i$  из множества  $X$ , при условии, что каждый элемент  $x^i \in X$  имеет вероятность выбора равную  $p^i$ , то есть это обозначение равнозначно предыдущему.

$\text{random}(a, b)$  — случайное действительное число из интервала  $[a; b]$ .

$\text{int}(a)$  — целая часть действительного числа  $a$ .

$\mu(X)$  — мощность множества  $X$ .

**Замечание.** Оператор присваивания обозначается через знак « $=$ », так же как и знак равенства.

**Замечание.** Индексация всех массивов в документе начинается с 1. Это стоит помнить при реализации алгоритма на С-подобных языках программирования, где индексация начинается с нуля.

**Замечание.** Вызывание трех функций:  $\text{Random}(X)$ ,  $\text{Random}(\{x_i \mid p_i\})$ ,  $\text{random}(a, b)$  — происходит каждый раз, когда по ходу выполнения формул, они встречаются. Если формула итерационная, то нельзя перед ее вызовом один раз определить, например,  $\text{random}(a, b)$  как константу и потом её использовать на протяжении всех итераций неизменной.

**Замечание.** Надстрочный индекс может обозначать как возведение в степень, так и индекс элемента. Конкретное обозначение определяется в контексте текста, в котором используется формула с надстрочным индексом.

**Замечание.** Если у нас имеется множество векторов, то подстрочный индекс обозначает номер компоненты конкретного вектора, а надстрочный индекс обозначает номер вектора во множестве, например,  $\bar{x}^i \in X$  ( $i = \overline{1, N}$ ),  $\bar{x}_j^i \in \{0; 1\}$ , ( $j = \overline{1, n}$ ). В случае, если вектор имеет свое обозначение в виде подстрочной надписи, то компоненты вектора проставляются за скобками, например,  $(\bar{x}_{\max})_j = 0$  ( $j = \overline{1, n}$ ).

**Замечание.** При выводе матриц и векторов элементы могут разделяться как пробелом, так и точкой с запятой, то есть обе записи  $(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)^T$  и  $(1; 1; 1; 1; 1; 1; 1; 1)^T$  допустимы.

**Замечание.** При выводе множеств элементы разделяются только точкой с запятой, то есть допустима только такая запись:  $\{1; 1; 1; 1; 1; 1; 1\}^T$ .

# Введение

На данный момент генетический алгоритм (ГА) является одним из наиболее исследуемых и развивающихся алгоритмов глобальной оптимизации. К сожалению, до сих пор, по крайней мере, в российской литературе, строго описания алгоритма автором не встречалось. Обычно даны лишь общие рекомендации, которые допускают разнотечение в процессе программирования. Либо предложенные схемы применимы лишь для решения тестовых задач. Из-за этого невозможно строго сопоставить друг другу различные исследования по эффективности алгоритмов. Как сказал один из исследователей: «Количество различных эволюционных алгоритмов совпадает с количеством исследователей, работающих в данной области!» [1]. Данная работа призвана определить некий стандарт генетического алгоритма (далее Стандарт).

Данный документ представляет его версию 3.5 от 6 июня 2013 г.

Последнюю версию документа можно найти по адресу:

<https://github.com/Harrix/Standard-Genetic-Algorithm>

**Цель Стандарта** — предоставить исследователям единое описание генетического алгоритма для последующей оценки и сравнения эффективности ГА и других алгоритмов.

Задача, которую решает Стандарт — определить технологию реализации сГА (стандартный генетический алгоритм). Технология обеспечивает единое понимание и одинаковую реализацию.

В качестве объекта в Стандарте выступает сГА однокритериальной оптимизации на бинарных или вещественных строках.

Стандарт предназначен, в первую очередь, для следующих пользователей:

- исследователей, занимающиеся разработкой и исследованиями модификаций ГА (студенты, аспиранты, докторанты, любители-энтузиасты и др.);
- экспертов, кто может участвовать в оценке результатов работ исследователей.

Автор выражает благодарности людям, помогающим в работе данного Стандарта: Бухтоярову Владимиру Викторовичу ([vladber@list.ru](mailto:vladber@list.ru)), Галушину Павлу Викторовичу ([galushin\\_pavel@mail.ru](mailto:galushin_pavel@mail.ru)), Семенкину Евгению Станиславовичу ([eugeneselemenkin@yandex.ru](mailto:eugeneselemenkin@yandex.ru)), Сергиенко Роману Борисовичу ([romaserg@list.ru](mailto:romaserg@list.ru)), Сопову Евгению Александровичу ([es\\_gt@mail.ru](mailto:es_gt@mail.ru)), Сопову Сергею Александровичу ([sopov.sa@gmail.com](mailto:sopov.sa@gmail.com)).

В данной версии документа по сравнению с предыдущем произведен перевод на  $\text{\LaTeX}$  систему и платформу GitHub, рисунки все переведены в векторный формат, исправлены ошибки и добавлены некоторые замечания.

С автором можно связаться по адресу [sergienkoanton@mail.ru](mailto:sergienkoanton@mail.ru) или <http://vk.com/harrix>.

Сайт автора, где публикуются последние новости: <http://blog.harrix.org/>, а проекты расположаются по адресу <http://harrix.org/>.

# Глава 1

## Постановка задачи оптимизации

Рассмотрим постановку задачи оптимизации в общем случае, решаемую стандартным генетическим алгоритмом (сГА).

Необходимо найти:

$$\begin{aligned}\bar{x}_{max} &= \arg \max_{\bar{x} \in X} f(\bar{x}), \text{ где} \\ g_i(\bar{x}) &\leq 0, i = \overline{1, m_1}, \\ h_j(\bar{x}) &= 0, j = \overline{1, m_2}.\end{aligned}\tag{1.1}$$

Здесь:

$X$  — множество всех возможных решений,

$f(\bar{x})$  — функционал, определенный на данном множестве, возвращающий действительное число из интервала  $(-\infty; \infty)$ ,

$m_1$  — число ограничений в виде неравенств,

$m_2$  — число ограничений в виде равенств.

$\bar{x} \in X$  имеет вид

$$\bar{x} = (x_1; \dots; x_i; \dots; x_n)^T. \tag{1.2}$$

В случае, если  $m_1 = 0$  и  $m_2 = 0$ , имеем задачу безусловной оптимизации. В противном случае — условной оптимизации.

В случае, если  $X$  — множество всех бинарных векторов длины  $n$  таких что  $x_i \in \{0; 1\}$  ( $i = \overline{1, n}$ ), то имеем задачу бинарной оптимизации (мощность поискового пространства  $X$  равна  $\mu(X) = 2^n$ ). Если  $X$  — множество всех вещественных векторов длины  $n$  таких что  $x_i \in \{Left_i; Right_i\}$  ( $i = \overline{1, n}$ ), то имеем задачу вещественной оптимизации.

Будем предполагать в дальнейшем, что  $f(\bar{x})$  может представлять собой многоэкстремальный функционал, и вычислению подлежат только значения  $f(\bar{x})$  (нет возможности вычислить производные от функционала и т. д.).

Стандартный генетический алгоритм, описанный в Стандарте, решает задачи оптимизации на бинарных и вещественных векторах. При решении задачи на вещественных строках происходит преобразования исходной задачи к задаче оптимизации на бинарной строках, на основе

стандартного представление целого числа — номера узла в сетке дискретизации или стандартным рефлексивным Грей-кодом.

**Пример.** Требуется найти бинарный вектор длины  $n = 8$ , сумма компонент которого максимальна:

$$\bar{x}_{max} = \arg \max_{\bar{x} \in X} f(\bar{x}), \text{ где} \quad (1.3)$$

$$f(\bar{x}) = \sum_{i=1}^n x_i.$$

Очевидно, что  $\bar{x}_{max} = (1; 1; 1; 1; 1; 1; 1; 1)^T$ , но алгоритм это «не знает», и его задача — найти это решение.

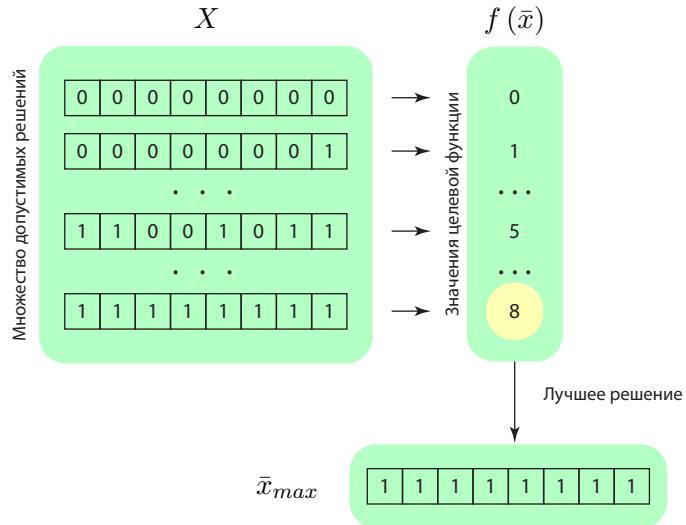


Рисунок 1.1. Пример задачи оптимизации

## Глава 2

# Определения генетического алгоритма

Введем некоторые понятия, которыми будем пользоваться в данном Стандарте.

**Стандартный генетический алгоритм** — генетический алгоритм, описанный в данном документе. Он считается базовым для исследования эффективности других ГА. Если не оговорено иное, то под генетическим алгоритмом понимается стандартный генетический алгоритм. Для обозначения также используется сокращение сГА.

**Бинарная строка** — вектор-столбец  $\bar{x}$  конечной длины  $n$ , компоненты которого могут принимать значения из множества  $\{0; 1\}$ . В принципе не совсем верное название, так как по сути это вектор-столбец, а не вектор-строка.

**Бинарный вектор** — то же самое, что и бинарная строка.

**Вещественная строка** — вектор-столбец  $\bar{x}$  конечной длины  $n$ , компоненты которого могут принимать значения из множества  $x_i \in \{Left_i; Right_i\} (i = \overline{1, n})$ . По своей сути не совсем верное название, так как по сути это вектор-столбец, а не вектор-строка.

**Вещественный вектор** — то же самое, что и вещественная строка.

**Решение** — любой элемент  $\bar{x}$  из множества  $X$ .

**Точка поискового пространства** — то же самое, что и решение.

**Возможное решение** — любое решение  $\bar{x}$  из множества  $X$ .

**Допустимое решение** — решение  $\bar{x}$  из множества  $X$ , которое удовлетворяет условиям  $g_i(\bar{x}) \leq 0 (i = \overline{1, m_1})$  и  $h_j(\bar{x}) = 0 (j = \overline{1, m_2})$ .

**Генотип** — то же самое, что и бинарная строка.

**Индивид** — то же самое, что и бинарная строка.

**Хромосома** — то же самое, что и бинарная строка.

**Ген** — компонент  $x_i$  бинарного вектора  $\bar{x} \in X$ .

**Бит** — компонент  $x_i$  бинарного вектора  $\bar{x} \in X$ .

**Фенотип** — некий объект, который был закодирован в бинарную строку при переходе от задачи оптимизации на произвольном пространстве поиска к задаче оптимизации на бинарных строках. Например, вещественный вектор преобразуется в бинарную строку с помощью кода Грея.

**Поколение** — одна итерация работы сГА.

**Популяция** — множество индивидов, обрабатываемых на текущем поколении.

**Целевая функция** — то же самое, что и функционал  $f(\bar{x})$ .

**Оптимизируемая функция** — то же самое, что и функционал  $f(\bar{x})$ .

**Функция пригодности** — преобразованная целевая функция для использования в генетическом алгоритме на бинарных строках.

**Пригодность** — значение функции пригодности.

**Поисковое пространство** — множество всех возможных векторов  $X$ .

**Множество решений** — то же самое что и поисковое пространство.

Остальные определения будут вводиться по мере их упоминания в тексте.

**Замечание.** Понятия «генотип», «хромосома», «индивиду» обозначают по своей сути одно и то же. Но в связи с тем, что генетический алгоритм имитирует настоящую эволюцию, то на различных этапах имитации более подходящими являются разные понятия из биологии.

Также стоит отметить, что здесь эти понятия отождествляются с бинарной строкой, как возможным решением, а не любым решением из первоначальной задачи оптимизации (там могут присутствовать также вещественные строки). Это объясняется тем, что в описанном в Стандарте генетическом алгоритме на уровне операторов используются именно бинарные строки. А задача вещественной оптимизации решается путем сведения к задаче бинарной оптимизации путем дискретизации поискового пространства, операторов кодирования и декодирования. Но это не означает, что в других вариантах генетического алгоритма, не описанных в Стандарте, понятия «генотип», «хромосома», «индивиду» будут отождествляться с бинарными строками.

# Глава 3

## Общая модель стандартного генетического алгоритма

Генетический алгоритм можно представить как некий стохастический оператор, который на выходе выдает **субоптимальное** решение  $\bar{x}_{submax}$  и значение функционала от этого решения  $f(\bar{x}_{submax})$ . То есть мы можем записать, что

$$\begin{pmatrix} \bar{x}_{submax} \\ f(\bar{x}_{submax}) \end{pmatrix} = GeneticAlgorithm \begin{pmatrix} X \\ f(\bar{x}) \\ g_i(\bar{x}) \\ h_j(\bar{x}) \\ Parameters \end{pmatrix}, \quad (3.1)$$

где  $i = \overline{1, m_1}$ ,  $j = \overline{1, m_2}$ ,

*GeneticAlgorithm* — непосредственно сам генетический алгоритм как оператор,

*Parameters* — параметры генетического алгоритма, которые определяет пользователь (будут рассмотрены ниже).

В виде схемы это можно представить так:

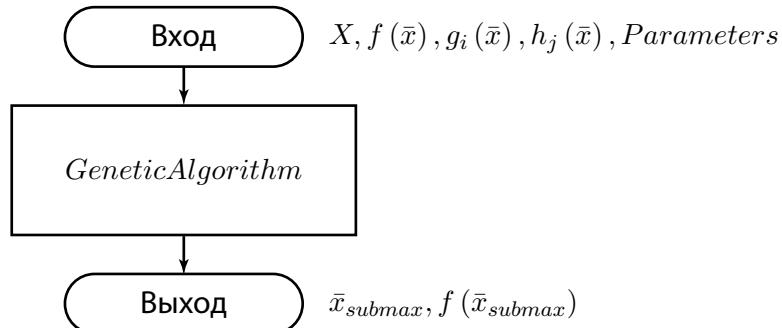


Рисунок 3.1. Модель черного ящика генетического алгоритма

Как было отмечено выше, стандартный генетический алгоритм (сГА) решает задачу оптимизации, как на бинарных строках, так и на вещественных. Поэтому стандартный генетический алгоритм включает в себя два алгоритма:

- стандартный генетический алгоритм на бинарных строках (*BinaryGeneticAlgorithm*);
- стандартный генетический алгоритм на вещественных строках.

При этом сГА на вещественных строках включает в себя сГА на бинарных строках, а также блок преобразования задачи вещественной оптимизации к задаче бинарной оптимизации и блок преобразования полученного бинарного решения к вещественному.

По этой причине вектор параметров сГА состоит из двух частей:

$$Parameters = \begin{pmatrix} ParametersOfBinaryGA \\ ParametersOfConvertingIntoBinaryGA \end{pmatrix}. \quad (3.2)$$

Здесь *ParametersOfBinaryGA* — параметры стандартного генетического алгоритма на бинарных строках, а *ParametersOfConvertingIntoBinaryGA* — параметры преобразования задачи оптимизации на вещественных векторах к задаче оптимизации на бинарных векторах. В случае если в конкретном случае решается задача бинарной оптимизации, то блок параметров *ParametersOfConvertingIntoBinaryGA* можно опустить.

Построим общую схему сГА, учитывая вышесказанное (рис. 3.2).

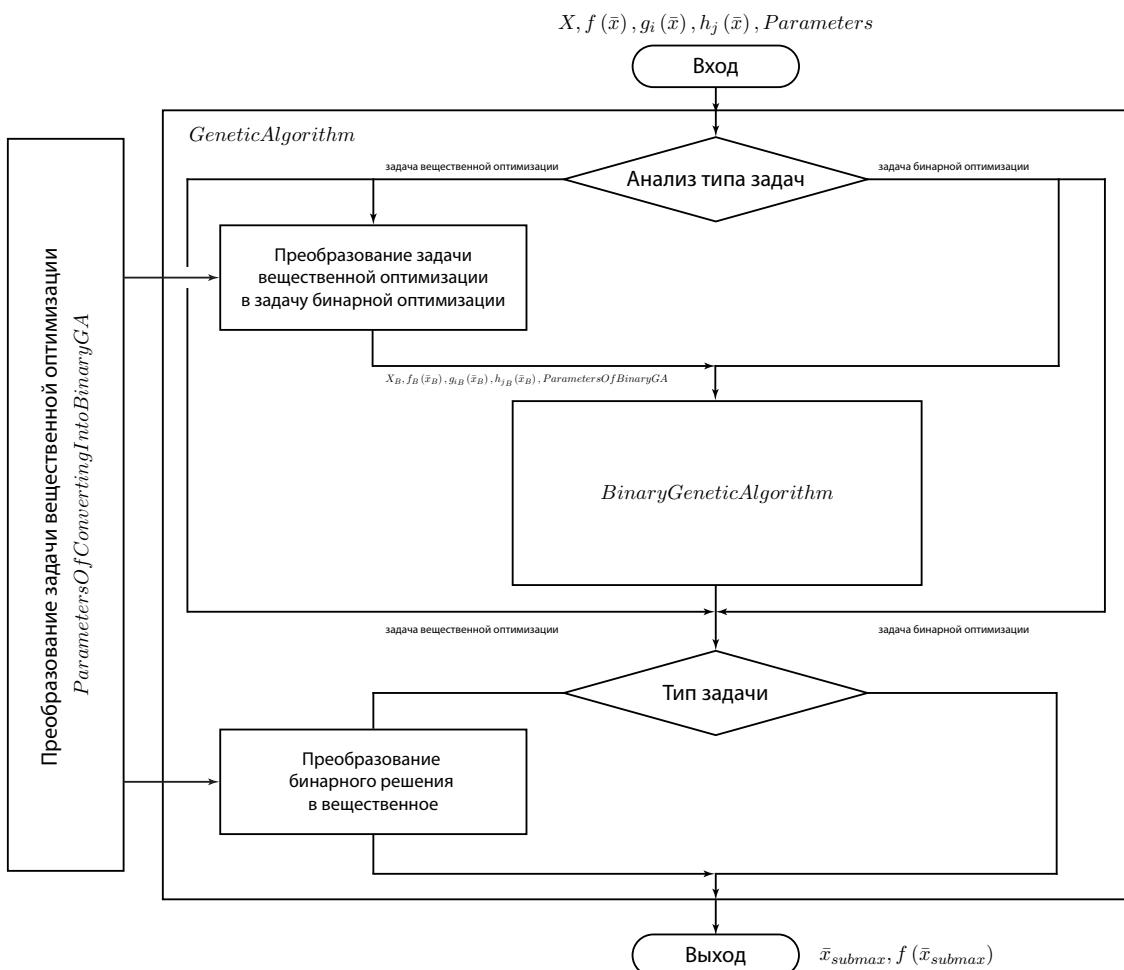


Рисунок 3.2. Общая модель стандартного генетического алгоритма

Распишем ее в виде алгоритма *GeneticAlgorithm*:

---

**Алгоритм 3.1.** Алгоритм *GeneticAlgorithm* (Часть 1)

**Начало алгоритма**

**Если**  $X$  представляет собой множество вещественных векторов **тогда**

**Начало блока.** *Преобразование задачи вещественной оптимизации в задачу бинарной оптимизации:*

Выполнить:

$$\begin{aligned} & \begin{pmatrix} X_B \\ f_B(\bar{x}_B) \\ g_{iB}(\bar{x}_B) \\ h_{jB}(\bar{x}_B) \\ ParametersOfBinaryGA \end{pmatrix} = \\ & = ConvertingIntoBinaryGA \begin{pmatrix} X \\ f(\bar{x}) \\ g_i(\bar{x}) \\ h_j(\bar{x}) \\ ParametersOfBinaryGA \\ ParametersOfConvertingIntoBinaryGA \end{pmatrix}; \end{aligned}$$

**Конец блока**

**Конец условия**

**Начало блока.** *Выполнение стандартного генетического алгоритма на бинарных строках:*

Если решается задача бинарной оптимизации, то подается вектор входных параметров самого алгоритма *GeneticAlgorithm* (без *ParametersOfConvertingIntoBinaryGA*, если он присутствует), иначе подается преобразованный вектор, полученный на предыдущем шаге:

$$A = \begin{cases} \left( \begin{array}{c} X \\ f(\bar{x}) \\ g_{iB}(\bar{x}) \\ h_j(\bar{x}) \\ ParametersOfBinaryGA \end{array} \right), \text{ где } X \text{ — множество бинарных векторов;} \\ \left( \begin{array}{c} X_B \\ f_B(\bar{x}_B) \\ g_{iB}(\bar{x}_B) \\ h_{jB}(\bar{x}_B) \\ ParametersOfBinaryGA \end{array} \right), \text{ где } X \text{ — множество вещественных векторов.} \end{cases}$$
$$\left( \begin{array}{c} \bar{x}_{submaxB} \\ f(\bar{x}_{submax})_B \end{array} \right) = BinaryGeneticAlgorithm(A);$$

**Конец блока**

**Начало блока.** *Тип задачи:*

Используем информацию, полученную на этапе «Анализ типа задачи»;

**Если**  $X$  представляет собой множество вещественных векторов **тогда**

**Возвратить**  $(\bar{x}_{submax}; f(\bar{x}_{submax}))^T$ ;

**Конец условия**

**Конец блока**

*Продолжение ниже*

---

---

**Алгоритм 3.2.** Алгоритм *GeneticAlgorithm* (Часть 2)

*Начало выше*

**Начало блока.** *Преобразование бинарного решения в вещественное:*

$$\begin{pmatrix} \bar{x}_{submax} \\ f(\bar{x}_{submax}) \end{pmatrix} = BinaryToReal \begin{pmatrix} \bar{x}_{submaxB} \\ f(\bar{x}_{submax})_B \\ ParametersOfConvertingIntoBinaryGA \end{pmatrix};$$

**Возвратить**  $(\bar{x}_{submax}; f(\bar{x}_{submax}))^T$ ;

**Конец блока**

**Конец алгоритма**

---

Операторы *ConvertingIntoBinaryGA* и *BinaryToReal* рассматриваются в главе 7. До него рассматривается только стандартный генетический алгоритм на бинарных строках.

**Замечание.** Существует вещественный генетический алгоритм, который работает непосредственно с вещественными строками без приведения к задаче бинарной оптимизации. В данном Стандарте он не рассматривается.

## Глава 4

# Схема работы стандартного генетического алгоритма на бинарных строках

В основу работы генетического алгоритма заложена имитация процесса эволюции живых организмов. Поэтому алгоритм представляет собой итерационный процесс (имитация смены поколений), работающий одновременно с несколькими взаимодействующими решениями (имитация популяции живых организмов).

Подробная схема, описывающая работу генетического алгоритма на бинарных строках (*BinaryGeneticAlgorithm*), представлена ниже на рисунке.

**Замечание.** Довольно часто в литературе (например, [2, с. 294]) рассматривается вариант, когда вначале отбираются потенциальные родители с помощью селекции (в «родительский пул»), а потом из них случайно отбираются родители для скрещивания. В данной модели генетического алгоритма это не происходит. Сразу после отбора двух родителей происходит скрещивание с целью получения одного потомка.

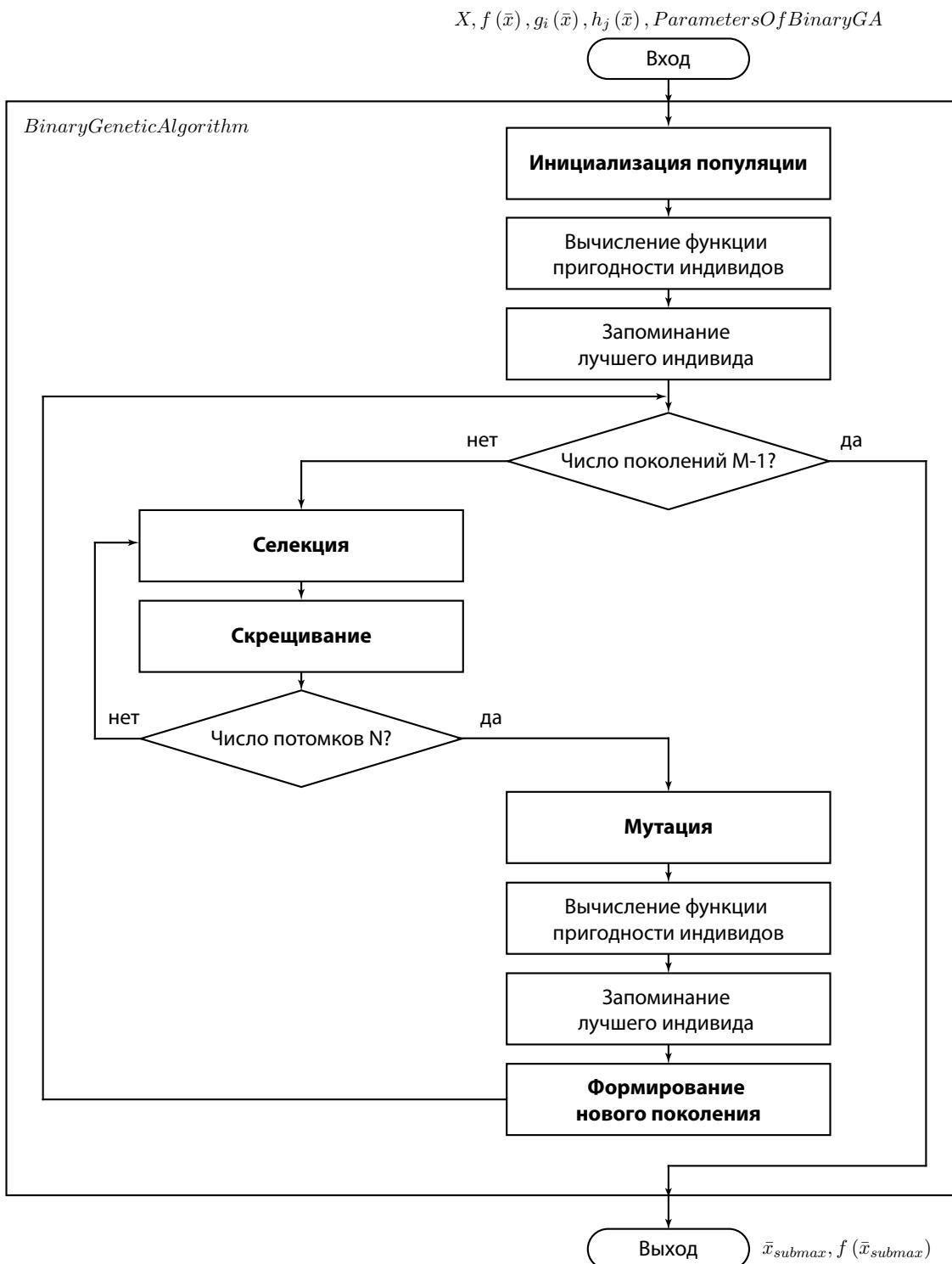


Рисунок 4.1. Схема генетического алгоритма на бинарных строках

# Глава 5

## Описание стандартного генетического алгоритма на бинарных строках

Теперь приступим непосредственно к описанию самого алгоритма *BinaryGeneticAlgorithm* согласно схеме, приведенной выше.

---

### Алгоритм 5.1. Алгоритм *BinaryGeneticAlgorithm* (Часть 1)

---

**Начало алгоритма**

**Начало блока. Инициализация популяции:**

$\text{Population} = \text{Initialization}(X)$ , где  $\text{Population} = \{\bar{x}^1; \bar{x}^2; \dots; \bar{x}^N\}$ ,  $\bar{x}^i \in X, i = \overline{1, N}$ ;

**Конец блока**

**Начало блока. Вычисление функции пригодности индивидов:**

$\text{Fitness} = \{f_{fit}(\bar{x}^1); f_{fit}(\bar{x}^2); \dots; f_{fit}(\bar{x}^N)\}$ ;

**Конец блока**

**Начало блока. Запоминание лучшего индивида и его значение целевой функции:**

$\overline{\text{Best}} = \arg \max_{\bar{x} \in \text{Population}} f(\bar{x})$ ;

$\text{BestFitness} = \max_{\bar{x} \in \text{Population}} f(\bar{x})$ ;

**Конец блока**

**Цикл.  $M - 1$  раз выполнять**

**Цикл. От  $j = 1$  до  $N$  выполнять**

**Начало блока. Селекция:**

Выбрать двух родителей:

$\overline{\text{Parent}}^1 = \text{Selection}(\text{Population}, \text{Fitness}, \text{DataOfSel})$ ;

$\overline{\text{Parent}}^2 = \text{Selection}(\text{Population}, \text{Fitness}, \text{DataOfSel})$ ;

$\overline{\text{Parent}}^1 \in X, \overline{\text{Parent}}^2 \in X$ ;

**Конец блока**

**Начало блока. Скрещивание:**

Получение потомка из родителей:

$\overline{\text{Child}}^j = \text{Crossover}(\overline{\text{Parent}}^1, \overline{\text{Parent}}^2, \text{DataOfCros})$ ;

**Конец блока**

**Конец цикла**

**Начало блока. Мутация всех потомков:**

Получение потомка из родителей:

*Продолжение ниже*

---

---

**Алгоритм 5.2.** Алгоритм *BinaryGeneticAlgorithm* (Часть 2)

**Начало выше**

$MutChildPopulation = Mutation(ChildPopulation, DataOfMut)$ ,

где  $ChildPopulation = \{\overline{Child}^1; \dots; \overline{Child}^N\}$ ,  $\overline{Child}^i \in X$ ,  $i = \overline{1, N}$ ;

$MutChildPopulation = \{\overline{MutChild}^1; \dots; \overline{MutChild}^N\}$ ,  $\overline{MutChild}^i \in X$ ,  $i = \overline{1, N}$ ;

**Конец блока**

**Начало блока. Вычисление функции пригодности индивидов:**

$FitnessOfMutChild = \{f_{fit}(\overline{MutChild}^1); f_{fit}(\overline{MutChild}^2); \dots; f_{fit}(\overline{MutChild}^N)\}$ ;

**Конец блока**

**Начало блока. Запоминание лучшего индивида и его значение целевой функции:**

Если есть потомок лучше сохраненного лучшего индивида  $\overline{Best}$ , то заменяем его новым индивидом с его значением целевой функции:

$\overline{Best} = \arg \max_{\bar{x} \in MutChildPopulation \cup \overline{Best}} f(\bar{x})$ ;

$BestFitness = \max_{\bar{x} \in MutChildPopulation \cup \overline{Best}} f(\bar{x})$ ;

**Конец блока**

**Начало блока. Формирование нового поколения:**

Из потомков и родителей формируется новое поколение, которое заменяет предыдущее поколение:

$$\begin{pmatrix} NewPopulation \\ NewFitness \end{pmatrix} = Forming \begin{pmatrix} Population \\ MutChildPopulation \\ Fitness \\ FitnessOfMutChild \\ DataOfForm \end{pmatrix};$$

$Population = NewPopulation$ ;

$Fitness = NewFitness$ ;

**Конец блока**

**Конец цикла**

**Начало блока. Выдаем лучшего индивида и его значение целевой функции:**

**Возвратить** пару  $(\overline{Best}, BestFitness)$ .

**Конец блока**

**Конец алгоритма**

---

Здесь,

$N$  — размер популяции,

$M$  — число поколений,

$DataOfSel$  — вектор параметров оператора селекции,

$DataOfCros$  — вектор параметров оператора скрещивания,

$DataOfMut$  — вектор параметров оператора мутации,

$DataOfForm$  — вектор параметров оператора формирования нового поколения.

При решении задачи нам известно количество вычислений целевой функции  $CountOfFitness$ , которые мы можем реализовать. Данная величина определяется по затратам

по вычислениям целевой функции в одной точке, временными ограничениями, экономической целесообразностью и другими причинами.

Вышеописанные переменные являются параметрами генетического алгоритма, варьируя которые мы получаем разные по эффективности реализации ГА. Во всех векторах параметров операторов главным параметром является тип данного оператора, так как существует несколько реализаций каждого из операторов.

При этом в сГА на бинарных строках число поколений и размер популяции берутся в таком соотношении, чтобы они были приблизительно одинаковыми. В таком случае мы можем исключить параметры  $M$  и  $N$  из числа определяемых пользователем, воспользовавшись формулой:

$$M = \text{int} \left( \sqrt{\text{CountOffFitness}} \right), \quad (5.1)$$

$$N = \text{int} \left( \frac{\text{CountOffFitness}}{M} \right).$$

Итак, определим вектор всех изменяемых параметров стандартного генетического алгоритма на бинарных строках:

$$\text{ParametersOfBinaryGA} = \begin{pmatrix} \text{CountOffFitness} \\ \text{DataOfSel} \\ \text{DataOfCros} \\ \text{DataOfMut} \\ \text{DataOfForm} \end{pmatrix}. \quad (5.2)$$

**Замечание.** В литературе часто в качестве критерия остановки алгоритма используется не число вычислений функции  $\text{CountOffFitness}$ , а время работы алгоритма или достижение заданной точности задачи или самого алгоритма.

Если взять в качестве критерия время работы алгоритма, то исследования по сравнению алгоритмов будут жестко привязаны к компьютерам, на которых проводились вычисления. При этом критерий будет характеризовать скорее ЭВМ, чем алгоритм, и результаты будет сложно проверить другими исследователями. Следует помнить, что часто в практических задачах основное время работы занимает вычисление целевой функции, а не работы самого алгоритма, которое пренебрежительно мало.

Если же взять в качестве критерия остановки достижение заданной точности алгоритмом (например,  $K$  поколений проходят без улучшения наилучшего решения  $\overline{\text{Best}}$ ), то сравнивать между собой можно только эволюционные алгоритмы, так как данный критерий не приемлем для других алгоритмов глобальной оптимизации.

Использование в качестве критерия остановки достижение заданной точности задачи (например, расстояние до точки глобального оптимума) не содержит каких либо явных противоречий при исследованиях различных алгоритмов, но в данной работе не рассматривается.

**Замечание.** Число вычислений функций  $\text{CountOffFitness}$  желательно выбирать из квадратов целых чисел:  $50^2, 100^2$  и т. д. Иначе настоящее число вычислений целевой функции будет меньше, чем  $\text{CountOffFitness}$ .

**Замечание.** *Population*, *MutChildPopulation*, *ChildPopulation* не являются подмножествами множества  $X$ , так как могут содержать одинаковые элементы, что противоречит определению подмножества.

**Замечание.** В главном цикле алгоритма число повторений равно  $M - 1$ , так как первые  $M$  вычислений функции пригодности приходится на вычисление пригодности индивидов после инициализации популяции.

**Замечание.** При реализации алгоритма следует следить, чтобы число вычислений целевой функции равнялось *CountOffFitness*. Часто встречаются реализации ГА, в которых целевая функция одного и того же индивида просчитывается несколько раз. Например, в схеме алгоритма для операторов селекции используется значения  $f_{fit}(\bar{x})$ , а при запоминании *BestFitness* используется значение  $f(\bar{x})$ . Иногда пересчитывают целевую функцию для определения как *BestFitness* так и  $f_{fit}(\bar{x})$  (трехкратное увеличение вычислений целевой функции). Рекомендуется хранить два массива: один для значений  $f(\bar{x})$ , другой для  $f_{fit}(\bar{x})$ , значения которого рассчитываются через первый массив. Данное замечание не касается модификации ГА, использующее хранение значение целевой функции для всех уже просчитанных индивидов, в которой прежде чем считать значение целевой функции индивид проверяют по базе.

**Замечание.** При запоминании лучшего индивида  $\overline{Best}$  и его значение функции пригодности в случае наступления ситуации, когда в популяции есть индивид с равной пригодностью, что и  $\overline{Best}$ , не нужно замещать  $\overline{Best}$  этим индивидом. Замещение происходит только в случае, когда находится индивид с лучшей, чем у  $\overline{Best}$ , пригодностью.

# Глава 6

## Описание операторов

Рассмотрим подробно каждый из операторов генетического алгоритма из его описания.

### 6.1 Инициализация популяции

**Инициализация популяции** — процесс формирования случайного массива с фиксированным числом  $N$  элементов из множества  $X$ . При этом должно выполняться условие, что для любых  $\bar{x}, \bar{y} \in X$  вероятности попадания в популяцию один или более раз одинаковы.

Итак, данный оператор определяется формулой:

$$\text{Initialization}(X) = \{\bar{x}^1; \bar{x}^2; \dots; \bar{x}^N\}, \bar{x}^i = \text{Random}(X), \bar{x}^i \in X, i = \overline{1, N}. \quad (6.1)$$

**Пример.** Допустим, решаем задачу из примера из формулы 1.3. Зададим размер популяции  $N = 6$ . Тогда в результате работы оператора инициализации мы можем получить популяцию:

$$\text{Population} = \left\{ \begin{array}{l} (0; 0; 1; 1; 0; 1; 1; 1)^T \\ (1; 0; 0; 0; 0; 0; 0; 0)^T \\ (1; 0; 0; 1; 1; 1; 1; 0)^T \\ (1; 1; 0; 1; 0; 1; 1; 1)^T \\ (0; 0; 1; 0; 0; 1; 1; 0)^T \\ (1; 0; 1; 1; 1; 1; 1; 1)^T \end{array} \right\}.$$

### 6.2 Вычисление функции пригодности

Генетический алгоритм не работает с функционалом  $f(\bar{x})$ , определяющий эффективность найденных решений. Как видно на схеме (рис. 4.1), ГА работает с функцией пригодности, которая для каждого индивида из популяции определяется по формуле:

$$f_{fit}(\bar{x}^k) = f_1(f_2(f(\bar{x}^k), g_i(\bar{x}^k), h_j(\bar{x}^k)), Y), \quad (6.2)$$

где  $\bar{x}^k \in Y$ ,  $Y = \text{Population} \cup \text{MutChildPopulation}$ ,  $k = \overline{1, N}$ ,  $i = \overline{1, m_1}$ ,  $j = \overline{1, m_2}$ .

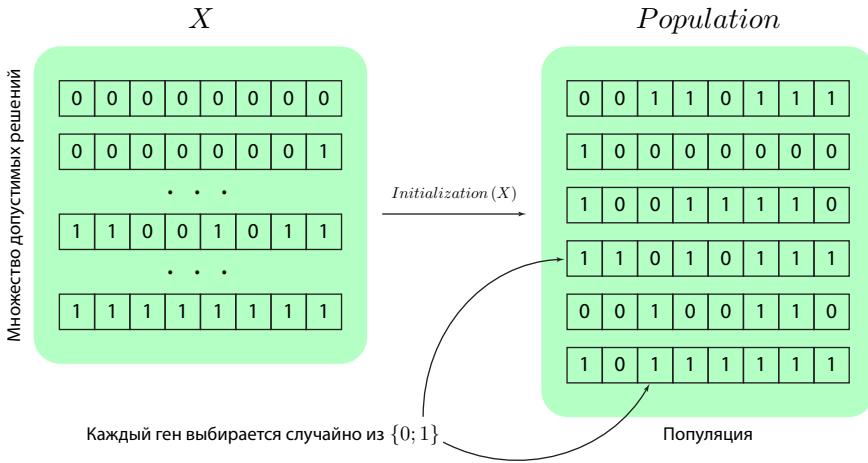


Рисунок 6.1. Инициализация популяции

Запишем более кратко для простоты последующих выкладок:

$$\begin{aligned}
 f_{fit}(\bar{x}^k) &\equiv f_1^k(f_2^k), \text{ где} \\
 f_2^k &\equiv f_2(f(\bar{x}^k), g_i(\bar{x}^k), h_j(\bar{x}^k)), \\
 f_1^k &\equiv f_1(f_2^k).
 \end{aligned} \tag{6.3}$$

То есть значение функционала  $f(\bar{x}^k)$  претерпевает два преобразования.

$f_2^k$  — преобразование с учетом ограничений, накладываемых на задачу оптимизацию. Например, это может быть штрафы (смертельные, аддитивные и др.). В случае если  $m_1 = 0$  и  $m_2 = 0$  (задача безусловной оптимизации), то:

$$f_2^k = f(\bar{x}^k), k = \overline{1, N}. \tag{6.4}$$

$f_1^k$  — преобразование непосредственно по определению функции пригодности. Ее значение требуется в операторе селекции. При использовании пропорциональной селекции выдвигается требование, чтобы показатель эффективности решения был неотрицательной величиной. Функционал  $f(\bar{x})$  таким свойством не обладает. В литературе не описывается способов приведения к соответствующему виду  $f(\bar{x}^k)$  (или точнее  $f_2^k$ ). Обычно сразу оговаривается, что  $f(\bar{x})$  обладает необходимыми условиями. Но тестовые функции, на которых проводятся исследования, этими свойствами не обладают. Из-за этого не адекватно сравнивать результаты исследования генетических алгоритмов. Например, при проверке одинаковых модификаций ГА у разных исследователей результаты могут отличаться, так как условия проведения экспериментов разные — по своей сути решаются разные задачи оптимизации. Поэтому предлагается следующая формула, при использовании которой будет всегда выполняться условие  $f_1^k \geq 0$ :

$$f_1^k = \begin{cases} \frac{f_2^k - I^{min}}{I^{max} - I^{min}}, & \text{если } I^{max} \neq I^{min}; \\ 1, & \text{если } I^{max} = I^{min}. \end{cases} \tag{6.5}$$

$$I^{min} = \min_{\bar{x}^i \in Y} f_2^i, i = \overline{1, N};$$

$$I^{max} = \max_{\bar{x}^i \in Y} f_2^i, i = \overline{1, N}.$$

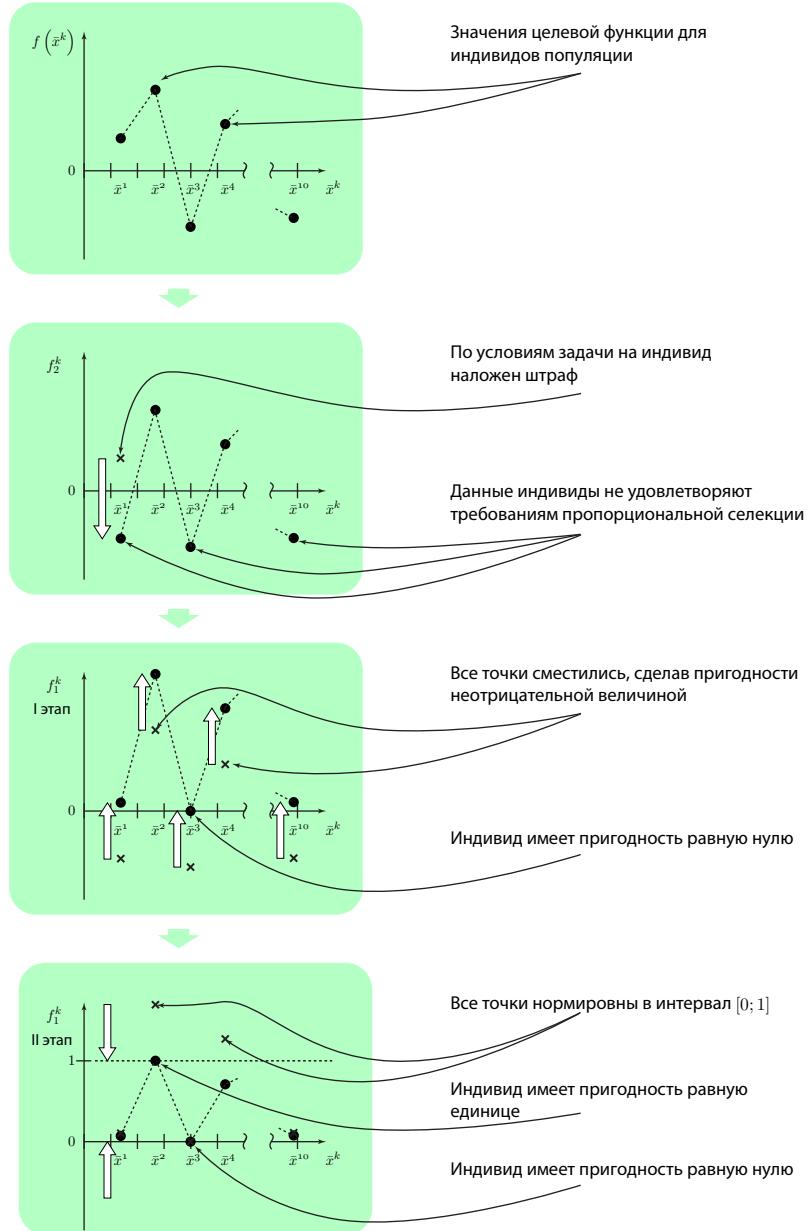


Рисунок 6.2. Пример получения пригодности индивидов популяции размера  $N = 10$

Отметим, что условие  $f_2^k \geq 0$  не означает удовлетворения целевой функции требованиям пропорциональной селекции на всем множестве решений, так как условие рассматривает только решения из текущей популяции.

**Замечание.** Для других типов селекции (ранговая и турнирная) нет требования о неотрицательности функции пригодности. Но монотонное преобразование  $f_1^k$  не изменяет вероятности выбора индивидов из популяции ввиду того, что в данных типах селекции используется только попарное сравнение пригодностей, а не их численных значений.

**Замечание.** При преобразовании (6.5) обязательно найдется индивид с вероятностью выбора равной нулю в пропорциональной селекции. То есть в таком случае, всегда хотя бы один индивид в текущем поколении не имеет шансов стать родителем (кроме вырожденного случая, когда все поколение состоит из одного и того же индивида). Существуют другие преобразования  $f_1^k$ , лишенные этого недостатка, но они не рассматриваются в данной работе.

**Замечание.** При использовании пропорциональной селекции в случае, когда целевая функция (точнее ее преобразование с учетом ограничений) может принимать отрицательные значения, то пригодность одного и того же индивида может изменяться в разных поколениях из-за преобразования (6.5).

**Замечание.** Вычисления функции пригодности при программной реализации генетического алгоритма должны проводиться только на этапах «**Вычисление функции пригодности индивидов**». Во всех остальных операторах, если в формулах используется функция пригодности в виде  $f(\bar{x})$  или ином виде, то значения функции пригодности должны браться из сохраненных значений, а не расчитываться заново. В противном случае количество вычислений целевой функции не будет совпадать с тем, что задается для запуска генетического алгоритма.

## 6.3 Селекция

Селекция — оператор случайного выбора одного индивида из популяции, основываясь на значениях функции пригодности всех индивидов текущей популяции, для использования его в операторе скрещивания. При этом вероятность выбора у индивидов с более высокой пригодностью выше, чем у индивидов с более низкой пригодностью.

В обычном генетическом алгоритме рассматривается три типа селекции, который определяется параметром  $TypeOfSelection$  из вектора  $DataOfSel$ .

### Пропорциональная селекция.

$$TypeOfSelection = ProportionalSelection. \quad (6.6)$$

Вероятность выбора элемента пропорциональна значению пригодности индивида. Данный вид селекции может работать только с неотрицательными значениями пригодности, чем, и вызвано преобразование (6.5).

Пропорциональная селекция определяется формулой:

$$Selection(Population, Fitness, DataOfSel) = Random(\{\bar{x}^i | p^i\}), \quad (6.7)$$

$$p^i = \begin{cases} \frac{f_{fit}(\bar{x}^i)}{\sum_{j=1}^N f_{fit}(\bar{x}^j)}, & \text{если } \exists f_{fit}(\bar{x}^k) \neq 0 \ (k = \overline{1, N}); \\ \frac{1}{N}, & \text{иначе.} \end{cases} \quad (6.8)$$

где  $\bar{x}^i \in Population, i = \overline{1, N}$ .

Как видим, формула определения вероятности выбора индивида имеет составной вид. Второе условие предназначено для маловероятного случая, когда в популяции все индивиды будут иметь пригодность равную нулю.

$DataOfSel$  не содержит каких-либо параметров относительно данного типа селекции.

**Пример.** Пусть  $Fitness = \{0,5; 0,2; 0,1; 0,6; 0,2; 0,4\}$ . Тогда вероятности выбора индивидов равны:

$$p_1 = \frac{0,5}{0,5 + 0,2 + 0,1 + 0,6 + 0,2 + 0,4} = 0,25;$$

$$p_2 = \frac{0,2}{0,5 + 0,2 + 0,1 + 0,6 + 0,2 + 0,4} = 0,1;$$

$$p_3 = \frac{0,1}{0,5 + 0,2 + 0,1 + 0,6 + 0,2 + 0,4} = 0,05;$$

$$p_4 = \frac{0,6}{0,5 + 0,2 + 0,1 + 0,6 + 0,2 + 0,4} = 0,3;$$

$$p_5 = \frac{0,2}{0,5 + 0,2 + 0,1 + 0,6 + 0,2 + 0,4} = 0,1;$$

$$p_6 = \frac{0,4}{0,5 + 0,2 + 0,1 + 0,6 + 0,2 + 0,4} = 0,2.$$

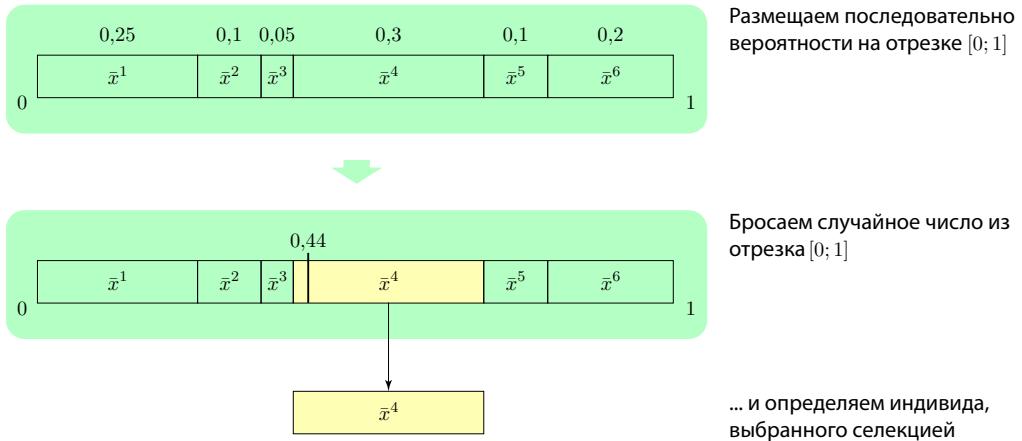


Рисунок 6.3. Механизм работы пропорциональной селекции

### Ранговая селекция.

$$TypeOfSelection = RankSelection. \quad (6.9)$$

Работает не с массивом пригодностей напрямую, а массивом нормированных рангов, присваиваемых индивидам на основе значений пригодности. Используется функция, которая проставляет ранги для элементов несортированного массива пригодностей, то есть номера, начиная с 1, в отсортированном массиве. Если в массиве есть несколько одинаковых элементов, то ранги им присуждаются как среднеарифметические ранги этих элементов в отсортированном массиве. Если это не сделать, то вероятность выбора индивидов одинаковых по функции пригодности будет не равна друг другу, что противоречит идеи оператора селекции. Далее для выбора индивидов используется пропорциональная селекция, работающая с массивом рангов.

Значит, ранговая селекция определяется формулой:

$$Selection (Population, Fitness, DataOfSel) = Random (\{\bar{x}^i | p^i\}) \quad (6.10)$$

$$p^i = \frac{Rank (f_{fit} (\bar{x}^i))}{\sum_{j=1}^N Rank (f_{fit} (\bar{x}^j))} \quad (6.11)$$

$$Rank(f_{fit}(\bar{x}^i)) = \frac{\sum_{j=1}^N NumberOfSorting(f_{fit}(\bar{x}^i), Fitness) \cdot S(f_{fit}(\bar{x}^i), f_{fit}(\bar{x}^j))}{\sum_{j=1}^N S(f_{fit}(\bar{x}^i), f_{fit}(\bar{x}^j))} \quad (6.12)$$

$$S(f_{fit}(\bar{x}^i), f_{fit}(\bar{x}^j)) = \begin{cases} 1, & \text{если } f_{fit}(\bar{x}^i) = f_{fit}(\bar{x}^j); \\ 0, & \text{если } f_{fit}(\bar{x}^i) \neq f_{fit}(\bar{x}^j). \end{cases} \quad (6.13)$$

где  $\bar{x}^i \in Population$ ,  $i = \overline{1, N}$ .

$NumberOfSorting(f_{fit}(\bar{x}^i), Fitness)$  — функция, возвращающая номер элемента  $f_{fit}(\bar{x}^i)$  в отсортированном массиве  $Fitness$  в порядке возрастания.

Формула (6.12) подсчитывает средние арифметические ранги при условии, что в массиве  $Fitness$  могут встречаться одинаковые элементы.

$DataOfSel$  также не содержит каких-либо параметров относительно данного типа селекции.

**Пример.** Пусть  $Fitness = \{0, 5; 0, 2; 0, 1; 0, 6; 0, 2; 0, 4\}$ . Тогда вероятности выбора индивидов равны:

$$\begin{aligned} p_1 &= \frac{5}{5 + 2, 5 + 1 + 6 + 2, 5 + 4} = 0, 238; \\ p_2 &= \frac{2, 5}{5 + 2, 5 + 1 + 6 + 2, 5 + 4} = 0, 119; \\ p_3 &= \frac{1}{5 + 2, 5 + 1 + 6 + 2, 5 + 4} = 0, 047; \\ p_4 &= \frac{6}{5 + 2, 5 + 1 + 6 + 2, 5 + 4} = 0, 286; \\ p_5 &= \frac{2, 5}{5 + 2, 5 + 1 + 6 + 2, 5 + 4} = 0, 119; \\ p_6 &= \frac{4}{5 + 2, 5 + 1 + 6 + 2, 5 + 4} = 0, 190. \end{aligned}$$

### Турнирная селекция.

$$TypeOfSelection = TournamentSelection. \quad (6.14)$$

Из популяции с равной вероятностью выбираются индивиды в количестве  $T$  (размер турнира), где  $2 \leq T \leq N$ . При этом каждый индивид может попасть в группу (турнир) только один раз (турнирная селекция без возвращения). Из данной группы выбирается индивид с наибольшей пригодностью.

Значит, турнирная селекция определяется формулой:

$$\begin{aligned} Selection(Population, Fitness, DataOfSel) &= \arg \max_{\bar{x} \in H} f_{fit}(\bar{x}), \text{ где} \\ H &= \{h^i | h_i = Random(Population / (\{h^1\} \cup \{h^2\} \cup \dots \cup \{h^{i-1}\}))\}, i = \overline{1, T}. \end{aligned} \quad (6.15)$$

Турнирная селекция является единственным типом данного оператора, который добавляет в  $DataOfSel$  дополнительный параметр — размер турнира  $T$ . Обычно выбирают значение этого параметра равное  $T = 2$ .



Рисунок 6.4. Механизм работы ранговой селекции

**Пример.** Массив возьмем тот же, что и в предыдущих примерах  $Fitness = \{0, 5; 0, 2; 0, 1; 0, 6; 0, 2; 0, 4\}$ . Размер турнира равен  $T = 3$ . Пример работы оператора показан на рисунке:

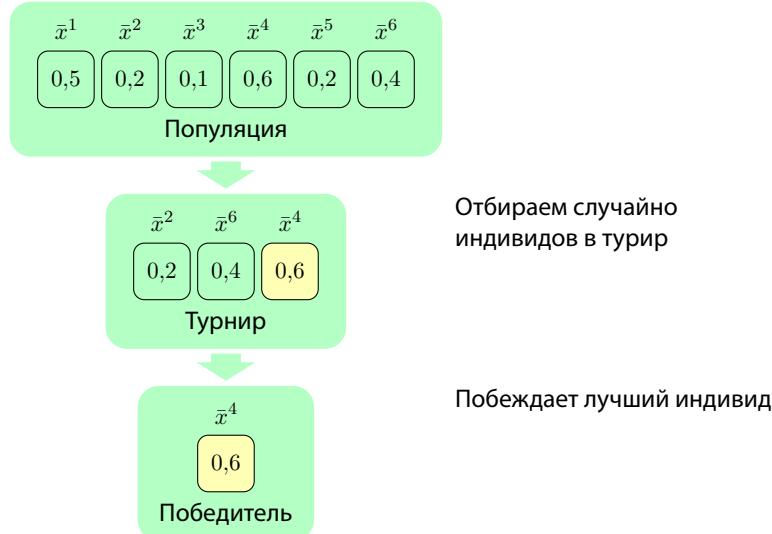


Рисунок 6.5. Механизм работы турнирной селекции

Итак, мы рассмотрели используемые варианты селекции. Тогда можно составить вектор параметров оператора селекции  $DataOfSel$ :

$$DataOfSel = \begin{pmatrix} TypeOfSel \\ T \end{pmatrix}. \quad (6.16)$$

**Замечание.** Часто в литературе наряду с рассмотренными ниже тремя видами селекции упоминается элитарная селекция (элитная селекция, селекция элитизма). Этот оператор не является селекцией по определению этого оператора. В стандарте включен в состав одного из видов оператора формирования нового поколения из родителей и потомков.

**Замечание.** Часто при программировании турнирной селекции реализуется такой вариант, при котором один и тот же индивид из популяции может быть выбран в турнир несколько раз. Это происходит из-за простоты программирования данного способа, но является недопустимым при реализации вышеупомянутой селекции. В противном случае мы получаем модифицированный вариант турнирной селекции с возвращением, который требует дополнительных исследований.

Разумеется, выбранный индивид для участия в турнире, может принять участие в других турнирах.

**Замечание.** Часто в литературе встречаются различные варианты ранговой селекции, связанные с преобразованием значений рангов (возведение в квадрат и др.) В данной работе эти варианты не рассматриваются.

**Замечание.** Программировать оператор селекции рекомендуется таким образом, чтобы он возвращал не индивида, а его номер в популяции.

**Замечание.** Если при выполнении селекции сравниваются два разных индивида с одинаковыми значениями функций пригодности, то выбирается первый из них.

## 6.4 Скрещивание

**Скрещивание (кроссовер)** — оператор случайного формирования нового индивида из двух выбранных родителей с сохранением признаков обоих родителей.

В обычном генетическом алгоритме рассматривается три типа скрещивания, который определяется параметром  $TypeOfCrossover$  из вектора  $DataOfCros$ .

### Одноточечное скрещивание.

$$TypeOfCrossover = SinglepointCrossover. \quad (6.17)$$

Пусть имеется два родителя (родительские хромосомы)  $\overline{Parent}^1$  и  $\overline{Parent}^2$ . В случайному месте происходит разрыв между двумя позициями генов в обеих хромосомах. После этого хромосомы обмениваются частями, в результате чего образуются два потомка. Из них выбирается случайно один потомок, который и передается в качестве результата оператора скрещивания. То есть скрещивание происходит по формулам:

$$Crossover \left( \overline{Parent}^1, \overline{Parent}^2, DataOfCros \right) = Random \left( \left\{ \overline{Offspring}^1; \overline{Offspring}^2 \right\} \right), \quad (6.18)$$

$R = Random (\{2; 3; \dots; n\})$ ;

$$\overline{Offspring}_i^1 = \overline{Parent}_i^1, i = \overline{1, R-1};$$

$$\overline{Offspring}_i^1 = \overline{Parent}_i^2, i = \overline{R, n};$$

$$\overline{Offspring}_i^2 = \overline{Parent}_i^2, i = \overline{1, R-1};$$

$$\overline{Offspring}_i^2 = \overline{Parent}_i^1, i = \overline{R, n};$$

$$\overline{Offspring}^1 \in X, \overline{Offspring}^2 \in X.$$

$DataOfCros$  не содержит каких-либо параметров относительно данного типа скрещивания.

**Пример.** Для всех видов скрещивания будем использовать двух родителей:  $\overline{Parent}^1 = (0; 1; 0; 1; 1; 1; 0; 0)^T$  и  $\overline{Parent}^2 = (1; 1; 0; 0; 1; 0; 1)^T$ . Одноточечное скрещивание показано на рисунке:

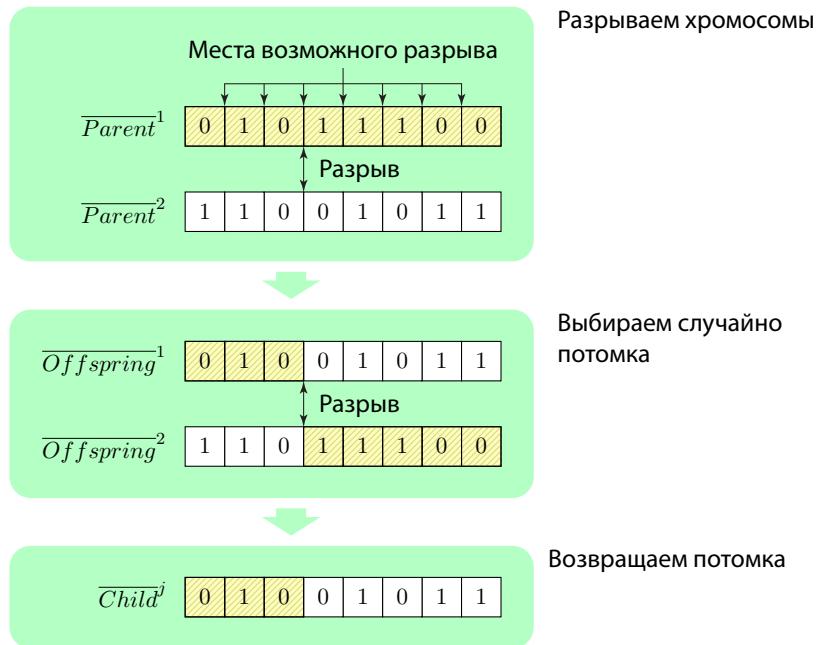


Рисунок 6.6. Механизм работы одноточечного скрещивания

### Двухточечное скрещивание.

$$TypeOfCrossover = TwoPointCrossover. \quad (6.19)$$

Пусть имеются два родителя (родительские хромосомы)  $\overline{Parent}^1$  и  $\overline{Parent}^2$ . В двух случайных местах происходят разрывы между двумя позициями генов в обеих хромосомах. После этого хромосомы обмениваются частями, в результате чего образуются два потомка. Из них выбирается случайно один потомок, который и передается в качестве результата оператора скрещивания. То есть скрещивание происходит по формулам:

$$\text{Crossover} \left( \overline{\text{Parent}}^1, \overline{\text{Parent}}^2, \text{DataOfCros} \right) = \text{Random} \left( \left\{ \overline{\text{Offspring}}^1; \overline{\text{Offspring}}^2 \right\} \right), \quad (6.20)$$

$$r_1 = \text{Random} (\{2; 3; \dots; n\});$$

$$r_2 = \text{Random} (\{2; 3; \dots; n\});$$

$$R_1 = \min (r_1, r_2);$$

$$R_2 = \max (r_1, r_2);$$

$$\overline{\text{Offspring}}_i^1 = \overline{\text{Parent}}_i^1, i = \overline{1, R_1 - 1};$$

$$\overline{\text{Offspring}}_i^1 = \overline{\text{Parent}}_i^2, i = \overline{R_1, R_2 - 1};$$

$$\overline{\text{Offspring}}_i^1 = \overline{\text{Parent}}_i^1, i = \overline{R_2, n};$$

$$\overline{\text{Offspring}}_i^2 = \overline{\text{Parent}}_i^2, i = \overline{1, R_1 - 1};$$

$$\overline{\text{Offspring}}_i^2 = \overline{\text{Parent}}_i^1, i = \overline{R_1, R_2 - 1};$$

$$\overline{\text{Offspring}}_i^2 = \overline{\text{Parent}}_i^2, i = \overline{R_2, n};$$

$$\overline{\text{Offspring}}^1 \in X, \overline{\text{Offspring}}^2 \in X.$$

*DataOfCros* не содержит каких-либо параметров относительно данного типа скрещивания.

**Пример.** Двухточечное скрещивание показано на рисунке:

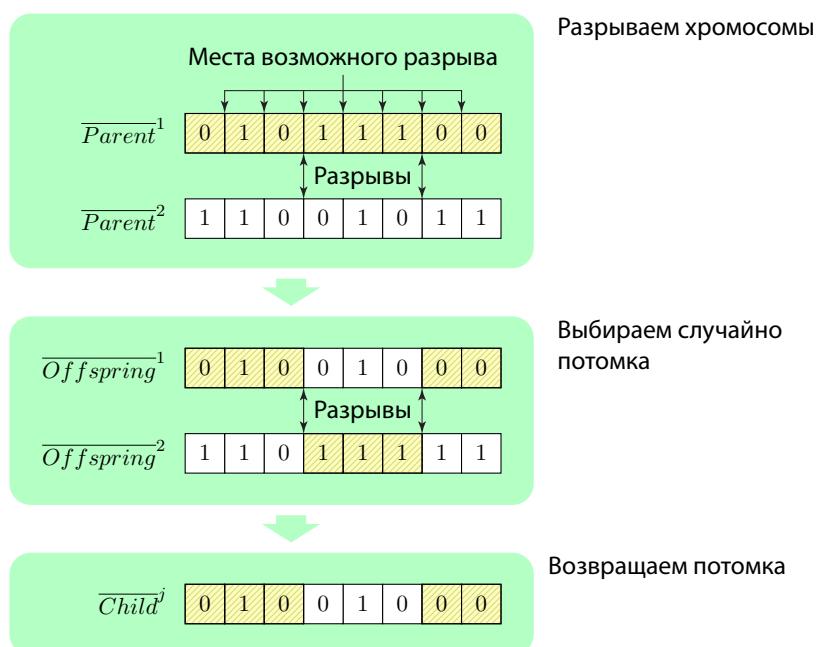


Рисунок 6.7. Механизм работы двухточечного скрещивания

### Равномерное скрещивание.

$$\text{TypeOfCrossover} = \text{UniformCrossover}. \quad (6.21)$$

Пусть имеется два родителя (родительские хромосомы)  $\overline{\text{Parent}}^1$  и  $\overline{\text{Parent}}^2$ . Потомок состоит из генов, каждый из которых выбран случайно из генов родителей на соответствующих позициях. То есть скрещивание происходит по формулам:

$$\begin{aligned}
 Crossover \left( \overline{Parent^1}, \overline{Parent^2}, DataOfCros \right) &= \overline{Offspring}; \\
 \overline{Offspring}_i &= Random \left( \left\{ \overline{Parent_i^1}; \overline{Parent_i^2} \right\} \right), i = \overline{1, n}; \\
 \overline{Offspring} &\in X.
 \end{aligned} \tag{6.22}$$

*DataOfCros* не содержит каких-либо параметров относительно данного типа скрещивания.

**Пример.** Равномерное скрещивание показано на рисунке:

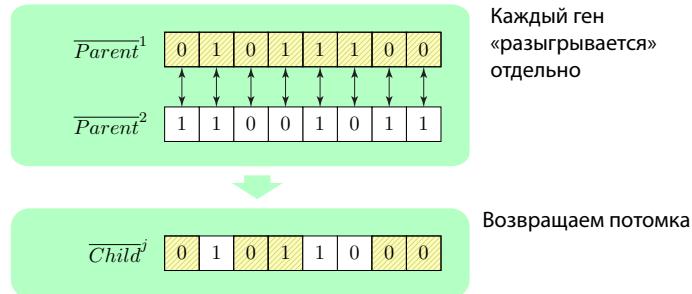


Рисунок 6.8. Механизм работы равномерного скрещивания

Итак, мы рассмотрели используемые варианты скрещивания. Тогда можно составить вектор параметров оператора скрещивания *DataOfCros*:

### Равномерное скрещивание.

$$DataOfCros = (TypeOfCros). \tag{6.23}$$

**Замечание.** В одноточечном и двухточечном скрещивании разрыв хромосомы может произойти только между двумя генами, то есть в начале или конце хромосомы разрыв произойти не может, что является частой ошибкой при программировании. Такая операция не даст новых решений.

**Замечание.** В литературе присутствует два подхода к оператору скрещивания. Согласно первому в результате скрещивания получаются два потомка, которые оба попадают в популяцию потомков. Но этот способ накладывает ограничение на размер популяции  $N$ : он должен быть четным. В данной работе используется второй подход, согласно которому в популяцию потомков попадает только один потомок, отобранный случайно. Есть варианты в литературе, когда отбирается потомок с наилучшей пригодностью, но это в два раза увеличивает число вычислений функции, причем половина из них идет на вычисление функции пригодности индивидов, которые не примут участие в существовании популяции.

**Замечание.** В качестве двух родителей может выбраться один и тот же индивид  $\overline{Parent^1} = \overline{Parent^2}$ . Это не запрещается в отличие от запрета попадания одного и того же индивида в турнир в турнирной селекции.

**Замечание.** В литературе в качестве одного из параметров скрещивания выступает вероятность скрещивания. Но часто она принимается за единицу, а многими исследователями вовсе опускается. В Стандарте (здесь описывается простейший вариант генетического алгоритма) данный параметр опущен.

Вероятность скрещивания меньше 1 исполняет роль операции клонирования. Хотя в Стандарте мы ее опустили, клонирование выполняется за счет селекции с повторением, то есть возможна реализация скрещивания одинаковых решений (см. предыдущее замечание) — аналог клонирования с очень низкой вероятностью.

**Замечание.** В двухточечном скрещивании точки разрыва могут совпасть. Тогда потомок будет совпадать с одним из родителей.

**Замечание.** Обратите внимание, что при описании оператора скрещивания для обозначения потомков используются  $\overline{Offspring}$ , в схеме работы стандартного генетического алгоритма на бинарных строках  $\overline{Child}$ . Это объясняется следующим образом.  $\overline{Child}$  используется для обозначения итогового потомка, который является результатом оператора скрещивания, тогда как  $\overline{Offspring}$  служит для обозначения временного потомка внутри работы оператора, и временных потомков может получиться несколько. Один из них, который выбирается, как описано выше, и становится возвращаемым  $\overline{Child}$ .

## 6.5 Мутация

**Мутация** — оператор случайного изменения всех потомков из популяции. Цель данного оператора не получить более лучшее решение, а разнообразить многообразие рассматриваемых индивидов. Обычно мутация предполагает незначительное изменение потомков. При выполнении оператора каждый ген каждого индивида с некоторой заданной вероятностью  $ProbabilityOfMutation$  мутирует, то есть меняет свое значение на противоположное. Мутация происходит по формулам:

$$\overline{MutChild}_j^i = \begin{cases} \overline{Child}_j^i, & \text{если } random(0, 1) > ProbabilityOfMutation; \\ 1 - \overline{Child}_j^i, & \text{иначе.} \end{cases} \quad (6.24)$$

$$\overline{Child}^i \in ChildPopulation, i = \overline{1, N},$$

$$\overline{MutChild}^i \in X, i = \overline{1, N}.$$

Обычно в генетическом алгоритме вероятность мутации выбирается из трех вариантов: слабая (*Weak*), средняя (*Average*) и сильная (*Strong*) мутация. Отсюда вероятность мутации определяется формулой:

$$ProbabilityOfMutation(TypeOfMutation) = \\ = \begin{cases} \frac{1}{3n}, & \text{если } TypeOfMutation = Weak; \\ \frac{1}{n}, & \text{если } TypeOfMutation = Average; \\ min\left(1, \frac{3}{n}\right), & \text{если } TypeOfMutation = Strong. \end{cases} \quad (6.25)$$

Здесь

$$TypeOfMutation \in \{Weak; Average; Strong\}, \quad (6.26)$$

$n$  — длина вектора  $\bar{x} \in X$  бинарной задачи оптимизации.

Составим вектор параметров оператора мутации  $DataOfMut$ :

$$DataOfMut = (TypeOfMutation). \quad (6.27)$$

**Замечание.** Определение сильной мутации через  $\min(1, \frac{3}{n})$  связано с тем, что при  $n < 3$  вероятность мутации становится больше 1, что недопустимо. Но при программировании это не вызывает отклонений в работе генетического алгоритма, к тому же использование ГА для столь малых размерностей нецелесообразно.

**Замечание.** Часто в литературе под мутацией понимается не оператор над всеми потомками, а оператор над каждым в отдельности. Никакой принципиальной разницы в двух этих подходах нет.

**Пример.** На рис. 6.9 показана мутация одного из индивидов из популяции.

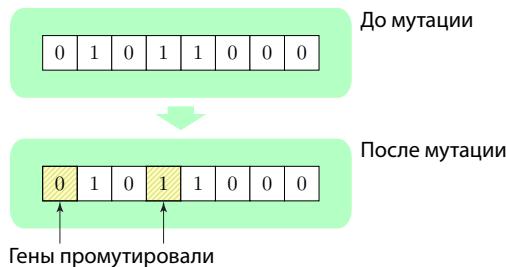


Рисунок 6.9. Механизм работы мутации

## 6.6 Формирование нового поколения

**Формирование нового поколения** — оператор формирования нового поколения из массива родителей и получившихся потомков с использованием уже известных значений функции пригодности, как родителей, так и потомков.

В обычном генетическом алгоритме используется два типа формирования нового поколения, который определяется параметром  $TypeOfGenerationForming$  из вектора  $DataOfForm$ .

**Только потомки.**

$$TypeOfGenerationForming = OnlyOffspringGenerationForming. \quad (6.28)$$

Данная схема предполагает формирование нового поколения из потомков и родителей таким, что в новое поколение попадают только потомки. Данный тип формирования нового поколения определяется формулой:

$$Forming \begin{pmatrix} Population \\ MutChildPopulation \\ Fitness \\ FitnessOfMutChild \\ DataOfForm \end{pmatrix} = \begin{pmatrix} MutChildPopulation \\ FitnessOfMutChild \end{pmatrix}. \quad (6.29)$$

$DataOfForm$  не содержит каких-либо параметров относительно данного типа формирования нового поколения.

**Только потомки и копия лучшего индивида.**

$$TypeOfGenerationForming = OnlyOffspringWithBestGenerationForming. \quad (6.30)$$

Данная схема предполагает формирование нового поколения из потомков и родителей таким, что в новое поколение попадают только потомки (без одного) и копия лучшего индивида  $Best$  (лучшего за всё время работы генетического алгоритма, а не только текущего поколения). В русской литературе данный способ часто называют селекцией элитизма. Данный тип формирования нового поколения определяется формулой:

$$Forming \begin{pmatrix} Population \\ MutChildPopulation \\ Fitness \\ FitnessOfMutChild \\ DataOfForm \end{pmatrix} = \begin{pmatrix} \{\bar{x}^i\} \\ \{f^i\} \end{pmatrix}, \quad (6.31)$$

$$\bar{x}^i = \begin{cases} \overline{Best}, & \text{если } i = 0; \\ \overline{MutChild}^i, & \text{если } i \neq 0; \end{cases}$$

$$f_i = f(\bar{x}_i), i = \overline{1, N}.$$

$DataOfForm$  не содержит каких-либо параметров относительно данного типа формирования нового поколения.

Итак, мы рассмотрели используемые варианты формирования нового поколения. Составим вектор параметров оператора формирования нового поколения:

$$DataOfForm = (TypeOfForm). \quad (6.32)$$

## Глава 7

# Решение задачи оптимизации на вещественных строках с помощью сГА

До данной главы рассматривался стандартный генетический алгоритм на бинарных строках, который является составной частью сГА *GeneticAlgorithm*. В случае, когда решается задача вещественной оптимизации, добавляются операторы преобразования задачи вещественной оптимизации в задачу бинарной оптимизации (*ConvertingIntoBinaryGA*) и преобразования бинарного решения в вещественное (*BinaryToReal*) (Алгоритм 3.1). При этом к вектору параметров сГА добавляется блок *ParametersOfConvertingIntoBinaryGA*.

Существует различные реализации данных операторов. В Стандарте рассматривается два из них:

- стандартное представление целого числа — номер узла в сетке дискретизации;
- стандартный рефлексивный Грэй-код.

Применяемый тип определяется параметром *TypeOfConverting* из *ParametersOfConvertingIntoBinaryGA*.

Итак, имеем:

- пространство поиска  $X$ , где  $\bar{x} \in X$ ,  $\bar{x} = (x_1; x_2; \dots; x_n)^T$ ,  $x_i \in [Left_i, Right_i]$ ,  $i = \overline{1, n}$ ;
- ограничения задачи оптимизации  $g_i(\bar{x}) \leq 0$  ( $i = \overline{1, m_1}$ ),  $h_j(\bar{x}) = 0$  ( $j = \overline{1, m_2}$ );
- целевую функцию  $f(\bar{x})$ ;

Необходимо произвести с этими данными преобразования.

### 7.1 Стандартное представление целого числа — номер узла в сетке дискретизации

$$TypeOfConverting = IntConverting. \quad (7.1)$$

Классическая схема, в которой каждая координата  $\bar{x}_i$  вектора  $\bar{x}$  дискретизируется на определенное число интервалов. Каждой точке на оси ставится в соответствие целое неотрицательное число — номер узла в сетке дискретизации (начиная с нуля). Данное целое число кодируется

в бинарную строку, длина которой равна наименьшей степени  $Length_i$  двойки при которой число  $2^{Length_i} \geq NumberOfParts_i + 1$ . Этот процесс осуществляется для всех компонент вектора  $\bar{x}$ . Итоговая бинарная строка, участвующая в сГА на бинарных строках, получается путем склейки бинарных строк всех компонент вектора.

Распишем алгоритм преобразования задачи *ConvertingIntoBinaryGA* (Алгоритм 7.1).

**Алгоритм 7.1.** Алгоритм *ConvertingIntoBinaryGA*: стандартное представление целого числа — номер узла в сетке дискретизации

**Начало алгоритма**

Рассчитаем необходимую длину каждой бинарной строки для каждой компоненты вектора  $\bar{x} \in X$ :

$$Length_i =$$

**Начало блока.**

$$S = 1, L = 0;$$

**До тех пока**  $S < NumberOfParts_i + 1$  **выполнять**

$$S = S \cdot 2;$$

$$L = L + 1;$$

**Конец цикла**

**Возвратить**  $L$ ;

**Конец блока**

Определим множество  $X_B$ , как множество бинарных векторов  $\bar{x}_B$  длины:

$$n_B = \sum_{i=1}^n Length_i$$

Рассчитаем шаг дискретизации для каждой компоненты вектора  $\bar{x} \in X$ :

$$h_i = \frac{|Left_i - Right_i|}{2^{Length_i}};$$

Получим целевую функцию задачи бинарной оптимизации:

$$f_B(\bar{x}_B) = f(\bar{x}), \text{ где}$$

$$\bar{x}_i = h_i \sum_{j=1}^{Length_i} \left( 2^{j-1} \cdot (\bar{x}_B)_{Begin_i+j} \right) + Left_i,$$

$$Begin_i = \sum_{j=1}^{i-1} Length_j, i = \overline{1, n};$$

Аналогично получим ограничения-неравенства для задачи бинарной оптимизации:

$$g_{j_B}(\bar{x}_B) = g_j(\bar{x}), \text{ где}$$

$$\bar{x}_i = h_i \sum_{k=1}^{Length_i} \left( 2^{k-1} \cdot (\bar{x}_B)_{Begin_i+k} \right) + Left_i,$$

$$Begin_i = \sum_{k=1}^{i-1} Length_k, i = \overline{1, n}, j = \overline{1, m_1};$$

Аналогично получим ограничения-равенства для задачи бинарной оптимизации:

$$h_{j_B}(\bar{x}_B) = g_j(\bar{x}), \text{ где}$$

$$\bar{x}_i = h_i \sum_{k=1}^{Length_i} \left( 2^{k-1} \cdot (\bar{x}_B)_{Begin_i+k} \right) + Left_i,$$

$$Begin_i = \sum_{k=1}^{i-1} Length_k, i = \overline{1, n}, j = \overline{1, m_2};$$

$$\text{Возвратить } \begin{pmatrix} X_B \\ f_B(\bar{x}_B) \\ g_{j_B}(\bar{x}_B) \\ h_{j_B}(\bar{x}_B) \\ ParametersOfBinaryGA \end{pmatrix}$$

**Конец алгоритма**

Алгоритм 7.1 использует следующие обозначения:

$NumberOfParts_i$  ( $i = \overline{1, n}$ ) — число интервалов, на которые предполагается разбивать каждую компоненту вектора  $\bar{x}$  в пределах своего изменения. В большинстве своем настоящее число интервалов будет получаться больше, чем запланировано;

$Begin_i$  — номер элемента (гена) бинарной строки  $\bar{x}_B$ , с которого начинают идти гены, кодирующую  $i$  компоненту вектора  $\bar{x}$ , ( $i = \overline{1, n}$ ).

$$NumberOfParts_i \in ParametersOfConvertingIntoBinaryGA, i = \overline{1, n}.$$

Аналогично приведенному методу преобразования бинарного решения в действительное (Алгоритм 7.1) получим формулу для оператора *BinaryToReal*:

$$\begin{aligned} BinaryToReal & \left( \begin{array}{c} \bar{x}_{submaxB} \\ f(\bar{x})_B \\ ParametersOfConvertingIntoBinaryGA \end{array} \right) = \left( \begin{array}{c} \bar{x}_{submax} \\ f(\bar{x}_{submax}) \end{array} \right), \quad \text{где} \quad (7.2) \\ \bar{x}_i &= h_i \sum_{j=1}^{Length_i} \left( 2^{j-1} \cdot (\bar{x}_B)_{Begin_i+j} \right) + Left_i, \\ Begin_i &= \sum_{j=1}^{i-1} Length_j, i = \overline{1, n}. \end{aligned}$$

**Замечание.** При программировании оператора *BinaryToReal* не следует заново вычислять значение целевой функции — следует использовать то значение, которое вычислялось при работе сГА на бинарных строках.

**Замечание.** При реализации генетического алгоритма операция кодирования вещественного числа в бинарную строку не требуется в отличие от операции декодирования.

**Замечание.** Существует схема, когда итоговая бинарная строка, участвующая в сГА на бинарных строках, получается не путем склейки бинарных строк всех компонент вещественного вектора  $\bar{x}$ , а путем покомпонентной склейки: вначале идут первые биты всех компонент вектора  $\bar{x}$ , потом вторых и т. д. В Стандарте он не рассматривается.

**Замечание.** Мы используем число  $NumberOfParts_i + 1$  для определения длины бинарной строки для кодирования каждой координаты, а не  $NumberOfParts_i$  ( $i = \overline{1, n}$ ), так как число точек больше интервалов, на которые мы разбиваем диапазон изменения вещественной координаты, на 1.

## 7.2 Стандартный рефлексивный Грей-код

$$TypOfConverting = GrayCodeConverting. \quad (7.3)$$

Так же, как в предыдущей схеме, каждая координата вектора  $\bar{x}$  дискретизируется на определенное число интервалов. Каждой точке на оси ставится в соответствие целое неотрицательное число — номер узла в сетке дискретизации (начиная с нуля). Данное целое число кодируется в бинарную строку, длина которой равна наименьшей степени двойки степени  $Length_i$  двойки при которой число  $2^{Length_i} \geq NumberOfParts_i + 1$ . Этот процесс осуществляется для всех компонент вектора  $\bar{x}$ . Итоговая бинарная строка, участвующая в сГА на бинарных строках, получается путем склейки бинарных строк всех компонент вектора. Всё то же самое.

Единственное отличие существует на этапе кодирования целого числа в бинарную строку. Бинарная строка не представляет собой двоичный код целого числа, а представляет код Грея. Его отличительной особенностью является то, что если два целых числа отличаются на единицу, то их коды Грея также будут отличаться только одним битом [3]. Двоичный код не обладает данным свойством.

Существует метод по переводу кода Грея в двоичный код: старший разряд (крайний левый бит) записывается без изменения, каждый следующий символ кода Грея нужно инвертировать, если в двоичном коде перед этим была получена «1», и оставить без изменения, если в двоичном коде был получен «0». Например [4], дан код Грея  $a = 1101$  целого числа  $x = 9$ . Двоичный код после преобразований:  $a' = 1001$ , что является двоичным представлением числа 9.

Благодаря этому методу мы можем операторы *ConvertingIntoBinaryGA* и *BinaryToReal* почти не изменять, а только добавить операцию по переводу бинарной строки из кода Грея в двоичный код.

Распишем алгоритм преобразования задачи вещественной оптимизации *ConvertingIntoBinaryGA* (Алгоритм 7.2):

---

**Алгоритм 7.2.** Алгоритм *ConvertingIntoBinaryGAGray*: стандартный рефлексивный Грей-код (I часть)

---

### Начало алгоритма

Рассчитаем необходимую длину каждой бинарной строки для каждой компоненты вектора  $\bar{x} \in X$ :

$Length_i =$

### Начало блока.

$S = 1, L = 0;$

**До тех пока**  $S < NumberOfParts_i + 1$  **выполнять**

$S = S \cdot 2;$

$L = L + 1;$

### Конец цикла

**Возвратить**  $L;$

### Конец блока

Определим множество  $X_B$ , как множество бинарных векторов  $\bar{x}_B$  длины:

$$n_B = \sum_{i=1}^n Length_i$$

Рассчитаем шаг дискретизации для каждой компоненты вектора  $\bar{x} \in X$ :

$$h_i = \frac{|Left_i - Right_i|}{2^{Length_i}}.$$

Получим целевую функцию задачи бинарной оптимизации:

$$f_B(\bar{x}_B) = f(\bar{x}), \text{ где}$$

$$\bar{x}_i = h_i \sum_{j=1}^{Length_i} \left( 2^{j-1} \cdot (\bar{x}_{BG})_{Begin_i+j} \right) + Left_i,$$

$$Begin_i = \sum_{j=1}^{i-1} Length_j, i = \overline{1, n},$$

$$(\bar{x}_{BG})_{Begin_i+j} = \begin{cases} (\bar{x}_B)_{Begin_i+j}, & \text{если } j = 1; \\ (\bar{x}_B)_{Begin_i+j}, & \text{если } j \neq 1 \text{ и } (\bar{x}_B)_{Begin_i+j-1} = 0; \\ 1 - (\bar{x}_B)_{Begin_i+j}, & \text{если } j \neq 1 \text{ и } (\bar{x}_B)_{Begin_i+j-1} = 1; \end{cases}$$

$$i = \overline{1, n};$$

*Продолжение ниже*

---

---

**Алгоритм 7.3.** Алгоритм *ConvertingIntoBinaryGAGray*: стандартный рефлексивный Грей-код (II часть)

---

*Начало выше*

Аналогично получим ограничения-неравенства для задачи бинарной оптимизации:

$$g_{j_B}(\bar{x}_B) = g_j(\bar{x}), \text{ где}$$

$$\bar{x}_i = h_i \sum_{k=1}^{Length_i} \left( 2^{k-1} \cdot (\bar{x}_{BG})_{Begin_i+k} \right) + Left_i,$$

$$Begin_i = \sum_{k=1}^{i-1} Length_k, i = \overline{1, n}, j = \overline{1, m_1},$$

$$(\bar{x}_{BG})_{Begin_i+j} = \begin{cases} (\bar{x}_B)_{Begin_i+j}, & \text{если } j = 1; \\ (\bar{x}_B)_{Begin_i+j}, & \text{если } j \neq 1 \text{ и } (\bar{x}_B)_{Begin_i+j-1} = 0; \\ 1 - (\bar{x}_B)_{Begin_i+j}, & \text{если } j \neq 1 \text{ и } (\bar{x}_B)_{Begin_i+j-1} = 1; \end{cases}$$

$$i = \overline{1, n};$$

Аналогично получим ограничения-равенства для задачи бинарной оптимизации:

$$h_{j_B}(\bar{x}_B) = g_j(\bar{x}), \text{ где}$$

$$\bar{x}_i = h_i \sum_{k=1}^{Length_i} \left( 2^{k-1} \cdot (\bar{x}_{BG})_{Begin_i+k} \right) + Left_i,$$

$$Begin_i = \sum_{k=1}^{i-1} Length_k, i = \overline{1, n}, j = \overline{1, m_2},$$

$$(\bar{x}_{BG})_{Begin_i+j} = \begin{cases} (\bar{x}_B)_{Begin_i+j}, & \text{если } j = 1; \\ (\bar{x}_B)_{Begin_i+j}, & \text{если } j \neq 1 \text{ и } (\bar{x}_B)_{Begin_i+j-1} = 0; \\ 1 - (\bar{x}_B)_{Begin_i+j}, & \text{если } j \neq 1 \text{ и } (\bar{x}_B)_{Begin_i+j-1} = 1; \end{cases}$$

$$i = \overline{1, n};$$

$$\text{Возвратить} \left( \begin{array}{c} X_B \\ f_B(\bar{x}_B) \\ g_{j_B}(\bar{x}_B) \\ h_{j_B}(\bar{x}_B) \\ ParametersOfBinaryGA \end{array} \right)$$

**Конец алгоритма**

---

Здесь (Алгоритм 7.2) следующие обозначения:

$NumberOfParts_i$  ( $i = \overline{1, n}$ ) — число интервалов, на которые предполагается разбивать каждую компоненту вектора  $\bar{x}$  в пределах своего изменения. В большинстве своем настоящее число интервалов будет получаться больше, чем запланировано;

$Begin_i$  — номер элемента (гена) бинарной строки  $\bar{x}_B$ , с которого начинают идти гены, кодирующую  $i$  компоненту вектора  $\bar{x}$ , ( $i = \overline{1, n}$ ).

$$NumberOfParts_i \in ParametersOfConvertingIntoBinaryGA, i = \overline{1, n}.$$

Аналогично приведенному методу преобразования бинарного решения в действительное (Алгоритм 7.2) получим формулу для оператора *BinaryToReal*:

$$BinaryToReal \left( \begin{array}{c} \bar{x}_{submax} \\ f(\bar{x})_B \\ ParametersOfConvertingIntoBinaryGA \end{array} \right) = \left( \begin{array}{c} \bar{x}_{submax} \\ f(\bar{x}_{submax}) \end{array} \right), \quad \text{где} \quad (7.4)$$

$$\bar{x}_i = h_i \sum_{j=1}^{Length_i} \left( 2^{j-1} \cdot (\bar{x}_{BG})_{Begin_i+j} \right) + Left_i,$$

$$Begin_i = \sum_{j=1}^{i-1} Length_j, i = \overline{1, n},$$

$$(\bar{x}_{BG})_{Begin_i+j} = \begin{cases} (\bar{x}_B)_{Begin_i+j}, & \text{если } j = 1; \\ (\bar{x}_B)_{Begin_i+j}, & \text{если } j \neq 1 \text{ и } (\bar{x}_B)_{Begin_i+j-1} = 0; \\ 1 - (\bar{x}_B)_{Begin_i+j}, & \text{если } j \neq 1 \text{ и } (\bar{x}_B)_{Begin_i+j-1} = 1; \end{cases}$$

$$i = \overline{1, n}.$$

## Глава 8

# Вектор параметров генетического алгоритма

Как мы помним вектор параметров сГА состоит из двух частей:

$$Parameters = \begin{pmatrix} ParametersOfBinaryGA \\ ParametersOfConvertingIntoBinaryGA \end{pmatrix}. \quad (8.1)$$

Распишем два этих компонента на основании предыдущего материала.

Вначале рассмотрим  $ParametersOfBinaryGA$  (формула 5.2), компоненты которого определяют параметры стандартного генетического алгоритма на бинарных строках:

$$ParametersOfBinaryGA = \begin{pmatrix} CountOfFitness \\ TypeOfSel \\ T \\ TypeOfCros \\ TypeOfMutation \\ TypeOfForm \end{pmatrix}.$$

Полагая, что в стандартном генетическом алгоритме при турнирной селекции размер турнира равен  $T = 2$ , то можно сократить вектор:

$$ParametersOfBinaryGA = \begin{pmatrix} CountOfFitness \\ TypeOfSel \\ TypeOfCros \\ TypeOfMutation \\ TypeOfForm \end{pmatrix}. \quad (8.2)$$

где

$CountOfFitness \in \mathbb{N}$ ,

$TypeOfSel \in \{ProportionalSelection; RankSelection; TournamentSelection\}$ ,

$TypeOfCros \in \{SinglepointCrossover; TwopointCrossover; UniformCrossover\}$ ,

$TypeOfMutation \in \{Weak; Average; Strong\}$ ,

$$TypeOfForm \in \left\{ \begin{array}{l} OnlyOffspringGenerationForming \\ OnlyOffspringWithBestGenerationForming \end{array} \right\}.$$

Так как  $\mu(TypeOfSel) = 3$ ,  $\mu(TypeOfCros) = 3$ ,  $\mu(TypeOfMutation) = 3$ ,  $\mu(TypeOfForm) = 2$ , то при фиксированных  $CountOfFitness$  и  $N$  (что обычно делают при исследованиях генетического алгоритма) число различных вариантов настроек генетического алгоритма равно:

$$CountOfSettings = \mu(TypeOfSel) \cdot \mu(TypeOfCros) \cdot \mu(TypeOfMutation) \cdot \mu(TypeOfForm) = 54. \quad (8.3)$$

Это может быть использовано для определения эффективных настроек генетического алгоритма.

Теперь рассмотрим вектор  $ParametersOfConvertingIntoBinaryGA$ , определяющий преобразование задачи вещественной оптимизации в задачу бинарной оптимизации.

$$ParametersOfConvertingIntoBinaryGA = \left( \begin{array}{l} TypeOfConverting \\ NumberOfParts_i \end{array} \right), \quad (8.4)$$

где

$$TypeOfConverting \in \{IntConverting; GrayodeConverting\},$$

$$NumberOfParts_i \in \{1; 2; \dots\}, (i = \overline{1, n}).$$

**Рекомендации.** Для некоторых параметров существуют рекомендации по выбору значений. Если выбирать не из предложенных вариантов, то алгоритм будет работать, но заданные величины не будут соответствовать реальным, которые используются внутри алгоритма (будут пересчитаны).

$$CountOfFitness = k_1^2, \text{ где } k_1 \in \mathbb{N}, \text{ например, } k_1 = 100. \quad (8.5)$$

$$NumberOfParts_i = 2^{(k_2)_i} - 1, \text{ где } (k_2)_i \in \mathbb{N}, \text{ например, } (k_2)_i = 14 (i = \overline{1, n}). \quad (8.6)$$

Также предлагается способ определения конкретных значений  $NumberOfParts_i$ , а точнее  $(k_2)_i$ . Для задач вещественной оптимизации в качестве одного из параметра выступает задаваемая точность вычислений  $\varepsilon$ . Даная величина несет в себе следующий смысл: запуск генетического алгоритма считается удачным, если найденное алгоритмом решение  $\bar{x}_{submax}$  удовлетворяет условию:

$$|(\bar{x}_{submax})_i - (\bar{x}_{max})_i| \leq \varepsilon, \text{ где } i = \overline{1, n}. \quad (8.7)$$

Шаг дискретизации в генетическом алгоритме берется в 10 раз меньше  $\varepsilon$ , чтобы при переходе к задаче бинарной оптимизации ГА мог найти точку рядом с глобальным оптимумом. Итак:

$$\begin{aligned} h_i &\leq \frac{\varepsilon}{10}; \\ \frac{Right_i - Left_i}{2^{(k_2)_i} - 1} &\leq \frac{\varepsilon}{10}; \\ 2^{(k_2)_i} &\geq \frac{10(Right_i - Left_i)}{\varepsilon} + 1, \end{aligned} \quad (8.8)$$

где  $i = \overline{1, n}$ ,  $(k_2)_i \in \mathbb{N}$ .

Тогда  $(k_2)_i$  определяется как минимальное натуральное число, которое удовлетворяет последнему неравенству 8.8. На практике выбор решается простым перебором натуральных чисел, начиная с 1. Это некритично с точки времени выполнения, так как перебор выполняется за всё время работы алгоритма только один раз, да и просмотреть нужно обычно не более пару десятков натуральных чисел начиная с 1.

Отметим, что мы не можем использовать только одно значение  $k_2$ , а нужно использовать целый вектор  $(k_2)_i$  ( $i = \overline{1, n}$ ). Это объясняется тем, что  $\varepsilon$  для всех вещественных координат одно и то же. Однако границы изменения каждой координаты могут быть различны, а, следовательно, разбиваться интервал каждой координаты при переходе к бинарной задаче оптимизации будет на разное число частей.

В случае если в задаче вещественной оптимизации не приводятся данные об  $\varepsilon$ , то выбор  $NumberOfParts_i$  производится пользователем по своему усмотрению.

**Замечание.** Когда мы рассматривали сГА как некий оператор (3.1), в его входных параметрах нигде не упоминается  $n$  — длина вектора  $\bar{x}$ ,  $Left_i$  и  $Right_i$  ( $i = \overline{1, n}$ ) — границы изменения компонент  $\bar{x}$ , если решается задача вещественной оптимизации. Это связано с тем, что данная информация заключена во входном параметре  $X$  — множестве всех возможных решений. Но при программировании лучше передавать только эти характеристики множества  $X$ , а не само всё множество.

# Заключение

**Стандартный генетический алгоритм** является стохастическим эвристическим итерационным эволюционным поисковым популяционным, основанным на решении дискретной задачи оптимизации, алгоритмом нулевого порядка глобальной условной или безусловной оптимизации многоэкстремального функционала, определенного на множестве бинарных или вещественных векторов конечной длины, находящий субоптимальное решение.

В работе была сделана попытка подробно расписать обычный вариант алгоритма со всеми тонкостями, которые могут возникнуть при реализации.

# Литература

1. 80.250.188.26. Разработка простого генетического алгоритма. 2007. [http://www.wikiznanie.ru/ru-wz/index.php/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0\\_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B3%D0%BE\\_%D0%B3%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE\\_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0](http://www.wikiznanie.ru/ru-wz/index.php/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B3%D0%BE_%D0%B3%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0).
2. Матвеев М. Г., Свиридов А. С., Алейникова Н. А. Модели и методы искусственного интеллекта. Применение в экономике: учеб. пособие. М.: Финансы и статистика, 2008. 448 с.
3. Код Грея. [http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4\\_%D0%93%D1%80%D0%B5%D1%8F](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%93%D1%80%D0%B5%D1%8F).
4. Савчук В. Л. Оптимальное кодирование. [http://www.ie.tusur.ru/books/COI/page\\_07.htm](http://www.ie.tusur.ru/books/COI/page_07.htm).