

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИBORОСТРОЕНИЯ

---

Ю. А. Скобцов

## ВЫЧИСЛИТЕЛЬНЫЙ ИНТЕЛЛЕКТ

Лабораторный практикум



Санкт-Петербург  
2022

УДК 004.8+519.7  
ББК 00  
С00

Рецензенты:

доктор технических наук, профессор *А. А. Большаков*;  
кандидат технических наук, доцент *В. Л. Оленев*

Утверждено

редакционно-издательским советом университета  
в качестве лабораторного практикума

Протокол № 2 от 30 марта 2022 г.

**Скобцов, Ю. А.**

С00      Вычислительный интеллект: лабораторный практикум / Ю. А. Скобцов. – СПб.: ГУАП, 2022. – 141 с.

В практикуме рассматриваются теоретические и прикладные вопросы вычислительного интеллекта – нового направления в теории искусственного интеллекта, которое в последнее время активно используется в инженерной практике.

Цель издания – способствовать успешному освоению студентами вычислительного интеллекта через совместное размещение в нем адаптированного теоретического материала и соответствующих конкретных заданий лабораторного практикума, выполняемого на персональных компьютерах.

Предназначено для подготовки бакалавров и магистров по направлениям 09.03.04 «Программная инженерия», 02.03.03 «Математическое обеспечение и администрирование информационных систем», 01.03.02 «Прикладная математика и информатика»; а также широкого круга других специальностей, где актуальны проблемы оптимизации большой размерности.

УДК 004.8+519.7  
ББК 000

© Санкт-Петербургский государственный  
университет аэрокосмического  
приборостроения, 2022

## ВВЕДЕНИЕ

В настоящее время интенсивно развивается новое направление в теории и практике искусственного интеллекта – эволюционные вычисления (ЭВ) [1–3]. Этот термин применяется для описания алгоритмов оптимизации, поиска или обучения, которые основаны на формализованных принципах естественной эволюции. Особенности эволюции и самоорганизации заключаются в том, что они прошли апробацию для биологических систем, развивающихся миллиарды лет. Формализация этих идей с успехом используется при разработке технических и, в особенности, программных систем.

В науке и технике эволюционные вычисления применяются в качестве адаптивных алгоритмов для решения практических задач. С помощью ЭВ было разработано много промышленных проектных решений, которые позволили сэкономить многие миллионы долларов. Следует отметить, что когда задача не может быть решена другими, более простыми методами, часто ЭВ могут найти (суб) оптимальные решения. При этом объем вычислений может оказаться достаточно большим, но скорость, с которой он растет, при увеличении размерности задачи обычно меньше, чем у остальных известных методов. После того, как компьютерные системы стали достаточно быстродействующими и недорогими, ЭВ превратились в важнейший инструмент исследования.

Отметим, что общество «Нейронные сети» IEEE (IEEE Neural Network Society) изменило свое название в 2004 году на IEEE Computational Intelligence Society (общество «Вычислительный интеллект») [4]. Вычислительный интеллект (ВИ) является одним из наиболее активно развивающихся и направлений в области информатики и техники и его методы широко используются при решении многих научно-технических проблем.

Первоначально ВИ был определен как комбинация нечеткой логики, нейронных сетей и генетических алгоритмов. Более широкое определение вычислительного интеллекта – изучение адаптивных механизмов, позволяющих или облегчающих интеллектуальное поведение в сложных, неопределенных и меняющихся условиях. Эти адаптивные механизмы включают в себя те парадигмы искусственного интеллекта (ИИ), которые демонстрируют способность учиться или адаптироваться к новым ситуациям, обобщать, обобщать, открывать и ассоциировать.

Парадигмы ВИ имитируют природные явления при решении сложных задач. Другими словами, вдохновленные биологией и при-

родой, эти парадигмы включают нейронные сети, эволюционные вычисления, роевой интеллект, нечеткие системы и искусственные иммунные системы. Вместе с такими разделами, как дедуктивное мышление, экспертные системы, логика и символика, системы машинного обучения, эти интеллектуальные алгоритмы образуют часть ИИ. ИИ с его многочисленными приемами, сочетанием нескольких научных дисциплин, например, биология, информатика, инженерия, нейро-биология, философия, социология. ИИ востребован как путь будущего развития науки в этой области. Вычислительный интеллект является преемником ИИ и является будущим вычислений. «Интеллект без вычислений похож на птиц без крыльев» [4]. С другой стороны, ВИ имеет отличия от ИИ. Некоторые из методов являются общими для ВИ и ИИ [4,20].

Существуют пять основных парадигм алгоритмов ВИ, а именно: (1) нейронные сети (НС), (2) эволюционные вычисления (ЭВ), (3) роевой интеллект (РА), (4) нечеткие (фаззи) системы (ФС) и (5) искусственные иммунные системы (ИИС). Широкий спектр алгоритмов ВИ из этих парадигм включает в себя: (1) – искусственные нейронные сети (ИНС); (2) – генетические алгоритмы (ГА), генетическое программирование (ГП), дифференциальная эволюция (ДЭ); (3) – оптимизация на основе роя частиц (), метод муравьиных колоний (ММК); (4) – нечеткие системы логического вывода (FIS); и (5) – искусственная иммунная система (ИИС). При этом из-за недостатка места рассматриваются только некоторые из популярных алгоритмов от основных парадигм. В данном пособии рассматриваются прежде всего эволюционные вычисления и роевой интеллект.

Автор читал лекции по вычислительному интеллекту с 2000 года в ряде университетов Украины и Санкт-Петербурга, что отражено в ранее опубликованных учебных пособиях [1–4, 13, 14], которые содержат необходимые теоретические материалы для выполнения лабораторных работ. Данное пособие написано на основе опыта преподавания эволюционных вычислений на кафедре «Компьютерные технологии и программная инженерия» Санкт-Петербургского государственного университета аэрокосмического приборостроения и кафедры «Телематика» Санкт-Петербургского политехнического университета Петра Великого.

Целью учебного пособия является – способствовать успешному освоению студентами основных разделов ЭВ, размещение в нем адаптированного теоретического материала, связанного с конкретными заданиями лабораторного практикума.

Учебное пособие содержит описание 8 лабораторных работ с конкретными указаниями и ссылками на теоретический материал, приводимый в данной работе:

1. Простой ГА.
2. ГА поиска экстремумов функций многих переменных.
3. Комбинаторная оптимизация на основе ГА (задача коммивояжера).
4. Символьная регрессия.
5. Эволюционная стратегия (экстремум функции многих переменных).
6. Муравьиный алгоритм (задача коммивояжера).
7. Роевой алгоритм (экстремум функции многих переменных).
8. Эволюционные алгоритмы оценки стоимости проектов в программной инженерии

Выполнение лабораторных работ желательно проводить в той последовательности, которая указана в учебном пособии

Для выполнения лабораторных работ кроме основ курса ЭВ желательны знания в объеме стандартных университетских курсов по методам оптимизации, теории вероятностей, дискретной математики и проектирования программного обеспечения. Минимальные теоретические знания, необходимые для выполнения лабораторных работ, приведены в пособии. В полном объеме они содержатся в авторских учебных пособиях и монографиях [1–3, 13, 14] и книгах других авторов [6–11]. Далее для каждой лабораторной работы даны ссылки на соответствующие литературные источники, содержащие необходимые теоретические материалы.

Тестирование лабораторных заданий выполняется на международных тестовых данных (benchmarks) [5–7], которые приведены в приложениях 1, 2, 3 и содержатся на многих сайтах в интернете.

Для лучшего усвоения материала пособие содержит контрольные вопросы для самопроверки. При выполнении лабораторных работ студенты могут использовать любой доступный язык программирования. Практика показывает, что наиболее популярными при выполнении работ являются МАТЛАБ [15–17], Питон [19], С++ и другие. Пример программной реализации в МАТЛАБ представлен в приложении. Кроме этого подобные примеры реализации в МАТЛАБ и С можно найти в англоязычных учебниках [15–17].

## Лабораторная работа №1

### ПРОСТОЙ ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

*Цель работы:* освоение базовых понятий эволюционных вычислений, кодирование потенциальных решений, получение первых практических навыков реализации генетических алгоритмов. Графическое отображение результатов оптимизации.

При выполнении лабораторной работы можно использовать следующие источники из прилагаемого списка литературы [1, 3, 4, 8–12, 15–17].

#### 1.1. Теоретическая часть

ГА используют принципы и терминологию, заимствованные у биологической науки – генетики. В ГА каждая особь представляет потенциальное решение некоторой проблемы. В классическом ГА особь кодируется строкой двоичных символов – хромосомой, каждый бит которой называется геном. Множество особей – потенциальных решений составляет популяцию. Поиск (суб)оптимального решения проблемы выполняется в процессе эволюции популяции – последовательного преобразования одного конечного множества решений в другое с помощью генетических операторов репродукции, кроссинговера и мутации.

ЭВ используют следующие механизмы естественной эволюции:

1) Первый принцип основан на концепции выживания сильнейших и естественного отбора по Дарвину, который был сформулирован им в 1859 году в книге «Происхождение видов путем естественного отбора». Согласно Дарвину особи, которые лучше способны решать задачи в своей среде, выживают и больше размножаются (репродуцируют). В генетических алгоритмах каждая особь представляет собой решение некоторой проблемы. По аналогии с этим принципом особи с лучшими значениями целевой (фитнесе) функции имеют большие шансы выжить и репродуцировать. Формализация этого принципа, как мы увидим далее, дает оператор репродукции.

2) Второй принцип обусловлен тем фактом, что хромосома потомка состоит из частей полученных из хромосом родителей. Этот принцип был открыт в 1865 году Менделем. Его формализация дает основу для оператора скрещивания (кроссинговера).

3) Третий принцип основан на концепции мутации, открытой в 1900 году де Вре. Первоначально этот термин использовался для описания существенных (резких) изменений свойств потомков и

приобретение ими свойств, отсутствующих у родителей. По аналогии с этим принципом генетические алгоритмы используют подобный механизм для резкого изменения свойств потомков и тем самым, повышают разнообразие (изменчивость) особей в популяции (множестве решений).

Эти три принципа составляют ядро ЭВ. Используя их, популяция (множество решений данной проблемы) эволюционирует от поколения к поколению. Эволюцию искусственной популяции – поиска множества решений некоторой проблемы формально можно описать алгоритмом, который представлен на рис. 1.1. ГА берет множество параметров оптимизационной проблемы и кодирует их последовательностями конечной длины в некотором конечном алфавите (в простейшем случае двоичный алфавит «0» и «1»).

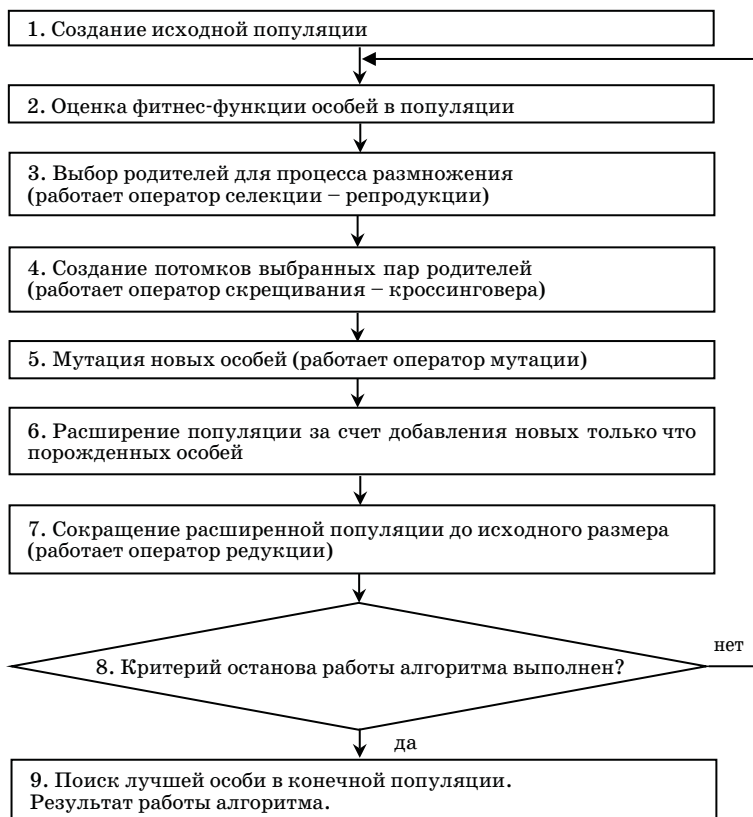


Рис. 1.1. Простой генетический алгоритм

Предварительно простой ГА случайным образом генерирует начальную популяцию стрингов (хромосом). Затем алгоритм генерирует следующее поколение (популяцию), с помощью трех основных генетических операторов:

- 1) Оператор репродукции (ОР);
- 2) Оператор скрещивания кроссинговера, ОК);
- 3) Оператор мутации (ОМ).

Генетические операторы являются математической формализацией приведенных выше трех основополагающих принципов Дарвина, Менделя и де Вре естественной эволюции. ГА работает до тех пор, пока не будет выполнено заданное количество поколений (итераций) процесса эволюции или на некоторой генерации будет получено заданное качество или вследствие преждевременной сходимости при попадании в некоторый локальный оптимум.

В каждом поколении множество искусственных особей создается с использованием старых и добавлением новых с хорошими свойствами. Генетические алгоритмы – не просто случайный поиск, они эффективно используют информацию, накопленную в процессе эволюции.

В отличие от других методов оптимизации ГА оптимизируют различные области пространства решений одновременно и более приспособлены к нахождению новых областей с лучшими значениями целевой функции за счет объединения квазиоптимальных решений из разных популяций.

### **Основная терминология в генетических алгоритмах**

Поскольку, как отмечалось выше, в ГА используются некоторые термины, заимствованные из генетики, то полезно привести их дать техническую интерпретацию.

*Ген* – элементарный код в хромосоме  $s_i$ , называемый также знаком, или детектором (в классическом ГА  $s_i=0,1$ ).

*Хромосома* – упорядоченная последовательность генов в виде закодированной структура данных  $S=(s_1, s_2, \dots, s_n)$ , определяющая решение (в простейшем случае двоичная последовательность – стринг, где  $s_i=0,1$ ).

*Локус* – местоположение (позиция, номер бита) данного гена в хромосоме.

*Аллель* – значение, которое принимает данный ген (например,  $\{0,1\}$ ).

*Особь* – одно потенциальное решение проблемы (представляемое хромосомой).



*Популяция* – множество особей – потенциальных решений, которые представляются хромосомами.

*Поклоение* – текущая популяция ГА (для текущей итерации алгоритма).

*Генотип* – набор хромосом данной особи (особями популяции могут быть генотипы либо отдельные хромосомы).

*Генофонд* – множество всех возможных генотипов.

*Фенотип* – набор значений, соответствующий данному генотипу, – декодированное множество параметров или структура данной задачи (например, фенотип – десятичное значение  $x$  его двоичного кода – геннотипа);

*Размер (мощность)* популяции  $N$  – число особей (решений) в популяции.

**Число поколений (генераций)** – количество итераций, в течение которых производится генетический поиск.

*Селекция* – комплекс правил, моделирующих выживание особей на основе их значений ЦФ.

*Эволюция популяции* – это чередование поколений, в которых хромосомы изменяют свои признаки, чтобы каждая новая популяция наилучшим образом приспособлялась к внешней среде.

*Фитнесс-функция* (полезность) – важнейшее понятие, определяющее меру приспособленности данной особи в популяции. В задачах оптимизации часто представляется целевой функцией или определяет меру близости к оптимальному решению. В обучении может принимать вид функции погрешности (ошибки). На каждой итерации ГА приспособленность каждой особи популяции оценивается с помощью фитнес-функции.

## Генетические операторы

Рассмотрим работу простого ГА на следующем примере. Надо найти (для простоты) целочисленное значение  $x$  в интервале от 0 до 31, при котором функции  $y = x^2$  принимает максимальное значение (рис. 1.2).

Здесь особи начальной популяции (двоичные коды значений переменных  $x$  – столбец 2) сгенерированы случайным образом. Двоичный код значения  $x$  называется хромосомой (она представляет генотип). Популяция образует множество потенциальных решений данной проблемы. В третьем столбце представлены их десятичные значения (фенотип). Далее на этом примере проиллюстрируем работу трех основных генетических операторов.

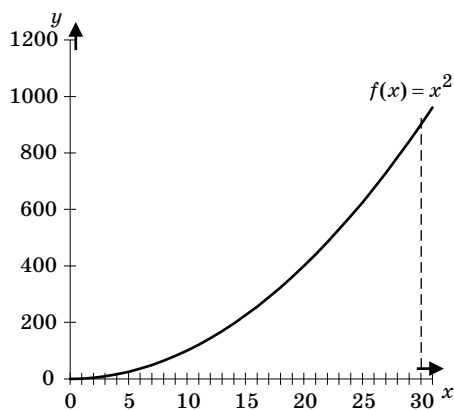


Рис. 1.2. Пример функции

### Репродукция

№ хромосомы	Начальная популяция особей	Десятичное значение $x$	Значение $f(x)$	$\frac{f(x_i)}{\sum f(x_j)}$	Среднее значение $\bar{f}(x)$	Максимальное значение $f_{\max}(x)$
1	01101	13	169	0,14	293	576
2	11000	24	576	0,49		
3	01000	8	64	0,06		
4	10011	19	361	0,31		



### Кроссинговер

№ хромосомы	Популяция после репродукции	Десятичное значение $x$	Значение $f(x)$	Пары хромосом для кроссинговера	Среднее значение $\bar{f}(x)$	Максимальное значение $f_{\max}(x)$
1	0 1 1 0 1	13	169	1–2	439	729
2	1 1 0 0 0	24	576	1–2		
3	1 1 0 0 0	24	576	3–4		
4	1 0 0 1 1	19	361	3–4		



### Мутация

№ хромосомы	Популяция после кроссинговера	Новая популяция после мутации	Десятичное значение $x$	Значение $f(x)$	Среднее значение $\bar{f}(x)$	Максимальное значение $f_{\max}(x)$
1	01100	01100	12	144	496.5	961
2	11001	11001	25	625		
3	11011	11111	31	961		
4	10000	10000	16	256		

Рис. 1.3. Эволюция популяции

Начальный этап работы ГА для данного примера приведен в верхней таблице (репродукция) рис. 1.3.

### Репродукция

Репродукция – это процесс, в котором хромосомы копируются в промежуточную популяцию для дальнейшего «размножения» согласно их значениям целевой (фитнес-) функции. При этом хромосомы с лучшими значениями целевой функции имеют большую вероятность попадания одного или более потомков в следующее поколение.

Очевидно, оператор репродукции (ОР) является искусственной версией естественной селекции – выживания сильнейших по Дарвину. Этот оператор представляется в алгоритмической форме различными способами. Самый простой (и популярный) метод реализации ОР – построение колеса рулетки, в которой каждая хромосома имеет сектор, пропорциональный ее значению ЦФ. Для нашего примера «колесо рулетки» имеет следующий вид, представленный на рис. 1.4.

Для селекции хромосом используется случайный поиск на основе колеса рулетки. При этом колесо рулетки вращается и после останова ее указатель определяет хромосому для селекции в промежуточную популяцию (родительский пул). Очевидно, что хромосома, которой соответствует больший сектор рулетки, имеет большую вероятность попасть в следующее поколение. В результате выполнения оператора репродукции формируется промежуточная популяция, хромосомы которой будут использованы для построения поколения с помощью операторов скрещивания.

В нашем примере выбираем хромосомы для промежуточной популяции, вращая колесо рулетки 4 раза, что соответствует мощности начальной популяции. Величину  $\frac{f(x_i)}{\sum f(x_j)}$  обозначим, как  $P(x_i)$ , тогда ожидаемое количество копий  $i$ -й хромосомы определяется  $M = P(x_i) \cdot N$ ,  $N$  – мощность популяции. Число копий хромосомы, переходящих в следующее поколение, иногда определяется и так:

$$\tilde{M} = \frac{f(x_i)}{\bar{f}(x)},$$

где  $\bar{f}(x)$  – среднее значение хромосомы в популяции.

Расчетные числа копий хромосом по приведенной формуле следующие: хромосома 1 – 0,56; хромосома 2 – 1,97; хромосома 3 – 0,22; хромосома 4 – 1,23. В результате, в промежуточную популяцию 1-я хромосома попадает в одном экземпляре, 2-я – в двух, 3-я – совсем

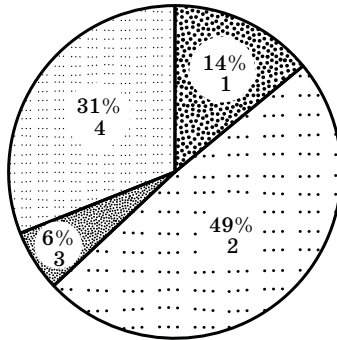


Рис. 1.4. Колесо рулетки

не попадает, 4-я – в одном экземпляре. Полученная промежуточная популяция является исходной для дальнейшего выполнения операторов кроссинговера и мутации.

### Оператор кроссинговера (скрещивания)

Одноточечный или простой оператор кроссинговера (ОК) с заданной вероятностью  $P_c$  выполняется в 3 этапа:

1-й этап. Две хромосомы (родители)

$$\begin{array}{l} A = a_1 a_2 \dots a_k \quad a_{k+1} \dots a_L \\ B = b_1 b_2 \dots b_k \quad b_{k+1} \dots b_L \end{array}$$

Точка кроссинговера

выбираются случайно из промежуточной популяции, сформированной при помощи оператора репродукции (ОР).

2-й этап. Случайно выбирается точка скрещивания – число  $k$  из диапазона  $[1, 2 \dots n - 1]$ , где  $n$  – длина хромосомы (число бит в двоичном коде)

3-й этап. Две новых хромосомы  $A'$ ,  $B'$  (потомки) формируются из  $A$  и  $B$  путем обмена подстроки после точки скрещивания:

$$\begin{array}{l} A' = a_1 a_2 \dots a_k \quad b_{k+1} \dots b_L \\ B' = b_1 b_2 \dots b_k \quad a_{k+1} \dots a_L \end{array}$$

Например, рассмотрим выполнение кроссинговера для хромосом 1 и 2 из промежуточной популяции:

$A = 0 \ 1 \ 1 \ 0 \ 1$

$B = 1 \ 1 \ 0 \ 0 \ 0$

$1 \leq k \leq 4, k=4$

A'=0 1 1 0 0

B'=1 1 0 0 1

Следует отметить, что ОЖ выполняется с заданной вероятностью  $P_c$  (отобранные два родителя не обязательно производят потомков). Обычно величина  $P_c \approx 0.5$

Таким образом, операторы репродукции и скрещивания очень просты – они выполняют копирование особей и частичный обмен частей хромосом. Продолжение нашего примера представлено на рис. 1.3 во второй таблице (кроссинговер).

Сравнение с предыдущей таблицей показывает, что в промежуточной популяции после скрещивания улучшились все показатели популяции (среднее и максимальные значения ЦФ).

### Мутация

Далее согласно схеме классического ГА с заданной вероятностью  $P_m$  выполняется оператор мутации. Иногда этот оператор играет вторичную роль. Обычно вероятность мутации мала –  $P_m \approx 0,001$ .

Оператор мутации (ОМ) выполняется в 2 этапа:

*1-й этап.* В хромосоме  $A = a_1 a_2 \dots a_L$  случайно выбирается  $k$ -я позиция (бит) мутации ( $1 \leq k \leq n$ ).

*2-й этап.* Производится инверсия значения гена в  $k$ -й позиции.

$$a'_k = \bar{a}_k$$

Например, для хромосомы 11011 выбирается  $k=3$  и после инверсии значения третьего бита получается новая хромосома – 11111. Продолжение нашего примера представлено в третьей таблице (мутация) рис. 1.3. Таким образом, в результате применения генетических операторов найдено оптимальное решение  $x=31$ .

В данном случае, поскольку пример искусственно подобран, мы нашли оптимальное решение за одну итерацию. В общем случае ГА работает до тех пор, пока не будет достигнут критерий окончания процесса поиска и в последнем поколении определяется лучшая особь.

### Представление вещественных решений в двоичной форме

В предыдущем примере мы рассматривали только целочисленные решения. Обобщим ГА на случай вещественных чисел на следующем примере, в котором для функции

$$f(x)=(1,85 - x)*\cos(3,5x - 0,5),$$

представленной на рис. 1.5 необходимо найти вещественное  $x \in [-10, +10]$ , которое максимизирует  $f$ , т. е. такое  $x_0$  для которого  $f(x_0) \geq f(x)$  для всех  $x \in [-10, +10]$ .

Нам необходимо построить ГА для решения этой задачи. Для представления вещественного решения (хромосомы)  $x$  будем использовать двоичный вектор, который применяется в классическом простом ГА. Его длина зависит от требуемой точности решения, которую в данном случае положим 3 знака после запятой.

Поскольку отрезок области решения имеет длину 20, для достижения заданной точности отрезок  $[-10, +10]$  должен быть разбит на равные части, число которых должно быть не менее  $20 \cdot 1000$ . В качестве двоичного представления используем двоичный код номера отрезка. Этот код позволяет определить соответствующее ему вещественное число, если известны границы области решения. Отсюда следует, что двоичный вектор для кодирования вещественного решения должен иметь 15 бит, поскольку

$$16384 = 2^{14} < 20000 \leq 2^{15} = 32768$$

Это позволяет разбить отрезок  $[-10, +10]$  на 32 768 частей и обеспечить необходимую точность. Отображение из двоичного пред-

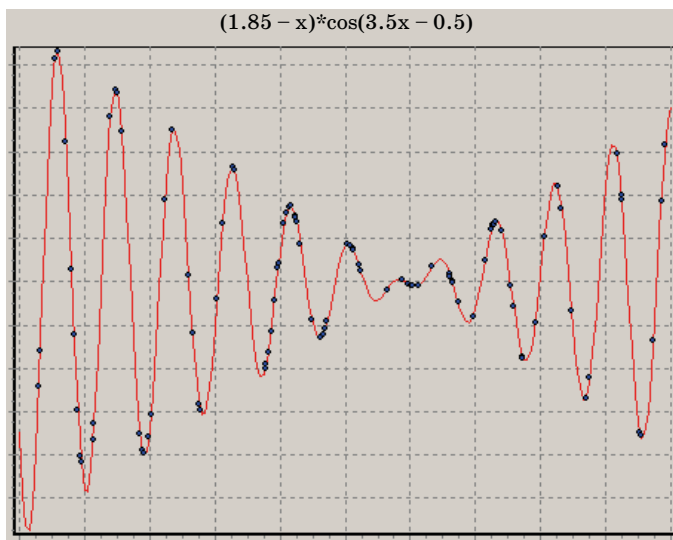


Рис. 1.5. Пример функции с популяцией особей в начале эволюции

ставления  $(b_{14}b_{13}...b_0)$  ( $b_i \in \{0,1\}$ ) в вещественное число из отрезка  $[-10,+10]$  выполняется в два шага.

1) Перевод двоичного числа в десятичное:

$$(<b_{14}b_{13}...b_0>)_2 = \left( \sum_{i=0}^{14} b_i 2^i \right)_{10} = X'$$

1) Вычисление соответствующего вещественного числа  $x$ :

$$x = -10 + x' \cdot \frac{10 - (-10)}{2^{15} - 1},$$

где  $-10$  левая граница области решения.

Естественно хромосомы

(000000000000000) и (111111111111111)

представляют границы отрезка  $-10$  и  $+10$  соответственно.

Очевидно, при данном двоичном представлении вещественных чисел можно использовать классический простой ГА. На рис. 1.5; рис. 1.8 представлено расположение особей – потенциальных решений на различных этапах ГА в процессе поиска решения. На рис. 1.5

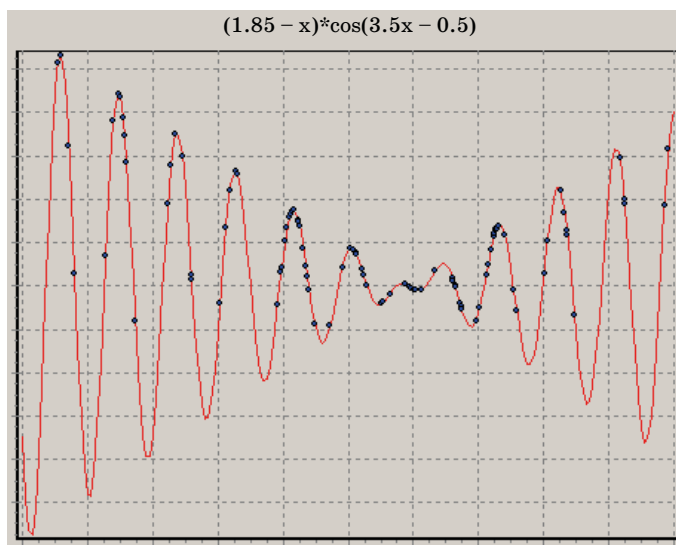


Рис. 1.6. Начальная «конденсация» особей популяции в окрестностях экстремумов

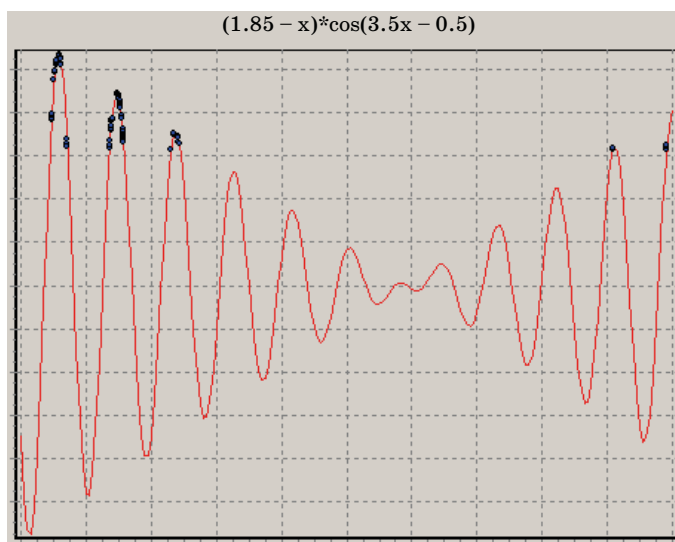


Рис. 1.7. «Конденсация» особей в окрестностях экстремумов

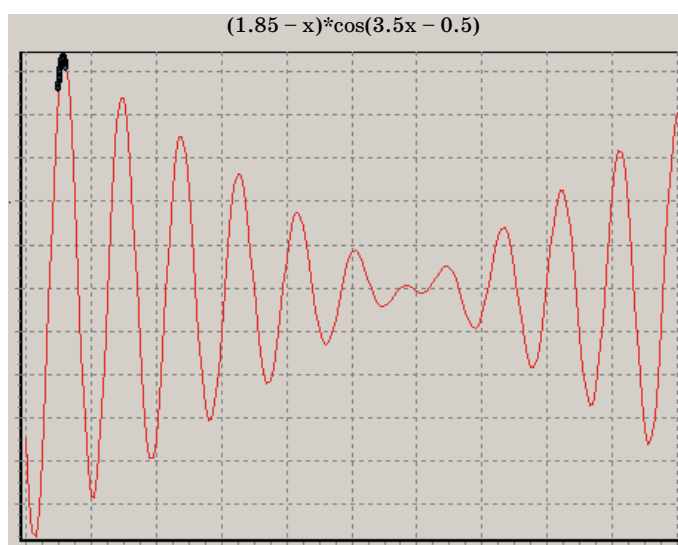


Рис. 1.8. Положение особей популяции в конце эволюции



показана начальная популяция потенциальных решений, которая равномерно покрывает область поиска решения. Далее явно видно, как постепенно с увеличением номера поколения особи «конденсируются» в окрестностях экстремумов и в конечном счете находится лучшее решение.

## **Логарифмическое кодирование**

Данный вид кодирования применяется для сокращения длины хромосом. При этом первый бит (a) кодовой последовательности используется для знака показательной функции, второй бит (b) – для знака степени этой функции, и остальные биты (str) представляют значение самой степени. Таким образом, двоичный код  $\langle a \ b \ str \rangle$  представляет вещественное число  $(-1)^b e^{(-1)^a [str]_{10}}$ . Здесь  $[str]_{10}$  означает десятичное число, представленное двоичным кодом str. Например, двоичный код  $\langle 10110 \rangle$  представляет вещественное число  $r = (-1)^0 e^{(-1)^1 [110]_{10}} = e^{-6} = 0,002478752$ . Следует отметить, что при этом кодировании пять битов позволяет кодировать вещественные числа из интервала  $[-e^7, e^7]$ , что значительно больше, чем метод кодирования вещественных чисел, описанный выше.

## **1.2. Практическая часть**

### **Порядок выполнения лабораторной работы**

#### **1. При домашней подготовке:**

- изучить теоретический материал;
- ознакомиться с вариантами кодирования хромосомы;
- рассмотреть способы выполнения операторов репродукции, кроссинговера и мутации;
- выполнить индивидуальное задание на любом языке высокого уровня с необходимыми комментариями и выводами.

#### **2. Во время занятия:**

- продемонстрировать результаты выполнения работы;
- получить допуск к защите лабораторной работы.

#### **3. Защитить отчет по лабораторной работе.**

## **Задание**

1. *Разработать* простой генетический алгоритм для нахождения оптимума заданной по варианту функции одной переменной (табл. 1.1).

Вид экстремума:

Вариант	Вид экстремума
$\leq 15$	Максимум
$> 15$	Минимум

2. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:

- число особей в популяции
- вероятность кроссинговера, мутации.

3. Вывести на экран график данной функции с указанием найденного экстремума для каждого поколения.

4. Сравнить найденное решение с действительным.

Таблица 1.1

#### Индивидуальные задания

Вариант	Вид функции	Промежуток нахождения решения
1	$(1.85 - x) \cdot \cos(3.5x - 0.5)$	$x \in [-10, 10]$
2	$\cos(\exp(x)) / \sin(\ln(x))$	$x \in [2, 4]$
3	$\sin(x) / x^2$	$x \in [3.1, 20]$
4	$\sin(2x) / x^2$	$x \in [-20, -3.1]$
5	$\cos(2x) / x^2$	$x \in [-20, -2.3]$
6	$(x - 1) \cos(3x - 15)$	$x \in [-10, 10]$
7	$\ln(x) \cos(3x - 15)$	$x \in [1, 10]$
8	$\cos(3x - 15) / \text{abs}(x) = 0$	$x \in [-10, -0.3], (0.3, 10]$ $x \in [-0.3, 0.3]$
9	$\cos(3x - 15) \cdot x$	$x \in [-9.6, 9.1]$
10	$\sin(x) / (1 + \exp(-x))$	$x \in [0.5, 10]$
11	$\cos(x) / (1 + \exp(-x))$	$x \in [0.5, 10]$
12	$(\exp(x) - \exp(-x)) \cos(x) / (\exp(x) + \exp(-x))$	$x \in [-5, 5]$
13	$(\exp(-x) - \exp(x)) \cos(x) / (\exp(x) + \exp(-x))$	$x \in [-5, 5]$
14	$\cos(x - 0.5) / \text{abs}(x)$	$x \in [-10, 0), (0, 10], \min$
15	$\cos(2x) / \text{abs}(x - 2)$	$x \in [-10, 2), (2, 10], \max$

## Содержание отчета

1. Титульный лист.
2. Индивидуальное задание по варианту.
3. Краткие теоретические сведения.
4. Программа и результаты выполнения индивидуального задания с комментариями и выводами.
5. Письменный ответ на контрольный вопрос по варианту (номер контрольного вопроса совпадает с номером варианта).

## Контрольные вопросы

1. Какие “источники” ГА?
2. Какие генетические операторы используются в ГА?
3. Какую роль в ГА играет оператор репродукции (ОР)?
4. Опишите реализацию ОР в виде колеса рулетки и приведите пример его работы.
5. Придумайте другую реализацию ОР.
6. Опишите 1-точечный оператор кроссинговера (ОК) и приведите пример его работы.
7. Придумайте другую реализацию ОК.
8. Какую роль играет оператор мутации (ОМ)?
9. Опишите ОМ и приведите пример его работы.
10. Придумайте другую реализацию ОМ.
11. Выполните программную реализацию простого ГА на одном из языков программирования для поиска максимума функции  $f(x) = 3x^3 + 2$  на интервале  $[-5, 5]$ .
12. Какие основные параметры ГА?
13. Исследуйте зависимость работы (скорость сходимости) ПГА от мощности популяции  $N$  (варьируя ее значение).
14. Исследуйте зависимость работы ПГА от значения вероятности  $OK P_c$ .
15. Исследуйте зависимость работы ПГА от значения вероятности  $OK P_m$ .

## Лабораторная работа №2

### ОПТИМИЗАЦИЯ МНОГОМЕРНЫХ ФУНКЦИЙ С ПОМОЩЬЮ ГА

*Цель работы:* модификация представления хромосомы и операторов рекомбинации ГА для оптимизации многомерных функций. Графическое отображение результатов оптимизации.

При выполнении лабораторной работы можно использовать следующие источники из прилагаемого списка литературы [1, 3, 4, 8–12, 14].

#### 2.1. Теоретическая часть

##### Представление хромосомы

При работе с оптимизационными задачами в непрерывных пространствах вполне естественно представлять гены напрямую вещественными числами. В этом случае хромосома есть вектор вещественных чисел. Их точность будет определяться исключительно разрядной сеткой той ЭВМ, на которой реализуется алгоритм. Длина хромосомы будет совпадать с длиной вектора-решения оптимизационной задачи, иначе говоря, каждый ген будет отвечать за одну переменную. Генотип объекта становится идентичным его фенотипу.

Вышесказанное определяет список основных преимуществ алгоритмов с вещественным кодированием:

1. Использование непрерывных генов делает возможным поиск в больших пространствах (даже в неизвестных), что трудно делать в случае двоичных генов, когда увеличение пространства поиска сокращает точность решения при неизменной длине хромосомы.

2. Одной из важных черт непрерывных ГА является их способность к локальной настройке решений.

3. Использование RGA для представления решений удобно, поскольку близко к постановке большинства прикладных задач. Кроме того, отсутствие операций кодирования/декодирования, которые необходимы в BGA, повышает скорость работы алгоритма.

Появление новых особей в популяции канонического ГА обеспечивают несколько биологических операторов: отбор, скрещивание и мутация. В качестве операторов отбора особей в родительскую пару здесь подходят любые известные из BGA: рулетка, турнирный, случайный. Однако операторы скрещивания и мутации в классических реализациях работают с битовыми строками. Необходимы реализации, учитывающие специфику real-coded алгоритмов.

Также рекомендуется использовать стратегию элитизма – лучшая особь сохраняется отдельно и не стирается при смене эпох, принимая при этом участие в отборе и рекомбинации.

### Операторы кроссинговера

Оператор скрещивания непрерывного ГА, или кроссовер, порождает одного или нескольких потомков от двух хромосом. Собственно говоря, требуется из двух векторов вещественных чисел получить новые векторы по каким-либо законам. Большинство real-coded алгоритмов генерируют новые векторы в окрестности родительских пар. Далее представлены простые и популярные кроссоверы.

Пусть  $C1=(c11,c21,...,cn1)$  и  $C2=(c12,c22,...,cn2)$  – две хромосомы, выбранные оператором селекции для скрещивания. После формулы для некоторых кроссоверов приводится рисунок – геометрическая интерпретация его работы. Предполагается, что  $ck1 \leq ck2$  и  $f(C1) \geq f(C2)$ .

**Плоский кроссовер (flat crossover):** создается потомок  $H=(h_1,...,h_k,...,h_n)$ , где  $h_k$ ,  $(k=1,...,n)$  – случайное число из интервала  $[c_{k1}, c_{k2}]$ .

**Простейший кроссовер (simple crossover):** случайным образом выбирается число  $k$  из интервала  $\{1,2,...,n-1\}$  и генерируются два потомка  $H1=(c_{11},c_{21},...,c_{k1},c_{k+12},...,c_{n2})$  и  $H2=(c_{12},c_{22},...,c_{k2},c_{k+11},...,c_{n1})$ .

**Арифметический кроссовер (arithmetical crossover):** создаются два потомка  $H1=(h_{11},...,h_{n1})$ ,  $H2=(h_{12},...,h_{n2})$ , где  $h_{k1}=w*c_{k1}+(1-w)*c_{k2}$ ,  $h_{k2}=w*c_{k2}+(1-w)*c_{k1}$ ,  $k=1,...,n$ ,  $w$  либо константа (равномерный арифметический кроссовер) из интервала  $[0;1]$ , либо изменяется с увеличением эпох (неравномерный арифметический кроссовер).

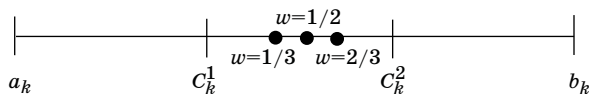


Рис. 2.1. Арифметический кроссовер

**Геометрический кроссовер (geometrical crossover):** создаются два потомка  $H1=(h_{11},...,h_{n1})$ ,  $H2=(h_{12},...,h_{n2})$ , где  $h_{k1}=(c_{k1})w*(c_{k2})(1-w)$ ,  $(c_{k2})w*(c_{k1})(1-w)$ , где  $w$  – случайное число из интервала  $[0;1]$ .

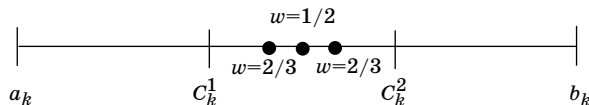


Рис. 2.2. Геометрический кроссовер

**Смешанный кроссовер** (blend, BLX-alpha crossover): генерирует-ся один потомок  $H=(h_1,...,h_k,...,h_n)$ , где  $h_k$  – случайное число из интервала  $[c_{\min}-I*\alpha, c_{\max}+I*\alpha]$ ,  $c_{\min}=\min(ck1, ck2)$ ,  $c_{\max}=\max(ck1, ck2)$ ,  $I=c_{\max}-c_{\min}$ . BLX-0.0 кроссовер превращается в плоский.

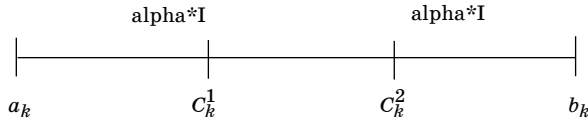


Рис. 2.3. Смешанный кроссовер

**Линейный кроссовер** (linear crossover): создаются три потомка  $Hq=(h_{1q},...,h_{kq},...,h_{nq})$ ,  $q=1,2,3$ , где  $h_{k1}=0.5*c_{k1}+0.5*c_{k2}$ ,  $h_{k2}=1.5*c_{k1}-0.5*c_{k2}$ ,  $h_{k3}=-0.5*c_{k1}+1.5*c_{k2}$ . На этапе селекции в этом кроссовере отбираются два наиболее сильных потомка.

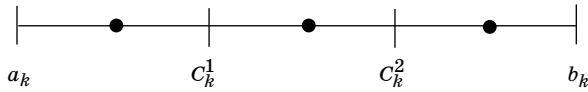


Рис. 2.4. Линейный кроссовер

**Дискретный кроссовер** (discrete crossover): каждый ген  $h_k$  выбирается случайно по равномерному закону из конечного множества  $\{c_{k1}, c_{k2}\}$ .



Рис. 2.5. Дискретный кроссовер

**Расширенный линейный кроссовер** (extended line crossover): ген  $hk=c_{k1}+w*(c_{k2}-c_{k1})$ ,  $w$  – случайное число из интервала  $[-0.25; 1.25]$ .

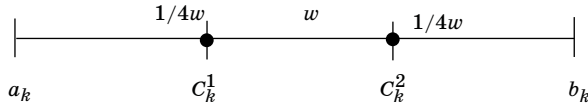


Рис. 2.6. Расширенный линейный кроссовер

**Эвристический кроссовер** (Wright's heuristic crossover). Пусть  $C1$  – один из двух родителей с лучшей приспособленностью. Тогда  $hk=w*(ck1-ck2)+ck1$ ,  $w$  – случайное число из интервала  $[0;1]$ .



Рис. 2.7. Эвристический кроссовер

### Мини-максный кроссинговер

Для представления хромосомы числами с плавающей точкой используется практически только мин-максный кроссовер или его модификации:

Для родителей  $C_w^t = (c_1, \dots, c_k, \dots, c_{3*(n+1)})$  и  $C_v^t = (c'_1, \dots, c'_k, \dots, c'_{3*(n+1)})$  получаются 4 потомка

$$C_1^{t+1} = \alpha * C_w^t + (1 - \alpha) * C_v^t,$$

$$C_2^{t+1} = \alpha * C_v^t + (1 - \alpha) * C_w^t,$$

$$C_3^{t+1} = \min\{c_k, c'_k\},$$

$$C_4^{t+1} = \max\{c_k, c'_k\}.$$

Выбираются два лучших из этих четырех потомков.

Рассмотренные кроссоверы исторически были предложены первыми, однако во многих задачах их эффективность оказывается невысокой. Исключение составляет BLX-кроссовер с параметром  $\alpha=0.5$  – он превосходит по эффективности большинство простых кроссоверов. Позднее были разработаны улучшенные операторы скрещивания, аналитическая формула которых и эффективность обоснованы теоретически. Один из таких кроссоверов – SBX.

### SBX кроссовер

SBX (англ.: Simulated Binary Crossover) – кроссовер, имитирующий двоичный. Был разработан в 1995 году исследовательской группой под руководством К. Deb'а. Как следует из его названия, он моделирует принципы работы двоичного оператора скрещивания.

SBX кроссовер был получен следующим способом. У двоичного кроссовера было обнаружено важное свойство – среднее значение функции приспособленности оставалось неизменным у родителей и их потомков, полученных путем скрещивания. Затем автором было введено понятие силы поиска кроссовера (search power). Это количественная величина, характеризующая распределение вероятно-

стей появления любого потомка от двух произвольных родителей. Первоначально была рассчитана сила поиска для одноточечного двоичного кроссовера, а затем был разработан вещественный SBX кроссовер с такой же силой поиска. В нем сила поиска характеризуется распределением вероятностей случайной величины  $\beta$ :

$$P(\beta) = \begin{cases} 0.5(n+1)\beta^n, & \text{при } \beta \leq 1, \\ 0.5(n+1)\beta^{-(n+2)}, & \text{при } \beta > 1. \end{cases}$$

Для генерации потомков используется следующий алгоритм, использующий выражение для  $P(\beta)$ . Создаются два потомка  $H_k = (h_{1k}, \dots, h_{jk}, \dots, h_{nk})$ ,  $k=1,2$ , где  $h_{j1} = 0.5[(1-\beta_k)c_{j2} + (1+\beta_k)c_{j2}]$ ,  $h_{j2} = 0.5[(1+\beta_k)c_{j1} + (1-\beta_k)c_{j2}]$ ,  $\beta_k \geq 0$  – число, полученное по формуле:

$$\beta(u) = \begin{cases} (2u)^{\frac{1}{n+1}}, & \text{при } u(0,1) \leq 0.5, \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{n+1}}, & \text{при } u(0,1) > 0.5. \end{cases}$$

В формуле  $u(0,1)$  – случайное число, распределенное по равномерному закону,  $n \in [2,5]$  – параметр кроссовера.

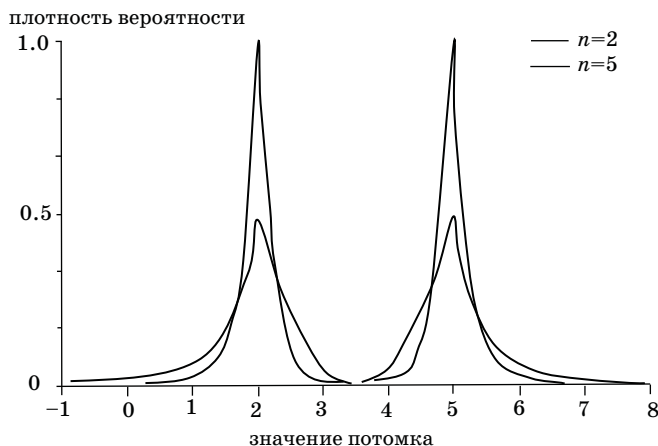


Рис. 2.8. SBX кроссовер



На рис. 2.8 приведена геометрическая интерпретация работы SBX кроссовера при скрещивании двух хромосом, соответствующих вещественным числам 2 и 5. Видно, как параметр  $n$  влияет на конечный результат: увеличение  $n$  влечет за собой увеличение вероятности появления потомка в окрестности родителя и наоборот.

Эксперименты автора SBX кроссовера показали, что он во многих случаях эффективнее BLX, хотя, очевидно, что не существует ни одного кроссовера, эффективного во всех случаях. Исследования показывают, что использование нескольких различных операторов кроссовера позволяет уменьшить вероятность преждевременной сходимости, т. е. улучшить эффективность алгоритма оптимизации в целом. Для этого могут использоваться специальные стратегии, изменяющие вероятность применения отдельного эволюционного оператора в зависимости от его «успешности», или использование гибридных кроссоверов, которых в настоящее время насчитывается несколько десятков.

### Операторы мутации

В качестве **оператора мутации** наибольшее распространение получили: случайная и неравномерная мутация (random and non-uniform mutation).

При **случайной мутации** ген, подлежащий изменению, принимает случайное значение из интервала своего изменения.

В **неравномерной мутации** из особи случайно выбирается точка  $c_k$  (с разрешенными пределами изменения  $[c_{kl} \ c_{kr}]$ ). Точка меняется на

$$c'_k = \begin{cases} c_k + \Delta(t, c_{kr} - c_k), & \text{при } a = 0, \\ c_k - \Delta(t, c_k - c_{kl}), & \text{при } a = 1, \end{cases}$$

где  $a$  – случайно выбранное направление изменения,

$\Delta(t, y)$  – функция, возвращающая случайную величину в пределах  $[0..y]$  таким образом, что при увеличении  $t$  среднее возвращаемое значение увеличивалось:

$$\Delta(t, y) = y \left( 1 - r \left( 1 - \frac{t}{T} \right)^b \right),$$

где  $r$  – случайная величина на интервале  $[0..1]$

$t$  – текущая эпоха работы генетического алгоритма

$T$  – общее разрешенное число эпох алгоритма

$b$  – задаваемый пользователем параметр, определяющий степень зависимости от числа эпох.

## 2.2. Практическая часть

### Тестовые примеры

Для данного вида задачи существует большое число тестовых примеров – Benchmark-ов. С некоторыми из них можно познакомиться, например, в [7]. Часть из них представлена в приложении 1. Для данных тестов произведено большое число исследований на скорость алгоритма, количество эпох для достижения результата и др. С результатами этих исследований можно ознакомиться в научной литературе, доступной в Internet.

Многочисленные исследования доказывают, что непрерывные ГА не менее эффективно, а часто гораздо лучше справляются с задачами оптимизации в многомерных пространствах, при этом более просты в реализации из-за отсутствия процедур кодирования и декодирования хромосом.

### Порядок выполнения лабораторной работы

1. Изучить теоретический материал.
2. Создать программу, использующую ГА для нахождения оптимума функции согласно таблице вариантов, приведенной в приложении 1. Для всех Benchmark-ов оптимумом является минимум. Программу выполнить на встроенном языке пакета Matlab (или любом доступном для вас языке программирования).
3. Для  $n=2$  вывести на экран график данной функции с указанием найденного экстремума, точек популяции. Для вывода графиков использовать стандартные возможности пакета Matlab. Предусмотреть возможность пошагового просмотра процесса поиска решения.
4. Повторить нахождение решения с использованием стандартного Genetic Algorithm toolbox. Сравнить полученные результаты.
5. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:
  - i. число особей в популяции
  - ii. вероятность кроссинговера, мутации.
6. Критерий остановки вычислений – повторение лучшего результата заданное количество раз или достижение популяцией определенного возраста (например, 100 эпох).
7. Повторить процесс поиска решения для  $n=3$ , сравнить результаты, скорость работы программы.

## **Содержание отчета**

1. Титульный лист установленной формы.
2. Условие задания с вариантом.
3. Распечатанный листинг программы.
4. Распечатка результатов выполнения программы (графиков);
5. Диаграммы исследованных зависимостей.

## **Контрольные вопросы**

1. Что такое «оптимальность»?
2. Опишите понятие «оптимизационная задача».
3. Что такое «критерий оптимизации»?
4. Что является целью оптимизационной задачи?
5. Что такое целевая функция в генетических алгоритмах?
6. Каким образом строится целевая функция?
7. Дайте понятие экстремума и оптимума целевой функции.
8. Что такое «локальный и глобальный оптимум»?
9. Каким образом в генетических алгоритмах осуществляется выбор способа представления решения?
10. Каким образом определяется эффективность генетического алгоритма?
11. Приведите основные цели и задачи генетических алгоритмов.
12. Выделите основные отличительные особенности генетических алгоритмов?
13. Поясните, как создается начальная популяция альтернативных решений?
14. Приведите основные понятия и определения генетических алгоритмов.
15. Опишите методику вещественного кодирования в ГА.

## ЛАБОРАТОРНАЯ РАБОТА №3

### РЕШЕНИЕ ЗАДАЧИ КОММИВОЯЖЕРА С ПОМОЩЬЮ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

*Цель работы:* Решение задач комбинаторной оптимизации с помощью генетических алгоритмов на примере задачи коммивояжера. Графическое отображение результатов оптимизации.

При выполнении лабораторной работы можно использовать следующие источники из прилагаемого списка литературы [1, 3, 8–10].

#### 3.1. Теоретическая часть

Задача коммивояжера (ЗК) считается классической задачей генетических алгоритмов. Она заключается в следующем: путешественник (или коммивояжер) должен посетить каждый из базового набора городов и вернуться к исходной точке. Имеется стоимость проезда из одного города в другой. Необходимо составить план путешествия, чтобы сумма затраченных средств была минимальной. Поисковое пространство для ЗК-множество из  $N$  городов. Любая комбинация из  $N$  городов, где города не повторяются, является решением. Оптимальное решение – такая комбинация, стоимость которой (сумма из стоимостей переезда между каждым из городов в комбинации) является минимальной.

ЗК – достаточно старая задача, она была сформулирована еще в 1759 году (под другим именем). Термин «Задача коммивояжера» был использован в 1932 г. в немецкой книге «The traveling salesman, how and what he should to get commissions and be successful in his business», написанную старым коммивояжером.

Задача коммивояжера была отнесена к NP-сложным задачам. Существуют строгие ограничения на последовательность, и количество городов может быть очень большим (существуют тесты, включающие несколько тысяч городов).

Кажется естественным, что представление тура – последовательность  $(i_1, i_2, \dots, i_n)$ , где  $(i_1, i_2, \dots, i_n)$  – числа из множества  $(1 \dots n)$ , представляющие определенный город. Двоичное представление городов неэффективно, так как требует специального ремонтирующего алгоритма: изменение одиночного бита может повлечь неправомерность тура.

В настоящее время существует три основных представления пути: соседское, порядковое и путевое. Каждое из этих представлений имеет собственные полностью различные операторы рекомбинации.

## Представление соседства

В представлении соседства тур является списком из  $n$  городов. Город  $J$  находится на позиции  $I$  только в том случае, если маршрут проходит из города  $I$  в город  $J$ . Например, вектор (2 4 8 3 9 7 1 5 6) представляет следующий тур: 1-2-4-3-8-5-9-6-7. Каждый маршрут имеет только одно соседское представление, но некоторые векторы в соседском представлении могут представлять неправильный маршрут. Например, вектор (2 4 8 1 9 3 5 7 6) обозначает маршрут 1-2-4-1..., т. е. часть маршрута – замкнутый цикл. Это представление не поддерживает классическую операцию кроссовера. Три операции кроссовера были определены и исследованы для соседского представления: *alternating edges* (альтернативные ребра), *subtour chunks* (куски подтуров), *heuristic crossovers* (евристический кроссовер).

Кроссовер обмен ребрами (*alternating edges*) строит потомков, выбрав (случайно) ребро от первого родителя, потом следующее ребро от второго, потом опять следующее от первого и т. д. Если новое ребро представляет замкнутый цикл, из того же родителя берут случайное ребро, которое еще не выбирался и не образуют замкнутого цикла. Для примера, один из потомков родителей

$P1 = (2\ 3\ 8\ 7\ 9\ 1\ 4\ 5\ 6)$  и

$P2 = (7\ 5\ 1\ 6\ 9\ 2\ 8\ 4\ 3)$

Может быть

$P1 = (2\ 5\ 8\ 7\ 9\ 1\ 6\ 4\ 3),$

где процесс начинался от угла (1,2) родителя  $P1$ , продолжая до угла (7,8), вместо которого выбран угол (7,8), поскольку тот образуют замкнутый цикл.

Кроссовер обмен подтуров (*subtour chunks*) создает потомков, выбирая (случайно) подтур от одного из родителей, затем случайной длины кусок от другого из родителей, и т. д. Как и в *alternating edges*, в случае образования замкнутого цикла, он ремонтируется аналогичным образом.

Эвристический кроссинговер (*heuristic crossover*) строит потомков, выбирая случайный город как стартовую точку для маршрута – потомка. Потом он сравнивает два соответствующих ребра от каждого из родителей и выбирает более короткое. Затем конечный город выбирается как начальный для выбора следующего более короткого ребра из этого города. Если на каком-то шаге получается замкнутый тур, тур продолжается любым случайным городом, который еще не посещался.

Преимущества этого представления – в том, что она позволяет схематически анализировать подобные маршруты. Это представление имеет в основании натуральные «строительные блоки» – ребра, маршруты между городами. Например, схема  $(** * 3 * 7 * **)$  описывает множество всех маршрутов с ребрами (4 3) и (6 7). Основным недостатком данного представления: множество операций бедно. Кроссовер *alternating edges* часто разрушает хорошие туры. Кроссовер *subtour chunks* имеет лучшие характеристики благодаря меньшим разрушительным свойствам. Но все равно его эксплуатационные качества все же достаточно низки. Кроссовер *heuristic crossover*, является наилучшим оператором для данного представления благодаря тому, что остальные операции «слепы». Но производительность этой операции нестабильна. В трех экспериментах на 50, 100 и 200 городах система нашла туры с 25%, 16% и 27% оптимального, приблизительно за 15 000, 20 000 и 25 000 итераций соответственно.

### Порядковое представление

Порядковое представление представляет тур как список из  $n$  городов;  $i$ -й элемент списка – номер от 1 до  $n-i-1$ . Идея порядкового представления состоит в следующем. Есть несколько упорядоченных списков городов  $C$ , которые служат как точки связи для списков с порядковым представлением. Предположим, для примера, что такой упорядоченный список прост:

(1 2 3 4 5 6 7 8 9).

Тогда тур

1-2-4-3-8-5-9-6-7

будет представлен как список  $l$  из ссылок,

$l=(1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$

и может быть интерпретирован следующим образом:

- Так как первый номер списка  $l$  1, берем первый город из списка  $C$  как первый город из тура (город номер 1), и исключаем его из списка. Часть маршрута – это 1

- Следующий номер в списке  $l$  также 1, поэтому берем первый номер из оставшегося списка. Так как мы исключили из списка  $C$  1-й город, следующий город – 2. Исключаем и этот город из списка. Маршрут:

1 – 2

- Следующий номер в списке  $l$  – 2. Берем из списка  $C$  2-й по порядку оставшийся город. Это – 4. Исключаем его из списка. Маршрут:

1 – 2 – 4

• Следующий номер в списке  $l - 1$ . Берем из списка  $C$  1-й город – с № 3. Имеем маршрут

1 – 2 – 4 – 3

• Следующий в списке  $l - 4$ , берем 4-й город из оставшегося списка (8).

Маршрут:

1 – 2 – 4 – 3 – 8

• Следующий номер в списке  $l - 1$ . Берем из списка  $C$  1-й город – с № 5. Маршрут:

1 – 2 – 4 – 3 – 8 – 5

• Следующий номер в списке  $l - 3$ . Берем следующий город из списка (9). Удаляем его из  $C$ . Маршрут:

1 – 2 – 4 – 3 – 8 – 5 – 9

• Следующий номер в списке  $l - 1$ , поэтому берем первый город из текущего списка  $C$  и следующий город маршрута (город номер 6), и удаляем его из  $C$ . Частичный маршрут имеет вид:

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6

• Последним номером в списке  $l$  всегда будет 1, поэтому берем последний оставшийся город из текущего списка  $C$  и последний город маршрута (город номер 7), и удаляем его из  $C$ . Окончательно маршрут имеет вид:

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7

Основное преимущество порядкового представления – в том, что классический кроссовер работает! Любые два маршрута в порядковом представлении, обрезанные на любой позиции и склеенные вместе, породят два потомка, каждый из которых будет правильным туром. Например, два родителя

$p1 = (1\ 1\ 2\ 1 | 4\ 1\ 3\ 1\ 1)$  и

$p2 = (5\ 1\ 5\ 5 | 5\ 3\ 3\ 2\ 1)$ ,

которые обозначают соответственно маршруты

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7 и

5 – 1 – 7 – 8 – 9 – 4 – 6 – 3 – 2,

с точкой разреза, обозначенной « $|$ » породят следующих потомков:

$o1 = (1\ 1\ 2\ 1\ 5\ 3\ 3\ 2\ 1)$  и

$o2 = (5\ 1\ 5\ 5\ 4\ 1\ 3\ 1\ 1)$ .

Эти потомки обозначают маршруты

1 – 2 – 4 – 3 – 9 – 7 – 8 – 6 – 5 и

5 – 1 – 7 – 8 – 6 – 2 – 9 – 3 – 4

Легко заметить, что части маршрута слева от линии разреза не изменились, тогда как части маршрута справа от линии разреза расположились в достаточно случайном порядке. Откровенно сла-

бые показатели этого представления также подтверждают первое впечатление о слабой возможности его использования.

### Путевое представление

Путевое представление – это, возможно, наиболее естественное представление тура. Например, тур

5 – 1 – 7 – 8 – 9 – 4 – 6 – 2 – 3

представлен просто как

(5 1 7 8 9 4 6 2 3).

Для путевого представления широко известны три операции кроссовера: частично отображенный – partially-mapped (PMX), порядковый – order (OX), циклический – cycle (CX) кроссоверы.

- PMX – строит потомков, выбирая подпоследовательность из тура от одного из родителей и сохраняя порядок и последовательность наибольшего из возможного числа городов другого родителя. Подпоследовательность маршрута выбирается двумя случайными точками разреза. Например, два родителя (разрезы отмечены “| “)

p1= (1 2 3 | 4 5 6 7 | 8 9) и

p2= (4 5 2 | 1 8 7 6 | 9 3)

могут получить потомков следующим способом. Во-первых, сегменты между точками обреза меняются местами (символом «X» обозначается неизвестный символ).

o1= (x x x | 4 5 6 7 | x x) и

o2= (x x x | 1 8 7 6 | x x)

также этот обмен определяет серию преобразований данных:

1 <-> 4, 8 <-> 5, 7 <-> 6 и 6 <-> 7

теперь мы можем заполнить остальные города, для которых нет конфликтов, из другого родителя:

o1= (x 2 3 | 4 5 6 7 | x 9) и

o2= (x x 2 | 1 8 7 6 | 9 3).

Напоследок, первый «x» из потомка o1 (который должен был быть 1, но был обнаружен конфликт) заменяется на 4 (см. серию преобразований данных). Таким же образом, второй «x» из потомка o1 меняем на 5 и «x»-ы в потомке o2 меняем на 1 и 8 соответственно. Имеем потомков:

o1= (4 2 3 | 4 5 6 7 | 5 9) и

o2= (1 8 2 | 1 8 7 6 | 9 3).

PMX разрабатывает важные общности в значениях и порядке следования, когда используется с соответствующим образом разработанным воспроизводственным планом.



• ОХ – строит потомков, выбирая кусок из одного родителя, остальные города – из другого, соблюдая очередность городов. Например, два родителя (разрезы отмечены “|”) “

p1= (1 2 3 | 4 5 6 7 | 8 9) и

p2= (4 5 2 | 1 8 7 6 | 9 3)

могут получить потомков следующим способом. Во-первых, сегменты между точками обреза копируются потомкам (символом «Х» обозначается неизвестный символ).

o1= (x x x | 4 5 6 7 | x x) и

o2= (x x x | 1 8 7 6 | x x)

далее, начиная от второй точки обреза другого родителя, записываются оставшиеся города в том же порядке, в котором они были в родителях.

o1= (2 1 8 | 4 5 6 7 | 9 3) и

o2= (3 4 5 | 1 8 7 6 | 9 2).

Кроссовер ОХ использует свойство путевого представления, что порядок городов важен, а первый город – нет. Туры:

1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 и

2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 1

являются фактически идентичными.

• СХ – строит потомков таким образом, что каждый город (и его позиция) приходят от одного из родителей. Два родителя

p1= (1 2 3 4 5 6 7 8 9) и

p2= (4 5 2 1 8 7 6 9 3)

будут порождать первого потомка, взяв первый город от первого родителя:

o1=(1 x x x x x x x).

После этого каждый город в потомке должен быть взят от одного из его родителей (с той же позиции, на которой находился предыдущий город в другом родителе). В нашем случае, это город 4. ставим его на ту же позицию, на которой он находился в p1

o1=(1 x x 4 x x x x).

Далее действуя таким же образом, находим город 8, находящийся «ниже» города 4

o1=(1 x x 4 x x 8 x).

Таким же образом ставим и города 3 и 2.

o1=(1 2 3 4 x x 8 x).

Дальнейшее заполнение тем же образом невозможно, так как ниже города 2 находится город 1, т. е. образуется замкнутый цикл. Остальные города берем из другого родителя.

o1=(1 2 3 4 7 6 9 8 5).

Действуя таким же образом, но начиная с родителя  $p_2$ , получим второго потомка

$O_2 = (4\ 1\ 2\ 8\ 5\ 6\ 7\ 3\ 9)$ .

СХ сохраняет абсолютную позицию элементов той же, что и у родителей.

Существуют и другие операции для путевого представления.

Путевое представление слишком бедно, чтобы представлять важные свойства тура, такие как ребра. Но по сравнению с другими векторными представлениями они показывают неплохие результаты, имеют достаточно широкие возможности.

## 3.2. Практическая часть

### Задание

1. Реализовать с использованием генетических алгоритмов решение задачи коммивояжера по индивидуальному заданию согласно номеру варианта (см. табл. 3.1 и прил. 2).

*Таблица 3.1*

**Варианты задания**

№ варианта	Название функции	Вид представления
1	Wi29.tsp	Представление соседства
2	Dj89.tsp	Представление соседства
3	Att48.tsp	Представление соседства
4	Bayg29.tsp	Представление соседства
5	Bays29.tsp	Представление соседства
6	Berlin52.tsp	Представление соседства
7	Eil51.tsp	Представление соседства
8	Eil76.tsp	Представление соседства
9	Wi29	Представление порядка
10	Dj89	Представление порядка
11	Att48.tsp	Представление порядка
12	Bayg29.tsp	Представление порядка
13	Bays29.tsp	Представление порядка
14	Berlin52.tsp	Представление порядка
15	Eil51.tsp	Представление порядка
16	Eil76.tsp	Представление порядка

№ варианта	Название функции	Вид представления
17	Wi29.tsp	Представление пути
18	Dj89.tsp	Представление пути
19	Att48.tsp	Представление пути
20	Bayg29.tsp	Представление пути
21	Bays29.tsp	Представление пути
22	Berlin52.tsp	Представление пути
23	Eil51.tsp	Представление пути
24	Eil76.tsp	Представление пути

2. Сравнить найденное решение с представленным в условии задачи оптимальным решением.

3. Представить графически найденное решение.

4. Проанализировать время выполнения и точность нахождения результата в зависимости от вероятности различных видов кроссовера, мутации.

### Содержание отчета

1. Титульный лист.
2. Индивидуальное задание по варианту.
3. Краткие теоретические сведения.
4. Программа и результаты выполнения индивидуального задания с комментариями и выводами.

### Контрольные вопросы

1. Поясните понятие пространства решений оптимизационной задачи?
2. В чем основная идея применения ГА для решения задачи коммивояжера?
3. Опишите структуру ГА для решения задачи коммивояжера.
4. Опишите структуру ГА для решения комбинаторных задач.
5. Тур в представлении соседства, кроссинговеры обмен ребер, обмен подтуров, эвристический.
6. Тур в порядковом представлении, используемые кроссинговеры.
7. Тур в представлении пути, кроссинговеры частично-отображенный (PMX), порядковый (OX), циклический (CX).

8. Какие оптимизационные задачи эффективно решать при помощи ГА?

9. Какие задачи называются NP-полными?

10. Почему неэффективно двоичное кодирование хромосомы при решении задачи коммивояжера?

11. Опишите основные виды недвоичного представления хромосомы для задачи коммивояжера.

12. Приведите пример задачи комбинаторной оптимизации, при которых может быть использован простой ГА с двоичным кодированием хромосомы.

## ЛАБОРАТОРНАЯ РАБОТА №4

### ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

*Цель работы:* Решение задачи символьной регрессии. Графическое отображение результатов оптимизации.

При выполнении лабораторной работы можно использовать следующие источники из прилагаемого списка литературы [1,3,5,8,12].

#### 4.1. Теоретическая часть

В генетическом программировании (ГП) в качестве особи выступает программа, представленная в определенном формате, которая решает некоторую задачу. Часто это выполняется с использованием обучающих данных и индуктивного вывода. ГП очень близко к машинному обучению и поэтому в качестве фитнес-функции как правило выступают функции ошибки. ГП работает с генетическим материалом переменной длины, что требует нестандартной формы представления генома и соответствующих генетических операторов.

Программы состояются из переменных, констант и функций, которые связаны некоторыми синтаксическими правилами. Поэтому определяется терминальное множество, содержащее константы и переменные, и функциональное множество, которое состоит, прежде всего, из операторов и необходимых элементарных функций ( $\exp(x)$ ,  $\sin(x)$  и т. п.). Следует отметить, что терминалы и функции играют различную роль. Терминалы обеспечивают входные значения в систему (программу), в то время как функции используются при обработке значений внутри системы. Термины «функции» и «терминалы» взяты из древовидного представления, и соответствуют узлам древовидных (или графоподобных) структур.

#### Терминальное множество

Терминальное множество включает в себя:

- 1) внешние входы в программу;
- 2) используемые в программе константы;
- 3) функции, которые не имеют аргументов.

Слово «терминал» используется, так как перечисленные выше объекты соответствуют терминальным (конечным, висячим) вершинам в древовидных структурах. Терминал дает некоторое (численное) значение, у него нет входных аргументов, и он имеет нулевую «арность».

Следует отметить тесную связь внешних входов с обучающими выборками, которые часто используются в ГП. При этом каждая переменная (признак, фактор и т. п.) обучающей выборки соответствует своему терминалу.

### **Функциональное множество**

Функциональное множество состоит из операторов (взятых у языков программирования) и различных функций. Оно может быть достаточно широким и включать типовые конструкции различных языков программирования, такие как:

- 1) булевы функции И, ИЛИ, НЕ и т. п.;
- 2) арифметические функции сложения, вычитания, умножения, деления;
- 3) трансцендентные функции (тригонометрические, логарифмические);
- 4) функции присваивания значения переменным ( $a:=2$ );
- 5) условные операторы (if then, else: case или switch операторы ветвления);
- 6) операторы переходов (go to, jump, call-вызов функции);
- 7) операторы цикла (while do, repeat until, for do);
- 8) подпрограммы и функции.

С одной стороны, терминальное и функциональное множество должны быть достаточно большими для представления потенциального решения. Но другой стороны не следует сильно без необходимости расширять функциональное множество, поскольку при этом резко возрастает пространство поиска решений. Набор функций существенно зависит от решаемой задачи.

### **Структуры для представления программ**

Терминалы и функции должны быть объединены по определенным правилам в некоторые структуры, которые могут представлять программы и использоваться при их выполнении. Выбор структуры существенно влияет на порядок выполнения программы, распределение и использование локальной или глобальной памяти.

В настоящее время наиболее распространенными структурами для представления особей (потенциальных решений проблемы) являются:

- 1) древовидное представление;
- 2) линейная структура;
- 3) графоподобная структура.

## Древовидное представление

В качестве примера рассмотрим арифметическую формулу, которую удобно представлять деревом. Рассмотрим арифметическую формулу  $\frac{d}{e} - a * (b + c)$  (в обычном представлении). Отметим, что дерево (генотип) рис. 4.1 также представляет эту формулу (фенотип)  $\frac{d}{e} - a * (b + c)$ .

При этом листья дерева соответствуют терминалам, а внутренние узлы – функциям.

Древовидная форма представления генотипа оказывается для данного класса задач более эффективной, и позволяет работать с программами или выражениями различной длины. Важным аспектом является также использование памяти при выполнении программы. Древовидная структура позволяет использовать только локальную память в процессе выполнения. Локальность памяти встроена в саму древовидную структуру. Значения переменных доступны для функции только в дереве, корню которого соответствует функция. Например, значения переменных  $d$ ,  $e$  являются локальными относительно узла  $/$ .

## Линейные структуры

Рассмотрим один из возможных вариантов линейной структуры ГП, ориентированного на программирование на языке Си. При этом каждая особь (программа) представлена последовательностью переменных длины операторов Си. Ниже представлен пример такой программы.

```
void int u
double v[8]
{
...
```

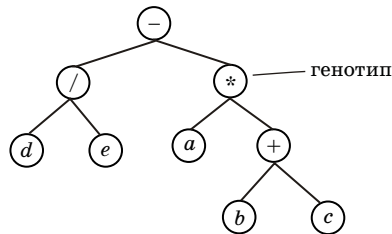


Рис. 4.1 Древовидное представление формулы  $d/e - a*(b+c)$

```

v[0]=v[5]+75;
v[7]=v[0]-69;
if (v[1]>0)
    if (v[4]>21)
        v[4]=v[2]*v[1];
v[2]=v[5]+v[4];
v[6]=v[4]-4;
v[1]=sin(v[6]);
if(v[0]>v[1])
    v[5]=v[7]+115;
if(v[1]<=v[6])
v[1]=sin(v[7]);
}

```

Здесь функциональное множество состоит из арифметических операций, условных операторов if и вызовов функций.

Переменные и константы вместе образуют множество терминальных символов. При этом подходе любой оператор кодируется 4-х мерным вектором, компонентами которого является тип оператора (одна компонента) и адреса (указатели) переменных и констант.

Например, оператор  $v_i = v_j + c$  представляется вектором  $(+, i, j, c)$ . Каждая компонента использует 1 байт памяти, следовательно, число переменных (и констант) ограничено сверху 256.

### Инициализация начальной популяции

Данный этап в ГП является не таким простым, как в классическом ГА, где генерация случайных двоичных строк не представляет особых проблем. Это связано с различной сложностью особей и их структурой. Естественно методы инициализации начальной популяции различны для разных форм представления программ. Одним из важнейших параметров в ГП является максимально возможный размер (сложность) программы.

### Инициализация древовидных структур

Для деревьев в качестве меры сложности используется максимальная глубина дерева или общее число узлов в дереве. Глубиной узла называется минимальное число узлов, которые необходимо пройти от корня дерева к этому узлу. Максимальной глубиной дерева  $D_m$  называется максимально возможная глубина в дереве для терминального символа (листа). Если арность каждого узла равна двум, то общее число узлов не превышает  $2^{B_b}$ , которое также используется в качестве меры сложности.



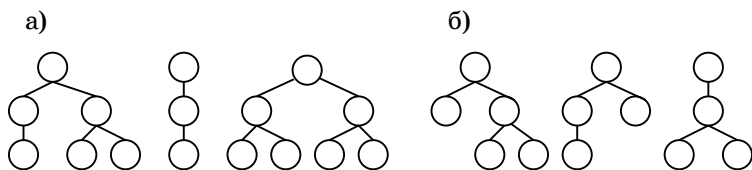


Рис. 4.2. Деревья, генерируемые при инициализации разными методами

Инициализация древовидных структур выполняется путем случайного выбора функциональных и терминальных символов при заданной максимальной глубине дерева. Применяются два основных метода: а) полная (full) и б) растущая (grow) инициализация, которые показаны на рис. 4.2.

В полном методе при генерации дерева, пока не достигнута максимальная глубина, допускается выбор только функциональных символов, а на последнем уровне (максимальной глубины) выбираются только терминальные символы.

В растущей инициализации генерируются нерегулярные деревья с различной глубиной листьев вследствие случайного на каждом шаге выбора функционального или терминального символа. Здесь при выборе терминального символа рост дерева прекращается по текущей ветви и поэтому дерево имеет нерегулярную структуру.

При использовании первого метода начальная популяция содержит однородное множество структур, что способствует вырождению генетического материала (и преждевременной сходимости в локальных экстремумах). Поэтому на практике часто эти два метода используют одновременно следующим образом. Начальная популяция генерируется так, чтобы в нее входили деревья с разной максимальной длиной примерно поровну (для нашего примера  $D_m=1$ ,  $D_m=2$ ,  $D_m=3$ ,  $D_m=4$ ). Для каждой глубины первая половина деревьев генерируется полным методом, а вторая – растущей инициализацией.

### Инициализация линейных структур

В этом случае процесс инициализации выполняется совершенно по-другому. Прежде всего, особь – программа разбивается на следующие четыре части:

- 1) Заголовок;
- 2) тело;
- 3) «подвал»;
- 4) выход (возврат).

Из них только тело программы генерируется с помощью эволюции, остальные части программы используются стандартные и заготавливаются заранее. Алгоритм инициализации можно сформулировать следующим образом:

- 1) Выбор случайной длины из заданного диапазона;
- 2) Копирование заготовленного заголовка;
- 3) Инициализация и пополнение собственно операторов в программу пока не достигнута длина, определенная в пункте 1. Операторы выбираются случайно, сначала тип, затем переменная или константа из заданного диапазона;
- 4) Копирование в конец программы заготовленного «подвал»;
- 5) Копирование в конец программы заготовленных операторов выхода.

### **Кроссинговер в генетическом программировании**

В начальной популяции особи (потенциальные решения), как правило, имеют низкие (плохие) значения фитнес-функции. В процессе эволюции с помощью генетических операторов популяция развивается и значения фитнес-функции улучшаются. В ГП используются те же, что и в ГА, генетические операторы кроссинговера, мутации и репродукции. Отметим, что в терминах машинного обучения они соответствуют операторам поиска решения. Далее мы рассмотрим выполнение генетических операторов на ранее определенных структурах.

### **Выполнение кроссинговера на древовидных структурах**

Для древообразной формы представления используются следующие три основных оператора кроссинговера (ОК):

- а) узловой ОК;
- б) кроссинговер поддеревьев;
- в) смешанный.

*В узловом операторе кроссинговера* выбираются два родителя (два дерева) и узлы в этих деревьях. Первый родитель называется доминантом, второй – рецессивом. Узлы в деревьях могут быть разного типа. Поэтому сначала необходимо убедиться, что выбранные узлы у родителей являются взаимозаменяемыми. Если узел во втором родителе не соответствует типу узла первого родителя, то случайным образом выбирается другой узел во втором родителе, который опять подлежит проверке на предмет совместимости. Далее производится обмен узлов.

В кроссинговере поддеревьев родители обмениваются не узлами, а определяемыми ими поддеревьями. Оператор выполняется следующим образом:

1. Выбираются родители (один – доминантный, другой – рецессивный). Далее необходимо убедиться, что выбранные узлы взаимозаменяемы, т. е. принадлежат одному типу. Иначе, как и в предыдущем случае в рецессивном дереве выбирается другой узел с последующей проверкой.

2. Затем производится обмен поддеревьев, которые определены этими узлами.

3. Вычисляется размер ожидаемых потомков. Если ожидаемый размер (сложность потомка) не превышает заданный порог, такой обмен ветвями запоминается.

Этот тип ОК является основным. При этом под размером (под)дерева понимается, как и ранее, либо его высота, либо число его вершин.

При смешанном операторе кроссинговера для некоторых узлов выполняется узловой ОК, а для других – кроссинговер поддеревьев.

### Кроссинговер на линейных структурах

Скрещивание на линейных структурах выполняется достаточно просто. Здесь у родителей выполняется обмен линейными сегментами, как это показано на рис. 4.3.

При этом в каждом из двух родителей выбирается сегмент, начиная со случайной позиции и имеющий случайную длину. Далее производится обмен выбранных сегментов программ. Если размер хотя бы одного из потомков превышает некоторый порог, то результаты ОК аннулируются. При этом точки скрещивания выбираются только между операторами.

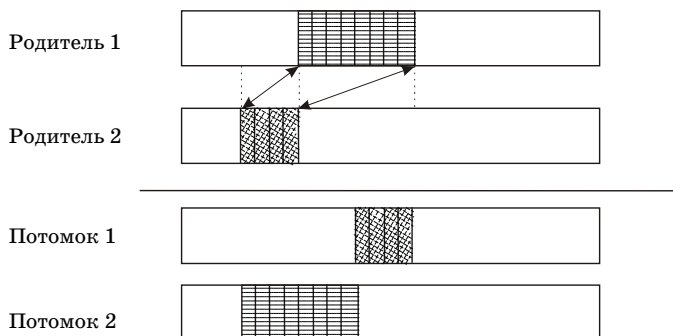


Рис. 4.3. Кроссинговер на линейных структурах

## Мутация в генетическом программировании

После выполнения кроссиговера с заданной малой вероятностью выполняется мутация для выбранной одной особи-программы.

### Выполнение мутации на древовидных структурах

Для деревьев используются следующие операторы мутации (ОМ):

- а) узловая;
- б) усекающая;
- в) растущая.

*Узловая мутация* выполняется следующим образом:

- выбрать случайным образом узел, подлежащий мутации, определить его тип;
- случайно выбрать из соответствующего множества вариантов узлов отличный от рассматриваемого узел;
- поменять исходный узел на выбранный.

*Усекающая мутация* производится так:

- определяется или выбирается узел;
- случайным образом выбирается терминальный символ из заданного множества;
- обрезается ветвь узла мутации;
- вместо обрезанной ветви помещается выбранный терминальный символ.

*Растущая мутация* выполняется следующим образом:

- определяется узел мутации;
- если узел нетерминальный то необходимо отсечь ветви исходящие из него, иначе выбрать другой узел.
- вычислить размер (сложность) остатка дерева;
- вместо отсеченного дерева вырастить случайным образом новое дерево так, чтобы размер нового построенного дерева не превышал заданный порог.

Это очень мощный оператор, который обладает большими возможностями.

### Выполнение мутации на линейных структурах

Здесь мутация выполняется совершенно другим способом. Прежде всего, из линейного сегмента (тела программы) случайно выбирается оператор (команда) и в нем производятся изменения одного из следующих видов:

- 1) имя переменной заменяется на другое случайно выбранное из заданного множества;

2) оператор может быть также изменен случайным образом на некоторый другой из функционального множества;

3) может быть случайно изменено значение константы на некоторое другое значение из заданного диапазона.

### Фитнесс-функция в генетическом программировании

В отличие от генетических алгоритмов, где часто при поиске экстремумов в качестве фитнесс-функции используется исходная целевая функция, в ГП фитнесс-функция обычно определяет меру близости между реальными  $y_i$  и требуемыми  $d_i$  выходными значениями (например, при использовании ГП в символьной регрессии). Поэтому в качестве фитнесс-функции часто используется абсолютная или квадратичная ошибка.

В этом случае фитнесс-функция использует обучающее множество данных, на котором выполняется обучение системы. С помощью фитнесс-функции в процессе обучения реализуется обратная связь, которая показывает насколько хорошо данная особь-программа реализует необходимую функцию на обучающем множестве. Для этого можно использовать различные виды фитнес-функций, некоторые из которых мы рассмотрим ниже.

Рассмотрим следующий пример [5] с обучающей выборкой, представленной табл. 6.7. Каждая строка таблицы определяет один элемент  $(x, y)$  обучающей выборки. Необходимо в процессе эволюции построить программу (или формулу в случае символьной регрессии), которая для каждого входного значения  $x$  вычисляет необходимое (в соответствии с табл. 6.5) значение  $y$  (фактически нам необходимо реализовать функцию  $f(x) = x^2 + x$ ).

Рассмотрим в качестве фитнесс-функции ошибку в метрике абсолютных значений  $f_a = \sum_{i=1}^n |y_i - d_i|$ , где суммирование выполняется по обучающей выборке. Эта фитнесс-функция соответствует перво-

Таблица 4.1

№	Вход $x$	Выход $d$
1	1	2
2	2	6
3	4	20
4	7	56
5	9	90

Таблица 4.2

№	Вход $x$	Выход $d$	Выход $y$	Ошибка $f_a$	Ошибка $f_s$
1	1	2	1	1	2
2	2	6	4	2	4
3	4	20	16	4	16
4	7	56	49	7	49
5	9	90	81	9	81
	Общая ошибка			23	151

му определению «непрерывной», поскольку чем ближе значения  $y_i$  к  $d_i$ , тем меньше значение фитнес-функции. Приведенная фитнес-функция является также стандартизованной, так как в случае идеального решения дает нулевое значение.

Часто в качестве фитнес-функции также используют квадратичную ошибку  $f_s = \sum_{i=1}^n (y_i - d_i)^2$ . Таблица 6.8 показывает различие

для этих двух фитнес-функций в том случае, если на некотором (промежуточном) этапе в качестве особи оценивается (плохо обученная) программа, реализующая функцию  $f(x) = x^2$ .

Мы рассмотрели использование в качестве фитнес-функции ошибки в двух метриках, которые характерны для применения ГП в качестве символьной регрессии.

Естественно разработано множество типов фитнес-функций вид которых существенно зависит от исследуемой проблемы. В некоторых случаях, кроме близости решений учитываются и другие критерии, например, длина или время выполнения программы. В этом случае говорят о многокритериальных фитнес-функциях.

### Общий алгоритм генетического программирования

Таким образом, для решения задачи с помощью ГП необходимо выполнить описанные выше предварительные этапы:

- 1) Определить терминальное множество;
- 2) Определить функциональное множество;
- 3) Определить фитнес-функцию;
- 4) Определить значения параметров, такие как мощность популяции, максимальный размер особи, вероятности кроссинговера и мутации, способ отбора родителей, критерий окончания эволюции (например, максимальное число поколений) и т. п.

После этого можно разрабатывать непосредственно сам эволюционный алгоритм, реализующий ГП для конкретной задачи.

Например, решение задачи на основе ГП можно представить следующей последовательностью действий.

- 1) установка параметров эволюции;
- 2) инициализация начальной популяции;
- 3)  $t:=0$ ;
- 4) оценка особей, входящих в популяцию;
- 5)  $t:=t+1$ ;
- 6) отбор родителей;
- 7) создание потомков выбранных пар родителей – выполнение оператора кроссинговера;
- 8) мутация новых особей;
- 9) расширение популяции новыми порожденными особями;
- 10) сокращение расширенной популяции до исходного размера;
- 11) если критерий останова алгоритма выполнен, то выбор лучшей особи в конечной популяции – результат работы алгоритма. Иначе переход на шаг 4.

### Символьная регрессия

Этот раздел является одним из важнейших приложений ГП. Данный термин подчеркивает то, что здесь объектом поиска является символьное описание модели, в отличие от множества коэффициентов в стандартных методах. Этот подход существенно отличается от других методов регрессии и использования нейросетей прямого распространения, где структура (и сложность) модели предполагается известной и фактически необходимо найти только ее коэффициенты. В случае символьной регрессии вид и сложность функции заранее неизвестны и могут изменяться в процессе поиска.

Задача регрессии может быть определена на основе множества значений входных независимых переменных  $x$  и зависимой выходной переменной  $y$ . Целью поиска является аппроксимация  $y$  с помощью переменных  $x$  и коэффициентов  $w$  следующим образом  $y = f(x, w) + \varepsilon$ , где  $\varepsilon$  представляет шум (ошибку).

В стандартных методах регрессии вид функции  $f$  предполагается известным, например, в линейной регрессии –  $f(x, w) = w_0 + w_1 x_1 + \dots + w_n x_n$ . Здесь коэффициенты  $w_i$  обычно находятся методом наименьших квадратов. В нелинейных методах, например с использованием нейронных сетей прямого распространения, функция имеет вид  $f(x, w) = w_0 \bullet g(w_n x)$ . Здесь коэффициенты  $w_0$  и  $w_n$  представляют

синаптические веса нейронной сети выходного и скрытых слоев соответственно.

Как уже отмечалось, символьная регрессия на основе ГП не использует некоторую заранее предопределенную форму функции  $f(x, w)$ . Здесь функция  $f(x, w)$  представляется древовидной структурой и строится эволюционным методом с использованием определенного функционального и терминального множеств. В качестве фитнес-функции обычно используется квадратичная ошибка, которая оценивает качество решения и обеспечивает обратную связь при поиске решения. Для определенности обозначим функции множества, зависящие от одной переменной через  $h_1, \dots, h_k$  и функции от двух переменных как  $g_1, \dots, g_l$ . В этой нотации функция  $f(x, w)$  представляется в виде суперпозиции функций  $h_i, g_j$  и например, может иметь следующий вид:  $f(x, w) = h_1(g_2(g_1(x_3, w_1), h_2(x_1)))$ .

Заметим, что в символьной регрессии при поиске решения не используются численные методы, например, градиентные или стохастические.

Далее рассмотрим детально пример использования символьной регрессии из для аппроксимации данных, представленных в табл. 4.3. Необходимо найти функцию  $f(x)$ , которая аппроксимирует с заданной точностью эти «экспериментальные» данные. Для решения задачи определим в соответствии с вышесказанным:

Терминальное множество: переменная  $x$  и константы в диапазоне  $[-5, 5]$ ;

Функциональное множество: арифметические функции  $+$ ,  $-$ ,  $*$ ,  $\%$  (защищенное деление).

Таблица 4.3

№	Вход $x$	Выход $y$
1	0.000	0.000
2	0.100	0.005
3	0.200	0.020
4	0.300	0.045
5	0.400	0.080
6	0.500	0.125
7	0.600	0.180
8	0.700	0.245
9	0.800	0.320
10	0.900	0.405



Кожа ввел следующую удобную и «прозрачную» форму для перечисления параметров, которая представлена в табл. 4.4.

Далее приведем некоторые полученные экспериментальные данные результатов эволюции для различных запусков программ из. В начальной популяции после инициализации лучшая особь представлена деревом рис. 4.4, которая реализует (не минимальным образом!) функцию  $f_0(x) = \frac{x}{3}$ . В последующих рис. 4.5–4.8 представлены лучшие особи последующих поколений.

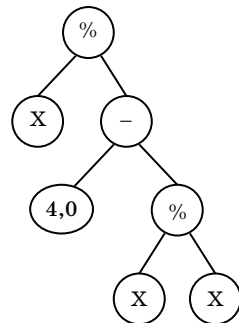


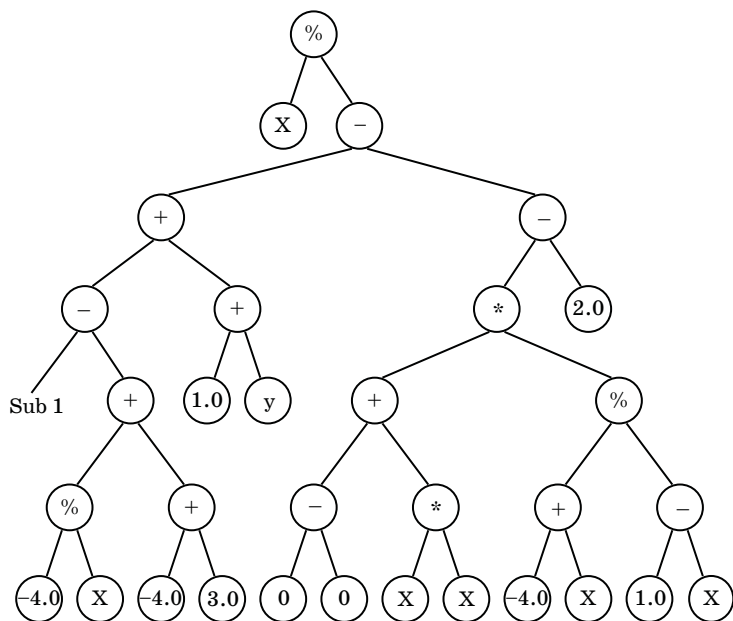
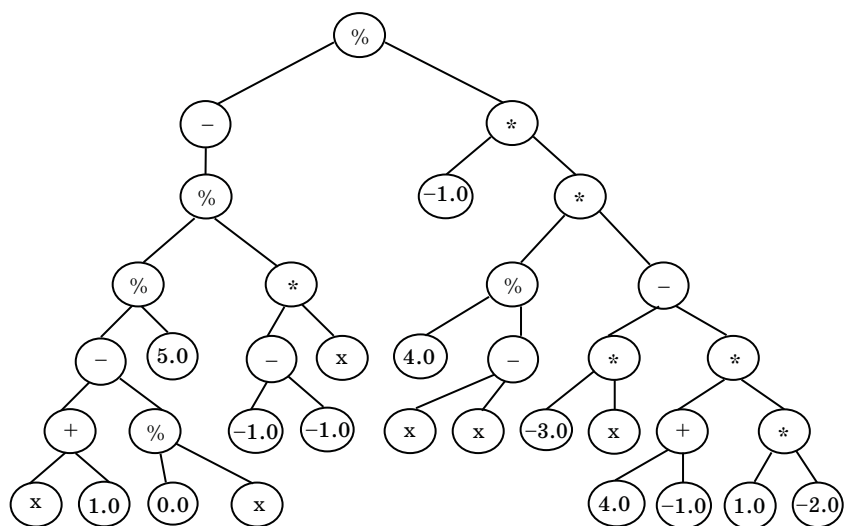
Рис. 4.4. Лучшая особь в поколении 0

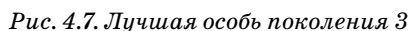
Здесь в первом поколении лучшая особь реализует  $f_1(x) = \frac{x}{6-3x}$ . В последующих рисунках рис. 4.6–4.8 представлены лучшие особи последующих поколений. Здесь в первом поколении лучшая особь реализует  $f_1(x) = \frac{x}{6-3x}$  и соответствующее дерево рис. 4.5 сильно избыточно. Аналогично во втором поколении лучшая особь реализует функцию  $f_2(x) = \frac{x}{x(x-4)-1 + \frac{4}{x} - \frac{9(x+1)}{6-3x} + x}$

$$f_2(x) = \frac{x}{x(x-4)-1 + \frac{4}{x} - \frac{9(x+1)}{6-3x} + x}$$

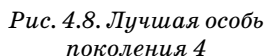
Таблица 4.4

Параметры	Значения
Цель:	Эволюция функции, аппроксимирующей данные табл. 7.6
Терминальное множество	Переменная $x$ , Целые от -5 до +5
Функциональное множество	ADD, SUB, MUL, DIV
Мощность популяции:	600
Вероятн. кроссинговера:	0.90
Вероятность мутации:	0.05
Отбор родителей:	турнирный, с мощностью тура 4
Максимальное число поколений:	100
Максимальная глубина после кроссинговера:	200
Максимальная глубина мутации:	4
Метод инициализации:	Растущая





**В табл. 6.11** для сравнения представлены значения функций лучших особей первых поколений (0-3). Отметим, что была выполнена еще одна итерация (4-е поколение). **На рис. 6.19** представлена лучшая особь четвертого



поколения, которая также реализует функцию  $f_4(x) = \frac{x^2}{2}$ , но избыточность соответствующего дерева выросла. Конечно, для данного примера можно было использовать и классические методы регрессии, но он носит чисто иллюстративный характер.

## Порядок выполнения лабораторной работы

1. При домашней подготовке:  
1) изучить теоретический материал;

2) ознакомиться с типами структур для представления программы;

3) рассмотреть способы инициализации, начальной популяции;

4) рассмотреть способы выполнения операторов кроссинговера и мутации;

5) выполнить индивидуальное задание на любом языке высокого уровня с необходимыми комментариями и выводами.

2. Во время занятия:

1) продемонстрировать результаты выполнения работы;

2) получить допуск к защите лабораторной работы.

3) защитить отчет по лабораторной работе.

### Задание

1. Разработать эволюционный алгоритм, реализующий ГП для нахождения заданной по варианту функции (табл. 4.1).

Таблица 4.5

#### Индивидуальные задания

№ вв.	Вид функции	Кол-во пер-ых N	Промежуток исследования
1	$f1(x)=\sum(x(i)^2, i=1:n;$	10	$-5.12 \leq x(i) \leq 5.12$
2	$f1a(x)=\sum(i \cdot x(i)^2, i=1:n;$	9	$-5.12 \leq x(i) \leq 5.12$
3	$f1b(x)=\sum(\sum(x(j)^2, j=1:i), i=1:n;$	8	$-5.536 \leq x(i) \leq 65.536$
4	$f2(x)=\sum(100 \cdot (x(i+1)-x(i)^2)^2 + (1-x(i))^2, i=1:n-1;$	7	$-2.048 \leq x(i) \leq 2.048$
5	$f6(x)=10 \cdot n + \sum(x(i)^2 \cdot 10 \cdot \cos(2 \cdot \pi \cdot x(i))), i=1:n;$	9	$-5.12 \leq x(i) \leq 5.12$
6	$f7(x)=\sum(-x(i) \cdot \sin(\sqrt{\text{abs}(x(i))})), i=1:n;$	10	$-500 \leq x(i) \leq 500$
7	$f8(x)=\sum(x(i)^2/4000) - \prod(\cos(x(i)/\sqrt{x(i)})) + 1; i=1:n;$	5	$-600 \leq x(i) \leq 600$
8	$f9(x)=\sum(\text{abs}(x(i))^{(i+1)}), i=1:n;$	8	$-1 \leq x(i) \leq 1$
9	$f10(x)=-a \cdot \exp(-b \cdot \sqrt{1/n \cdot \sum(x(i)^2)}) - \exp(1/n \cdot \sum(\cos(c \cdot x(i)))) + a \cdot \exp(1); a=20; b=0.2; c=2 \cdot \pi; i=1:n;$	4	$-2.768 \leq x(i) \leq 32.768$
10	$f11(x)=-\sum(c(i) \cdot (\exp(-1/\pi \cdot \sum((x-A(i))^2)) \cdot \cos(\pi \cdot \sum((x-A(i))^2))), i=1:m, m=5; A(i), C(i) > 0, m=5$	4	$0 \leq x(i) \leq 10$

№ вв.	Вид функции	Кол-во пер-ых N	Промежуток исследования
11	$f12(x) = -\sum_{i=1:n} (\sin(x(i)) \cdot (\sin(i \cdot x(i)^2 / \pi))^2 \cdot m)$ , $i=1:n, m=10$ ;	5	$0 \leq x(i) \leq \pi$
12	$fBran(x1, x2) = a \cdot (x2 - b \cdot x1^2 + c \cdot x1 - d)^2 + e \cdot (1 - f) \cdot \cos(x1) + e$ ; $a=1, b=5.1/(4 \cdot \pi^2), c=5/\pi, d=6, e=10, f=1/(8 \cdot \pi)$ ;	2	$-5 \leq x1 \leq 10$ , $0 \leq x2 \leq 15$
13	$fEaso(x1, x2) = -\cos(x1) \cdot \cos(x2) \cdot \exp(-((x1 - \pi)^2 + (x2 - \pi)^2))$ ;	2	$-100 \leq x(i) \leq 100, i=1:2$
14	$fGold(x1, x2) = [1 + (x1 + x2 + 1)^2 \cdot (19 - 14 \cdot x1 + 3 \cdot x1^2 - 14 \cdot x2 + 6 \cdot x1 \cdot x2 + 3 \cdot x2^2)] \cdot [30 + (2 \cdot x1 - 3 \cdot x2)^2 \cdot (18 - 32 \cdot x1 + 12 \cdot x1^2 + 48 \cdot x2 - 36 \cdot x1 \cdot x2 + 27 \cdot x2^2)]$	2	$-2 \leq x(i) \leq 2$ , $i=1:2$
15	$fSixh(x1, x2) = (4 - 2.1 \cdot x1^2 + x1^4/3) \cdot x1^2 + x1 \cdot x2 + (-4 + 4 \cdot x2^2) \cdot x2^2$ ;	2	$-3 \leq x1 \leq 3$ , $-2 \leq x2 \leq 2$

2. Структура для представления программы – древовидное представление.

3. Терминальное множество: переменные  $x_1, x_2, x_3, \dots, x_n$ , и константы в соответствии с заданием по варианту.

4. Функциональное множество: +, -, \*, /, abs(), sin(), cos(), exp(), возведение в степень.

5. *Фитнесс-функция – мера близости между реальными значениями выхода и требуемыми.*

6. Представить графически найденное решение на каждой итерации.

7. Сравнить найденное решение с представленным в условии задачи.

### Содержание отчета

1. Титульный лист.
2. Индивидуальное задание по варианту.
3. Краткие теоретические сведения.
4. Программа и результаты выполнения индивидуального задания с комментариями и выводами.
5. Письменный ответ на контрольный вопрос по варианту (номер контрольного вопроса совпадает с номером варианта).

### Контрольные вопросы

1. Чем отличаются терминальное и функциональное множества?

2. Какие структуры используются для представления программ в ГП?

3. Опишите древовидное представление.

4. Опишите линейное представление программы.

5. Опишите представление программы в виде графа.

6. Какие знаете методы инициализации древовидных структур?

7. Как производится инициализация линейных структур?

8. Какие виды кроссинговера вы знаете для древовидных структур?

9. Как выполняется кроссинговер на линейных структурах?

10. Какие виды кроссинговера вы знаете для графоподобных структур?

11. Какие виды мутации вы знаете для древовидных структур?

12. Как производится мутация на линейных структурах?

13. Как можно определить фитнесс-функцию в ГП?

14. Что такое интроны?

15. Приведите общий алгоритм ГП.

## Лабораторная работа №5

# ОПТИМИЗАЦИЯ МНОГОМЕРНЫХ ФУНКЦИЙ С ПОМОЩЬЮ ЭВОЛЮЦИОННОЙ СТРАТЕГИИ

### 5.1. Теоретическая часть

*Цель работы:* оптимизация функций многих переменных модификация методом эволюционной стратегии. Графическое отображение результатов оптимизации.

При выполнении лабораторной работы можно использовать следующие источники из прилагаемого списка литературы [1,3,7,11].

#### Общие сведения

Эволюционные стратегии (ЭС), также как и предыдущие парадигмы, основаны на эволюции популяции потенциальных решений, но, в отличие от них, здесь используются генетические операторы на уровне фенотипа, а не генотипа, как это делается в ГА. Разница в том, что ГА работают в пространстве генотипа – кодов решений, в то время как ЭС производят поиск в пространстве фенотипа – векторном пространстве вещественных чисел. В ЭС учитываются свойства хромосомы «в целом», в отличие от ГА, где при поиске решений исследуются отдельные гены. В природе один ген может одновременно влиять на несколько свойств организма. С другой стороны одно свойство особи может определяться несколькими генами. Естественная эволюция основана на исследовании совокупности генов, а не отдельного (изолированного) гена.

В эволюционных стратегиях целью является движение особей популяции по направлению к лучшей области ландшафта фитнес-функции. ЭС изначально разработаны для решения многомерных оптимизационных задач, где пространство поиска – многомерное пространство вещественных чисел. Иногда при решении задачи накладываются некоторые ограничения, например, вида  $g_i(x) > 0$ .

Ранние эволюционные стратегии (ЭС) основывались на популяции, состоящей из одной особи, и в них использовался только один генетический оператор – мутация. Здесь для представления особи (потенциального решения) была использована идея, не представленная в классическом генетическом алгоритме, которая заключается в следующем.

Здесь особь представляется парой действительных векторов

$$v = (\bar{x}, \bar{\sigma}), \quad (5.1)$$

где  $\bar{x}$  – точка в пространстве решений и  $\bar{\sigma}$  – вектор стандартных отклонений (вариабельность) от решения. В общем случае особь популяции определяется вектором потенциального решения и вектором «стратегических параметров» эволюции. Обычно это вектор стандартных отклонений (дисперсия), хотя допускаются (и иногда используются) и другие статистики.

Единственным генетическим оператором в классической ЭС [1] является оператор мутации, который выполняется путем сложения координат вектора-родителя со случайными числами, подчиняющимися закону нормального распределения, следующим образом:

$$\bar{x}^{t+1} = \bar{x}^t + N(0, \bar{\sigma}), \quad (5.2)$$

где  $N(0, \bar{\sigma})$  – вектор независимых случайных чисел, генерируемых согласно распределению Гаусса (например, табличным способом) с нулевым средним значением и стандартным отклонением  $\sigma$ . Как видно из приведенной формулы величина мутации управляется нетрадиционным способом. Иногда эволюционный процесс используется для изменения и самих стратегических параметров  $\sigma$ , в этом случае величина мутации эволюционирует вместе с искомым потенциальным решением. Это соответствует адаптивному ГА с изменяемым шагом мутации.

Интуитивно ясно, что увеличение отклонения подобно увеличению шага поиска на поверхности ландшафта. Высокая вариабельность способствует расширению пространства поиска и эффективна при нахождении потенциальных зон (суб)оптимальных решений и соответствует высоким значениям коэффициента мутации. В тоже время малые значения вариабельности позволяют сфокусироваться на поиске решения в перспективной области. В данном случае стратегические параметры стохастически определяют величину шага поиска: большая вариабельность ведет к большим шагам. Отметим, что поскольку отклонения генерируются стохастически (по нормальному закону), то большая вариабельность может давать маленький шаг и наоборот. Известно, что 68,26% случайных чисел при нормальном распределении попадают в интервал, определяемый стандартным отклонением  $\sigma$ ; 95% чисел попадают в интервал  $1,96\sigma$  и т. д.

### Двукратная эволюционная (1+1) – стратегия

Здесь потомок принимается в качестве нового члена популяции (он заменяет своего родителя), если значение фитнес функции (ЦФ) на нем лучше, чем у его родителя и выполняются все ограничения.



Иначе, (если значение фитнес-функции на нем хуже, чем у родителей), потомок уничтожается и популяция остается неизменной.

Рассмотрим выполнение оператора мутации на конкретном примере следующей функции [2]

$$f(x_1, x_2) = 21,5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$$

$$\begin{aligned} -3.0 \leq x_1 \leq 12.1 & \quad \bar{x} = (x_1, x_2), \\ 4.1 \leq x_2 \leq 5.8 & \quad \bar{\sigma} = (\sigma_1, \sigma_2) \end{aligned} \quad (5.3)$$

в предположении поиска максимума.

Для определенности предположим, что в  $t$ -поколении текущая особь имеет вид:

$$(\bar{x}^t, \sigma) = ((5.3; 4.9), (1.0; 1.0)). \quad (5.4)$$

Тогда потомок определяется следующим образом:

$$\left. \begin{aligned} x_1^{t+1} &= x_1^t + N(0; 1.0) = 5.3 + 0.4 = 5.7 \\ x_2^{t+1} &= x_2^t + N(0; 1.0) = 4.9 - 0.3 = 4.6 \end{aligned} \right\} \text{потомок}, \quad (5.5)$$

где числа 0.4 и 0.3 получены случайным образом в соответствии с распределением Гаусса.

Поскольку

$$f(x^t) = f(5.3; 4.9) = 18.383705 < 24.849532 = f(5.7; 4.6) = f(x^{t+1})$$

(значение ЦФ потомка лучше, чем у родителя), то полученный потомок заменяет родителя.

В целом алгоритм процесса эволюции двукратной (1+1) – эволюционной стратегии можно сформулировать следующим образом.

1. Выбрать множество  $P$  параметров  $X$ , необходимых для представления решения данной проблемы, и определить диапазон допустимых изменений каждого параметра:

$$\{x_{1\min}, x_{1\max}\}, \{x_{2\min}, x_{2\max}\}, \dots, \{x_{P\min}, x_{P\max}\},$$

установить номер поколения (итерации)  $t=0$ ;

задать стандартное отклонение  $\sigma_i$  для каждого параметра, функцию  $f$ , для которой необходимо найти оптимум, и максимальное число поколений  $k$ .

2. Для каждого параметра случайным образом выбрать начальное значение из допустимого диапазона: множество этих значений составляет начальную популяцию (из одной особи)  $X^t = (x_1, x_2, \dots, x_P)$ .

3. Вычислить значение оптимизируемой функции  $f$  для родительской особи  $F^p=f(X^t)$ .

4. Создать новую особь-потомка в соответствии с (5.2)

$$\bar{x}^* = \bar{x}^t + N(0, \bar{\sigma}).$$

5. Вычислить значение  $f$  для особи-потомка  $F^o=f(X^*)$ .

6. Сравнить значения функций  $f$  для родителя и потомка; если значение потомка  $F^o$  лучше, чем у родительской особи, то заменить родителя на потомка

$$\bar{x}^t = \bar{x}^*,$$

иначе оставить в популяции родителя.

7. Увеличить номер поколения  $t=t+1$ ;

8. Если не достигнуто максимальное число поколений  $t < k$ , то переход на шаг 4, иначе выдать найденное решение  $X^t$ .

Несмотря на то, что фактически здесь популяция состоит из одной особи, рассмотренная стратегия называется двукратной ЭС. Причина в том, что здесь фактически происходит конкуренция потомка и родителя. Обычно вектор стандартных отклонений  $\sigma$  остается неизменным в течении всего процесса эволюции.

Поэтому, чтобы оптимизировать скорость сходимости этого процесса, И. Решенберг (основоположник ЭС) предложил правило успеха «1/5».

Смысл его заключается в следующем – правило применяется после каждых  $k$  поколений процесса (где  $k$  – параметр этого метода):

$$\sigma^{t+1} = \begin{cases} c_d \cdot \sigma^t, & \text{если } \varphi(k) < 1/5 \\ c_i \cdot \sigma^t, & \text{если } \varphi(k) > 1/5, \\ \sigma^t, & \text{если } \varphi(k) = 1/5 \end{cases} \quad (5.6)$$

где  $\varphi(k)$  – отношение числа успешных мутаций к общему числу произведенных мутаций  $k$  (число успехов, деленное на  $k$ ), которое называется коэффициентом успеха для оператора мутации в течении  $k$  последних поколений; величина  $c_i > 1$ ,  $c_d < 1$  – регулирует увеличение/уменьшение отклонения мутации.

Обычно на практике оптимальные значения полагают равными следующим величинам:  $c_d=0.82$ ;  $c_i=1/0.82=1.22$ . Смысл этого правила в следующем:

– если коэффициент успеха  $\varphi(k) > 1/5$ , то отклонение  $\sigma^{t+1}$  увеличивается (мы идем более крупными шагами);

– если коэффициент успеха  $\varphi(k) < 1/5$ , то отклонение  $\sigma^{t+1}$  уменьшается (шаг поиска уменьшается).

Иногда рекомендуется устанавливать коэффициент мутации обратно пропорционально числу переменных в потенциальном решении (особи) и прямо пропорционально расстоянию от точки оптимального решения. Конечно, в реальных приложениях точное расположение оптимума неизвестно. Однако иногда может быть известна априорная информация об оптимуме (например, порядок величины). Даже ограниченная информация может быть полезна в процессе поиска в ЭС.

### Многократная эволюционная стратегия

По сравнению с двукратной многократная эволюция отличается не только размером популяции ( $N > 2$ ), но и имеет некоторые дополнительные отличия:

– все особи в поколении имеют одинаковую вероятность выбора для мутации;

– имеется возможность введения оператора рекомбинации (например, однородного ОК в ГА, рассмотренного в разделе 4), где два случайно выбранных родителя производят потомка по следующей схеме:

$$\begin{aligned} (\bar{x}^1, \bar{\sigma}^1) &= ((x_1^1, \dots, x_n^1), (\sigma_1^1, \dots, \sigma_n^1)) \\ (\bar{x}^2, \bar{\sigma}^2) &= ((x_1^2, \dots, x_n^2), (\sigma_1^2, \dots, \sigma_n^2))', \\ (\bar{x}, \bar{\sigma}) &= ((x_1^{q_1}, \dots, x_n^{q_n}), (\sigma_1^{q_1}, \dots, \sigma_n^{q_n})) \end{aligned} \quad (5.7)$$

где  $q_i=1$  или  $q_i=2$ ,  $i=1, \dots, n$  (т. е. каждая компонента потомка копируется из первого или второго родителя).

Имеется еще одно сходство между двукратными и многократными эволюционными стратегиями. При обоих видах ЭС производится только один потомок. В двукратных стратегиях потомок соревнуется со своим родителем. В многократной стратегии самая слабая особь уничтожается.

В современной литературе используются следующие обозначения:

(1+1) – ЭС – двукратная стратегия (1 родитель производит 1 потомка);

( $\mu$ +1) – ЭС– многократная стратегия ( $\mu$  родителей производят 1 потомка);

( $\mu$ + $\lambda$ ) – ЭС, где  $\mu$ -родителей производят  $\lambda$ -потомков и отбор  $\mu$  лучших представителей производится среди объединенного множества (( $\mu$ + $\lambda$ ) особей) родителей и потомков;

$(\mu, \lambda)$  – ЭС, где  $\mu$  особей родителей порождает  $\lambda$  потомков, причем  $\lambda > \mu$  и процесс выбора лучших производится только на множестве потомков.

Следует подчеркнуть, что в обоих последних видах ЭС обычно число потомков существенно больше числа родителей  $\lambda > \mu$  (иногда полагают  $\lambda/\mu=7$ ).

Укрупненный алгоритм решения задачи с помощью ЭС можно представить следующим образом.

Установка счетчика поколений  $t=0$ ;

Инициализация параметров;

Инициализация популяции  $C(0)$  из  $\mu$  особей;

```
for каждой особи  $x_i(t) \in C(t)$  do
    оценка значения фитнес-функции  $f(x_i(t))$ ;
end
while условие останова не выполнено do
    for  $i=1, \dots, \lambda$  do
        случайный выбор  $\rho \geq 2$  родительских особей;
        построение особи-потомка путем кроссинговера
        на генотипе
        и параметрах родительских особей;
        мутация генотипа и параметров особи-потомка;
        оценка значения фитнес-функции потомка;
    end
    Отбор следующего поколения популяции  $C(t+1)$ ;
     $t=t+1$ ;
end
```

Здесь на этапе инициализации генерируются особи начальной популяции со значениями в пределах ограничений и задаются начальные значения параметров. Для оценки качества особи используется абсолютное значение фитнес-функции. Далее выполняются генетические операторы отбора, кроссинговера и мутации, наиболее распространенные варианты которых представлены ниже. В качестве критерия останова может быть использован любой из рассмотренных ранее.

### Тестовые примеры

Для данного вида задачи существует большое число тестовых примеров – Benchmark-ов. С некоторыми из них можно познакомиться, например, в. Для данных тестов произведено большое число исследований на скорость алгоритма, количество эпох для достижения результата и пр. С результатами этих исследований можно ознакомиться в научной литературе, доступной в Internet.

Многочисленные исследования доказывают, что ЭС не менее эффективно, а часто гораздо лучше справляются с задачами оптимизации в многомерных пространствах, при этом более просты в реализации из-за отсутствия процедур кодирования и декодирования хромосом.

## **5.2. Практическая часть**

### **Порядок выполнения лабораторной работы**

1. Выбрать свой вариант задания.

2. Создать программу, использующую ЭС для нахождения оптимума функции согласно таблице вариантов, приведенной в приложении А. Для всех Benchmark-ов оптимумом является минимум. Программу выполнить на встроенном языке пакета Matlab (или любым, доступным вам, языке программирования).

3. Для  $n=2$  вывести на экран график данной функции с указанием найденного экстремума, точек популяции. Для вывода графиков использовать стандартные возможности пакета Matlab. Предусмотреть возможность пошагового просмотра процесса поиска решения.

4. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:

- число особей в популяции
- вероятность мутации.

Критерий остановки вычислений – повторение лучшего результата заданное количество раз или достижение популяцией определенного возраста (например, 100 эпох).

5. Повторить процесс поиска решения для  $n=3$ , сравнить результаты, скорость работы программы.

### **Содержание отчета**

1. Титульный лист установленной формы.
2. Условие задания с вариантом.
3. Распечатанный листинг программы.
4. Распечатка результатов выполнения программы (графиков).
5. Диаграммы исследованных зависимостей.

### **Контрольные вопросы**

1. Как представляется потенциальное решение в ЭС?
2. Какой генетический оператор применяется в ЭС?

3. Как выполняется мутация в ЭС?
4. Опишите двукратную ЭС.
5. Сформулируйте правило успеха в ЭС.
6. Что такое двукратная ЭС?
7. Что такое многократная ЭС?
8. Приведите основные параметры ЭС.
9. Что такое самоадаптация в ЭС?
10. Какие параметры, кроме отклонений, можно использовать в ЭС?
11. Как можно использовать углы вращения в ЭС?
12. Приведите основные стратегии самоадаптации.
13. Приведите основные типы операторов отбора в ЭС.
14. Что такое локальный и глобальный оператор кроссинговера?
15. Какие операторы рекомбинации могут быть использованы в ЭС?
16. Сформулируйте общий алгоритм решения задачи с использованием ЭС.
17. Как выполняется оператор мутации в ЭС?
18. Что такое направленная мутация?
19. Что общего между ГА и ЭС?
20. Каковы различия между ГА и ЭС?

## ЛАБОРАТОРНАЯ РАБОТА №6

### ОПТИМИЗАЦИЯ ПУТЕЙ НА ГРАФАХ С ПОМОЩЬЮ МУРАВЬИНЫХ АЛГОРИТМОВ

*Цель работы:* Решение задач комбинаторной оптимизации с помощью муравьиных алгоритмов на примере задачи коммивояжера. Графическое отображение результатов оптимизации.

При выполнении лабораторной работы можно использовать следующие источники из прилагаемого списка литературы [2,3,11,13].

#### 6.1. Теоретическая часть

Муравьиные алгоритмы (МА) основаны на использовании популяции потенциальных решений и разработаны для решения задач комбинаторной оптимизации, прежде всего, поиска различных путей на графах. Кооперация между особями (искусственными муравьями) здесь реализуется на основе моделирования. При этом каждый агент, называемый искусственным муравьем, ищет решение поставленной задачи. Искусственные муравьи последовательно строят решение задачи, передвигаясь по графу, откладывают феромон и при выборе дальнейшего участка пути учитывают концентрацию этого фермента. Чем больше концентрация феромона в последующем участке, тем больше вероятность его выбора.

#### Простой муравьиный алгоритм

Рассмотрим простой муравьиный алгоритм (ПМА) (simple ant colony optimization – SACO), в котором представлены основные аспекты муравьиных алгоритмов (МА).

В качестве иллюстрации возьмем задачу поиска кратчайшего пути между двумя узлами графа  $G=(V,E)$ , где  $V$  – множество узлов (вершин), а  $E$  – матрица, которая представляет связи между узлами. Пусть  $n_G = |V|$  – число узлов в графе. Обозначим  $L^k$  – длину пути в графе, пройденного  $k$ -м муравьем, которая равна числу пройденных дуг (ребер) от первой до последней вершины пути. Пример графа с выделенным путем представлен на рис. 6.1. С каждой дугой, соединяющей вершины  $(i,j)$  ассоциируем концентрацию феромона  $\tau_{ij}$ .

Строго говоря, в начальный момент времени концентрация феромона для каждой дуги графа нулевая, но мы для удобства каждой дуге присвоим небольшое случайное число  $\tau_{ij}(0)$ .

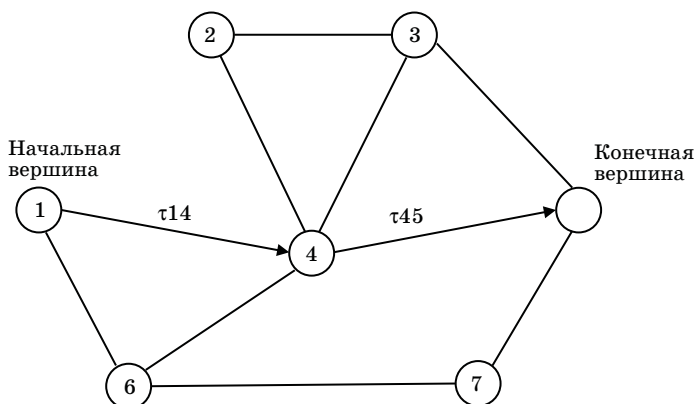


Рис. 6.1. Пример графа

Муравей выбирает следующую дугу пути следующим образом. Множество муравьев  $k=1, \dots, n_k$  помещаются в начальную вершину. В каждой итерации ПМА каждый муравей пошагово строит путь до конечной вершины. При этом в каждой вершине каждый муравей должен выбрать следующую дугу пути. Если  $k$ -й муравей находится в  $i$ -й вершине, то он выбирает следующую вершину  $j \in N_i^k$  на основе вероятностей перехода:

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha(t)}{\sum_{j \in N_i^k} \tau_{ij}^\alpha(t)}, & \text{если } j \in N_i^k \\ 0, & \text{если } j \notin N_i^k \end{cases} \quad (6.1)$$

Здесь  $N_i^k$  представляет множество возможных вершин, связанных с  $i$ -й вершиной, для  $k$ -го муравья. Если для любого  $i$ -го узла и  $k$ -го муравья  $N_i^k \equiv \emptyset$ , тогда предшественник узла  $i$  включается в  $N_i^k$ . В этом случае в пути возможны петли. Эти петли удаляются при достижении конечного города пути. В (6.1)  $\alpha$  – положительная константа, которая определяет влияние концентрации феромона. Очевидно большие значения  $\alpha$  повышают влияние концентрации феромона. Это особенно существенно в начальной стадии для начальных случайных значений концентрации, что может привести к преждевременной сходимости к субоптимальным решениям. Когда все муравьи построили полный путь от начальной до конечной вершины, удаляются петли в путях, и каждый муравей помечает



свой построенный путь, откладывая для каждой дуги феромон в соответствии со следующей формулой:

$$\Delta\tau_{ij}^k(t) \propto \frac{1}{L^k(t)} \quad (6.2)$$

Здесь  $L^k(t)$  – длина пути, построенного  $k$ -м муравьем в момент времени  $t$ .

Таким образом, для каждой дуги графа концентрация феромона определяется следующим образом:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t) \quad (6.3)$$

где  $n_k$  – число муравьев. Из (4.2) следует, что общая концентрация феромона для данной дуги пропорциональна «качеству» путей, в которые входит эта дуга, поскольку откладываемое количество феромона согласно (6.2) отражает «качество» соответствующего пути. В данном случае «качество» обратно пропорционально длине пути (числу дуг, вошедших в путь). Но в общем случае может быть использована и другая мера качества (например, стоимость проезда по данному пути или геометрическое расстояние и т. п.). Пусть  $x^k(t)$  обозначает решение в момент  $t$ , и некоторая функция  $f(x^k(t))$  выражает качество решения. Если  $\Delta\tau^k$  не пропорционально качеству решения и все муравьи откладывают одинаковое количество феромона ( $\Delta\tau_{ij}^1 = \Delta\tau_{ij}^2 = \dots = \Delta\tau_{ij}^{n_k}$ ), то существует только один фактор, который зависит от длины пути и способствует выбору коротких путей. Это ведет к двум основным способам оценки качества решений, которые используются в МА:

- неявная оценка, где муравьи используют отличие в длине путей относительно построенных путей другими муравьями;
- явная оценка, количество феромона пропорционально некоторой мере качества построенного решения.

В нашем случае (6.2) мы имеем явную оценку качества решения согласно, которая ведет к тому, что дуги, входящие в длинные пути, становятся менее привлекательными для окончательных решений.

**Алгоритм:**

```

Инициализация  $\tau_{ij}(0)$  малыми случайными значениями;
 $t=0$ ;
поместить  $n_k$  муравьев на начальную вершину;
repeat
  for каждого муравья  $k=1, \dots, n_k$  do

```

```

// построение пути  $x^k(t)$ 
 $x^k(t)=0$ ;
repeat
    выбрать следующую вершину согласно вероятности,
    определяемой (4.1);
    добавить дугу  $(i,j)$  в путь  $x^k(t)$ ;
    until конечная вершина не достигнута;
удалить петли из  $x^k(t)$ ;
вычислить длину пути  $f(x^k(t))$ 
end
for каждой дуги графа  $(i,j)$  do
    //испарение феромона
    Уменьшить концентрацию феромона согласно (4.3);
end
for каждого муравья  $k=1,...,n_k$  do
    for каждой дуги  $(i,j)$  пути  $x^k(t)$  do
        
$$\Delta\tau^k = \frac{1}{f(x^k(t))};$$

        Коррекция  $\tau_{ij}$  согласно (4.4);
    End
end
     $t=t+1$ ;
until не выполнен критерий останова;
Возврат решения - пути с наименьшим значением  $f(x^k(t))$ ;

```

В описанном алгоритме могут быть использованы различные критерии окончания, например,

- окончание при превышении заданного числа итераций;
- окончание по найденному приемлемому решению  $f(x^k(t)) \leq \varepsilon$ ;
- окончание, когда все муравьи следуют одним и тем же путем.

Для предотвращения преждевременной сходимости и расширения пространства поиска можно ввести искусственное испарение феромона на каждой итерации алгоритма следующим образом:

$$\tau_{ij}(t) \leftarrow (1-\rho)\tau_{ij}(t)\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t) \quad (4)$$

где  $\rho \in [0,1]$ . При этом константа  $\rho$  определяет скорость испарения, которое заставляет муравьи «забывать» предыдущие решения. Очевидно, что при больших значениях  $\rho$  феромон испаряется быстро, в то время как малые значения  $\rho$  способствуют медленному испарению. Отметим, что чем больше испаряется феромон, тем поиск становится более случайным.

Так при  $\rho=1$  мы имеем случайный поиск.

Следует отметить, построение решения является результатом совместного поведения, которое определяется простым поведением отдельных муравьев: каждый муравей выбирает следующий участок пути на основе информации, предоставляемой другими муравьями в форме отложений феромона. При этом при выборе муравей использует информацию только локального окружения.

Эксперименты (DorigoM., DiCaro) показали, что:

- ПМА работает хорошо для очень маленьких графов и в большинстве случаев находит кратчайший путь;
- для больших графов характеристики ухудшаются, алгоритм становится менее стабильным и более чувствительным к выбору параметров;
- сходимости к кратчайшему пути хорошая при малом числе муравьев, в то время как большое количество муравьев часто ведет к тому, что процесс поиска не сходится;
- эффект испарения более важен для сложных графов. В этом случае при  $\rho=0$  (нет испарения) алгоритм часто не сходится. С другой стороны, если феромон испаряется слишком быстро (большие значения  $\rho$ ), алгоритм часто сходится к субоптимальным решениям;
- при малых значениях  $\alpha$  алгоритм в основном сходится к кратчайшему пути. Для сложных задач (высокой размерности) большое значение  $\alpha$  ведет к плохой сходимости.

Эти исследования показали (как и для других эволюционных алгоритмов) важность проблемы эксплуатации-расширения пространства поиска. Характеристики ПМА можно значительно улучшить путем включения эвристической информации при выборе следующей дуги, запоминания локальной информации для предотвращения преждевременных циклов, использования различных значений коэффициентов на различных стадиях поиска.

### **Муравьиная система**

Первый муравьиный алгоритм был разработан М. Дориго. По современной классификации он относится к (antsystem) муравьиной системе (МС). По сравнению с простым муравьиным алгоритмом в МС улучшены характеристики за счет изменения метода вычисления вероятности выбора следующей вершины путем учета эвристической информации и ввода списка запрещенных вершин (tabulist). Конкретно, в МС вероятность перехода из  $i$ -й вершины в  $j$ -ю вершину определяется следующим образом

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha(t) \eta_{ij}^\beta(t)}{\sum_{u \in N_i^k} \tau_{iu}^\alpha(t) \eta_{iu}^\beta(t)}, & \text{если } j \in N_i^k(t) \\ 0, & \text{если } j \notin N_i^k(t) \end{cases}, \quad (6.5)$$

где 1)  $\tau_{ij}$  представляет апостериорную эффективность перехода из вершины  $i$  в  $j$ , которая определяется интенсивностью феромона для соответствующей дуги; 2)  $\eta_{ij}$  представляет априорную эффективность перехода из  $i$  в  $j$  на основе некоторой эвристики.

Вероятность перехода в МС, определяемая (6.5), отличается от аналога в ПМА, заданной (6.1), двумя аспектами:

1) При вычислении вероятности перехода в МС предпринята попытка сбалансировать влияние интенсивности феромона  $\tau_{ij}$  (отражающее предысторию успешных действий) и эвристической информации  $\eta_{ij}$  (выражающее предпочтительность некоторого выбора). Этот баланс управляет процессом эксплуатации-расширения в пространстве поиска решения. Баланс регулируется значениями коэффициентов  $\alpha$  и  $\beta$ . При  $\alpha=0$  информация о концентрации феромона не используется и предыдущий опыт игнорируется. Если  $\beta=0$ , то не учитывается эвристическая информация и мы имеем простой МА. Эвристическая информация о предпочтительности выбора следующей вершины может представляться в различной форме и зависит от задачи. Например, для выбора кратчайшего пути можно использовать  $\eta_{ij} = \frac{1}{d_{ij}}$ , где  $d_{ij}$  – расстояние между вершинами  $i$  и  $j$ . Очевидно, что в этом случае предпочтительней короткая дуга, исходящая из вершины  $i$ .

2) Множество  $N_i^k$  определяет множество допустимых вершин для  $k$ -го муравья. Это множество может включать соседние к  $i$  вершины, которые не посещались  $k$ -м муравьем. Для этого для каждого муравья создается и отслеживается табу-список. Вершины из этого списка удаляются из  $N_i^k$ , поскольку каждая вершина может посещаться только один раз.

Некоторые авторы вместо (6.5) в МС используют другую форму выражения для вероятности:

$$p_{ij}^k(t) = \begin{cases} \frac{\alpha \tau_{ij}(t) + (1-\alpha) \eta_{ij}(t)}{\sum_{u \in N_i^k} (\alpha \tau_{iu}(t) + (1-\alpha) \eta_{iu}(t))}, & \text{если } j \in N_i^k(t) \\ 0, & \text{если } j \notin N_i^k(t) \end{cases}. \quad (6.6)$$

Здесь параметр  $\alpha$  определяет относительную важность концентрации феромона  $\tau_{ij}(t)$  по сравнению с эвристикой  $\tau_{ij}$ . Данный вариант МС по сравнению с предыдущим не требует задания параметра  $\beta$ .

Испарение феромона реализуется согласно (6.7) – после построения пути каждым муравьем, концентрация феромона на каждой дуге корректируется следующим образом:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \Delta\tau_{ij}(t), \quad (6.7)$$

где

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t) \quad (12.10)$$

и  $\Delta\tau_{ij}^k(t)$  – количество феромона, откладываемое муравьем  $k$  на дуге  $(i, j)$  в момент времени  $t$ .

М. Дориго разработал три модификации МС, которые отличаются методом вычисления  $\Delta\tau_{ij}^k$  (в предположении, что решается задача минимизации):

1) Ant-cycle AS, где

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{Q}{f(x^k(t))}, & \text{если дуга } (i, j) \text{ есть в пути } x^k(t) \\ 0, & \text{иначе,} \end{cases} \quad (6.8)$$

где  $Q$  – положительная константа. Здесь количество феромона откладывается обратно пропорционально качеству  $f(x^k(t))$  на дугах полного пути, построенного муравьем. При этом для изменения концентрации феромона используется глобальная информация.

При решении задач максимизации в этом случае

$$\Delta\tau_{ij}^k(t) = \begin{cases} Qf(x^k(t)), & \text{если дуга } (i, j) \text{ есть в пути } x^k(t) \\ 0, & \text{иначе.} \end{cases} \quad (6.9)$$

1) Ant-density AS, где

$$\Delta\tau_{ij}^k(t) = \begin{cases} Q, & \text{если дуга } (i, j) \text{ есть в пути } x^k(t) \\ 0, & \text{иначе.} \end{cases} \quad (6.10)$$

В этой модификации каждый муравей откладывает одинаковое количество феромона на любой дуге построенного пути. Этот подход

учитывает только количество муравьев, прошедших по данной дуге  $(i,j)$ . Чем выше плотность трафика на дуге, тем более она привлекательна для окончательного решения.

1) Ant-quantity AS, для которой

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{Q}{d_{ij}}, & \text{если дуга } (i,j) \text{ есть в пути } x^k(t) \\ 0, & \text{иначе} \end{cases} \quad (6.11)$$

В этом случае при коррекции концентрации феромона используется только локальная информация – расстояние  $d_{ij}$  и МС предпочитает выбирать короткие дуги.

В целом МС-алгоритм представлен ниже псевдокодом А2. Здесь на этапе инициализации размещение муравьев определяется решаемой задачей. Если целью является поиск кратчайшего пути между заданными вершинами графа, то все  $n_k$  муравьев размещаются на начальной вершине. С другой стороны, если целью является построение кратчайшего гамильтонова цикла (соединяющего все вершины), то  $n_k$  муравьев случайно размещаются на всем графе. Это расширяет пространство поиска. Инициализация феромона выполняется с помощью либо малой константы  $\tau_0$ , либо небольших значений из диапазона  $[0, \tau_0]$ .

*Алгоритм А2*

```
t=0;
инициализация всех параметров  $\alpha, \beta, \rho, Q, n_k, \tau_0$ ;
поместить  $n_k$  муравьев на соответствующие вершины;
for каждой дуги  $(i,j)$  do
     $\tau_{ij}(t) \sim U(0, \tau_0)$ ;
end
repeat
    for каждого муравья  $k=1, \dots, n_k$  do
         $x^k(t) = \emptyset$ ;
        repeat
            в текущей вершине  $i$  выбрать следующую вершину  $j$ 
            согласно вероятности, определяемой (12.6);
            добавить дугу  $(i,j)$  в путь  $x^k(t) = x^k(t) \cup (i,j)$ ;
        until полный путь не построен;
        вычислить  $f(x^k(t))$ ;
    end
    for каждой дуги графа  $(i,j)$  do
```

```

//испарение феромона
Уменьшить концентрацию феромона согласно (12.5);
Вычислить  $\tau_{ij}(t)$  в соответствии с (12.10);
Изменить концентрацию феромона согласно (12.4);
end
    for каждой дуги (i,j) do
         $\tau_{ij}(t+1) = \tau_{ij}(t)$  ;
    End
t=t+1;
until не выполнен критерий останова;
Возврат  $x^k(t): f(x^k(t)) = \min_{k'=1 \dots n_k} \{f(x^{k'}(t))\}$  ;

```

Автор МС М.Дориго исследовал характеристики всех трех приведенных модификаций, прежде всего, при решении задачи коммивояжера. Версия Ant-cycle AS работала быстрее, в силу использования глобальной информации.

Кроме этого Дориго ввел стратегию элитизма, где в дополнение коррекции феромона согласно (12.4) дополнительно добавляется количество феромона, пропорциональное длине лучшего пути для всех его дуг следующим образом:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \Delta\tau_{ij}(t) + n_e \Delta\tau_{ij}^e(t), \quad (6.12)$$

где

$$\Delta\tau_{ij}(t) = \begin{cases} \frac{Q}{f(\tilde{x}(t))}, & \text{if } (i,j) \in \tilde{x}(t) \\ 0, & \text{иначе} \end{cases}. \quad (12.16)$$

Здесь  $e$  – число элитных муравьев,  $\tilde{x}(t)$  – лучшее корректное решение с

$$f(\tilde{x}(t)) = \min_{k=1, \dots, n_k} \{f(x^k(t))\}.$$

## 6.2. Практическая часть

### Порядок выполнения лабораторной работы

#### Часть 1

1. Создать программу, использующую МА для решения задачи поиска гамильтонова пути. Индивидуальное заданию выбирается по таблице В.1 в приложении В согласно номеру варианта.

2. Представить графически найденное решение. Предусмотреть возможность пошагового просмотра процесса поиска решения.

3. Сравнить найденное решение с представленным в условии задачи оптимальным решением.

## **Часть 2**

Реализовать с использованием муравьиных алгоритмов решение задачи коммивояжера по индивидуальному заданию согласно номеру варианта (см. табл. 3.1 и прил. Б).

Представить графически найденное решение.

Сравнить найденное решение с представленным в условии задачи оптимальным решением и результатами, полученными в лабораторной работе №3.

Проанализировать время выполнения и точность нахождения результата в зависимости от вероятности различных видов кроссовера, мутации.

## **Содержание отчета**

1. Титульный лист.
2. Индивидуальное задание по варианту.
3. Краткие теоретические сведения.
4. Программа и результаты выполнения индивидуального задания с комментариями и выводами.

## **Контрольные вопросы**

1. Как представляется потенциальное решение задачи в МА?
2. Что отражает и как определяется концентрация феромона в простом МА?
3. Зачем нужно и как определяется испарение феромона в простом МА?
4. Опишите простой МА.
5. Объясните влияние параметров  $\alpha, \beta$  в правиле выбора следующей вершины (12.2).
6. Как оценивается качество построенного решения в МА?
7. Какие критерии окончания могут быть использованы в простом МА?
8. Опишите основные параметры МА.
9. Приведите общий алгоритм МА.
10. Что общего между генетическими и муравьиными алгоритмами.
11. Чем отличаются муравьиные и генетические алгоритмы?



## Варианты задания

№ варианта	Название функции
1	Wi29.tsp
2	Dj89.tsp
3	Att48.tsp
4	Bayg29.tsp
5	Bays29.tsp
6	Berlin52.tsp
7	Eil51.tsp
8	Eil76.tsp
9	Wi29
10	Dj89
11	Att48.tsp
12	Bayg29.tsp
13	Bays29.tsp
14	Berlin52.tsp
15	Eil51.tsp
16	Eil76.tsp
17	Wi29.tsp
18	Dj89.tsp
19	Att48.tsp
20	Bayg29.tsp
21	Bays29.tsp
22	Berlin52.tsp
23	Eil51.tsp
24	Eil76.tsp

## Содержание отчета

1. Титульный лист установленной формы.
2. Задание, учитывая свой вариант.
3. Краткие теоретические сведения.
4. Распечатанный листинг программы.
5. Распечатка результатов выполнения программы (графиков).

## Лабораторная работа №7

### ОПТИМИЗАЦИЯ ФУНКЦИЙ МНОГИХ ПЕРЕМЕННЫХ С ПОМОЩЬЮ РОЕВЫХ АЛГОРИТМОВ

*Цель работы:* оптимизация функций многих переменных методом роевого интеллекта. Графическое отображение результатов оптимизации.

При выполнении лабораторной работы можно использовать следующие источники из прилагаемого списка литературы [2,3,11,13].

#### 7.1. Теоретическая часть

##### Основной роевой алгоритм

Роевой алгоритм (РА) использует рой частиц, где каждая частица представляет потенциальное решение проблемы.

Поведение частицы в гиперпространстве поиска решения все время подстраивается в соответствии со своим опытом и опытом своих соседей.

Кроме этого, каждая частица помнит свою лучшую позицию с достигнутым локальным лучшим значением целевой (фитнесс-) функции и знает наилучшую позицию частиц – своих соседей, где достигнут глобальный на текущий момент оптимум.

В процессе поиска частицы роя обмениваются информацией о достигнутых лучших результатах и изменяют свои позиции и скорости по определенным правилам на основе имеющейся на текущий момент информации о локальных и глобальных достижениях.

При этом глобальный лучший результат известен всем частицам и немедленно корректируется в том случае, когда некоторая частица роя находит лучшую позицию с результатом, превосходящим текущий глобальный оптимум.

Каждая частица сохраняет значения координат своей траектории с соответствующими лучшими значениями целевой функции, которые обозначим  $y_i$ , которая отражает когнитивную компоненту.

Аналогично значение глобального оптимума, достигнутого частицами роя, будем обозначать  $\hat{y}_i$ , которое отражает социальную компоненту.

Каждая  $i$ -я частица характеризуется в момент времени  $t$  своей позицией  $x_i(t)$  в гиперпространстве и скоростью движения  $v_i(t)$ .

Позиция частицы изменяется в соответствии со следующей формулой:

$$x_i(t+1) = x_i(t) + v_i(t+1), \text{ где } x_i(0) \sim (x_{\min}, x_{\max}). \quad (7.1)$$

Вектор скорости  $v_i(t+1)$  управляет процессом поиска решения и его компоненты определяются с учетом когнитивной и социальной составляющей следующим образом:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t) [y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t) [\hat{y}_j(t) - x_{ij}(t)] \quad (7.2)$$

Здесь  $v_{ij}(t)$  –  $j$ -я компонента скорости ( $j=1, \dots, n_x$ )  $i$ -й частицы в момент времени  $t$ ,  $x_{ij}(t)$  –  $j$ -я координата позиции  $i$ -й частицы,  $c_1$  и  $c_2$  – положительные коэффициенты ускорения (часто полагаемые 2), регулирующие вклад когнитивной и социальной компонент,  $r_{1j}(t), r_{2j}(t) \sim (0,1)$  – случайные числа из диапазона  $[0,1]$ , которые генерируются в соответствии с нормальным распределением и вносят элемент случайности в процесс поиска. Кроме этого  $y_{ij}(t)$  – персональная лучшая позиция по  $j$ -й координате  $i$ -й частицы, а  $\hat{y}_j(t)$  – лучшая глобальная позиция роя, где целевая функция имеет экстремальное значение.

При решении задач минимизации персональная лучшая позиция в следующий момент времени  $(t+1)$  определяется следующим образом:

$$y_i(t+1) = \begin{cases} y_i(t) & \text{iff } (x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & \text{iff } (x_i(t+1)) < f(y_i(t)) \end{cases}, \quad (7.3)$$

где  $f: R^{n_x} \rightarrow R$  – фитнес-функция.

Как и в эволюционных алгоритмах фитнес-функция измеряет близость текущего решения к оптимуму.

Глобальная лучшая позиция  $\hat{y}_j(t)$  в момент  $t$  определяется в соответствии с

$$\hat{y}_j(t) \in \{y_0(t), \dots, y_{n_s}(t)\} | f(\hat{y}_j(t)) = \min \{f(y_0(t)), \dots, f(y_{n_s}(t))\}, \quad (7.4)$$

где  $n_s$  – общее число частиц в рое.

В процессе поиска решения описанные действия выполняются для каждой частицы роя. Укрупненный основной роевой алгоритм представлен ниже.

#### Глобальный роевой

```

Создание инициализация  $n_x$ -мерного роя;
repeat
  for каждой частицы  $i=1, \dots, n_s$  do
    // определить персональную лучшую позицию
    If  $f(x_i) < f(y_i)$  then
       $y_i = x_i$ ;
end
```

```

// определить глобальную лучшую позицию
    if  $f(y_i) < f(\hat{y})$  then
 $(\hat{y}) = y_i$ ;
end
end
for каждой частицы  $i=1, \dots, n_s$  do
    коррекция скорости согласно (11.2);
    коррекция позиции согласно (11.1);
end
until критерий останова не выполнен;

```

Рассмотрим влияние различных составляющих при вычислении скорости частицы в соответствии с (7.2).

Первое слагаемое в (7.2)  $v_i(t)$  сохраняет предыдущее направление скорости  $i$ -й частицы и может рассматриваться как момент, который препятствует резкому изменению направления скорости и выступает в роли инерционной компоненты.

Когнитивная компонента  $c_1 r_1 (y_i - x_i)$  определяет характеристики частицы относительно ее предистории, которая хранит лучшую позицию данной частицы.

Эффект этого слагаемого в том, что оно пытается вернуть частицу назад в лучшую достигнутую позицию.

Третье слагаемое  $c_2 r_2 (\hat{y} - x_i)$  определяет социальную компоненту, которая характеризует частицу относительно своих соседей.

Эффект социальной компоненты в том, что она пытается направить каждую частицу в сторону достигнутого роём (или его некоторым ближайшим окружением) глобального оптимума.

Графически это наглядно иллюстрируется для двумерного случая, как это показано рис. 7.1.



Рис. 7.1 Коррекция скорости частицы

Представленный основной роевой алгоритм часто называют глобальным РА (Global Best PSO), поскольку здесь при коррекции скорости частицы используется информация о положении достигнутого глобального оптимума, которая определяется на основании информации, передаваемой всеми частицами роя.

В противоположность этому подходу часто используется локальный РА, где при коррекции скорости частицы используется информация, передаваемая только в каком-то смысле ближайшими соседними частицами роя.

## **7.2. Практическая часть**

### **Тестовые примеры**

Для данного вида задачи существует большое число тестовых примеров – Benchmark-ов. Для данных тестов произведено большое число исследований на скорость алгоритма, количество эпох для достижения результата и пр. С результатами этих исследований можно ознакомиться в научной литературе, доступной в Internet.

Многочисленные исследования доказывают, что РА не менее эффективны, а часто гораздо лучше справляются с задачами оптимизации в многомерных пространствах, при этом более просты в реализации из-за отсутствия процедур кодирования и декодирования хромосом.

### **Порядок выполнения лабораторной работы**

1. Разработать программу, использующую РА для нахождения оптимума функции согласно таблице вариантов, приведенной в приложении А. Для всех Benchmark-ов оптимумом является минимум. Программу выполнить на встроенном языке пакета Matlab (или любом, доступном вам, языке программирования).

2. Для  $n=2$  вывести на экран график данной функции с указанием найденного экстремума, точек популяции. Для вывода графиков использовать стандартные возможности пакета Matlab. Предусмотреть возможность пошагового просмотра процесса поиска решения.

3. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:

- i. число особей в популяции
- ii. вероятность мутации.

iii. Критерий остановки вычислений – повторение лучшего результата заданное количество раз или достижение популяцией определенного возраста (например, 100 эпох).

1. Повторить процесс поиска решения для  $n=3$ ,  $n=5$ ,  $n=10$ , сравнить результаты, скорость работы программы.

### **Содержание отчета**

1. Титульный лист установленной формы.
2. Условие задания с вариантом.
3. Распечатанный листинг программы.
4. Распечатка результатов выполнения программы (графиков).
5. Диаграммы исследованных зависимостей.

### **Контрольные вопросы**

1. Как представляется потенциальное решение в РА?
2. Как производится коррекция скорости частицы?
3. Что такое когнитивная составляющая?
4. Что такое социальная составляющая?
5. Опишите глобальный РА.
6. Чем отличаются глобальный и локальный РА?
7. Какие используются социальные структуры?
8. Опишите локальный РА.
9. Определите персональную лучшую позицию.
10. Определите глобальную лучшую позицию.
11. Приведите основные аспекты РА.
12. Приведите основные условия останова РА.
13. Как выполняется инициализация в РА?
14. Приведите основные параметры РА.
15. Какие существуют модификации РА?
16. Что общего между РА, ЭС и ГА?
17. Какие различия имеют место между РА, ГА и ЭС?

### ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ ОЦЕНКИ СТОИМОСТИ ПРОЕКТОВ В ПРОГРАММНОЙ ИНЖЕНЕРИИ

*Цель работы:* разработка эволюционного алгоритма оценки стоимости программных проектов. Графическое отображение результатов.

При выполнении лабораторной работы можно использовать следующие источники из прилагаемого списка литературы [14,18,1].

#### 8.1. Методика СОСОМО

Одной из самых популярных моделей, используемых для оценки сложности проектируемого программного обеспечения (ПО), является модель СОСОМО (COnstructive COst Model), предложенная Boehm [18]. Эта модель разработана на основе фактически статистики 63 проектов ПО (НАСА). Модель позволяет определить математическую зависимость между сложностью ПО, выраженную в килостроках кода, и затратами на его разработку, которые оцениваются в человеко-месяцах. Ядром модели является следующая формула  $Ef = aL^b$ , где  $L$  – длина кода ПО в килостроках;  $Ef$  – оценка сложности проекта в человеко-месяцах;  $a$  и  $b$  – коэффициенты (параметры) модели, которые для различных типов ПО имеют различные значения. Основная проблема модели СОСОМО заключается в том, что она не обеспечивает реальных оценок на затраты при проектировании ПО в современных условиях. То есть оценка программного обеспечения на основе существующих параметров не всегда дает точный результат; из-за этого часто требуется настройка параметров для получения более точных результатов. Поэтому в настоящее время идет активный поиск новых моделей (или развития и модификаций существующих). Это ограничение модели СОСОМО можно преодолеть путем применения методов искусственного интеллекта, таких как искусственные нейронные сети, генетические алгоритмы и другие метаэвристики.

В данной лабораторной работе для определения значений коэффициентов  $a$  и  $b$  используются генетический или роевой алгоритм в соответствии с заданным вариантом. Фактически задача сводится к машинному обучению на заданной обучающей выборке. В этом случае обучающая выборка строится на основе следующей таблицы, которая дает реальные данные для 18 проектов НАСА, на основе которых мы ищем зависимость между  $L$  и  $Ef$ .

Таблица 8.1

## Экспериментальные данные проектов НАСА

Номер проекта	$L$	$Me$	$E_f$	$E_{fm}$	$E_{fm2}$
1	90,2000	30,0000	115,8000	124,8585	134,0202
2	46,2000	20,0000	96,0000	74,8467	84,1616
3	46,5000	19,0000	79,0000	75,4852	85,0112
4	54,5000	20,0000	909,8000	85,4349	94,9828
5	31,1000	35,0000	39,6000	50,5815	56,6580
6	67,5000	29,000	98,4000	99,0504	107,2609
7	12,8000	26,000	18,9000	24,1480	32,6461
8	10,5000	34,0000	10,3000	18,0105	25,0755
9	21,5000	31,0000	28,5000	37,2724	44,3086
10	3,1000	26,000	7,0000	4,5849	14,4563
11	4,2000	19,0000	9,0000	8,9384	19,9759
12	7,8000	31,0000	7,3000	13,5926	21,5763
13	2,1000	28,0000	5,0000	1,5100	11,2703
14	5,0000	29,0000	8,4000	8,2544	17,0887
15	78,6000	35,0000	98,7000	110,5249	118,0378
16	9,7000	27,0000	15,6000	18,2559	26,8312
17	12,5000	27,0000	23,9000	23,3690	31,6864
18	100,8000	34,0000	138,3000	135,4825	144,4587

При этом данные по 13 проектам (из 18) можно использовать в качестве обучающего множества, а данные остальных 5 проектов – в качестве тестового множества (для проверки – тестирования) полученных значений коэффициентов  $a$  и  $b$ . Иногда для обучения используется расширенное множество проектов НАСА, представленное в табл. 8.2.

Таблица 8.2

## Расширенное множество проектов НАСА

№	$L$ – в кило- строках	$E_f$ – реальная стоимость в человеко- месяцах	COCOMO с ошибкой MRE	ИНС с MRE	ГА с MRE	Гибрид с MRE
1	2.2	8.4	24.15	13.65	8.95	6.32
2	3.5	10.8	3.95	5.26	4.69	1.13
3	5.5	18	7.36	5.21	6.75	4.35
4	6	24	58.88	34.10	27.63	28.02
5	9.7	25.2	20.05	11.50	13.49	7.61



Продолжение табл. 8.2

№	$L$ – в кило- строках	$Ef$ – реальная стоимость в человеко- месяцах	СОСОМО с ошибкой MRE	ИНС с MRE	ГА с MRE	Гибрид с MRE
6	7.7	31.2	23.91	12.35	7.54	12.42
7	11/3	36	30.83	17.45	12.45	13.35
8	8.2	36	29.55	16.68	14.23	11.21
9	6.5	42	28.32	18.52	11.64	13.42
10	8	42	22.22	13.21	15.47	9.34
11	20	48	27.21	14.65	16.32	12.16
12	10	48	41.66	23.98	19.84	19.84
13	15	48	46.19	28.04	23.11	26.74
14	10.4	50	34.90	25.47	17.02	21.95
15	13	60	9.36	6.53	5.31	7.15
16	14	60	25.88	15.41	17.54	8.46
17	19.7	60	6,10	7.21	4.21	2.54
18	32.5	60	93.91	47.35	56.47	36.10
19	31.5	60	3.81	6.52	5.46	1.07
20	12.5	62	27.96	13.11	10.84	4.31
21	15.4	70	22.51	10.13	12.76	7.02
22	20	72	60.76	45.68	33.82	27.11
23	7.5	72	41.75	32.61	24.15	15.04
24	16.3	82	29.79	23.40	17.37	7.46
25	15	90	39.54	27.68	21.51	19.01
26	11.4	98.8	42.04	25.10	19.07	21.74
27	21	107	36.75	24.55	16.53	9.02
28	16	114	34.48	24.55	16.53	9.92
29	25.9	117.6	27.85	19.36	11.57	17.09
30	24.6	117.6	31.65	21.87	16.34	14.82
31	29.5	120	18.94	11.15	7.13	6.44
32	19.3	155	35.78	17.30	21.06	16.72
33	32.6	170	29.88	19.54	15.19	5.68
34	35.5	192	32.10	16.35	8.37	13.06
35	38	210	28.46	13.19	19.50	15.43
36	48.5	239	24.31	8.43	12.07	7.94
37	47.5	252	37.81	21.36	18.64	11.83
38	70	278	21.28	9.42	11.46	6.24
39	66.6	300	23.76	11.30	16.79	9.22

№	$L$ – в кило- строках	$Ef$ – реальная стоимость в человеко- месяцах	СОСОМО с ошибкой MRE	ИНС с MRE	ГА с MRE	Гибрид с MRE
40	66.6	352.8	35.17	19.25	11.20	13.62
41	50	370	36.90	23.54	13.48	7.42
42	79	400	45.74	31.29	22.97	18.06
43	90	450	38.29	20.11	31.73	15.94
44	78	571.4	24.50	13.64	8.03	5.21
45	100	215	120.66	86.14	61.42	51.04
46	150	324	49.50	26.80	13.09	23.83
47	100	360	44.97	17.67	25.07	12.62
48	100	360	15.85	6.23	8.62	9.84
49	190	420	1.89	4.87	3.84	2.65
50	115.8	480	11.37	16.49	5.32	5.42
51	101	750	19.87	10.67	6.46	12.71
52	161.1	815	4.76	10.25	8.41	5.95
53	284.7	973	38.36	21.43	17.09	10.14
54	227	1181	3.93	2.36	6.31	4.62
55	177.9	1228	3.64	9.84	5.08	2.06
56	282.1	1368	17.21	9.46	11.36	7.92
57	219	2120	29.00	21.03	15.81	8.31
58	423	2300	25.78	16.07	7.44	9.02
59	302	2400	0.46	3.24	5.64	2.54
50	370	3240	25.21	8.62	3.21	6.87

Напомним, что для того, чтобы применить генетический алгоритм для решения некоторой проблемы необходимо, прежде всего, определить:

1. Кодирование (представление потенциального решения);
2. Для определенного кодирования выбрать или разработать генетические операторы кроссовера, мутации и репродукции.
3. Фитнесс-функцию из условия задачи.
4. Определить параметры ГА: число особей в популяции, значения вероятностей кроссовера  $P_c$  и  $P_m$ .

В данном случае потенциальное решение представляется вектором значений параметров  $(a, b)$ . Значения каждого параметра лежат в некотором диапазоне  $a \in [L_a, U_a]$ ,  $b \in [L_b, U_b]$ .

Для кодирования значений вектора  $(a,b)$  можно использовать как двоичное кодирование, так и непосредственное представление потенциального решения в виде вектора вещественных чисел  $(a,b)$ . Кодирование решения определяется вариантом курсовой работы согласно приведенной далее таблице. В случае двоичного кодирования можно использовать стандартный 1-точечный, 2-точечный (или однородный) кроссовер и стандартный оператор мутации. В случае вещественного кодирования следует использовать какой-либо вещественный кроссовер (например, в виде линейной комбинации родительских векторов) и вещественную мутацию. Значения параметров ГА следует подобрать экспериментально в ходе эксперимента.

### Фитнесс-функция

В качестве фитнес-функции в данном случае следует взять различие между реальными значениями стоимостей проектов и модельными значениями (оценками) стоимостей этих же проектов, которые вычислены согласно приведенной формуле с найденными с помощью ГА коэффициентами  $a$  и  $b$ . Это различие (расстояние между оценками) можно оценить по-разному – в различной метрике. Можно взять, например, метрику абсолютных значений (Манхэттен – метрика городских кварталов), где это различие определяется с помощью следующей формулы

$$MD = \sum_{i=1}^n |Ef_i - Efm_i|.$$

Здесь  $Ef_i$  – реальная (измеренная) стоимость  $i$ -го проекта в человеко-месяцах и  $Efm_i$  – модельная оценка того же проекта, вычисленная с помощью приведенной формулы с найденными путем применения ГА коэффициентами  $a$  и  $b$ . Для оценки различия можно использовать и другие метрики [18], например:

– среднее значение относительной погрешности (MMRE)

$$MMRE = \frac{1}{n} \sum_{i=1}^n \frac{|Ef_i - Efm_i|}{Ef_i},$$

– корень квадратный среднеквадратичной ошибки (RMS)

$$RMS = \sqrt{\frac{1}{n} \sum_{i=1}^n |Ef_i - Efm_i|},$$

– отклонение (дисперсия) (VAF)  $VAF = \left[ 1 - \frac{\text{var}(Ef - Efm)}{\text{var}(Ef)} \right] \times 100\%$ ,

– Евклидово расстояние  $ED = \sqrt{\frac{1}{n} \sum_{i=1}^n (Ef_i - Efm_i)^2}$ .

## 8.2. Практическая часть

### Тестовые примеры

Для данного вида задачи существует большое число тестовых примеров – Benchmark-ов. Например, можно использовать сокращенное (табл.?) или расширенное множество проектов НАСА(табл

### Порядок выполнения лабораторной работы

1. Разобраться в теоретическом описании математического метода оценки стоимости программного проекта – модели СОСОМО.

2. Из приведенной выше табл. 8.1 (или табл. 8.2) экспериментальных данных (программных проектов НАСА) отобрать из 18 проектов в качестве обучающего множества 13 (40) проектов.

3. В соответствии с вариантом лабораторной работы, заданного табл. 8.3 определить тип используемого эволюционного алгоритма (генетический или роевой алгоритм, генетическое программирование), кодирование потенциального решения, вид ошибки в целевой функции, вид генетических операторов кроссовера, мутации и репродукции

4. Отработать алгоритм решения задачи с помощью заданного метода на обучающем множестве.

5. Разработать программу на языке MATLAB или другом доступном вам языке программирования, включающую в себя реализацию пользовательского интерфейса в виде диалогового меню, реализацию алгоритма решения поставленной задачи заданным методом.

6. Протестировать разработанную программу: вычислить заданный тип ошибки на тестовом множестве – оставшихся 5 (из 18) проектов табл. 8.1 (или табл. 8.2).

5. Выполнить вывод полученного решения в виде текста и графиков.

В отчете должны быть представлены следующие пункты:

1. Титульный лист
2. Постановка задачи.
3. Теоретические сведения.
4. Входные данные – таблица.

Таблица 8.3

## Выбор варианта лабораторной работы

№ варианта	Тип эволюционного алгоритма	Кодирование решения	Фитнесс-функция	Оператор кроссовера	Оператор мутации	Оператор репродукции
1	ГА	Двоичн. стр.	MD	1-точечный	Классическая	рулетка
2	ГА	Веществ. вектор	MMRE	арифметич.	Арифметич.	ранговый
3	ГА	Двоичн. стр.	RMS	2-точечный	Классическая	Турнир
4	ГА	Веществ. вектор	ED	арифметич.	арифметич.	рулетка
5	ГП	Дерево	MD	поддержив	Усекающ. Растущ.	ранговый
6	ГА	Дерево	MMRE	Поддержив	Усекающ. Растущ.	Турнир
7	ГП	Дерево	RMS	поддержив	Усекающ. Растущ.	ранговый
8	РА	Веществ. вектор	ED	–	–	–
9	РА	Веществ. вектор	MD	–	–	–
10	РА	Веществ. вектор	MMRE	–	–	–

5. Разработанный генетический алгоритм в виде псевдокода или блок-схемы.

6. Используемое кодирование потенциального решения в ГА (двоичное или вещественное).

7. Используемые генетические операторы: отбора родителей, кроссовера и мутации.

8. Метод отбора обучающего и тестового множества из входных данных (лучше случайный).

9. Используемая фитнес-функция (тип ошибки).

10. Используемые значения параметров (мощность популяции, вероятность кроссовера и мутации).

11. График обучения (падение значения ошибки в процессе обучения по итерациям-эпохам).

12. Диаграмма значений ошибки на обучающем и тестовом множестве.

13. Найденные значения коэффициентов для модели СОСОМО и формула в целом.
14. Выводы по полученным результатам.
15. Список использованных источников (литература).
16. Листинг программы (в приложении).

### **Контрольные вопросы**

1. Приведите основную формулу методики СОСОМО.
2. Какие данные необходимы для использования методики СОСОМО?
3. Что позволяет оценивать методика СОСОМО?
4. Какие данные можно использовать при обучении (поиске коэффициентов  $a$ ,  $b$ )?
5. Как можно разбить данные на обучающее и тестовое множество?
6. Какое кодирование потенциальных решений можно использовать в этой задаче?
7. Какие типы генетических операторов можно использовать в этой задаче?
8. Какую фитнес-функцию можно использовать в этой задаче?
9. Какие виды ошибок можно использовать в этой задаче?
10. Приведите основную формулу метрики абсолютных значений.
11. Приведите основную формулу метрики среднего значения относительной погрешности.
12. Приведите основную формулу отклонения.
13. Как можно улучшить методику СОСОМО?
14. Как можно применить роевой алгоритм для поиска коэффициентов  $a$ ,  $b$ ?
15. Как можно применить эволюционную стратегию для поиска коэффициентов  $a$ ,  $b$ ?

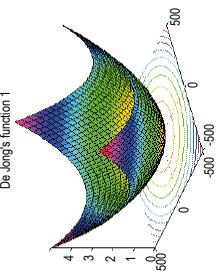
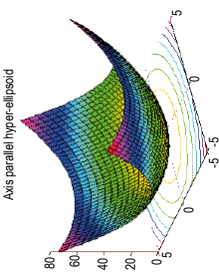
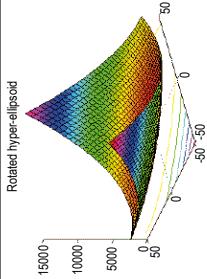
## ЛИТЕРАТУРА

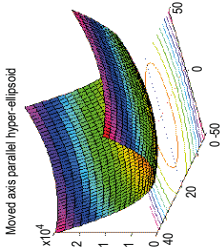
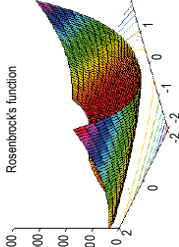
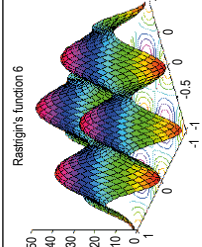
1. Скобцов Ю. А., Сперанский Д. В. Эволюционные вычисления: учеб. пособие. М.: Национальный открытый университет «ИНТУ-ИТ», 2015. 2-е изд. М.: ИНТУИТ, 2016. 429 с. Текст: электронный // Лань: электронно-библиотечная система. e.lanbook.com>book/100264
2. Скобцов Ю. А. Основы эволюционных вычислений: учеб. пособие. Донецк: ДонНТУ, 2008. 326 с.
3. Скобцов Ю. А., Федоров Е. Е. Метаэвристики: монография. Донецк: Изд-во «Ноулидж», 2013. 428 с.
4. Скобцов Ю. А. Междисциплинарный курс «Вычислительный интеллект» / Сб. тр. Междунар. конф. «Математические методы в технике и технологиях». Изд-во СПбПУ. Т. 7–1. 2020. С. 66–69.
5. Тестовые наборы для задачи комивояжера. URL: <http://www.tsp.gatech.edu/world/countries.html>
6. Тестовые наборы для задачи комивояжера. URL: <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/>
7. Функции для многопараметрической оптимизации генетическими алгоритмами // Hartmut Pohlheim. 1996–2004. URL: [http://www.geatbx.com/docu/fcnindex-01.html#P89\\_3085](http://www.geatbx.com/docu/fcnindex-01.html#P89_3085).
8. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, second edition, Berlin, 1994.
9. Курейчик В. В., Курейчик В. М., Родзин С. И. Теория эволюционных вычислений. М.: ФИЗМАТЛИТ, 2012, 260 с.
10. Гладков Л. А., Курейчик В. В., Курейчик В. М., Сороколетов П. В. Биоинспирированные методы в оптимизации. М.: ФИЗМАТЛИТ, 2009, 384 с.
11. Карпенко А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учеб. пособие. М.: Изд-во МГТУ им. Н. Э. Баумана, 2014. 446 с.
12. Семенкин Е. С. Эволюционные методы моделирования и оптимизации сложных систем // Электронные публикации НГУ. URL: [http://library.krasu.ru/ft/\\_umkd/22/u\\_lectures.pdf](http://library.krasu.ru/ft/_umkd/22/u_lectures.pdf).
13. Родзин С. И., Скобцов Ю. А., Эль-Хатиб С. А. Биоэвристики: теория, алгоритмы и приложения: монография. Чебоксары: ИД «Среда», 2019. 224 с.
14. Скобцов Ю. А. Эволюционные методы в программной инженерии: учеб. пособие. СПб.: ГУАП. 2020. 128 с.
15. Sumathi T. Hamsapriya P. Surekha. Evolutionary Intelligence. An Introduction to Theory and Applications with Matlab. 2008 Springer-Verlag Berlin Heidelberg. 599 p.

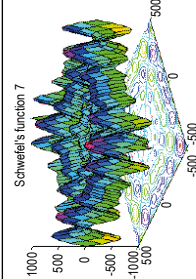
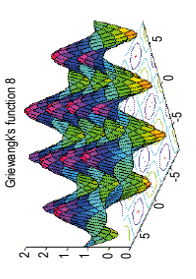
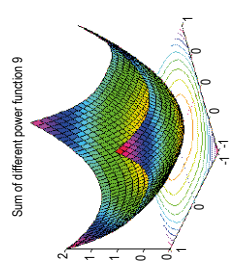
16. *Haupt R.L., Haupt S.L.* Practical genetic algorithms. 2nd ed. A John Wiley & Sons, Inc., Publication. 2004. 261 p.
17. *Sivanandam S.N., Deepa S.N.* Introduction to Genetic Algorithms. Springer-Verlag Berlin Heidelberg 2008. 453 p.
18. *Соммервилл Иан.* Инженерия программного обеспечения, 6-е изд. / Пер. с англ. М.: Изд. дом «Вильямс», 2002. 624 с.
19. *Вурсански Э.* Генетические алгоритмы на Python / Пер. с англ. А. А. Слинкина. М.: ДМК Пресс, 2020. 286 с.
20. *Yuriy Skobtsov.* Prospects of the Interdisciplinary Course “Computational Intelligence” in Engineering Education// Studies in Systems, Decision and Control, Volume 342, Cyber-Physical Systems: Design and Application for Industry 4.0. P. 431–442.



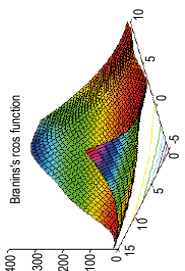
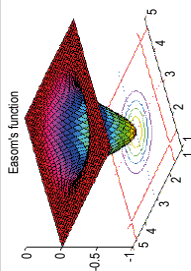
Индивидуальные задания на лабораторную работу №2

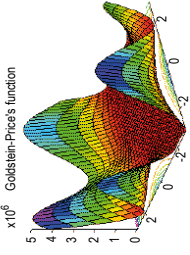
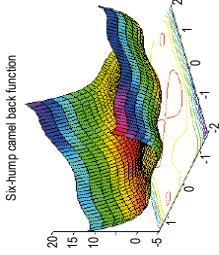
№ вв.	Название	Оптимум	Вид функции	График функции
1	De Jong's function 1	global minimum $f(x)=0; x(i)=0,$ $i=1:n.$	$f_1(x) = \sum_{i=1}^n x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f1(x)=\text{sum}(x(i)^2),$ $i=1:n;$	
2	Axis parallel hyper-ellipsoid function	global minimum $f(x)=0; x(i)=0,$ $i=1:n.$	$f_{1a}(x) = \sum_{i=1}^n i \cdot x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f1a(x)=\text{sum}(i \cdot x(i)^2),$ $i=1:n;$	
3	Rotated hyper-ellipsoid function	global minimum $f(x)=0; x(i)=0,$ $i=1:n$	$f_{1b}(x) = \sum_{i=1}^n \left( \sum_{j=1}^i x_j \right)^2$ $-65.536 \leq x_i \leq 65.536$ $f1b(x)=\text{sum}(\text{sum}(x(j)^2),$ $j=1:i), i=1:n;$	

№ вв.	Название	Оптимум	Вид функции	График функции
4	Moved axis parallel hyper- ellipsoid function	global minimum $f(x)=0$ ; $x(i)=5 \cdot i$ , $i=1:n$	$f_{1c}(x) = \sum_{i=1}^n 5i \cdot x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f1c(x) = \text{sum}(5 \cdot i \cdot x(i)^2),$ $i=1:n;$	 <p>Moved axis parallel hyper-ellipsoid</p>
5	Rosenbrock's valley (De Jong's function 2)	global minimum $f(x)=0$ ; $x(i)=1$ , $i=1:n$ .	$f_2(x) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2$ $-2.048 \leq x_i \leq 2.048$ $f2(x) = \text{sum}(100 \cdot (x(i+1) - x(i)^2)^2 + (1 - x(i))^2),$ $i=1:n-1;$	 <p>Rosenbrock's function</p>
6	Rastrigin's function 6	global minimum $f(x)=0$ ; $x(i)=0$ , $i=1:n$ .	$f_6(x) = 10 \cdot n + \sum_{i=1}^{n-1} (x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i))$ $-5.12 \leq x_i \leq 5.12$ $f6(x) = 10 \cdot n + \text{sum}(x(i)^2 - 10 \cdot \cos(2 \cdot \pi \cdot x(i))),$ $i=1:n;$	 <p>Rastrigin's function 6</p>

№ вв.	Название	Оптимум	Вид функции	График функции
7	Schwefel's function 7	global minimum f(x)=n·418.9829; x(i)=420.9687, i=1:n.	$f_7(x) = \sum_{i=1}^n -x_i \cdot \sin\left(\sqrt{ x_i }\right)$ $-500 \leq x_i \leq 500$ $f_7(x) = \text{sum}(-x(i) \cdot \sin(\sqrt{\text{abs}(x(i))})),$ $i=1:n;$	 <p>Schwefel's function 7</p>
8	Griewangk's function 8	global minimum f(x)=0; x(i)=0, i=1:n	$f_8(x) = \sum_{i=1}^n \frac{x_i^2}{4000} - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right)$ $-600 \leq x_i \leq 600$ $f_8(x) = \text{sum}(x(i)^2 / 4000) - \text{prod}(\cos(x(i) / \sqrt{i})),$ $i=1:n;$	 <p>Griewangk's function 8</p>
9	Sum of different power function 9	global minimum f(x)=0; x(i)=0, i=1:n.	$f_9(x) = \sum_{i=1}^n  x_i ^{(i+1)}$ $-1 \leq x_i \leq 1$ $f_9(x) = \text{sum}(\text{abs}(x(i))^{(i+1)}),$ $i=1:n;$	 <p>Sum of different power function 9</p>

№ вв.	Название	Оптимум	Вид функции	График функции
10	Ackley's Path function 10	global minimum $f(x)=0$ ; $x(i)=0$ , $i=1:n$ .	$f_{10}(x) = -a \cdot e^{-b \sqrt{\frac{1}{n} \sum_{i=1}^n \frac{\cos(c \cdot x_i)}{i+1}}} - a + e^1$ $-1 \leq x_i \leq 1$ $f_{10}(x) = -a \cdot \exp(-b \cdot \sqrt{1/n \cdot \sum (x(i)^2)}) - \exp(1/n \cdot \sum (\cos(c \cdot x(i)))) + a + \exp(1);$ $a=20; b=0.2; c=2 \cdot \pi; i=1:n;$	Ackley's Path function 10 
11	Langermann's function 11	global minimum $f(x)=-1.4$ (for $m=5$ ); $x(i)=???$ , $i=1:n$ .	$f_{11}(x) = -\sum_{i=1}^m c_i \left( e^{-\frac{\ x-A(i)\ ^2}{\pi}} \cdot \cos(\pi \cdot \ \bar{x} - A(i)\ ^2) \right)$ $i=1:m, 2 \leq m \leq 10, 0 \leq x_i \leq 10$ $f_{11}(x) = \sum (c(i) \cdot (\exp(-1/\pi \cdot \sum ((x-A(i))^2))) \cdot \cos(\pi \cdot \sum ((x-A(i))^2))),$ $i=1:m, m=5; A(i), C(i) > 0, m=5$	Langermann's function 11 
12	Michalewicz's function 12	global minimum $f(x)=-4.687$ ( $n=5$ ); $x(i)=???$ , $i=1:n$ . $f(x)=-9.66$ ( $n=10$ ); $x(i)=???$ , $i=1:n$ .	$f_{12}(x) = -\sum_{i=1}^n \sin(x_i) \cdot \sin\left(\frac{i \cdot x_i^2}{\pi}\right)^{2m}$ $i=1:n, m=10, 0 \leq x_i \leq \pi$ $f_{12}(x) = -\sum (\sin(x(i)) \cdot (\sin(i \cdot x(i)^2/\pi))^{2m}),$ $i=1:n, m=10; \text{проверить для } n=5, 10$	Michalewicz's function 12 

№ вв.	Название	Оптимум	Вид функции	График функции
13	Branins's roos function	global minimum f(x1,x2)= 0.397887; (x1,x2)= (-pi,12.275), (pi,2.275), (9.42478,2.475).	$f_{Bran}(x_1, x_2) = a \cdot (x_2 - b \cdot x_1^2 + c \cdot x_1 - d)^2 + e \cdot (1 - f) \cdot \cos(x_1) + e$ $a = 1, b = \frac{5.1}{4 \cdot \pi^2}, c = \frac{5}{\pi}, d = 6, e = 10, f = \frac{1}{8 \cdot \pi}$ $-5 \leq x_1 \leq 10, 0 \leq x_2 \leq 15$ $f_{Bran}(x_1, x_2) = a \cdot (x_2 - b \cdot x_1^2 + c \cdot x_1 - d)^2 + e \cdot (1 - f) \cdot \cos(x_1) + e;$ $a = 1, b = 5.1 / (4 \cdot \pi^2), c = 5 / \pi, d = 6, e = 10, f = 1 / (8 \cdot \pi);$	
14	Easom's function	global minimum f(x1,x2)=-1; (x1,x2)=(pi,pi).	$f_{Easo}(x_1, x_2) = -\cos(x_1) \cdot \cos(x_2) \cdot e^{-((x_1 - \pi)^2 + (x_2 - \pi)^2)}$ $-100 \leq x_i \leq 100, i = 1:2$ $f_{Easo}(x_1, x_2) = -\cos(x_1) \cdot \cos(x_2) \cdot \exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2));$	

№ вв.	Название	Оптимум	Вид функции	График функции
15	Goldstein-Price's function	global minimum $f(x_1, x_2)=3$ ; $(x_1, x_2)=(0, -1)$ .	$f_{Gold}(x_1, x_2) = (1 + (x_1 + x_2 + 1)^2) \cdot$ $(19 - 14 \cdot x_1 + 3x_1^2 - 14x_2 + 6 \cdot x_1x_2 + 3x_2^2) \cdot$ $(30 + (2x_1 - 3x_2)^2 \cdot (18 - 32x_1 + 12x_1^2 +$ $+ 48x_2 - 36x_1x_2 + 27 \cdot x_2^2))$ $-2 \leq x_i \leq 2, i = 1:2$ $f_{Gold}(x_1, x_2) = [1 + (x_1 + x_2 + 1)^2 \cdot (19 -$ $14x_1 + 3x_1^2 -$ $14x_2 + 6x_1x_2 + 3x_2^2)] \cdot [30 + (2x_1 - 3x_2)^2 \cdot (18 -$ $32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	
16	Six-hump camel back function	global minimum $f(x_1, x_2) = -1.0316$ ; $(x_1, x_2) = (-$ $0.0898, 0.7126)$ , $(0.0898, -0.7126)$ .	$f_{Sixh}(x_1, x_2) = (4 - 2 \cdot 1 \cdot x_1^2 + x_1^{4/3}) \cdot$ $x_1^2 + x_1 \cdot x_2 + (-4 + 4 \cdot x_2^2) \cdot x_2^2$ $-3 \leq x_i \leq 3, -2 \leq x_1 \leq 2$ $f_{Sixh}(x_1, x_2) = (4 - 2 \cdot 1 \cdot x_1^2 + x_1^{4/3}) \cdot$ $x_1^2 + x_1 \cdot x_2 + (-4 + 4 \cdot x_2^2) \cdot x_2^2;$	

Для остальных вариантов берется строчка, соответствующая остатку от деления номера варианта на 16 (для 17-1, 18-2 и т.д.)

### Тестовые наборы к лабораторной работе №3

В приложении представлены наборы в трех формах:

1. Эвклидовы координаты городов. Матрица расстояний получается путем нахождения эвклидовых расстояний между координатами города по формуле:  $Dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ . В случае эвклидовых координат городов они представлены в формате: №\_города, координата x, координата y (через пробел).

2. Полная матрица расстояний. Не обрабатывается, переписывается без изменений из файла.

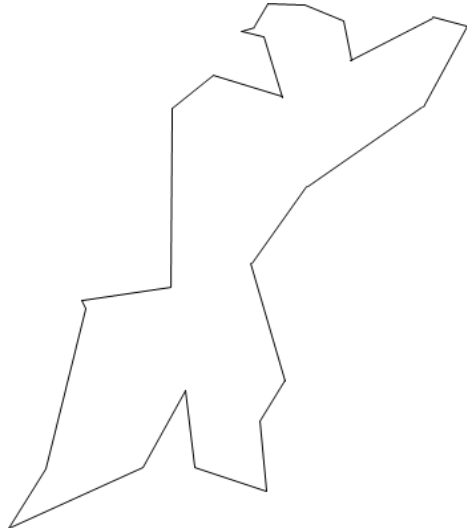
Диагональная матрица расстояний. Данную матрицу необходимо транспонировать, после чего заполнить верхнюю половину матрицы расстояний (от главной диагонали). Нижняя половина заполняется из верхней, с соблюдением условия  $Dist_{ij} = Dist_{ji}$ . wi29: 29 городов в Западной Сахаре [4].

Тип данных: эвклидовы  
координаты городов

Оптимальное решение wi29:

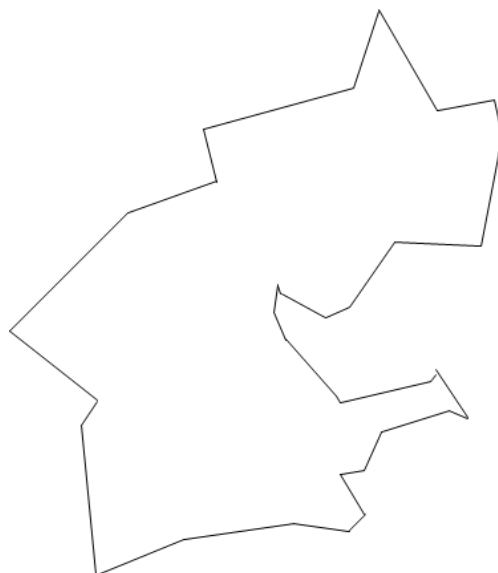
1	20833.3333	17100.0000
2	20900.0000	17066.6667
3	21300.0000	13016.6667
4	21600.0000	14150.0000
5	21600.0000	14966.6667
6	21600.0000	16500.0000
7	22183.3333	13133.3333
8	22583.3333	14300.0000
9	22683.3333	12716.6667
10	23616.6667	15866.6667
11	23700.0000	15933.3333
12	23883.3333	14533.3333
13	24166.6667	13250.0000
14	25149.1667	12365.8333
15	26133.3333	14500.0000
16	26150.0000	10550.0000
17	26283.3333	12766.6667
18	26433.3333	13433.3333
19	26550.0000	13850.0000
20	26733.3333	11683.3333
21	27026.1111	13051.9444
22	27096.1111	13415.8333
23	27153.6111	13203.3333
24	27166.6667	9833.3333
25	27233.3333	10450.0000
26	27233.3333	11783.3333
27	27266.6667	10383.3333
28	27433.3333	12400.0000
29	27462.5000	12992.2222

EOF



**dj89: 89 городов в Джибути [4]****Оптимальный тур dj89:****Тип данных: эвклидовы****координаты городов**

1	11511.3889	42106.3889
2	11503.0556	42855.2778
3	11438.3333	42057.2222
4	11438.3333	42057.2222
5	11438.3333	42057.2222
6	11785.2778	42884.4444
7	11785.2778	42884.4444
8	11785.2778	42884.4444
9	11785.2778	42884.4444
10	12363.3333	43189.1667
11	11846.9444	42660.5556
12	11503.0556	42855.2778
13	11963.0556	43290.5556
14	11963.0556	43290.5556
15	12300.0000	42433.3333
16	11973.0556	43026.1111
17	11973.0556	43026.1111
18	11461.1111	43252.7778
19	11461.1111	43252.7778
20	11461.1111	43252.7778
21	11461.1111	43252.7778
22	11600.0000	43150.0000
23	12386.6667	43334.7222
24	12386.6667	43334.7222
25	11595.0000	43148.0556
26	11595.0000	43148.0556
27	11569.4444	43136.6667
28	11310.2778	42929.4444
29	11310.2778	42929.4444
30	11310.2778	42929.4444
31	11963.0556	43290.5556
32	11416.6667	42983.3333
33	11416.6667	42983.3333
34	11595.0000	43148.0556
35	12149.4444	42477.5000
36	11595.0000	43148.0556
37	11595.0000	43148.0556
38	11108.6111	42373.8889
39	11108.6111	42373.8889
40	11108.6111	42373.8889
41	11108.6111	42373.8889
42	11183.3333	42933.3333
43	12372.7778	42711.3889
44	11583.3333	43150.0000
45	11583.3333	43150.0000
46	11583.3333	43150.0000
47	11583.3333	43150.0000
48	11583.3333	43150.0000
49	11822.7778	42673.6111
50	11822.7778	42673.6111
51	12058.3333	42195.5556
52	11003.6111	42102.5000





53 11003.6111 42102.5000  
54 11003.6111 42102.5000  
55 11522.2222 42841.9444  
56 12386.6667 43334.7222  
57 12386.6667 43334.7222  
58 12386.6667 43334.7222  
59 11569.4444 43136.6667  
60 11569.4444 43136.6667  
61 11569.4444 43136.6667  
62 11155.8333 42712.5000  
63 11155.8333 42712.5000  
64 11155.8333 42712.5000  
65 11155.8333 42712.5000  
66 11133.3333 42885.8333  
67 11133.3333 42885.8333  
68 11133.3333 42885.8333  
69 11133.3333 42885.8333  
70 11133.3333 42885.8333  
71 11003.6111 42102.5000  
72 11770.2778 42651.9444  
73 11133.3333 42885.8333  
74 11690.5556 42686.6667  
75 11690.5556 42686.6667  
76 11751.1111 42814.4444  
77 12645.0000 42973.3333  
78 12421.6667 42895.5556  
79 12421.6667 42895.5556  
80 11485.5556 43187.2222  
81 11423.8889 43000.2778  
82 11423.8889 43000.2778  
83 11715.8333 41836.1111  
84 11297.5000 42853.3333  
85 11297.5000 42853.3333  
86 11583.3333 43150.0000  
87 11569.4444 43136.6667  
88 12286.9444 43355.5556  
89 12355.8333 43156.3889  
EOF

**att48: 48 городских центров США  
(Padberg/Rinaldi) [6]**

Тип данных: координаты городов	Оптимальное решение att48
1 6734 1453	1
2 2233 10	8
3 5530 1424	38
4 401 841	31
5 3082 1644	44
6 7608 4458	18
7 7573 3716	7
8 7265 1268	28
9 6898 1885	6
10 1112 2049	37
11 5468 2606	19
12 5989 2873	27
13 4706 2674	17
14 4612 2035	43
15 6347 2683	30
16 6107 669	36
17 7611 5184	46
18 7462 3590	33
19 7732 4723	20
20 5900 3561	47
21 4483 3369	21
22 6101 1110	32
23 5199 2182	39
24 1633 2809	48
25 4307 2322	5
26 675 1006	42
27 7555 4819	24
28 7541 3981	10
29 3177 756	45
30 7352 4506	35
31 7545 2801	4
32 3245 3305	26
33 6426 3173	2
34 4608 1198	29
35 23 2216	34
36 7248 3779	41
37 7762 4595	16
38 7392 2244	22
39 3484 2829	3
40 6271 2135	23
41 4985 140	14
42 1916 1569	25
43 7280 4899	13
44 7509 3239	11
45 10 2676	12
46 6807 2993	15
47 5185 3258	40
48 3023 1942	9
EOF	-1
	EOF

**bayg29: 29 городов в Баварии, географические расстояния (Groetschel, Juenger, Reinelt) [6]**

**Тип данных – транспонированная диагональная матрица**

```
97 205 139 86 60 220 65 111 115 227 95 82 225 168 103 266 205
149 120 58 257 152 52 180 136 82 34 145
129 103 71 105 258 154 112 65 204 150 87 176 137 142 204 148 148
49 41 211 226 116 197 89 153 124 74
219 125 175 386 269 134 184 313 201 215 267 248 271 274 236 272
160 151 300 350 239 322 78 276 220 60
167 182 180 162 208 39 102 227 60 86 34 96 129 69 58 60 120 119
192 114 110 192 136 173 173
51 296 150 42 131 268 88 131 245 201 175 275 218 202 119 50 281
238 131 244 51 166 95 69
279 114 56 150 278 46 133 266 214 162 302 242 203 146 67 300 205
111 238 98 139 52 120
178 328 206 147 308 172 203 165 121 251 216 122 231 249 209 111
169 72 338 144 237 331
169 151 227 133 104 242 182 84 290 230 146 165 121 270 91 48 158
200 39 64 210
172 309 68 169 286 242 208 315 259 240 160 90 322 260 160 281
57 192 107 90
140 195 51 117 72 104 153 93 88 25 85 152 200 104 139 154 134
149 135
320 146 64 68 143 106 88 81 159 219 63 216 187 88 293 191 258 272
174 311 258 196 347 288 243 192 113 345 222 144 274 124 165 71 153
144 86 57 189 128 71 71 82 176 150 56 114 168 83 115 160
61 165 51 32 105 127 201 36 254 196 136 260 212 258 234
106 110 56 49 91 153 91 197 136 94 225 151 201 205
215 159 64 126 128 190 98 53 78 218 48 127 214
61 155 157 235 47 305 243 186 282 261 300 252
105 100 176 66 253 183 146 231 203 239 204
113 152 127 150 106 52 235 112 179 221
79 163 220 119 164 135 152 153 114
236 201 90 195 90 127 84 91
273 226 148 296 238 291 269
112 130 286 74 155 291
130 178 38 75 180
281 120 205 270
213 145 36
94 217
162
```

**bayg29: координаты городов:**

```
1 1150.0 1760.0
2 630.0 1660.0
3 40.0 2090.0
4 750.0 1100.0
5 750.0 2030.0
6 1030.0 2070.0
7 1650.0 650.0
8 1490.0 1630.0
9 790.0 2260.0
10 710.0 1310.0
11 840.0 550.0
12 1170.0 2300.0
13 970.0 1340.0
14 510.0 700.0
15 750.0 900.0
16 1280.0 1200.0
17 230.0 590.0
18 460.0 860.0
19 1040.0 950.0
20 590.0 1390.0
21 830.0 1770.0
22 490.0 500.0
23 1840.0 1240.0
24 1260.0 1500.0
25 1280.0 790.0
26 490.0 2130.0
27 1460.0 1420.0
28 1260.0 1910.0
29 360.0 1980.0
EOF
```

**bayg29:лучшие решения**

```
1
28
6
12
9
26
3
29
5
21
2
20
10
4
15
18
14
17
22
11
19
25
7
23
8
27
16
13
24
-1
EOF
```

**bays29: 29 городов в Баварии, расстояния по дорогам (Groetschel, Juenger, Reinelt) [6]**

**Тип данных: полная матрица**

```
0 107 241 190 124 80 316 76 152 157 283 133 113 297 228 129 348
276 188 150 65 341 184 67 221 169 108 45 167
107 0 148 137 88 127 336 183 134 95 254 180 101 234 175 176 265
199 182 67 42 278 271 146 251 105 191 139 79
241 148 0 374 171 259 509 317 217 232 491 312 280 391 412 349 422
356 355 204 182 435 417 292 424 116 337 273 77
190 137 374 0 202 234 222 192 248 42 117 287 79 107 38 121 152
86 68 70 137 151 239 135 137 242 165 228 205
124 88 171 202 0 61 392 202 46 160 319 112 163 322 240 232 314
287 238 155 65 366 300 175 307 57 220 121 97
80 127 259 234 61 0 386 141 72 167 351 55 157 331 272 226 362
296 232 164 85 375 249 147 301 118 188 60 185
316 336 509 222 392 386 0 233 438 254 202 439 235 254 210 187
313 266 154 282 321 298 168 249 95 437 190 314 435
76 183 317 192 202 141 233 0 213 188 272 193 131 302 233 98 344
289 177 216 141 346 108 57 190 245 43 81 243
152 134 217 248 46 72 438 213 0 206 365 89 209 368 286 278 360
333 284 201 111 412 321 221 353 72 266 132 111
157 95 232 42 160 167 254 188 206 0 159 220 57 149 80 132 193
127 100 28 95 193 241 131 169 200 161 189 163
283 254 491 117 319 351 202 272 365 159 0 404 176 106 79 161 165
141 95 187 254 103 279 215 117 359 216 308 322
133 180 312 287 112 55 439 193 89 220 404 0 210 384 325 279 415
349 285 217 138 428 310 200 354 169 241 112 238
113 101 280 79 163 157 235 131 209 57 176 210 0 186 117 75 231
165 81 85 92 230 184 74 150 208 104 158 206
297 234 391 107 322 331 254 302 368 149 106 384 186 0 69 191 59
35 125 167 255 44 309 245 169 327 246 335 288
228 175 412 38 240 272 210 233 286 80 79 325 117 69 0 122 122
56 56 108 175 113 240 176 125 280 177 266 243
129 176 349 121 232 226 187 98 278 132 161 279 75 191 122 0 244
178 66 160 161 235 118 62 92 277 55 155 275
348 265 422 152 314 362 313 344 360 193 165 415 231 59 122 244
0 66 178 198 286 77 362 287 228 358 299 380 319
276 199 356 86 287 296 266 289 333 127 141 349 165 35 56 178 66
0 112 132 220 79 296 232 181 292 233 314 253
188 182 355 68 238 232 154 177 284 100 95 285 81 125 56 66 178
112 0 128 167 169 179 120 69 283 121 213 281
150 67 204 70 155 164 282 216 201 28 187 217 85 167 108 160 198
132 128 0 88 211 269 159 197 172 189 182 135
65 42 182 137 65 85 321 141 111 95 254 138 92 255 175 161 286
220 167 88 0 299 229 104 236 110 149 97 108
341 278 435 151 366 375 298 346 412 193 103 428 230 44 113 235
77 79 169 211 299 0 353 289 213 371 290 379 332
184 271 417 239 300 249 168 108 321 241 279 310 184 309 240 118
362 296 179 269 229 353 0 121 162 345 80 189 342
67 146 292 135 175 147 249 57 221 131 215 200 74 245 176 62 287
232 120 159 104 289 121 0 154 220 41 93 218
221 251 424 137 307 301 95 190 353 169 117 354 150 169 125 92
228 181 69 197 236 213 162 154 0 352 147 247 350
169 105 116 242 57 118 437 245 72 200 359 169 208 327 280 277
358 292 283 172 110 371 345 220 352 0 265 178 39
```

108 191 337 165 220 188 190 43 266 161 216 241 104 246 177 55  
299 233 121 189 149 290 80 41 147 265 0 124 263  
45 139 273 228 121 60 314 81 132 189 308 112 158 335 266 155 380  
314 213 182 97 379 189 93 247 178 124 0 199  
167 79 77 205 97 185 435 243 111 163 322 238 206 288 243 275 319  
253 281 135 108 332 342 218 350 39 263 199 0

**bays29: координаты городов:**

```
1 1150.0 1760.0
2 630.0 1660.0
3 40.0 2090.0
4 750.0 1100.0
5 750.0 2030.0
6 1030.0 2070.0
7 1650.0 650.0
8 1490.0 1630.0
9 790.0 2260.0
10 710.0 1310.0
11 840.0 550.0
12 1170.0 2300.0
13 970.0 1340.0
14 510.0 700.0
15 750.0 900.0
16 1280.0 1200.0
17 230.0 590.0
18 460.0 860.0
19 1040.0 950.0
20 590.0 1390.0
21 830.0 1770.0
22 490.0 500.0
23 1840.0 1240.0
24 1260.0 1500.0
25 1280.0 790.0
26 490.0 2130.0
27 1460.0 1420.0
28 1260.0 1910.0
29 360.0 1980.0
EOF
```

**bays29: лучший тур**

```
1
28
6
12
9
26
3
29
5
21
2
20
10
4
15
18
14
17
22
11
19
25
7
23
8
27
16
13
24
-1
EOF
```

# berlin52: 52 здания in Berlin (Groetschel) [6]

Тип данных: эвклидовы координаты

```
1 565.0 575.0
2 25.0 185.0
3 345.0 750.0
4 945.0 685.0
5 845.0 655.0
6 880.0 660.0
7 25.0 230.0
8 525.0 1000.0
9 580.0 1175.0
10 650.0 1130.0
11 1605.0 620.0
12 1220.0 580.0
13 1465.0 200.0
14 1530.0 5.0
15 845.0 680.0
16 725.0 370.0
17 145.0 665.0
18 415.0 635.0
19 510.0 875.0
20 560.0 365.0
21 300.0 465.0
22 520.0 585.0
23 480.0 415.0
24 835.0 625.0
25 975.0 580.0
26 1215.0 245.0
27 1320.0 315.0
28 1250.0 400.0
29 660.0 180.0
30 410.0 250.0
31 420.0 555.0
32 575.0 665.0
33 1150.0 1160.0
34 700.0 580.0
35 685.0 595.0
36 685.0 610.0
37 770.0 610.0
38 795.0 645.0
39 720.0 635.0
40 760.0 650.0
41 475.0 960.0
42 95.0 260.0
43 875.0 920.0
44 700.0 500.0
45 555.0 815.0
46 830.0 485.0
47 1170.0 65.0
48 830.0 610.0
49 605.0 625.0
50 595.0 360.0
51 1340.0 725.0
52 1740.0 245.0
EOF
```

berlin52: лучший тур

```
1
49
32
45
19
41
8
9
10
43
33
51
11
52
14
13
47
26
27
28
12
25
4
6
15
5
24
48
38
37
40
39
36
35
34
44
46
16
29
50
20
23
30
2
7
42
21
17
3
18
31
22
-1
EOF
```



# eil51: 51 город (Christofides/Eilon) [6]

Тип данных: эвклидовы координаты городов

```
1 37 52
2 49 49
3 52 64
4 20 26
5 40 30
6 21 47
7 17 63
8 31 62
9 52 33
10 51 21
11 42 41
12 31 32
13 5 25
14 12 42
15 36 16
16 52 41
17 27 23
18 17 33
19 13 13
20 57 58
21 62 42
22 42 57
23 16 57
24 8 52
25 7 38
26 27 68
27 30 48
28 43 67
29 58 48
30 58 27
31 37 69
32 38 46
33 46 10
34 61 33
35 62 63
36 63 69
37 32 22
38 45 35
39 59 15
40 5 6
41 10 17
42 21 10
43 5 64
44 30 15
45 39 10
46 32 39
47 25 32
48 25 55
49 48 28
50 56 37
51 30 40
```

EOF

eil51: лучший тур

```
1
22
8
26
31
28
3
36
35
20
2
29
21
16
50
34
30
9
49
10
39
33
45
15
44
42
40
19
41
13
25
14
24
43
7
23
48
6
27
51
46
12
47
18
4
17
37
5
38
11
32
-1
EOF
```

# **eil76: 76 городов (Christofides/Eilon) [6]**

**Тип данных: эвклидовы координаты городов**

1 22 22  
2 36 26  
3 21 45  
4 45 35  
5 55 20  
6 33 34  
7 50 50  
8 55 45  
9 26 59  
10 40 66  
11 55 65  
12 35 51  
13 62 35  
14 62 57  
15 62 24  
16 21 36  
17 33 44  
18 9 56  
19 62 48  
20 66 14  
21 44 13  
22 26 13  
23 11 28  
24 7 43  
25 17 64  
26 41 46  
27 55 34  
28 35 16  
29 52 26  
30 43 26  
31 31 76  
32 22 53  
33 26 29  
34 50 40  
35 55 50  
36 54 10  
37 60 15  
38 47 66  
39 30 60  
40 30 50  
41 12 17  
42 15 14  
43 16 19  
44 21 48  
45 50 30  
46 51 42  
47 50 15  
48 48 21  
49 12 38  
50 15 56  
51 29 39  
52 54 38

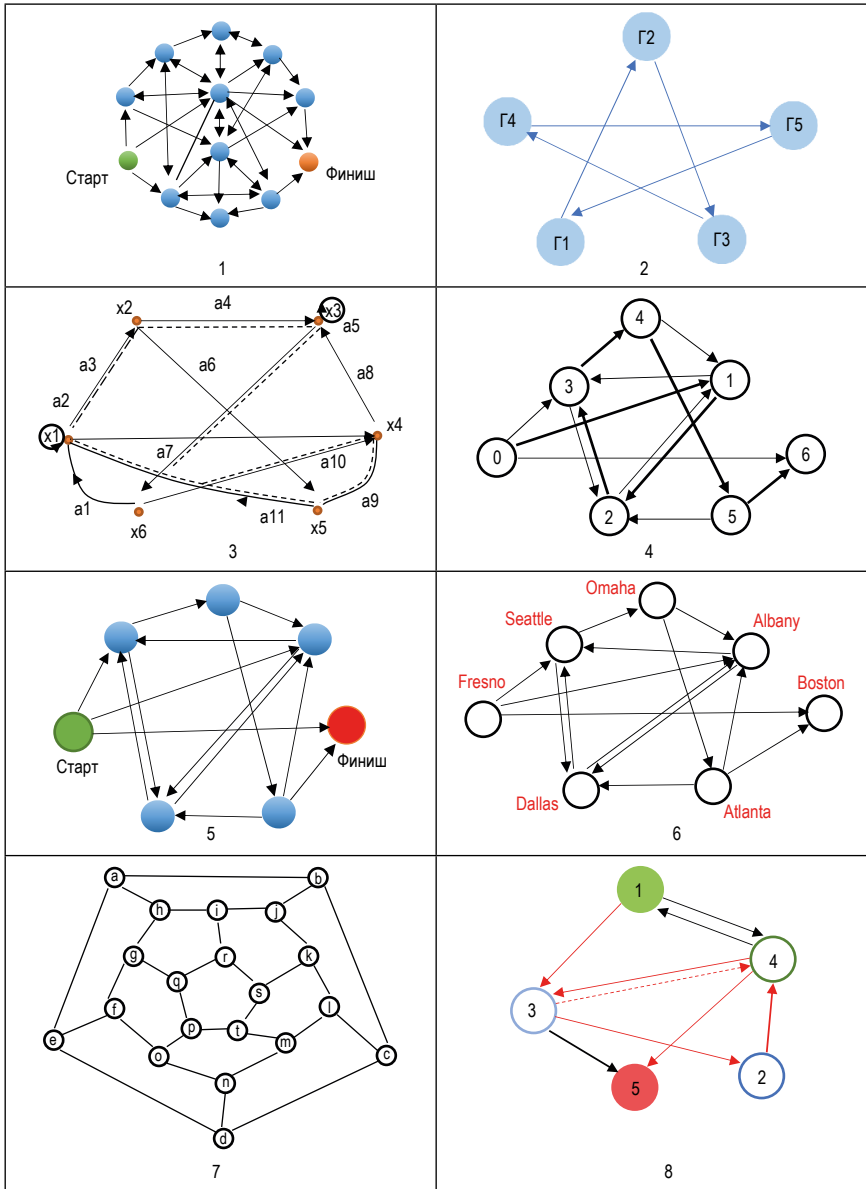
**eil76: лучший тур**

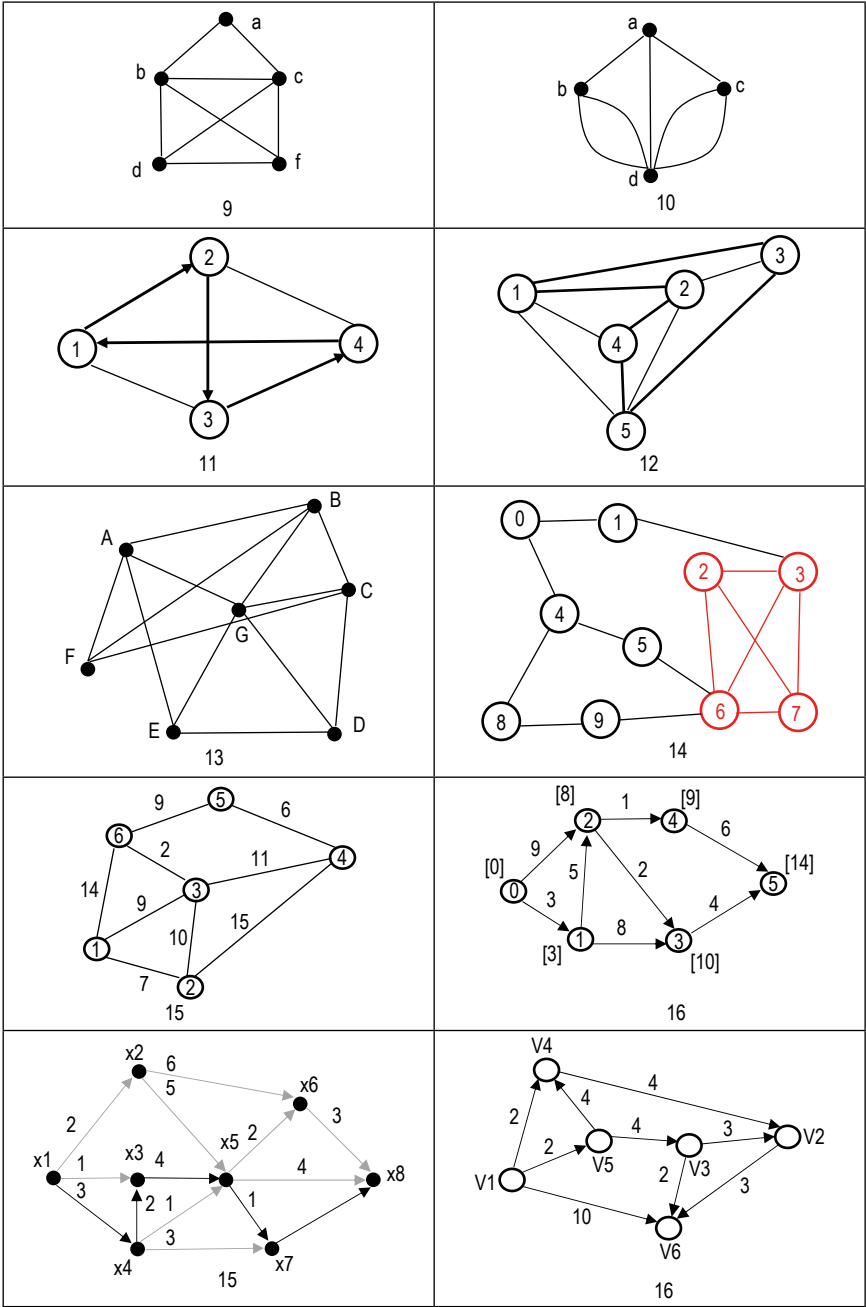
1  
33  
63  
16  
3  
44  
32  
9  
39  
72  
58  
10  
31  
55  
25  
50  
18  
24  
49  
23  
56  
41  
43  
42  
64  
22  
61  
21  
47  
36  
69  
71  
60  
70  
20  
37  
5  
15  
57  
13  
54  
19  
14  
59  
66  
65  
38  
11  
53  
7  
35  
8

53 55 57  
54 67 41  
55 10 70  
56 6 25  
57 65 27  
58 40 60  
59 70 64  
60 64 4  
61 36 6  
62 30 20  
63 20 30  
64 15 5  
65 50 70  
66 57 72  
67 45 42  
68 38 33  
69 50 4  
70 66 8  
71 59 5  
72 35 60  
73 27 24  
74 40 20  
75 40 37  
76 40 40  
EOF

46  
34  
52  
27  
45  
29  
48  
30  
4  
75  
76  
67  
26  
12  
40  
17  
51  
6  
68  
2  
74  
28  
62  
73  
-1  
EOF

Индивидуальные задания на лабораторную работу №4





**Листинг (МАТЛАБ)**

**Пример реализации ГА**

**PROGRAM 1: BINARY GENETIC ALGORITHM**

```
%
    Binary Genetic Algorithm
%
% minimizes the objective function designated in ff
% Before beginning, set all the parameters in parts I, II,
% and III
% Haupt & Haupt
% 2003
clear
%

%-----
%
% I. Setup the GA
ff='testfunction'; % objective function
npar=2; % number of optimization variables
%

%-----
%
% II. Stopping criteria
maxit=100; % max number of iterations
mincost=-9999999; % minimum cost
%

%-----
%
% III. GA parameters
popsize=100; % set population size
nntrate=.1; % set mutation rate
selection=.5; % fraction of population
% kept
nbits=8; % number of bits in each
% parameter
Nt=nbits*npar; % total number of bits
% in a chromosome
keep=floor (selection*popsize) ; % #population members
% that survive
%

%-----
%
% Create the initial population
iga=0; % generation counter
% initialized
pop=round(rand(popsize,Nt)); % random population of
```

```

par=gadecode(pop,0.10, nbits); % Is and Os
cost=feval (ff .par); % convert binary to
% continuous values
[cost,ind]=sort(cost); % calculates population
% cost using ff
par=par(ind,:);pop=pop(ind,:); % min cost in element 1
% sorts population with
% lowest cost first
minc(1)smin(cost); % minc contains min of
% population
meanc(1)smean(cost); % meanc contains mean
% of population
%

% Iterate through generations
while iga<maxit
    iga=iga+1; % increments generation counter
%

% Pair and mate
Msceil((popsize-keep)/2); % number of matings
prob=flipud([1 :keep] 'sum( [1:keep] )); % weights
% chromosomes based
% upon position in
% list
odds=[0 cumsum(prob(1:keep))'1; % probability
distribution function

pick1=rand(1,M); % mate #1
pick2=rand(1,M) ; % mate #2

% ma and pa contain the indicies of the chromosomes
% that will mate
ic=1;
while ic<=M
    for id=2:keep+1
        if pick1(ic)<=odds(id) & pick1(ic)>odds(id-1)
            ma(ic)=id-1;
        end % if
        if pick2(ic)<=odds(id) & pick2(ic)>odds(id-1)
            pa(ic)=id-1;
        end % if
    end % id
    ic=ic*1;
end % while
%

```

```

%           Performs mating using single point crossover
ix=1:2:keep; % index of mate #1
xp=ceil(rand(1,M)*(Nt-1)); % crossover point
pop(keep*ix,:)= [pop(ma,1:xp) pop(pa,xp+1:Nt) ];
% first offspring
pop(keep*ix*1,:)= [pop(pa,1:xp) pop(ma,xp+1:Nt)];
% second offspring
%

```

---

```

%           Mutate the population
nmut=ceil((popsize-1)*Nt*mutrate); % total number
% of mutations
mrow=ceil(rand(1,nmut)*(popsize-1))*1; % row to mutate
mcol=ceil(rand(1,nmut)*Nt); % column to mutate
for ii=1:nmut
    pop(mrow(ii),mcol(ii))=abs(pop(mrow(ii),mcol(ii))-1);
    % toggles bits
end
% ii
%

```

---

```

%           The population is re-evaluated for cost
par(2:popsize,:)=gadecode(pop(2:popsize,:),0,10.nbits);
% decode
cost(2:popsize)=feval(ff,par(2:popsize,:));
%

```

---

```

%           Sort the costs and associated parameters
[cost,ind]=sort(cost);
par=par (ind, : ) , - pop=pop (ind, : ) ;
%

```

---

```

% Do statistics for a single nonaveraging run
mine(iga+1)=min(cost);
meanc(iga*1)=mean(cost);
%

```

---

```

%           Stopping criteria
if iga>maxit | cost(1)<mincost
    break
end
[iga cost(1)]
end %iga
%

```

---



```

%           Displays the output
day=clock;
disp(datestr(datetime(day(1).day(2).day(3),day(4).day(5).
    day(6)),0))
disp(['optimized function is ' ff] )
format short g
disp(['popsize = ' num2str(popsize) ' mutrate = '
num2str (mutrate) ' # par = ' num2str(npar)])
disp(['#generations=' num2str(iga) ' best cost='
num2str(cost(1))])
disp(['best solution'])
disp([num2str(par(1,:))])
disp('binary genetic algorithm')
disp(['each parameter represented by ' num2str(nbits)
'bits'])
figure(24)
iters=0:length(minc)-1;
plot (iters,minc,iters,meanc,'-');
xlabel('generation') ; ylabel ('cost');
text (0,minc (1), 'best'); text (1,minc(2). 'population
average')
%



---


%           Sort the costs and associated parameters
[cost,ind]=sort(cost);
par=par (ind, : ) ,- pop=pop (ind, :) ;

%



---


%           Do statistics for a single nonaveraging run
mine(iga+1)=min(cost);
meanc(iga*1)=mean(cost);
%



---


%           Stopping criteria
if iga>maxit | cost(1)<mincost
    break
end

[iga cost(1)]

end %iga
%



---



```

```

%                               Displays the output
day=clock;
disp(datestr(datumum(day(1).day(2).day(3),day(4).day(5),
day(6)),0))
disp(['optimized function is ' ff] )
format short g
disp(['popsize = ' num2str(popsize) ' mutrate = '
num2str (mutrate)' # par = ' num2str(npar)])
disp(['#generations=' num2str(iga) ' best cost='
num2str(cost(1))])
disp(['best solution'])
disp([num2str(par(1,:))])
disp('binary genetic algorithm')
disp(['each parameter represented by ' num2str(nbits)
'bits'])
figure(24)
iters=0:length(minc)-1;
plot (iters,minc,iters,meanc, '-');
xlabel ('generation');ylabel ('cost');
text (0,minc (1), 'best'); text (1,minc(2),'population
average')

```

## PROGRAM 2: CONVERTS BINARY CHROMOSOME TO CONTINUOUS VARIABLES

```

%       gadecode.m
%       Decodes binary encripted parameters
%       f=gadecode(chrom,lo,hi,bits,gray)
%       chrocn = population
%       lo =      minimum parameter   value
%       hi =      maximum parameter   value
%       bits   = number of bits/parameter
% Haupt & Haupt
% 2003
function f=gadecode(chrom,lo,hi,bits)
[M,N]=size(chrom);
npar=N/bits;                               % number of variables
quant=(0.5.*[1:bits]');                   % quantization levels
quant=quant/sum(quant);                   % quantization levels
normalized ct=reshape(chrom'.bits,npar'M) % each column contains
                                           % one variable
par=((ct"quant)*(hi-lo)*lo>;              % DA conversion and
                                           % unnormalize variables
f=reshape(par,npar,M)';                   % reassemble population

```

### PROGRAM 3: CONTINUOUS GENETIC ALGORITHM

```
%           Continuous Genetic Algorithm
%
% minimizes the objective function designated in ff
% Before beginning, set all the parameters in parts
% I, II, and III
% Haupt & Haupt
%      2003
%


---


%           I Setup the GA
ff='testfunction';    % objective function
npar=2;               % number of optimization variables
varhi=10; varlo=0;    % variable limits
%


---


%           II Stopping criteria
maxit=100;             %max number of iterations
mincost=-9999999;      %minimum cost
%


---


%           III GA parameters
popsize=12;            % set population size
mutrate=.2;           % set mutation rate
selection=0- 5;        % fraction of population kept
Nt=npar;               % continuous parameter GA Nt=#variables
keep=floor (selection*popsize); % #population
                                % members that survive
nmut=ceil((popsize-1)*Nt*mutrate); % total number of
                                % mutations
M=ceil((popsize-keep)/2); % number of matings
%


---


%           Create the initial population
iga=0;                 % generation counter
initialized
par=(varhi-varlo)'rand(popsize,npar)»varlo; % random
cost=feval(ff.par);    % calculates population cost
                                % using ff
[cost,ind]=sort(cost); % min cost in element 1
par=par(ind,:);        % sort continuous
minc(1)=min(cost);     % mine contains min of
meanc(1)=mean(cost);  % meanc contains mean of
population
%
```

---

```

%               Iterate through generations
while iga<maxit
    iga=iga+1;    % increments generation counter
%


---


%               Pair and mate
M=ceil((popsize-keep)/2);    % number of matings
prob=flipud([1:keep] ' /sum([1:keep])));    % weights
                                % chromosomes
odds=[0 cumsum(prob(1:keep))'];    % probability
                                % distribution
                                % function
pick1=rand(1.M);    % mate #1
pick2=rand(1.M);    % mate #2

% ma and pa contain the indicies of the chromosomes
% that will mate
ic=1;
    while ic<=M
        for id=2:keep+1
            if pick1(ic)<=odds(id)
                & pick1(ic)>odds(id-1) ma(ic)=id-1;
            end
        if pick2(ic)<=odds(id) i pick2(ic)>odds(id-1) pa(ic)=id-1;
        end
        end
        ic=ic+1;
    end
%


---


% Performs mating using single point crossover ix=1:2:keep;
                                % index of mate #1
xp=ceil(rand(1.M)*Nt);    % crossover point
r=rand(1,M);    % mixing parameter
for ic=1:M xy=par(ma(ic),xp(ic))-par(pa(ic),xp(ic));
                                % ma and pa % mate
par(keep*ix(ic),:)=par(ma(ic),:);    % 1st offspring
    par(keep*ix(ic)*1,:)=par(pa(ic),:);    % 2nd offspring
    par(keep*ix(ic),xp(ic))=par(ma(ic),xp(ic))-r(ic).'xy;
% 1st par(keep*ix(ic)*1,xp(ic))=par(pa(ic),xp(ic))*r(ic)."xy; % 2nd
    if xp(ic)<npar % crossover when last variable not selected
par(keep+ix(ic),:)=par(keep*ix(ic),1:xp(ic)) par(keep*ix(ic)*1,x
p(ic)*1:npar)1;
par(keep*ix(ic)*1,:)=par(keep*ix(ic)*1,1:xp(ic)) par(keep*
ix(ic),xp(ic)*1:npar)];
        end % if end
%


---



```

```

%      Mutate the population
mrowssort(ceil(rand(1,nmut)*(popsize-1))*1);
mcol=ceil(rand(1,nmut)*Nt);

for ii=1:nmut
par(mrow(ii).mcol(ii))=(varhi-varlo)*rand+varlo;
% mutation
end % ii
%

%      The new offspring and mutated chromosomes are % evaluated
cost=feval(ff.par);
%

%      Sort the costs and associated parameters
[cost,ind]=sort(cost);
par=par(ind,:);
%

%      Do statistics for a single nonaveraging run
mine(iga*1)smin(cost);
meanc(iga*1)=mean(cost)
%

%      Stopping criteria
if iga>maxit | cost(1)<mincost
    break
end
[iga cost(1)]
end %iga
%

%      Displays the output
daysclock;
disp(datestr(datenum(day(1),day(2).day(3),day(4),day(5),
day(6)).0))
disp(['optimized function is ' ff])
format short g
disp(['popsize = ' num2str(popsize) ' mutrate = '
num2str(mutrate) '# par = ' num2str(npar)])
disp(['#generations=' num2str(iga) ' best cost='
num2str(cost(1))])
disp(['best solution'])
disp([num2str(par(1,:))])
disp('continuous genetic algorithm')

```

```
figure (24)
iters=0:length(mine)-1;
plot(iters,mine,iters,meanc,'-');
xlabel ('generation'); ylabel('cost');
text(0,mine(1), 'best'); text(1,meanc(2), 'population
average')
```

**Пример реализации ГА на С**

```

https://people.sc.fsu.edu/~jburkardt/cpp_src/simple_ga/
simple_ga.cpp
# include <cmath>
# include <cstdlib>
# include <iostream>
# include <iomanip>
# include <fstream>
# include <iomanip>
# include <ctime>
# include <cstring>
using namespace std;
//
//   Change any of these parameters to match your needs
//
# define POPSIZE 50
# define MAXGENS 1000
# define NVAR 3
# define PXOVER 0.8
# define PMUTATION 0.15
//
//   Each GENOTYPE is a member of the population, with
// gene: a string of variables,
// fitness: the fitness
// upper: the variable upper bounds,
// lower: the variable lower bounds,
// rfitness: the relative fitness,
// cfitness: the cumulative fitness.
//
struct genotype
{
    double gene[NVAR];
    double fitness;
    double upper[NVAR];
    double lower[NVAR];
    double rfitness;
    double cfitness;
};
struct genotype population[POPSIZE+1];
struct genotype newpopulation[POPSIZE+1];
int main ();
void crossover (int &seed);
void elitist ();

```

```

void evaluate ();
int i4_uniform_ab (int a, int b, int &seed);
void initialize (string filename, int &seed);
void keep_the_best ();
void mutate (int &seed);
double r8_uniform_ab (double a, double b, int &seed);
void report (int generation);
void selector (int &seed);
void timestamp ();
void Xover (int one, int two, int &seed);
//*****
***80
int main ()
//*****
***80
//
// Purpose:
//
// MAIN supervises the genetic algorithm.
//
// Discussion:
//
// Each generation involves selecting the best
// members, performing crossover & mutation and then
// evaluating the resulting population, until the terminating
// condition is satisfied
//
// This is a simple genetic algorithm implementation where
the
// evaluation function takes positive values only and the
// fitness of an individual is the same as the value of the
// objective function.
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//
// 28 April 2014
//
// Author:
//
// Original version by Dennis Cormier and Sita Raghavan.
// This C++ version by John Burkardt.
//

```



```

// Reference:
//
// Zbigniew Michalewicz,
// Genetic Algorithms + Data Structures = Evolution Programs,
// Third Edition,
// Springer, 1996,
// ISBN: 3-540-60676-9,
// LC: QA76.618.M53.
//
// Parameters:
//
// MAXGENS is the maximum number of generations.
//
// NVARs is the number of problem variables.
//
// PMUTATION is the probability of mutation.
//
// POPSIZE is the population size.
//
// PXOVER is the probability of crossover.
//
{
string filename = "input.txt";
int generation;
int i;
int seed;
timestamp ();
cout << "\n";
cout << "SIMPLE_GA:\n";
cout << " C++ version\n";
cout << " A simple example of a genetic algorithm.\n";
if (NVARs < 2)
{
    cout << "\n";
    cout << " The crossover modification will not be available,\n";
    cout << " since it requires 2 <= NVARs.\n";
}
seed = 123456789;
initialize (filename, seed);
evaluate ();
keep_the_best ();
for (generation = 0; generation < MAXGENS; generation++)
{
    selector (seed);
    crossover (seed);
    mutate (seed);
}

```

```

    report (generation);
    evaluate ();
    elitist ();
}
cout << "\n";
cout << " Best member after " << MAXGENS << " generations:\n";
cout << "\n";
for (i = 0; i < NVARs; i++)
{
    cout << " var(" << i << ") = " << population[POPSIZE].gene[i]
        << "\n";
}
cout << "\n";
cout << " Best fitness = " << population[POPSIZE].fitness
<< "\n";
//
// Terminate.
//
cout << "\n";
cout << "SIMPLE_GA:\n";
cout << " Normal end of execution.\n";
cout << "\n";
timestamp ();
return 0;
}
//*****
***80
void crossover (int &seed)
//*****
***80
//
// Purpose:
//
// CROSSOVER selects two parents for the single point crossover.
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//
// 28 April 2014
//
// Author:
//
// Original version by Dennis Cormier and Sita Raghavan.

```

```

// This C++ version by John Burkardt.
//
//   Local parameters:
//
// Local, int FIRST, is a count of the number of members
chosen.
//
//   Parameters:
//
// Input/output, int &SEED, a seed for the random number
generator.
//
{
  const double a = 0.0;
  const double b = 1.0;
  int mem;
  int one;
  int first = 0;
  double x;
  for (mem = 0; mem < POPSIZE; ++mem)
  {
    x = r8_uniform_ab (a, b, seed);
    if (x < PXOVER)
    {
      ++first;
      if (first % 2 == 0)
      {
        Xover (one, mem, seed);
      }
      else
      {
        one = mem;
      }
    }
  }
  return;
}

//*****
***80
void elitist ()
//*****
***80
//
//   Purpose:
//

```

```

// ELITIST stores the best member of the previous generation.
//
// Discussion:
//
// The best member of the previous generation is stored as
// the last in the array. If the best member of the current
// generation is worse then the best member of the previous
// generation, the latter one would replace the worst member
// of the current population.
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//
// 29 December 2007
//
// Author:
//
// Original version by Dennis Cormier and Sita Raghavan.
// This C++ version by John Burkardt.
//
// Local parameters:
//
// Local, double BEST, the best fitness value.
//
// Local, double WORST, the worst fitness value.
//
{
    int i;
    double best;
    int best_mem;
    double worst;
    int worst_mem;
    best = population[0].fitness;
    worst = population[0].fitness;
    for (i = 0; i < POPSIZE - 1; ++i)
    {
        if (population[i+1].fitness < population[i].fitness)
        {
            if (best <= population[i].fitness)
            {
                best = population[i].fitness;
                best_mem = i;
            }
        }
    }
}

```

```

        if (population[i+1].fitness <= worst)
        {
            worst = population[i+1].fitness;
            worst_mem = i + 1;
        }
    }
else
{
    if (population[i].fitness <= worst)
    {
        worst = population[i].fitness;
        worst_mem = i;
    }
    if (best <= population[i+1].fitness)
    {
        best = population[i+1].fitness;
        best_mem = i + 1;
    }
}
}
//
// If the best individual from the new population is better
// than
// the best individual from the previous population, then
// copy the best from the new population; else replace the
// worst individual from the current population with the
// best one from the previous generation
//
if (population[POPSIZE].fitness <= best)
{
    for (i = 0; i < NVAR; i++)
    {
        population[POPSIZE].gene[i] = population[best_mem].gene[i];
    }
    population[POPSIZE].fitness = population[best_mem].fitness;
}
else
{
    for (i = 0; i < NVAR; i++)
    {
        population[worst_mem].gene[i] = population[POPSIZE].gene[i];
    }
    population[worst_mem].fitness = population[POPSIZE].fitness;
}
return;
}

```

```

//*****
***80
void evaluate ()
//*****
***80
//
// Purpose:
//
// EVALUATE implements the user-defined valuation function
//
// Discussion:
//
// Each time this is changed, the code has to be recompiled.
// The current function is:  $x[1]^2 - x[1] * x[2] + x[3]$ 
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//
// 29 December 2007
//
// Author:
//
// Original version by Dennis Cormier and Sita Raghavan.
// This C++ version by John Burkardt.
//
{
    int member;
    int i;
    double x[NVARS+1];
    for (member = 0; member < POPSIZE; member++)
    {
        for (i = 0; i < NVARS; i++)
        {
            x[i+1] = population[member].gene[i];
        }
        population[member].fitness = (x[1] * x[1]) - (x[1] * x[2]) + x[3];
    }
    return;
}
//*****
***80
int i4_uniform_ab (int a, int b, int &seed)

```

```

//*****
***80
//
// Purpose:
//
// I4_UNIFORM_AB returns a scaled pseudorandom I4 between
// A and B.
//
// Discussion:
//
// The pseudorandom number should be uniformly distributed
// between A and B.
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//
// 02 October 2012
//
// Author:
//
// John Burkardt
//
// Reference:
//
// Paul Bratley, Bennett Fox, Linus Schrage,
// A Guide to Simulation,
// Second Edition,
// Springer, 1987,
// ISBN: 0387964673,
// LC: QA76.9.C65.B73.
//
// Bennett Fox,
// Algorithm 647:
// Implementation and Relative Efficiency of Quasirandom
// Sequence Generators,
// ACM Transactions on Mathematical Software,
// Volume 12, Number 4, December 1986, pages 362-376.
//
// Pierre L'Ecuyer,
// Random Number Generation,
// in Handbook of Simulation,
// edited by Jerry Banks,
// Wiley, 1998,

```

```

// ISBN: 0471134031,
// LC: T57.62.H37.
//
// Peter Lewis, Allen Goodman, James Miller,
// A Pseudo-Random Number Generator for the System/360,
// IBM Systems Journal,
// Volume 8, Number 2, 1969, pages 136-143.
//
// Parameters:
//
// Input, int A, B, the limits of the interval.
//
// Input/output, int &SEED, the "seed" value, which should
NOT be 0.
// On output, SEED has been updated.
//
// Output, int I4_UNIFORM, a number between A and B.
//
{
    int c;
    const int i4_huge = 2147483647;
    int k;
    float r;
    int value;
    if (seed == 0)
    {
        cerr << "\n";
        cerr << "I4_UNIFORM_AB - Fatal error!\n";
        cerr << " Input value of SEED = 0.\n";
        exit (1);
    }
//
// Guarantee A <= B.
//
    if (b < a)
    {
        c = a;
        a = b;
        b = c;
    }
    k = seed / 127773;
    seed = 16807 * (seed - k * 127773) - k * 2836;
    if (seed < 0)
    {
        seed = seed + i4_huge;
    }
}

```



```

r = (float) (seed) * 4.656612875E-10;
//
// Scale R to lie between A-0.5 and B+0.5.
//
r = (1.0 - r) * ((float) a - 0.5)
  + r * ((float) b + 0.5);
//
// Use rounding to convert R to an integer between A and B.
//
value = round (r);
//
// Guarantee A <= VALUE <= B.
//
    if (value < a)
    {
        value = a;
    }
    if (b < value)
    {
        value = b;
    }
    return value;
}
//*****
***80
void initialize (string filename, int &seed)
//*****
***80
//
//   Purpose:
//
//   INITIALIZE initializes the genes within the variables
//   bounds.
//
//   Discussion:
//
//   It also initializes (to zero) all fitness values for each
//   member of the population. It reads upper and lower bounds
//   of each variable from the input file `gadata.txt'. It
//   randomly generates values between these bounds for each
//   gene of each genotype in the population. The format of
//   the input file `gadata.txt' is
//
//   var1_lower_bound var1_upper bound
//   var2_lower_bound var2_upper bound...
//

```

```

//   Licensing:
//
//   This code is distributed under the GNU LGPL license.
//
//   Modified:
//
//   28 April 2014
//
//   Author:
//
//   Original version by Dennis Cormier and Sita Raghavan.
//   This C++ version by John Burkardt.
//
//   Parameters:
//
//   Input, string FILENAME, the name of the input file.
//
//   Input/output, int &SEED, a seed for the random number
//   generator.
//
//
{
    int i;
    ifstream input;
    int j;
    double lbound;
    double ubound;
    input.open (filename.c_str ());
    if (!input)
    {
        cerr << "\n";
        cerr << "INITIALIZE - Fatal error!\n";
        cerr << " Cannot open the input file!\n";
        exit (1);
    }
//
//   Initialize variables within the bounds
//
    for (i = 0; i < NVAR; i++)
    {
        input >> lbound >> ubound;
        for (j = 0; j < POPSIZE; j++)
        {
            population[j].fitness = 0;
            population[j].rfitness = 0;
            population[j].cfitness = 0;
            population[j].lower[i] = lbound;

```

```

        population[j].upper[i]= ubound;
        population[j].gene[i] = r8_uniform_ab (lbound,
        ubound, seed);
    }
}
input.close ();
return;
}
//*****
***80
void keep_the_best ()
//*****
***80
//
// Purpose:
//
// KEEP_THE_BEST keeps track of the best member of the
// population.
//
// Discussion:
//
// Note that the last entry in the array Population holds a
// copy of the best individual.
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//
// 29 December 2007
//
// Author:
//
// Original version by Dennis Cormier and Sita Raghavan.
// This C++ version by John Burkardt.
//
// Local parameters:
//
// Local, int CUR_BEST, the index of the best individual.
//
{
    int cur_best;
    int mem;
    int i;
    cur_best = 0;

```

```

    for (mem = 0; mem < POPSIZE; mem++)
    {
        if (population[POPSIZE].fitness < population[mem].fitness)
        {
            cur_best = mem;
            population[POPSIZE].fitness = population[mem].fitness;
        }
    }
//
// Once the best member in the population is found, copy the
genes.
//
    for (i = 0; i < NVAR; i++)
    {
        population[POPSIZE].gene[i] = population[cur_best].
gene[i];
    }
    return;
}
//*****
***80
void mutate (int &seed)
//*****
***80
//
// Purpose:
//
// MUTATE performs a random uniform mutation.
//
// Discussion:
//
// A variable selected for mutation is replaced by a random
value
// between the lower and upper bounds of this variable.
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//
// 28 April 2014
//
// Author:
//
// Original version by Dennis Cormier and Sita Raghavan.

```

```

// This C++ version by John Burkardt.
//
// Parameters:
//
// Input/output, int &SEED, a seed for the random number
// generator.
//
{
    const double a = 0.0;
    const double b = 1.0;
    int i;
    int j;
    double lbound;
    double ubound;
    double x;
    for (i = 0; i < POPSIZE; i++)
    {
        for (j = 0; j < NVAR; j++)
        {
            x = r8_uniform_ab (a, b, seed);
            if (x < PMUTATION)
            {
                lbound = population[i].lower[j];
                ubound = population[i].upper[j];
                population[i].gene[j] = r8_uniform_ab
                    (lbound, ubound, seed);
            }
        }
    }
    return;
}

//*****
***80
double r8_uniform_ab (double a, double b, int &seed)
//*****
***80
//
// Purpose:
//
// R8_UNIFORM_AB returns a scaled pseudorandom R8.
//
// Discussion:
//
// The pseudorandom number should be uniformly distributed
// between A and B.
//

```

```

//   Licensing:
//
//   This code is distributed under the GNU LGPL license.
//
//   Modified:
//
//   09 April 2012
//
//   Author:
//
//   John Burkardt
//
//   Parameters:
//
//   Input, double A, B, the limits of the interval.
//
//   Input/output, int &SEED, the "seed" value, which should
//   NOT be 0.
//   On output, SEED has been updated.
//
//   Output, double R8_UNIFORM_AB, a number strictly between
//   A and B.
//
{
    int i4_huge = 2147483647;
    int k;
    double value;
    if (seed == 0)
    {
        cerr << "\n";
        cerr << "R8_UNIFORM_AB - Fatal error!\n";
        cerr << " Input value of SEED = 0.\n";
        exit (1);
    }
    k = seed / 127773;
    seed = 16807 * (seed - k * 127773) - k * 2836;
    if (seed < 0)
    {
        seed = seed + i4_huge;
    }
    value = (double) (seed) * 4.656612875E-10;
    value = a + (b - a) * value;
    return value;
}
//*****
***80

```

```

void report (int generation)
//*****
***80
//
//   Purpose:
//
//   REPORT reports progress of the simulation.
//
//   Licensing:
//
//   This code is distributed under the GNU LGPL license.
//
//   Modified:
//
//   29 December 2007
//
//   Author:
//
//   Original version by Dennis Cormier and Sita Raghavan.
//   This C++ version by John Burkardt.
//
//   Local parameters:
//
//   Local, double avg, the average population fitness.
//
//   Local, best_val, the best population fitness.
//
//   Local, double square_sum, square of sum for std calc.
//
//   Local, double stddev, standard deviation of population
fitness.
//
//   Local, double sum, the total population fitness.
//
//   Local, double sum_square, sum of squares for std calc.
//
{
    double avg;
    double best_val;
    int i;
    double square_sum;
    double stddev;
    double sum;
    double sum_square;
    if (generation == 0)
    {

```

```

        cout << "\n";
        cout << " Generation Best Average Standard \n";
        cout << " number value fitness deviation \n";
        cout << "\n";
    }
    sum = 0.0;
    sum_square = 0.0;
    for (i = 0; i < POPSIZE; i++)
    {
        sum = sum + population[i].fitness;
        sum_square = sum_square + population[i].fitness
            * population[i].fitness;
    }
    avg = sum / (double) POPSIZE;
    square_sum = avg * avg * POPSIZE;
    stddev = sqrt ((sum_square - square_sum) / (POPSIZE - 1));
    best_val = population[POPSIZE].fitness;
    cout << " " << setw(8) << generation
        << " " << setw(14) << best_val
        << " " << setw(14) << avg
        << " " << setw(14) << stddev << "\n";
    return;
}
//*****
***80
void selector (int &seed)
//*****
***80
//
// Purpose:
//
// SELECTOR is the selection function.
//
// Discussion:
//
// Standard proportional selection for maximization problems
incorporating
// the elitist model. This makes sure that the best member
always survives.
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//

```



```

// 28 April 2014
//
//   Author:
//
// Original version by Dennis Cormier and Sita Raghavan.
// This C++ version by John Burkardt.
//
//   Parameters:
//
// Input/output, int &SEED, a seed for the random number
generator.
//
{
    const double a = 0.0;
    const double b = 1.0;
    int i;
    int j;
    int mem;
    double p;
    double sum;
//
// Find the total fitness of the population.
//
    sum = 0.0;
    for (mem = 0; mem < POPSIZE; mem++)
    {
        sum = sum + population[mem].fitness;
    }
//
// Calculate the relative fitness of each member.
//
    for (mem = 0; mem < POPSIZE; mem++)
    {
        population[mem].rfitness = population[mem].fitness / sum;
    }
//
// Calculate the cumulative fitness.
//
    population[0].cfitness = population[0].rfitness;
    for (mem = 1; mem < POPSIZE; mem++)
    {
        population[mem].cfitness = population[mem-1].cfitness +
            population[mem].rfitness;
    }
//
// Select survivors using cumulative fitness.

```

```

//
for (i = 0; i < POPSIZE; i++)
{
    p = r8_uniform_ab (a, b, seed);
    if (p < population[0].cfitness)
    {
        newpopulation[i] = population[0];
    }
    else
    {
        for (j = 0; j < POPSIZE; j++)
        {
            if (population[j].cfitness <= p && p < population[j+1].
                cfitness)
            {
                newpopulation[i] = population[j+1];
            }
        }
    }
}
//
// Overwrite the old population with the new one.
//
for (i = 0; i < POPSIZE; i++)
{
    population[i] = newpopulation[i];
}
return;
}
//*****
***80
void timestamp ()
//*****
***80
//
// Purpose:
//
// TIMESTAMP prints the current YMDHMS date as a time stamp.
//
// Example:
//
// May 31 2001 09:45:54 AM
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.

```

```

//
//   Modified:
//
//   04 October 2003
//
//   Author:
//
//   John Burkardt
//
{
# define TIME_SIZE 40
static char time_buffer[TIME_SIZE];
const struct tm *tm;
time_t now;
now = time (NULL);
tm = localtime (&now);
strftime (time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p",
tm);
cout << time_buffer << "\n";
return;
# undef TIME_SIZE
}
//*****
***80
void Xover (int one, int two, int &seed)
//*****
***80
//
//   Purpose:
//
//   XOVER performs crossover of the two selected parents.
//
//   Licensing:
//
//   This code is distributed under the GNU LGPL license.
//
//   Modified:
//
//   28 April 2014
//
//   Author:
//
//   Original version by Dennis Cormier and Sita Raghavan.
//   This C++ version by John Burkardt.
//
//   Local parameters:

```

```

//
// Local, int point, the crossover point.
//
// Parameters:
//
// Input, int ONE, TWO, the indices of the two parents.
//
// Input/output, int &SEED, a seed for the random number
// generator.
//
{
    int i;
    int point;
    double t;
//
// Select the crossover point.
//
    point = i4_uniform_ab (0, NVARs - 1, seed);
//
// Swap genes in positions 0 through POINT-1.
//
    for (i = 0; i < point; i++)
    {
        t = population[one].gene[i];
        population[one].gene[i] = population[two].gene[i];
        population[two].gene[i] = t;
    }
    return;
}

```

## СОДЕРЖАНИЕ

Введение .....	3
Лабораторная работа №1. Простой генетический алгоритм .....	6
1.1. Теоретическая часть .....	6
1.2. Практическая часть .....	17
Лабораторная работа №2. Оптимизация многомерных функций с помощью ГА.....	20
2.1. Теоретическая часть .....	20
2.2. Практическая часть .....	26
Лабораторная работа №3. Решение задачи коммивояжера с помощью генетических алгоритмов .....	28
3.1. Теоретическая часть .....	28
3.2. Практическая часть .....	34
Лабораторная работа №4. Генетическое программирование .....	37
4.1. Теоретическая часть .....	37
4.2. Практическая часть .....	51
Лабораторная работа №5. Оптимизация многомерных функций с помощью эволюционной стратегии.....	55
5.1. Теоретическая часть .....	55
5.2. Практическая часть .....	61
Лабораторная работа №6. Оптимизация путей на графах с помощью муравьиных алгоритмов .....	63
6.1. Теоретическая часть .....	63
6.2. Практическая часть .....	71
Лабораторная работа №7. Оптимизация функций многих переменных с помощью роевых алгоритмов .....	74
7.1. Теоретическая часть .....	74
7.2. Практическая часть .....	77
Лабораторная работа №8. Эволюционные алгоритмы оценки стоимости проектов в программной инженерии .....	79
8.1. Методика СОСОМО .....	79
8.2. Практическая часть .....	84
Литература .....	87
Приложение 1. Индивидуальные задания на лабораторную работу №2 .....	89
Приложение 2. Тестовые наборы к лабораторной работе №3 .....	95
Приложение 3. Индивидуальные задания на лабораторную работу №4 .....	108
Приложение 4. Листинг (МАТЛАБ). Пример реализации ГА .....	110
Приложение 5. Пример реализации ГА на С.....	119

Учебное издание

**Скобцов Юрий Александрович**

**ВЫЧИСЛИТЕЛЬНЫЙ ИНТЕЛЛЕКТ**

**Лабораторный практикум**

Публикуется в авторской редакции  
Компьютерная верстка *С. Б. Мацапуры*

---

Подписано к печати 07.05.2022. Формат 60 × 84 1/16.

Усл. печ. л. 0,0. Уч.-изд. л. 0,0.

Тираж 50 экз. Заказ № 000.

---

Редакционно-издательский центр ГУАП  
190000, г. Санкт-Петербург, ул. Б. Морская, д. 67, лит. А