

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ  
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**  
**для проведения контрольных работ по дисциплине**  
**«Интеллектуальный анализ данных»**

Донецк  
2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ  
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**  
**для проведения контрольных работ по дисциплине**  
**«Интеллектуальный анализ данных»**

для обучающихся по направлениям подготовки  
09.04.01 «Информатика и вычислительная техника»  
заочной формы обучения

РАССМОТРЕНО  
на заседании кафедры  
«Автоматизированные системы  
управления»  
Протокол № 6 от 17.01.2022 г.

УТВЕРЖДЕНО  
на заседании учебно-  
издательского совета ДОННТУ  
Протокол № 1 от 26.01.2022 г.

Донецк  
2022

УДК 004.8(076)  
ББК 32.813я73  
М 54

**Составитель:**

Васяева Татьяна Александровна – кандидат технических наук, доцент, доцент кафедры автоматизированных системы управления.

М 54

**Методические рекомендации для проведения контрольных работ по дисциплине «Интеллектуальный анализ данных»** [Электронный ресурс]: для обучающихся по направлениям подготовки 09.04.01 «Информатика и вычислительная техника», заочной формы обучения/ ГОУВПО «ДОННТУ», каф. автоматизированных системы управления; сост. Т. А. Васяева. – Донецк: ДОННТУ, 2021. – Систем. требования: Acrobat Reader. – Загл. с титул. экрана.

Методические рекомендации разработаны с целью оказания помощи обучающимся в усвоении теоретического материала и получении практических навыков по дисциплине «Интеллектуальный анализ данных», которые содержат задания для выполнения контрольных работ по курсу.

УДК 004.8(076)  
ББК 32.813я73

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
Контрольная работа №1.....	8
Контрольная работа №2.....	13
Контрольная работа №3.....	29
СПИСОК ЛИТЕРАТУРЫ.....	34
Приложение А. Индивидуальные задания на контрольную работу №1.....	35
Приложение Б. Индивидуальные задания на контрольную работу №7.....	43

## ВВЕДЕНИЕ

Дисциплина рассматривает вопросы получения данных из различных источников, контроля целостности и качества полученных данных, интеллектуального анализа данных, машинного обучения и глубокого обучения.

Цели дисциплины: ознакомление с теоретическими аспектами технологии Data Mining, Machine Learning and Deep Learning; ознакомление студентов с различными типами данных, методами их получения и оценкой качества; ознакомление с классическими задачами интеллектуального анализа данных; рассмотрение моделей для решения базовых задач интеллектуального анализа данных и изучение методов для их разработки; приобретение практических навыков по использованию инструментальных средств Data Mining, Machine Learning and Deep Learning.

В результате освоения дисциплины студент должен знать: место и значение анализа данных, основные этапы интеллектуального анализа данных; основные понятия, задачи, практическое применение, модели и методы интеллектуального анализа данных; принципы сбора данных из различных источников и их обработки; основы машинного и глубокого обучения; основы эволюционных вычислений; уметь: применять полученные знания при разработке, внедрении и эксплуатации автоматизированных и информационно-аналитических систем; получать данные из различных источников, выполнять их анализ и предварительную обработку; подбирать необходимые методы интеллектуального анализа в соответствии с задачами аналитической работы; выполнять программную реализацию методов интеллектуального анализа данных для решения типовых практических задач; решать задачи связанные с обработкой естественного языка; решать задачи вычислительной и комбинаторной оптимизации на базе эволюционных вычислений.

Эволюционные вычисления (ЭВ) – относительно новое направление в теории и практике искусственного интеллекта. Этот термин обычно используется для общего описания алгоритмов поиска, оптимизации или обучения основанных на некоторых формализованных принципах естественного эволюционного отбора. Особенности идей эволюции и самоорганизации заключаются в том, что они находят подтверждение не только для биологических систем развивающихся много миллиардов лет. Эти идеи в настоящее время с успехом используются при разработке многих технических и, в особенности, программных систем.

История эволюционных вычислений началась в 60-е годы XX века, когда различные группы ученых в области кибернетики независимо друг от друга исследовали возможности применения принципов эволюции биологических систем при решении различных технических проблем, как правило, требующих решения задач оптимизации. Таким образом, было основано новое научное направление, которое в настоящее время принято называть «эволюционными вычислениями».

Методы ЭВ обычно используются для оценки и выбора (суб)оптимальных непрерывных параметров моделей большой размерности, для решения различных NP-полных комбинаторных задач и многих других областях науки и техники. Следует отметить, что когда задача не может быть решена другими, более простыми методами, ЭВ часто могут найти оптимальные или близкие к ним решения.

Основные идеи генетических алгоритмов и их структура довольно просты, как и большинство генетических операторов. Поэтому вполне посильно разработать программу, реализующую генетический алгоритм решения конкретной задачи, с нуля. Однако, как часто бывает при разработке программного обеспечения, использование проверенной специализированной библиотеки может облегчить жизнь. Она помогает создавать решения быстрее и с меньшим числом ошибок, а также предлагает на выбор много отработанных заготовок.

Согласно отчету RedMonk за январь 2020 года, Python стал вторым по популярности языком программирования после Java Script. Ранее эту позицию на протяжении длительного времени уверенно удерживал Java, однако в начале года этот ЯП сместился на третью строчку рейтинга, который формируется на базе информации репозитория GitHub. Если быть точными, то пара Java Script и Java удерживали топ-2 популярности языков программирования с момента начала формирования указанного рейтинга, то есть с 2012 года. Этот язык программирования используется в самых разных целях, от веб-разработки до работы с данными и DevOps. Также стоит отметить, что в последнее время становятся все более распространенными приложения, разработанные на Python. Наиболее часто Python применяют в сферах анализа данных, машинного обучения и научных исследованиях. Быстрее всего растет популярность отдельных библиотек, например, pandas, numpy, matplotlib, seaborn, Scikit-Learn, BeautifulSoup, Requests, Tensorflow и Keras.

Для работы с генетическими алгоритмами так же есть несколько на Python, например, GAFT, Pyevolve и PyGMO. При выполнении лабораторных работ по курсу рекомендовано использовать DEAP, поскольку он прост в использовании и предлагает широкий набор функций, поддерживает расширяемость и может похвастаться подробной документацией.

DEAP был разработан в канадском университете Лавалья в 2009 г. и предлагается на условиях лицензии GNU Lesser General Public License (LGPL). Исходный код DEAP доступен по адресу <https://github.com/DEAP/deap>, а документация размещена по адресу <https://deap.readthedocs.io/en/master/>.

В методических указаниях приведены темы, необходимый теоретический и иллюстративный материал для выполнения лабораторных работ.

## Контрольная работа №1

Тема: решения задачи оптимизации многомерной функции эволюционными стратегиями (ЭС).

Цель: изучение основных принципов ЭС; научиться использовать у ЭС для оптимизации многомерной функции.

### 1.1. Общие сведения

Эволюционные стратегии (ЭС) основаны на эволюции популяции потенциальных решений, но, в отличие от большинства других парадигм, здесь используются генетические операторы на уровне фенотипа, а не генотипа, как это делается в например в ГА. 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 эволюционных стратегиях целью является движение особей популяции по направлению к лучшей области ландшафта фитнес-функции.

ЭС разработаны для решения многомерных оптимизационных задач, где пространство поиска – многомерное пространство вещественных чисел.

Иногда при решении задачи накладываются некоторые ограничения.

В ЭС 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) решения.

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



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

где  $N(0, \bar{\sigma})$  - вектор независимых случайных чисел Гаусса с нулевым средним значением и стандартным отклонением  $\bar{\sigma}$ .

## 1.2. Двукратная эволюционная стратегия

Здесь потомок принимается в качестве нового члена популяции (он заменяет своего родителя), если значение фитнес функции 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 на конкретном примере следующей функции:

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

$$-3.0 \leq x_1 \leq 12.1 \quad \bar{x} = (x_1, x_2)$$

$$4.1 \leq x_2 \leq 5.8 \quad \bar{\sigma} = (\sigma_1, \sigma_2),$$

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

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

Тогда потомок определяется следующим образом:

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

Поскольку (значение фитнес функции потомка лучше, чем у родителя), то полученный потомок заменяет родителя.

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

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:  $\{X_{1min}, X_{1max}\}, \{X_{2min}, X_{2max}\}, \dots, \{X_{Pmin}, X_{Pmax}\}$ , 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).

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

### 1.3. Многократная эволюционная стратегия

Отличия многократной ЭС:

- все особи в поколении имеют одинаковую вероятность выбора для мутации;
- имеется возможность введения оператора рекомбинации (типа – однородного ОК в ГА), где два случайно выбранных родителя производят потомка по следующей схеме (рис. 5.1):

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

Рис. 5.1. Оператор кроссинговера

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 и типы ЭС:

- $(1+1)$  – 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.

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

В кратных стратегиях часто используется двухуровневое обучение, где параметр  $\sigma$  не является постоянным, но изменяется по некоторому детерминированному алгоритму.

#### 1.4. Укрупненный алгоритм многократной ЭС:

```

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**

В  $(m, \lambda)$  алгоритме родители просто замещаются в следующем поколении потомками, в алгоритме  $(m + \lambda)$  в следующее поколение попадают и  $m$  родителей, и  $\lambda$  потомков. Таким образом, родители конкурируют с потомками и во всех поколениях после начального размер популяции равен  $m + \lambda$ .

### 1.5. Порядок выполнения лабораторной работы

1. Разработать ЭС нахождения оптимума согласно варианта в приложении А.
2. Выполнить программную реализацию разработанного алгоритма на языке высокого уровня. Рекомендованный язык Python. Предусмотреть возможность просмотра процесса поиска решения.
3. Для  $n=2$  вывести на экран график данной функции с указанием найденного экстремума. Предусмотреть возможность пошагового просмотра процесса поиска решения (с выводом всех точек популяции и лучшего решения другим цветом).
4. Исследовать зависимость числа поколений (генераций), точности нахождения решения от основных параметров ЭС.

### 1.6. 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);
6. Диаграммы исследованных зависимостей.

## Контрольная работа №2

Тема: решения задачи оптимизации многомерной функции роевыми алгоритмами (РА).

Цель: изучение основных принципов РА; научиться использовать РА для оптимизации многомерной функции.

### 2.1. Общие сведения

Роевые алгоритмы (РА), также, как и эволюционные, используют популяцию особей – потенциальных решений проблемы и метод стохастической оптимизации, который навеян (моделирует) социальным поведением птиц или рыб в стае или насекомых в рое. Аналогично эволюционным алгоритмам здесь начальная популяция потенциальных решений также генерируется случайным образом и далее ищется (суб)оптимальное решение проблемы в процессе выполнения РА. Первоначально в РА предпринята попытка моделировать поведение стаи птиц, которая обладает способностью порой внезапно и синхронно перегруппироваться и изменять направление полета при выполнении некоторой задачи. В отличие от ГА здесь не используются генетические операторы, в РА особи (называемые частицами - particles) летают в процессе поиска в гиперпространстве поиска решений и учитывают успехи своих соседей. Если одна частица видит хороший (перспективный) путь (в поисках пищи или защиты от хищников), то остальные частицы способны быстро последовать за ней, даже если они находились в другом конце роя.

### 2.2. 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  $u_i$ , 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:

$$x_i(t + 1) = x_i(t) + v_i(t + 1) \quad (2.1)$$

где  $x_i(0) \sim (x_{min}, x_{max})$ .

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

$$v_{ij}(t + 1) = v_{ij}(t) + c_1 r_1(t) [y_{ij}(t) - x_{ij}(t)] + c_2 r_2(t) [\hat{y}_j(t) - x_{ij}(t)] \quad (2.2)$$

Здесь:  $v_{ij}(t)$  –  $j$ -ая компонента скорости ( $j = 1, \dots, n_x$ )  $i$ -ой 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-ой 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:

$$y_i = \begin{cases} y_i(t) & \text{if } f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & \text{if } f(x_i(t+1)) < f(y_i(t)) \end{cases} \quad (2.3)$$

где  $f: R^{n_\infty} \rightarrow R$  - фитнес-функция.

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

Глобальная лучшая позиция в момент t определяется в соответствии с

$$\hat{y}_j(t) \in \{y_0(t), \dots, y_{n_s}(t)\} | f(\hat{y}_j(t)) = \min \{f(y_0(t)), \dots, f(y_{n_s}(t))\} \quad (2.4)$$

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.

### 2.3. Глобальный роевой алгоритм

Создание инициализация  $n_x$ -мерного роя;

```
repeat
for каждой частицы i=1,...,ns do
// определить персональную лучшую позицию
If f(xi) < f(yi) then
yi = xi;
end
// определить глобальную лучшую позицию
```

```

if  $f(y_i) < f(\hat{y})$  then  $(\hat{y}) = y_i$ ;
end
end
for каждой частицы  $i=1, \dots, ns$  do
коррекция скорости согласно (6.2);
коррекция позиции согласно (6.1);
end
until критерий останова не выполнен;

```

Рассмотрим влияние различных составляющих при вычислении скорости частицы в соответствии с (6.2).

Первое слагаемое в (6.2)  $v_i(t)$  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 роем (или его некоторым ближайшим окружением) глобального оптимума.

Графически это наглядно иллюстрируется для двумерного случая, как это показано на рис. 6.1.

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.



Рисунок 2.1 – Эффект социальной компоненты.

#### 2.4. Локальный роевой алгоритм

Локальный РА (Local Best PSO) использует для коррекции вектора скорости частицы только локальный оптимум, который определяется на множестве соседних (ближайших в некотором смысле) частиц.

То есть считается, что данной частице может передавать полезную информацию только ее ближайшее окружение.

При этом отношение соседства задается некоторой «социальной» сетевой структурой, которая образует перекрывающиеся множества соседних частиц, которые могут влиять друг на друга.

Соседние частицы обмениваются между собой информацией о достигнутых лучших результатах и поэтому стремятся двигаться в сторону локального в данной окрестности оптимума.

Характеристики роевого локального алгоритма сильно зависят от структуры используемой «социальной сети».

Поток информации через «социальную сеть» зависит от:

- 1) степени связности узлов сети,
- 2) числа кластеров,
- 3) среднего расстояния между узлами сети.

В сильно связанной социальной сети большинство частиц могут сообщаться друг с другом, что способствует быстрому распространению информации о достигнутых оптимумах и вследствие этого высокой скорости сходимости процесса поиска решения в отличие от мало связанных сетей.

Однако это часто достигается ценой преждевременной сходимости к локальным экстремумам.

С другой стороны, для мало связанных сетей с большим числом кластеров возможна ситуация, когда пространство поиска покрывается неудовлетворительно, вследствие чего трудно получить глобальное оптимальное решение.

Каждый кластер содержит сильно связанные особи и покрывает только часть пространства поиска.

Сетевая структура обычно содержит несколько кластеров, которые слабо связаны между собой.

Следовательно, исследуется информация только в ограниченной части пространства поиска.

В РА используются различные социальные структуры, типовые сетевые структуры которых представлены на рис.2.2.

На рис. 2.2 а) представлена структура «звезда», где все частицы связаны друг с другом (образуют полный граф) и могут соответственно обмениваться информацией.

В этом случае каждая частица стремится сместиться в сторону глобальной лучшей позиции, которую нашел рой.

Очевидно, что основной РА, рассмотренный в предыдущем разделе, использует по умолчанию фактически структуру «звезда» на всем рое. Исследования показывают, что РА на этой структуре имеет лучшую сходимость, но и имеет тенденцию попадать в ловушки локальных экстремумов. Поэтому данную структуру можно рекомендовать, прежде всего, для унимодальных задач с одним экстремумом.

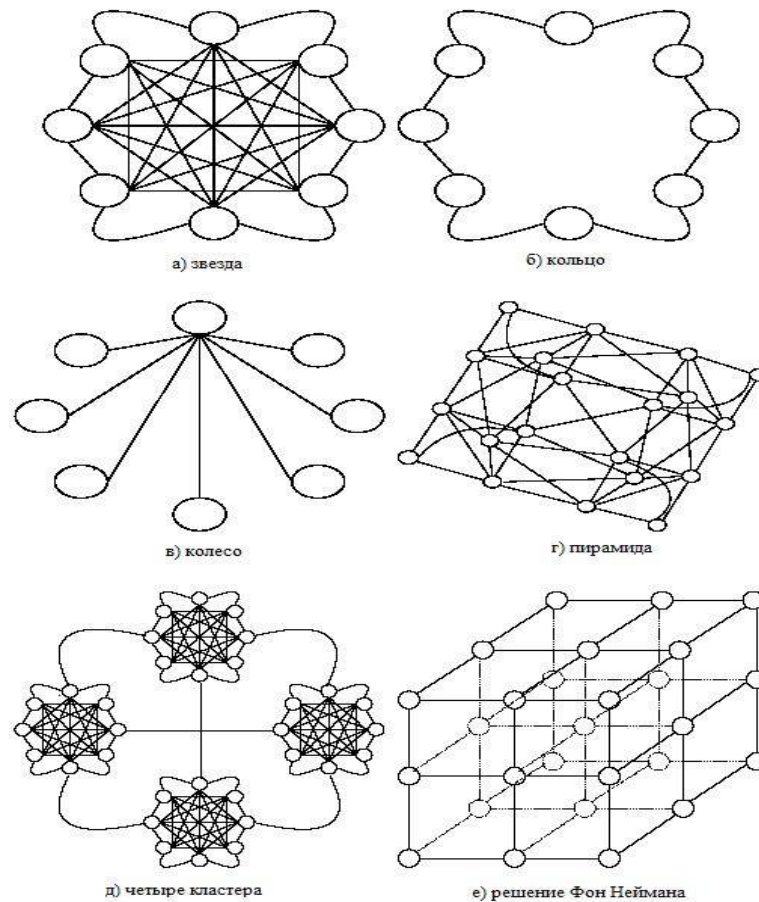


Рисунок 2.2 – Типовые сетевые структуры

На следующем рис. 2.6 б) приведена сетевая структура «кольцо», где каждая частица общается со своими  $n_N$  ближайшими соседями. При  $n_N=2$  каждая частица связана только с двумя ближайшими соседями по кольцу, как это показано на рисунке. В этом случае каждая частица пытается сместиться в сторону лучшего соседа. Следует отметить, что множества соседних окружений перекрываются и вследствие этого возможен обмен информацией не только между ближайшими соседями. Поэтому данная структура позволяет

находить и глобальный экстремум, но с меньшей скоростью. Данная структура лучше зарекомендовала себя при решении мульти-модальных задач (со многими экстремумами).

В сетевой структуре «колесо», показанной на рис. 2.2 в), особи фактически изолированы друг от друга. Только одна частица выступает в качестве «фокальной точки», через которую идет обмен информации. В этом случае «фокальная частица» сравнивает характеристики всех соседних частиц и стремится в сторону лучшего соседа. Если новая позиция «фокальной частицы» имеет лучшие характеристики, то она сообщает это всем своим соседям. Данная сетевая структура замедляет распространение хороших решений через рой.

Социальная структура «пирамида» образует трехмерный каркас, как это показано на рис. 2.2 г).

Далее на рис. 2.2 д) представлена четырех-кластерная социальная структура, в которой четыре кластера (клики) связаны друг с другом двумя соединениями.

Наконец, на рис. 2.2 е) приведена социальная структура, где частицы объединены в решетку, которая часто называется социальной сетевой структурой фон Неймана.

Данный тип согласно проведенным экспериментальным исследованиям показывает лучшие результаты при решении многих задач.

Следует отметить, что нет единого рецепта по использованию некоторой сетевой структуры. Для различных задач эффективными могут быть различные сетевые структуры, определяющие отношение соседства. В целом полно связная структура (звезда) дает лучшие результаты для унимодальных задач, в то время, как слабо связные структуры предпочтительнее для мульти модальных задач. При коррекции скорости частицы в локальных РА вклад данной частицы пропорционален расстоянию между ней и лучшей позицией своего окружения, которое задается одной из рассмотренных сетевых структур.

Таким образом, скорость частицы вычисляется следующим образом:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(y(t) - x_{ij}(t) + c_2 r_{2j}(t)[\widehat{y}_{lj}(t) - x_{ij}(t)] \quad (2.5)$$

где  $\widehat{y}_{lj}$ - лучшая позиция, которая найдена по координате  $j$  соседями частицы  $i$ . При этом локальная лучшая позиция  $\widehat{y}_{lj}$  определяется как лучшая позиция в окружении  $N_i$  в соответствии с выражением.

$$\widehat{y}_i(t+1) \in \{N_i | f(\widehat{y}_i(t+1)) = \min\{f(x)\}, \forall x \in N_i\} \quad (2.6)$$

где

$$N_i = \{y_{i-nN_i}(t), y_{i-nN_i+1}(t), \dots, y_{i-1}(t), y_i(t), y_{i+1}(t), \dots, y_{i+nN_i}(t)\} \quad (2.7)$$

при числе соседей  $nN_i$ .

Здесь локальная лучшая позиция относится к лучшей позиции соседнего окружения.

## 2.5. Локальный роевой алгоритм

*Создание инициализация nx-мерного роя;*

*repeat*

*for* каждой частицы  $i=1, \dots, ns$  *do*

*//определить персональную лучшую позицию*

*If*  $f(x_i) < f(y_i)$  *then*

$y_i = x_i$ ;

*end*

*//определить лучшую позицию окружения*

*If*  $f(y_i) < f(\widehat{y}_i)$  *then*  $f(\widehat{y}_i) = y_i$ ;

*end*

*end*

*for* каждой частицы  $i=1, \dots, ns$  *do*

*коррекция скорости согласно (6.6);*

*коррекция позиции согласно (6.1);*

*until* критерий останова не выполнен;

Существуют, по крайней мере, два основных различия между этими двумя подходами относительно их характеристик (свойств) сходимости:

1. благодаря большему взаимодействию частиц в глобальном РА он сходится быстрее, чем локальный РА. Однако эта быстрая сходимость достигается ценой сужения пространства поиска.
2. вследствие большего разнообразия потенциальных решений локальный РА менее подвержен преждевременной сходимости к локальным экстремумам. Часто сетевые социальные структуры (например, такие как «кольцо») позволяют улучшить характеристики РА для многих задач.

## 2.6. Реализация оптимизации методом роя частиц

Основные части программы.

1. Для начала зададим значения различных констант. Размерность задачи в нашем случае равна для начала 2, а она, в свою очередь, определяет размерность положения и скорости частиц. Следующая константа – размер популяции, т. е. количество частиц в рое, и, наконец, количество поколений, или итераций алгоритма.

```
DIMENSIONS = 2
```

```
POPULATION_SIZE = 20
```

```
MAX_GENERATIONS = 500
```

2. Далее зададим несколько дополнительных констант, влияющих на порядок создания и обновления частиц.

```
MIN_START_POSITION, MAX_START_POSITION = -5, 5
```

```
MIN_SPEED, MAX_SPEED = -3, 3
```

```
MAX_LOCAL_UPDATE_FACTOR = MAX_GLOBAL_UPDATE_FACTOR = 2.0
```

3. Будем стремимся найти минимум функции определим единственную цель – минимизирующую стратегию приспособления:

```
creator.create("FitnessMin", base.Fitness,
weights=(-1.0,))
```

4. Теперь создадим класс `Particle`. Поскольку этот класс представляет положение в непрерывном пространстве, мы могли бы унаследовать его от обычного списка чисел с плавающей точкой. Однако здесь мы решили воспользоваться N-мерным массивом (`ndarray`) из библиотеки `numpy`, так как он особенно удобен для выполнения поэлементных алгебраических операций, в частности сложения и умножения, которые понадобятся при обновлении положения частицы. Помимо текущего положения, методу создания объекта класса `Particle` передается несколько дополнительных атрибутов:

- `fitness` – определенная выше минимизирующая стратегия приспособления;
- `speed` – здесь будут храниться компоненты текущего вектора скорости частицы. Начальное значение равно `None`, но позже скорость будет инициализирована с помощью другого массива `ndarray`;
- `best` – представляет лучшее найденное на данный момент положение этой частицы (локально лучшее).

В итоге определение метода создания объекта класса `Particle` выглядит следующим образом:

```
creator.create("Particle", np.ndarray,
               fitness=creator.FitnessMin, speed=None, best=None)
```

5. Для конструирования одной частицы нам понадобится вспомогательная функция, которая создает частицу и инициализирует ее случайным образом. Мы воспользуемся функцией `random.uniform()` из библиотеки `numpy`, чтобы сгенерировать случайное положение и скорость частицы в заданных пределах:

```
def createParticle():
    particle = creator.Particle(np.random.uniform(
MIN_START_POSITION, MAX_START_POSITION, DIMENSIONS))
    particle.speed = np.random.uniform(MIN_SPEED,
MAX_SPEED, DIMENSIONS)
    return particle
```

6. Эта функция используется в определении оператора, который создает экземпляр частицы, а он, в свою очередь, применяется в операторе создания популяции:

```
toolbox.register("particleCreator", createParticle)
toolbox.register("populationCreator",
tools.initRepeat, list,
toolbox.particleCreator)
```

7. Далее определим сердце алгоритма – метод `updateParticle()`, отвечающий за обновление положения и скорости каждой частицы в популяции. Ему передается одна частица и лучшее найденное на данный момент положение.

Первым делом метод создает два случайных множителя – для локального и глобального обновлений – в заранее заданном диапазоне. Затем он вычисляет два обновления скорости (локальное и глобальное) и прибавляет их к текущей скорости частицы.

Заметим, что все участвующие в вычислениях значения имеют тип массива `ndarray`, в нашем случае двумерного, а вычисления производятся поэлементно.

Обновленная скорость частицы – это комбинация ее исходной скорости (представляющей инерцию), ее лучшего известного положения (когнитивная сила) и лучшего известного положения частицы во всей популяции (социальная сила):

```
def updateParticle(particle, best):
    localUpdateFactor =
np.random.uniform(0,MAX_LOCAL_UPDATE_FACTOR,particle.size)
    globalUpdateFactor =
np.random.uniform(0,MAX_GLOBAL_UPDATE_FACTOR,particle.size)
```



```

        localSpeedUpdate      =      localUpdateFactor      *
        (particle.best - particle)
        globalSpeedUpdate = globalUpdateFactor * (best -
particle)
        particle.speed = particle.speed + (localSpeedUpdate
+ llobalSpeedUpdate)

```

8. Метод `updateParticle()` далее проверяет, что новая скорость не выходит за установленные пределы, и обновляет положение частицы с учетом пересчитанной скорости. Выше мы отмечали, что положение и скорость имеют тип `ndarray`, содержащий по одному элементу в каждом измерении:

```

        particle.speed = np.clip(particle.speed, MIN_SPEED,
MAX_SPEED)
        particle[:] = particle + particle.speed

```

9. Теперь регистрируем метод `updateParticle()` как оператор инструментария, который будет впоследствии использоваться в главном цикле:

```

toolbox.register("update", updateParticle)

```

10. Нам еще нужно определить подлежащую оптимизации функцию зарегистрировать ее как оператор вычисления приспособленности:

```

def himmelblau(particle):
    x = particle[0]
    y = particle[1]
    f = (x ** 2 + y - 11) ** 2 + (x + y ** 2 - 7) ** 2
    return f, # return a tuple
toolbox.register("evaluate", himmelblau)

```

11. Вот мы, наконец, можем начать с создания популяции частиц:

```

population
=
toolbox.populationCreator(n=POPULATION_SIZE)

```

12. Прежде чем входить в главный цикл, нужно создать объект `stats`, необходимый для вычисления статистики, и объект `logbook`, в котором будет сохраняться статистика на каждой итерации:

```
stats = tools.Statistics(lambda ind:
ind.fitness.values)
stats.register("min", np.min)
stats.register("avg", np.mean)
logbook = tools.Logbook()
logbook.header = ["gen", "evals"] + stats.fields
```

13. Главный цикл программы содержит внешний цикл по поколениям, в котором производится обновление частиц. На каждой итерации имеется два вспомогательных цикла, в каждом из которых обходятся все частицы в популяции. В первом цикле, показанном ниже, к каждой частице применяется оптимизируемая функция и при необходимости обновляются локально и глобально лучшие частицы:

```
particle.fitness.values =
toolbox.evaluate(particle)
# локально лучшая
if particle.best is None or particle.best.size == 0
or
particle.best.fitness < particle.fitness:
particle.best = creator.Particle(particle)
particle.best.fitness.values =
particle.fitness.values
# глобально лучшая
if best is None or best.size == 0 or best.fitness <
particle.fitness:
best = creator.Particle(particle)
best.fitness.values = particle.fitness.values
```

14. Во втором внутреннем цикле вызывается оператор `update`. Ранее мы видели, что этот оператор обновляет скорость и положение частицы на основе комбинации инерции, когнитивной силы и социальной силы:

```
toolbox.update(particle, best)
```

15. В конце внешнего цикла мы сохраняем и печатаем статистику для текущего поколения:

```
logbook.record(gen=generation,
evals=len(population),
**stats.compile(population))
print(logbook.stream)
```

16. По завершении внешнего цикла печатается информация о лучшем положении, найденном за все время работы алгоритма. Оно и считается решением задачи:

```
# печать информации о лучшем найденном решении
print("-- Лучшая частица = ", best)
print("-- Лучшая приспособленность = ",
best.fitness.values[0])
```

Программа печатает следующий результат:

```
gen evals min avg
0 20 8.74399 167.468
1 20 19.0871 357.577
2 20 32.4961 219.132
...
...
...
497 20 7.2162 412.189
498 20 6.87945 273.712
499 20 16.1034 272.385
-- Лучшая частица = [-3.77695478 -3.28649153]
-- Лучшая приспособленность = 0.0010248367255068806
```

Как видим, алгоритм смог найти один минимум рядом с точкой  $x = -3.77$ ,  $y = -3.28$ . Статистика показывает, что лучший результат был достигнут в поколении 480. Также видно, что частицы колеблются, то приближаясь к лучшему результату, то отдаляясь от него.

Для нахождения других минимумов можно перезапустить алгоритм с иным начальным значением генератора случайных чисел.

Еще один подход – использовать несколько роев для нахождения всех минимумов в одном прогоне.

## 2.7. Порядок выполнения лабораторной работы

1. Изучить теоретический материал.
2. Разработать роевый алгоритм для нахождения оптимума заданной по варианту функции в контрольной работе №1. Для четных – локальный роевый алгоритм, для нечетных вариантов – глобальный.
3. Выполнить программную реализацию разработанного алгоритма на языке высокого уровня. Рекомендованный язык Python с применением DEAP. Предусмотреть возможность просмотра процесса поиска решения.
4. Исследовать зависимость числа поколений (генераций), точности нахождения решения от основных параметров роевого алгоритма.
5. Сравнить найденное решение с действительным и с найденным решением в лабораторной работе №2 и №5. Сделать выводы.

## 2.8. 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).

### Контрольная работа №3

Тема: решения задачи задачи поиска гамильтонова пути с помощью муравьиных алгоритмов (МА).

Цель: изучение основных принципов МА; научиться использовать МА для гамильтонова пути.

#### 3.1. Общие сведения

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.

#### 3.2. Простой муравьиный алгоритм

Рассмотрим простой муравьиный алгоритм (ПМА) (simple ant colony optimization – SACO), в котором фактически формализованы приведенные выше экспериментальные исследования и 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. 3.1. С каждой дугой, соединяющей вершины  $(i,j)$  ассоциируем концентрацию феромона  $\tau_{ij}$ .

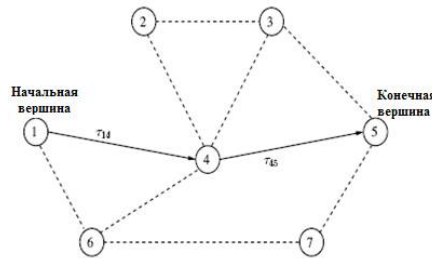


Рис. 3.1. Пример графа.

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

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  $i$ -ой вершине, то он выбирает следующую вершину  $j \in N_i^k$  на основе вероятностей перехода:

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha(t)}{\sum_{j \in N_i^k} \tau_{ij}^\alpha(t)}, & \text{если } j \in N_i^k \\ 0, & \text{если } j \notin N_i^k \end{cases} \quad (3.1)$$

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 (7.1)  $\alpha$  - 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:

$$\Delta\tau_{ij}^k(t) \propto \frac{1}{L^k(t)} \quad (3.2)$$

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:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t) \quad (3.3)$$

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 (3.2) 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 (3.2) 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 (3.3);
  end
  for каждого муравья  $k=1, \dots, n_k$  do
    for каждой дуги  $(i, j)$  пути  $x^k(t)$  do
      
$$\Delta\tau^k = \frac{1}{f(x^k(t))} ;$$

      Коррекция  $\tau_{ij}$  согласно (3.4);
    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:

$$\tau_{ij}(t) \leftarrow (1 - \rho)\tau_{ij}(t) \quad (3.4)$$

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.

### 3.3. Порядок выполнения лабораторной работы

1. Разработать МА для решение задачи поиска гамильтонова пути. Индивидуальное заданию выбирается по таблице Б.1 в приложении Б согласно номеру варианта.
2. Выполнить программную реализацию разработанного алгоритма на языке высокого уровня. Рекомендованный язык Python с применением DEAP. Предусмотреть возможность просмотра процесса поиска решения.
3. Представить графически найденное решение. Предусмотреть возможность пошагового просмотра процесса поиска решения.
4. Сравнить найденное решение с представленным в условии задачи оптимальным решением.

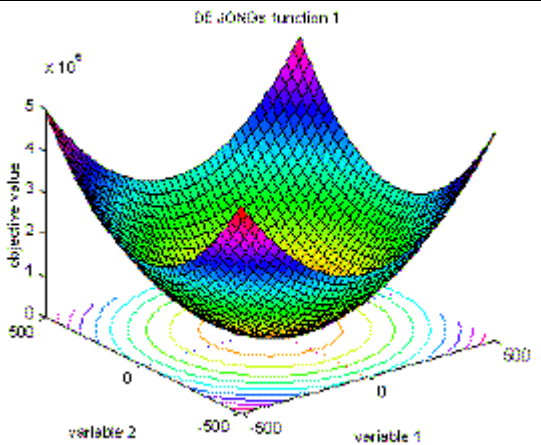
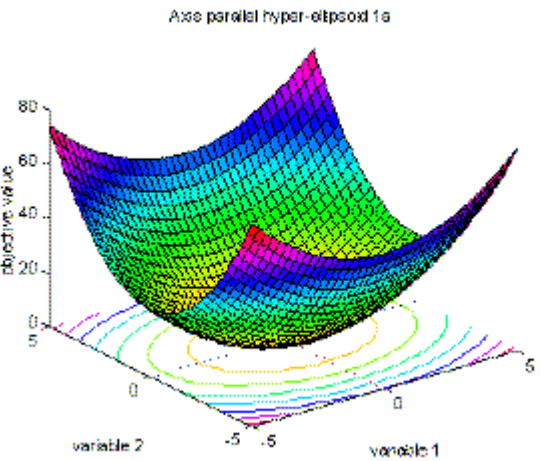
### 3.4. Содержание отчета

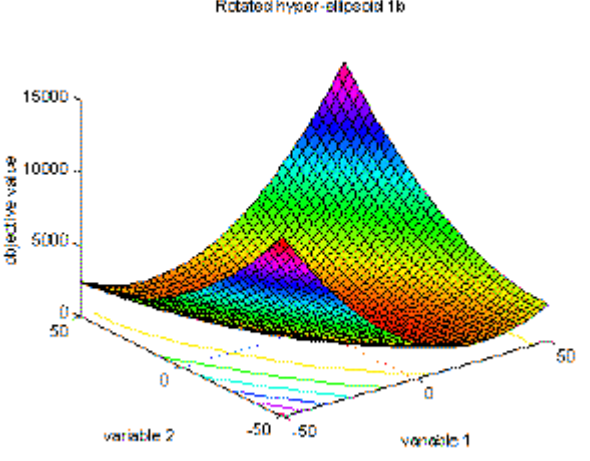
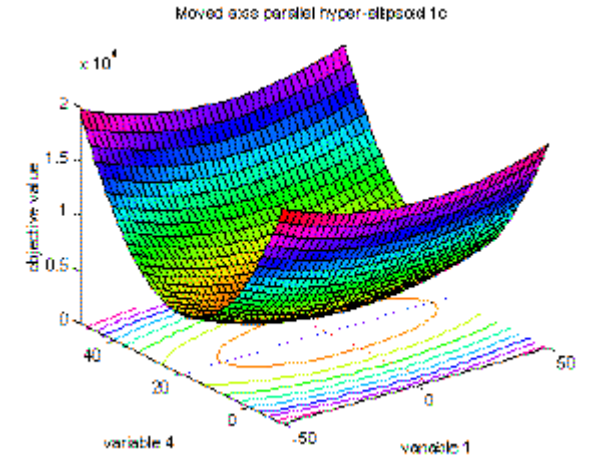
1. Титульный лист установленной формы.
2. Задание, учитывая свой вариант.
3. Краткие теоретические сведения.
4. Распечатанный листинг программы.
5. Распечатка результатов выполнения программы.

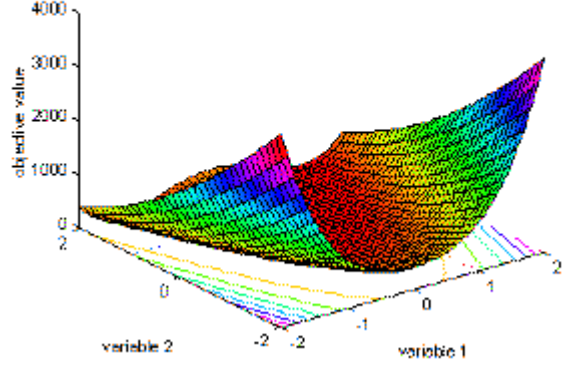
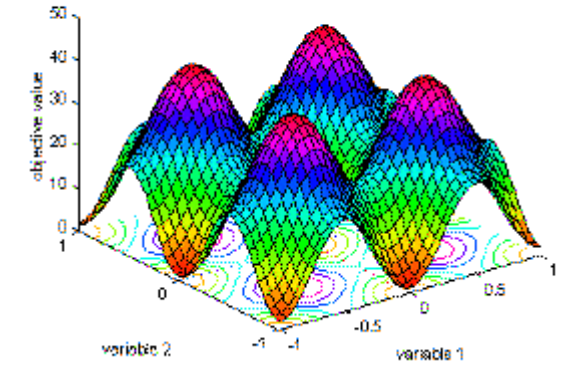
## СПИСОК ЛИТЕРАТУРЫ

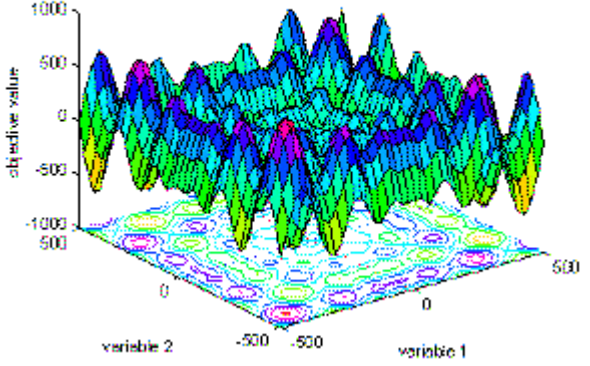
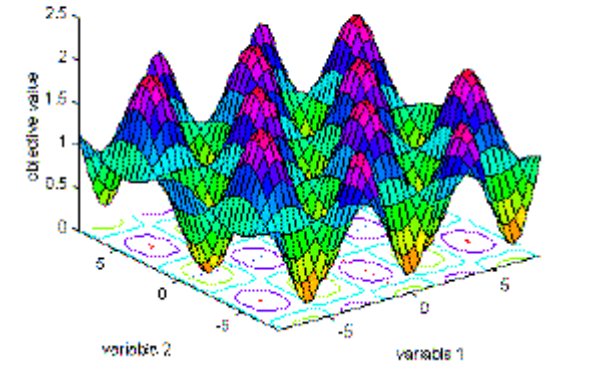
1. Барсегян А. А. Анализ данных и процессов: учебное пособие [Электронный ресурс] / А. А. Барсегян, М. С. Куприянов, И. И. Холод, М. Д. Тесс, С. И. Елизаров. – 3-е изд., перераб. и доп. – 9Мб. — СПб.: БХВ-Петербург, 2009. – 1 файл. – Систем. требования: Acrobat Reader.
3. Антонио Джулли. Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow [Электронный ресурс] Антонио Джулли, Суджит Пал / пер. с англ. Слинкин А.А. – М.: ДМК Пресс, 2018. -294 с.: ил. 1 файл. – Систем. требования: Acrobat Reader.
4. Жерон, Орельен. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем. [Электронный ресурс]. Пер. с англ. - СПб.: ООО «Альфа-книга»: 2018. - 688 с.: ил. 1 файл. – Систем. требования: Acrobat Reader.
5. Скобцов, Ю.А. Эволюционные вычисления [Электронный ресурс]: учебное пособие / Ю. А. Скобцов, Д. В. Сперанский; Ю.А. Скобцов, Д.В. Сперанский; Нац. Открытый Ун-т "ИНТУИТ". - 5 Мб. - М. : Нац. Откр. Ун-т "ИНТУИТ", 2015. - 1 файл. - (Основы информационных технологий). - Систем. требования: Acrobat Reader. <http://ed.donntu.org/books/cd3220.pdf>
6. Скобцов Ю.А. Метаэвристики [Электронный ресурс]: монография / Ю. А. Скобцов, Е.Е. Федоров; Ю.А. Скобцов, Е.Е. Федоров ; ГВУЗ "ДонНТУ", Донец. акад. автомоб. транспорта. - 13 Мб. - Донецк : Изд-во "Ноулидж". Донецк. отд-ние, 2013. - 1 файл. - Систем. требования: Acrobat Reader. - ISBN 978-5-89070-682-9. <http://ed.donntu.org/books/cd3217.pdf> 3.
7. Курейчик В.В. Теория эволюционных вычислений [Электронный ресурс]: монография / В. В. Курейчик, В. М. Курейчик, С. И. Родзин; В.В. Курейчик, В.М. Курейчик, С.И. Родзин. - 3 Мб. - Москва: Физматлит, 2012. - 1 файл. - Систем. требования: Просмотрщик djvu-файлов. <http://ed.donntu.org/books/17/cd8014.djvu> .

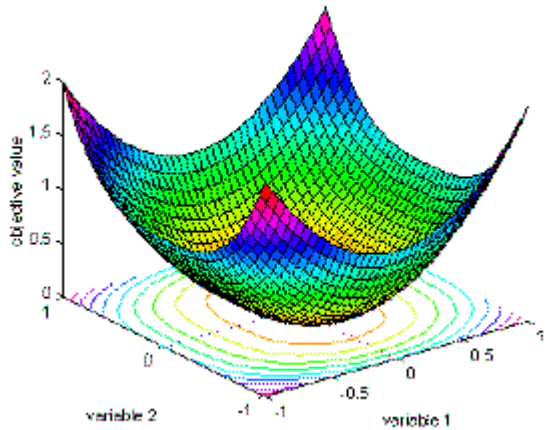
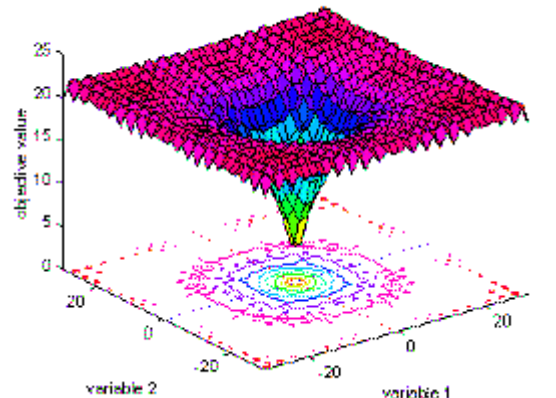
Таблица А.1. Индивидуальные задания на контрольную работу №1

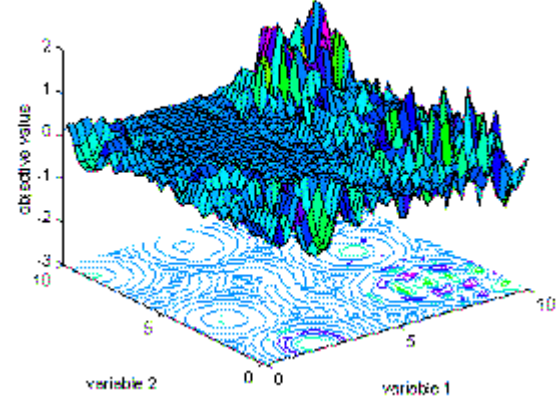
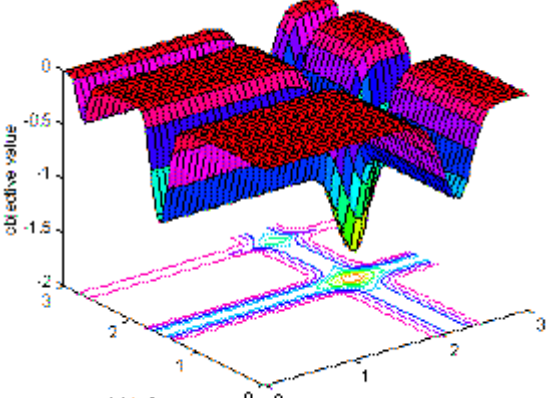
№ вв.	Название	Оптимум	Вид функции	График функции
1	De Jong's function 1	global minimum $f(x)=0$ ; $x(i)=0$ , $i=1:n$ .	$f_1(x) = \sum_{i=1}^n x_i^2 \quad -5.12 \leq x_i \leq 5.12$ $f_1(x) = \sum_{i=1}^n x(i)^2,$ $i=1:n;$	
2	Axis parallel hyper-ellipsoid function	global minimum $f(x)=0$ ; $x(i)=0$ , $i=1:n$ .	$f_{1a}(x) = \sum_{i=1}^n i \cdot x_i^2 \quad -5.12 \leq x_i \leq 5.12$ $f_{1a}(x) = \sum_{i=1}^n (i \cdot x(i)^2),$ $i=1:n;$	

3	Rotated hyper-ellipsoid function	global minimum $f(x)=0$ ; $x(i)=0$ , $i=1:n$	$f_{1b}(x) = \sum_{i=1}^n \left( \sum_{j=1}^i x_j \right)^2 \quad -65.536 \leq x_i \leq 65.536$ $f1b(x) = \text{sum}(\text{sum}(x(j)^2, j=1:i), i=1:n);$	<p>Rotated hyper-ellipsoid 1b</p> 
4	Moved axis parallel hyper-ellipsoid function	global minimum $f(x)=0$ ; $x(i)=5*i$ , $i=1:n$	$f_{1c}(x) = \sum_{i=1}^n 5i \cdot x_i^2 \quad -5.12 \leq x_i \leq 5.12$ $f1c(x) = \text{sum}(5*i*x(i)^2, i=1:n);$	<p>Moved axis parallel hyper-ellipsoid 1c</p> 

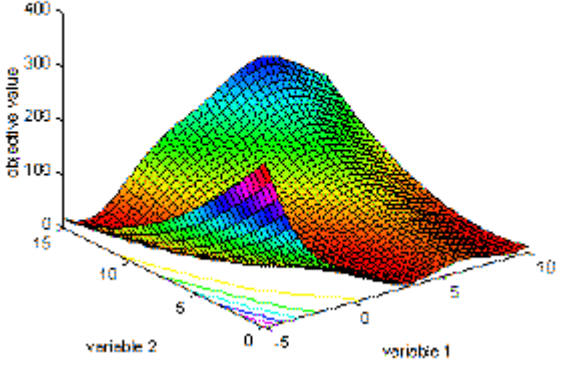
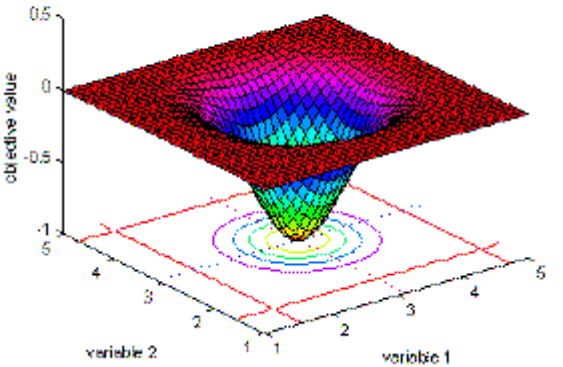
5	Rosenbrock's valley (De Jong's function 2)	global minimum $f(x)=0$ ; $x(i)=1$ , $i=1:n$ .	$f_2(x) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \quad -2.048 \leq x_i \leq 2.048$ $f2(x) = \text{sum}(100 \cdot (x(i+1) - x(i)^2)^2 + (1 - x(i))^2),$ $i=1:n-1;$	<p>ROSENBROCK's function 2</p> 
6	Rastrigin's function 6	global minimum $f(x)=0$ ; $x(i)=0$ , $i=1:n$ .	$f_6(x) = 10 \cdot n + \sum_{i=1}^n (x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i)) \quad -5.12 \leq x_i \leq 5.12$ $f6(x) = 10 \cdot n + \text{sum}(x(i)^2 - 10 \cdot \cos(2 \cdot \pi \cdot x(i))),$ $i=1:n;$	<p>RASTRIGIN's function 6</p> 

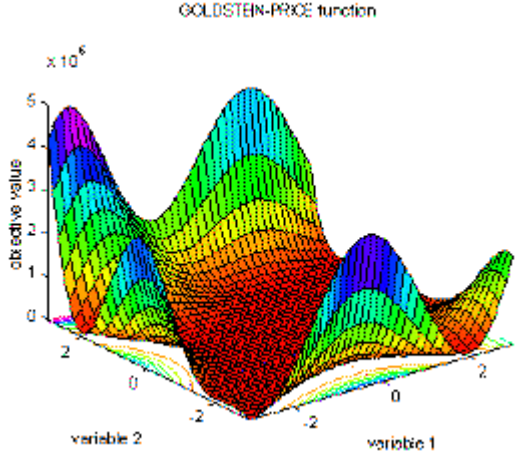
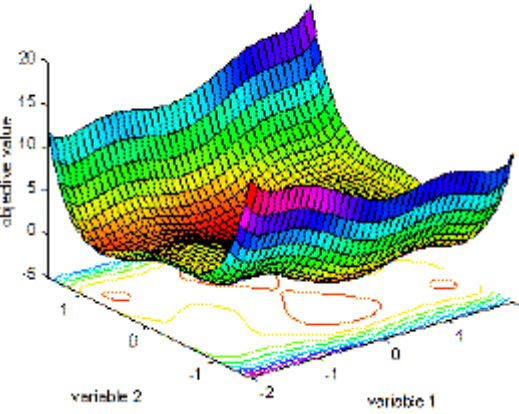
7	Schwefel's function 7	global minimum $f(x) = n \cdot 418.9829$ ; $x(i) = 420.9687$ , $i = 1:n$ .	$f_7(x) = \sum_{i=1}^n -x_i \cdot \sin(\sqrt{ x_i }) \quad -500 \leq x_i \leq 500$ $f7(x) = \sum_{i=1}^n (-x(i) \cdot \sin(\sqrt{\text{abs}(x(i))})),$ $i = 1:n;$	<p>SCHWEFEL's function 7</p> 
8	Griewangk's function 8	global minimum $f(x) = 0$ ; $x(i) = 0$ , $i = 1:n$	$f_8(x) = \sum_{i=1}^n \frac{x_i^2}{4000} - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad -600 \leq x_i \leq 600$ $f8(x) = \sum_{i=1}^n (x(i)^2/4000) - \text{prod}(\cos(x(i)/\sqrt{i}))+1,$ $i = 1:n;$	<p>GRIEWANGK's function 8</p> 

9	Sum of different power function 9	global minimum $f(x)=0$ ; $x(i)=0$ , $i=1:n$ .	$f_9(x) = \sum_{i=1}^n  x_i ^{(i+1)} \quad -1 \leq x_i \leq 1$ $f9(x) = \sum (abs(x(i))^{(i+1)}),$ $i=1:n;$	<p>Sum of different power function 9</p> 
10	Ackley's Path function 10	global minimum $f(x)=0$ ; $x(i)=0$ , $i=1:n$ .	$f_{10}(x) = -a \cdot e^{-b \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} - \frac{\sum_{i=1}^n \cos(c \cdot x_i)}{n} + a + e^1} \quad -1 \leq x_i \leq 1$ $f10(x) = -a \cdot \exp(-b \cdot \sqrt{1/n \cdot \sum (x(i)^2)}) - \exp(1/n \cdot \sum (\cos(c \cdot x(i)))) + a + \exp(1);$ $a=20; b=0.2; c=2 \cdot \pi; i=1:n;$	<p>ACKLEY's PATH function 10</p> 

11	Langermann's function 11	<p>global minimum  <math>f(x)=-1.4</math> (for  <math>m=5</math>); <math>x(i)=???</math>,  <math>i=1:n</math>.</p>	$f_{11}(x) = -\sum_{i=1}^m c_i \left( e^{-\frac{\ x-A(i)\ ^2}{\pi}} \cdot \cos(\pi \cdot \ x-A(i)\ ^2) \right) \quad i = 1:m, 2 \leq m \leq 10, 0 \leq x_i \leq 10$ <p>fl1(x)=-sum(c(i)·(exp(-1/pi·sum((x-A(i))^2))·cos(pi·sum((x-A(i))^2)))),  <math>i=1:m, m=5; A(i),C(i)&gt;0, m=5</math></p>	<p>LANGERMANN's function 11</p> 
12	Michalewicz's function 12	<p>global minimum  <math>f(x)=-4.687</math>  <math>(n=5)</math>; <math>x(i)=???</math>,  <math>i=1:n</math>.  <math>f(x)=-9.66</math>  <math>(n=10)</math>; <math>x(i)=???</math>,  <math>i=1:n</math>.</p>	$f_{12}(x) = -\sum_{i=1}^n \sin(x_i) \cdot \left( \sin\left(\frac{i \cdot x_i^2}{\pi}\right) \right)^{2 \cdot m} \quad i = 1:n, m = 10, 0 \leq x_i \leq \pi$ <p>fl2(x)=-sum(sin(x(i))·(sin(i·x(i)^2/pi))^(2·m)),  <math>i=1:n, m=10</math>; проверить для <math>n=5,10</math></p>	<p>MICHALEWICZ's function 12</p> 

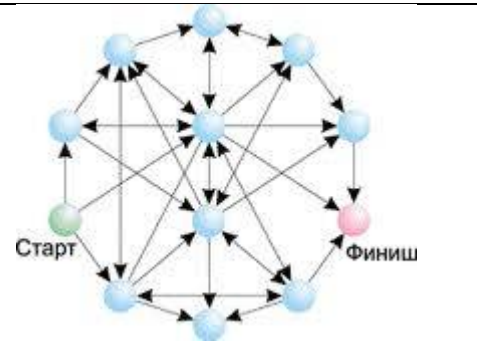
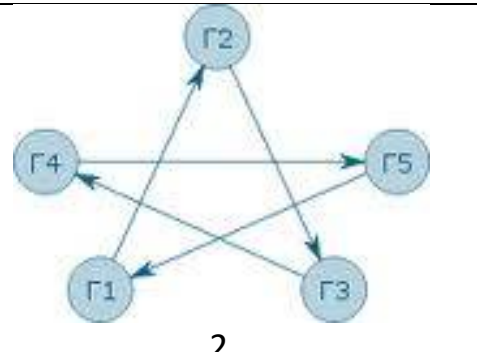
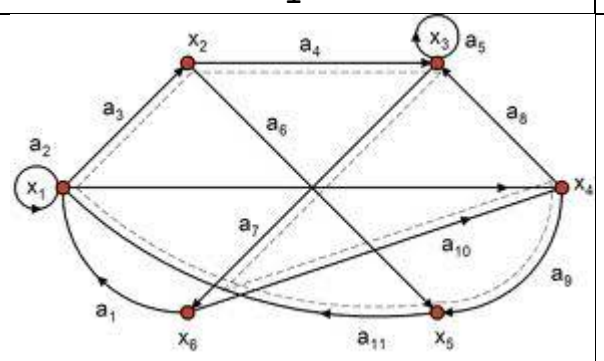
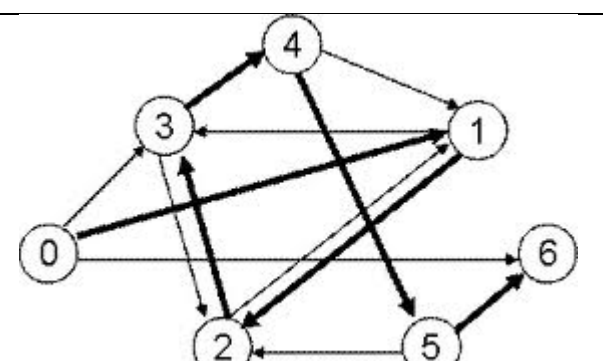
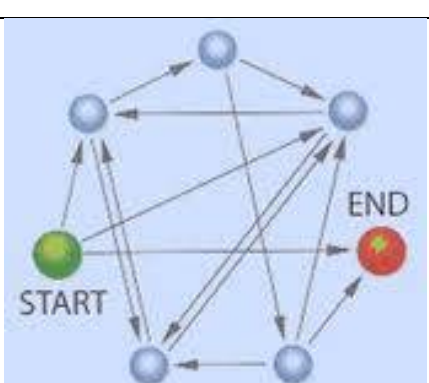
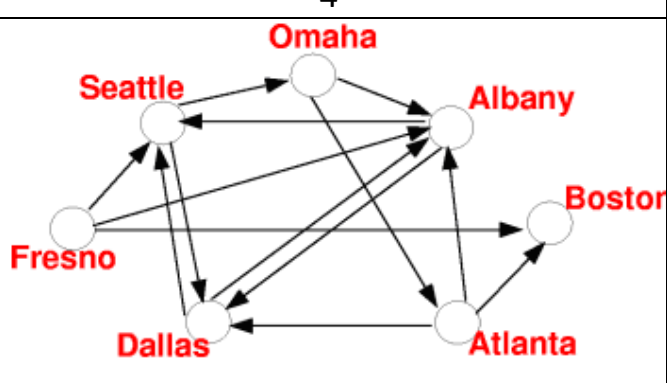
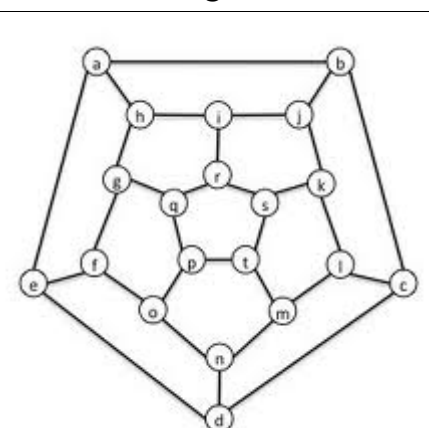
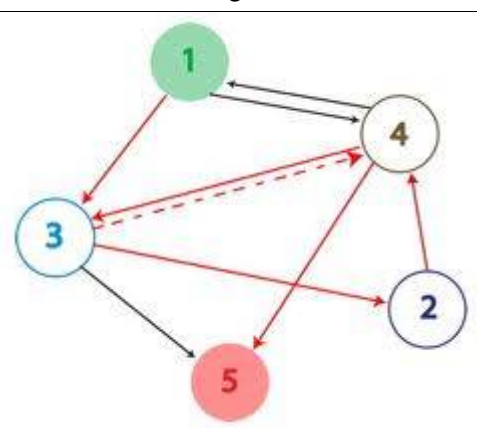


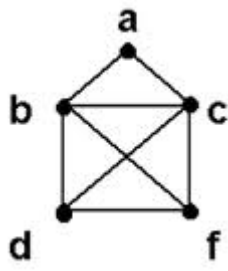
13	Branin's rcos function	<p>global minimum  <math>f(x_1, x_2) = 0.397887</math>;  <math>(x_1, x_2) = (-\pi, 12.275)</math>,  <math>(\pi, 2.275)</math>,  <math>(9.42478, 2.475)</math>.</p>	$f_{Bran}(x_1, x_2) = a \cdot (x_2 - b \cdot x_1^2 + c \cdot x_1 - d)^2 + e \cdot (1 - f) \cdot \cos(x_1) + e \quad -5 \leq x_1 \leq 10, 0 \leq x_2 \leq 15$ $a = 1, \quad b = \frac{5.1}{4 \cdot \pi^2}, \quad c = \frac{5}{\pi}, \quad d = 6, \quad e = 10, \quad f = \frac{1}{8 \cdot \pi}$ $f_{Bran}(x_1, x_2) = a \cdot (x_2 - b \cdot x_1^2 + c \cdot x_1 - d)^2 + e \cdot (1 - f) \cdot \cos(x_1) + e;$ $a = 1, \quad b = 5.1 / (4 \cdot \pi^2), \quad c = 5 / \pi, \quad d = 6, \quad e = 10, \quad f = 1 / (8 \cdot \pi);$	<p>BRANIN's Rcos function</p> 
14	Easom's function	<p>global minimum  <math>f(x_1, x_2) = -1</math>;  <math>(x_1, x_2) = (\pi, \pi)</math>.</p>	$f_{Easo}(x_1, x_2) = -\cos(x_1) \cdot \cos(x_2) \cdot e^{-((x_1 - \pi)^2 + (x_2 - \pi)^2)} \quad -100 \leq x_i \leq 100, i = 1:2$ $f_{Easo}(x_1, x_2) = -\cos(x_1) \cdot \cos(x_2) \cdot \exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2));$	<p>EASOM's function</p> 

15	Goldstein-Price's function	global minimum $f(x_1, x_2) = 3$ ; $(x_1, x_2) = (0, -1)$ .	$f_{Gold}(x_1, x_2) = (1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \cdot (30 + (2x_1 - 3x_2)^2 \cdot (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2))$ $-2 \leq x_i \leq 2, i = 1: 2$ $f_{Gold}(x_1, x_2) = [1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14 \cdot x_1 + 3 \cdot x_1^2 - 14 \cdot x_2 + 6 \cdot x_1 \cdot x_2 + 3 \cdot x_2^2)] \cdot [30 + (2 \cdot x_1 - 3 \cdot x_2)^2 \cdot (18 - 32 \cdot x_1 + 12 \cdot x_1^2 + 48 \cdot x_2 - 36 \cdot x_1 \cdot x_2 + 27 \cdot x_2^2)]$	
16	Six-hump camel back function	global minimum $f(x_1, x_2) = -1.0316$ ; $(x_1, x_2) = (-0.0898, 0.7126)$ , $(0.0898, -0.7126)$ .	$f_{Sixh}(x_1, x_2) = (4 - 2.1x_1^2 + x_1^{4/3}) \cdot x_1^2 + x_1x_2 + (-4 + 4x_2^2) \cdot x_2^2 \quad -3 \leq x_1 \leq 3, -2 \leq x_2 \leq 2$ $f_{Sixh}(x_1, x_2) = (4 - 2.1 \cdot x_1^2 + x_1^{4/3}) \cdot x_1^2 + x_1 \cdot x_2 + (-4 + 4 \cdot x_2^2) \cdot x_2^2$	

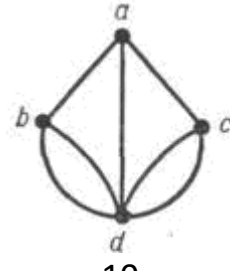
Для остальных вариантов берется строчка, соответствующая остатку от деления номера варианта на 16 (для 17-1, 18-2 и т.д.)

Таблица Б.1. Индивидуальные задания на контрольную работу №3.

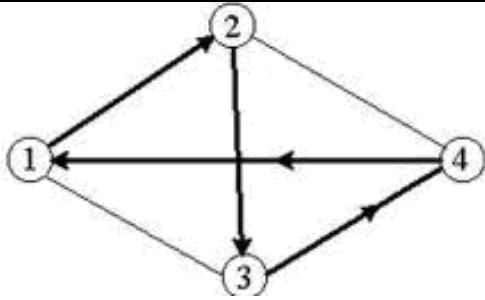
 <p>1</p>	 <p>2</p>
 <p>3</p>	 <p>4</p>
 <p>5</p>	 <p>6</p>
 <p>7</p>	 <p>8</p>



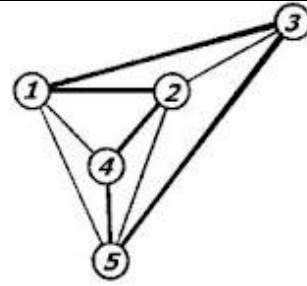
9



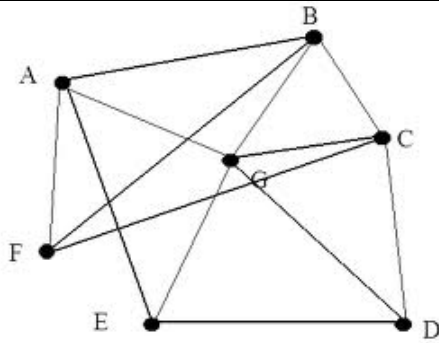
10



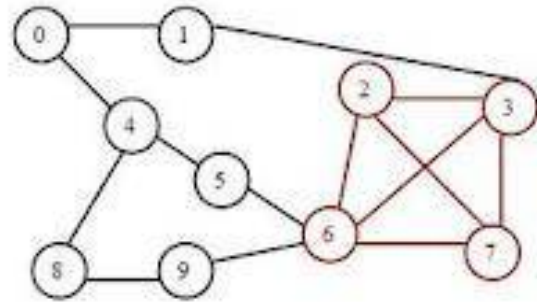
11



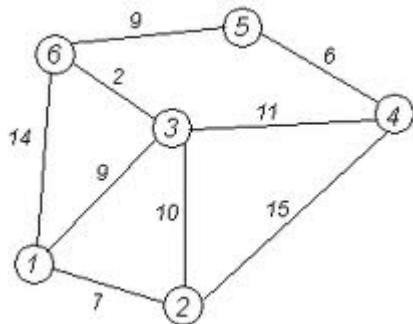
12



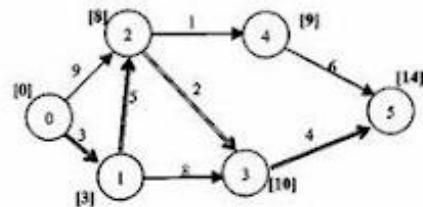
13



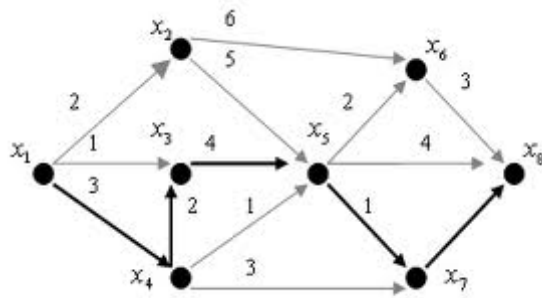
14



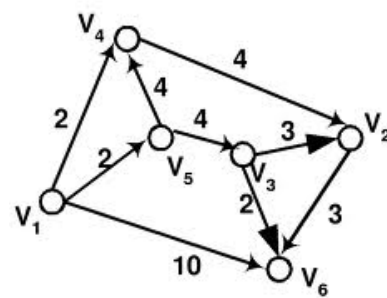
15



16



17



18

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**  
**для проведения контрольных работ по дисциплине**  
**«Интеллектуальный анализ данных»**

**Составители:**

Васяева Татьяна Александровна – кандидат технических наук, доцент, декан факультета информационных систем и технологий, доцент кафедры автоматизированных систем управления ГОУВПО «ДОННТУ»

**Ответственный за выпуск:**

Секирин Александр Иванович – кандидат технических наук, доцент, заведующий кафедрой автоматизированных систем управления ГОУВПО «ДОННТУ»