

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

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

для обучающихся по направлениям подготовки
09.04.01 «Информатика и вычислительная техника»,
дневной 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 лабораторных работ по дисциплине «Интеллектуальный анализ данных» [Электронный ресурс]: для обучающихся по направлениям подготовки 09.04.01 «Информатика и вычислительная техника», дневной 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 проведения лабораторных работ по курсу.

УДК 004.8(076)
ББК 32.813я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
Лабораторная работа №1.....	8
Лабораторная работа №2.....	18
Лабораторная работа №3.....	31
Лабораторная работа №4.....	68
Лабораторная работа №5.....	81
Лабораторная работа №6.....	86
Лабораторная работа №7.....	102
СПИСОК ЛИТЕРАТУРЫ.....	107
Приложение А. Описание HTML тегов.....	108
Приложение Б. Листинг программы metacritic.ipynb	114
Приложение В. Листинг программы.....	116
Приложение Д. Листинг программы.....	119
Приложение Е. Листинг программы.....	121
ПРИЛОЖЕНИЕ Ж. Ж.1. Листинг программы «Решение задачи OneMax с помощью DEAP».....	125
Приложение Ж. Ж.2. Листинг программы «Решение задачи OneMax с помощью DEAP».....	129
Приложение И. Индивидуальные задания на лабораторную работу №4.....	132
Приложение К. Индивидуальные задания на лабораторную работу №7.....	140

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.

Лабораторная работа №1

Тема: применение библиотек BeautifulSoup и Requests для сбора данных с сайтов.

Цель: изучение способов получения данных из сети интернет; научиться использовать библиотеки BeautifulSoup и Requests для поиска и извлечения данных.

1.1 Получение данных из веб-ресурсов

Веб-скрапинг – процесс извлечения неструктурированных данных (как правило просто помеченных тегами HTML) из веб-страниц для дальнейшего приведения их к структурируемому виду. Существует четыре принципиальных подхода извлечения данных с веб-ресурсов:

Метод Copy-Paste. Метод основан на ручном копировании необходимой информации с вебресурсов и помещению их в структурированный вид (база данных, файлы форматов Excel, CSV и т.д.). Метод позволяет получить достаточно точные результаты, при этом метод слишком долгий для больших объемов данных. Применяется для извлечения отдельных значений (не большого количества значений), когда писать средства автоматизации займет дольше времени, чем извлечь все данные вручную.

Поиск по регулярным выражениям. Метод основан на сопоставлении данных некоторому шаблону и извлечению их с дальнейшим переводом к нужному формату. Особенность метода заключается в необходимости разработать паттерн позволяющий извлечь все (максимум) нужные данные.

Интерфейс API. Метод предполагает, что сайт предоставляет необходимый функционал для извлечения данных. Самый удобный из всех методов, однако возможность реализации этого метода зависит от разработчика веб-сервиса.

Парсинг DOM. Модель DOM представляет иерархическую структуру (рисунок 1.1). Извлечение данных в этом случае осуществляется с использованием соответствующих тегов, которыми помечены данные.

1.2 Структура DOM

Объектная модель документа (DOM) – это интерфейс, с помощью которого программы могут работать с контентом, структурой и стилями веб-страницы. DOM – это объектное представление исходного HTML-документа, попытка преобразовать его структуру и содержимое в объектную модель, с которой смогли бы работать различные программы.

В ближайшем приближении DOM – это "дерево узлов" (рис. 1.1). У него единый корень, который разветвляется на множество дочерних ветвей, каждая из которых может ветвиться сама и заканчивается "листьями". Корень – это элемент html, а ветви – вложенные элементы.

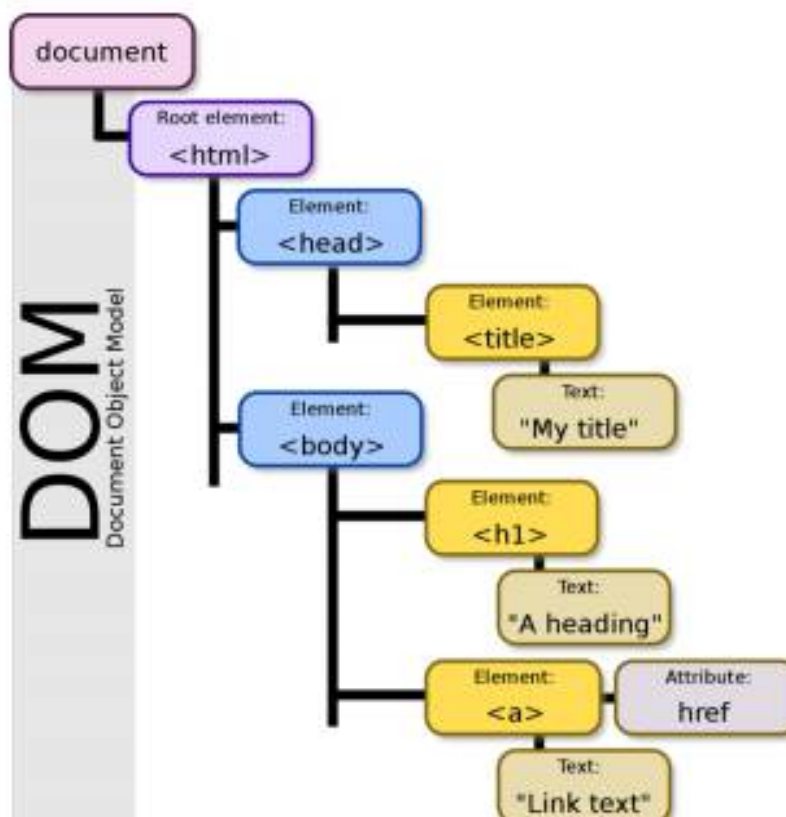


Рисунок 1.1. Объектная модель документа на примере HTML-документа

Глядя на схему (рис.1.2), легко понять связи тегов – можно сказать:

- предком называется тот тег, который включает в себя другие теги. В нашей схеме `html` – это предок для `head` и `body`, а `head`, в свою очередь, является предком для `title` и `script`. Тег `body` – предок для `h1`, `h2`, `p`;
- потомком называется тег, который расположен в одном или нескольких тегах. Например, теги `head` и `body` являются потомками тега `html`, `title` и `script` – потомки и для `head`, и для `html`, теги `h1`, `h2` и `p` – потомки и для `body`, и для `html`, а `span` – потомок для `p`, `body` и `html`;
- родительским называется тот элемент, который находится на один уровень выше относительно другого. В схеме родителем является `html` по отношению к `head` и `body`. Тег `head` – родитель тегов `title` и `script`. Тег `body` – родитель для `h1`, `h2` и `p`. А тег `p` является родителем для `span`;
- дочерним, соответственно, называется элемент, который находится под родительским элементом. Теги `h1`, `h2`, `p` – дочерние для `body`. Но при этом тег `span` является дочерним только для `p`;
- соседними называются элементы, у которых есть общий родитель. Например, соседними между собой являются теги `head` и `body`, а также теги `h1`, `h2`, `p`.

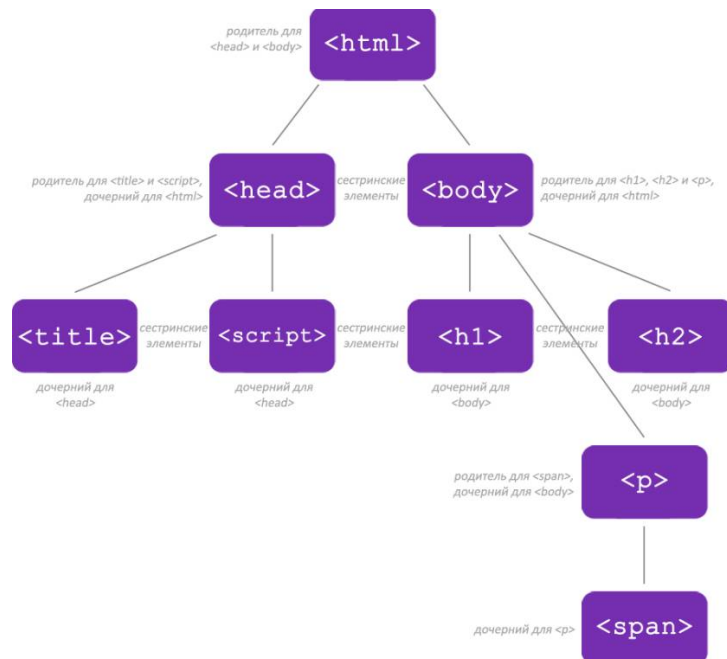


Рисунок 1.2. Пример HTML-документа

Описание всех тегов находится в приложении А.

1.3 Интерфейс прикладного программирования – API

Интерфейс прикладного программирования (application programming interfaces, или API) – средства для обмена информацией между несколькими различными приложениями. Не имеет значения, что приложения написаны разными программистами, с использованием различных архитектур или даже на разных языках, API выступают в качестве общепринятого языка для различных программ, которые должны обмениваться информацией друг с другом. Как правило, программист выполняет запрос к API с помощью HTTP, чтобы получить некоторые типы данных, а API возвращает эти данные в виде XML или JSON.

Есть четыре способа отправить запрос к веб-серверу с помощью HTTP:

- GET;
- POST;
- PUT;
- DELETE.

Современные API перед использованием часто требуют аутентификации. Методы аутентификации построены на использовании определенного токена, который передается на веб-сервер с каждым вызовом API. Токен можно передать в URL-адресе запроса напрямую, или с помощью cookie в заголовке запроса.

При извлечении данных с помощью GET-запроса URL-путь описывает интересующие нас данные, а параметры запроса выступают в качестве фильтров. Например, вы можете отправить запрос, чтобы извлечь все посты пользователя с ID = 1234, написанные в течение августа 2014 года (сайт <http://socialmediasite.com>):

<http://socialmediasite.com/users/1234/posts?from=08012014&to=08312014>

Также можно работать и с другими сайтами, которые предоставляют API. Получить ответ на API-запрос можно в JSON или XML, который вы можете парсить. Библиотека, используемая для парсинга JSON-данных,

является частью стандартной библиотеки языка Python. Пример кода (рис. 1.3) вернет код страны для IPадреса: 5.153.176.84.

```
In [10]: import json
from urllib.request import urlopen
response = urlopen("http://api.ipstack.com/5.153.176.84?access_key=9b68726509586630a6b38674396b3afd").read()
responseJson = json.loads(response)
print(json.dumps(responseJson,indent=2))

{
  "ip": "5.153.176.84",
  "type": "ipv4",
  "continent_code": "EU",
  "continent_name": "Europe",
  "country_code": "UA",
  "country_name": "Ukraine",
  "region_code": "14",
  "region_name": "Donetsk",
  "city": "Khartsyzk",
  "zip": "86100",
  "latitude": 47.98958969116211,
  "longitude": 38.045989990234375,
  "location": {
    "geoname_id": 706466,
    "capital": "Kyiv",
    "languages": [
      {
        "code": "uk",
        "name": "Ukrainian",
        "native": "\u0422\u0430\u0440\u0430\u0457\u043d\u0441\u044c\u043a\u0430\u0430"
      }
    ]
  },
  "country_flag": "http://assets.ipstack.com/flags/ua.svg",
  "country_flag_emoji": "\ud83c\uddfa\uud83c\uddde6",
  "country_flag_emoji_unicode": "U+1F1FA U+1F1E6",
  "calling_code": "380",
  "is_eu": false
}
```

Рисунок 1.3. Полученные данные с помощью api в JSON формате

1.4 Основы работы с библиотеками requests и beautifulsoup

Beautifulsoup предполагает универсальное средство для веб-скрапинга, реализованного по модели парсинга DOM.

Основные методы, используемые для веб-скрапинга (табл. 1.1):

Таблица 1.1. Описание методов BeautifulSoup

Метод	Параметры	Описание
prettify		При помощи этого метода можно добиться того, чтобы HTML-код выглядел аккуратнее.
find	tag, param	При помощи этого метода можно найти элементы страницы, используя различные опорные параметры, id в том числе.
find_all	tag, param	При помощи этого метода можно найти все элементы, которые соответствуют заданным критериям.
select	selector	Метод для нахождения запрашиваемого элемента можно использовать некоторые CSS селекторы.

Метод	Параметры	Описание
<code>select_one</code>	<code>selector</code>	Метод для нахождения запрашиваемых элементов можно использовать некоторые CSS селекторы.
<code>append</code>	<code>pos, tag</code>	Метод добавляет в рассматриваемый HTML-документ новый тег.
<code>insert</code>	<code>tag</code>	Метод позволяет вставить тег в определенно выбранное место.
<code>replace_with</code>	<code>newTag</code>	Метод заменяет содержимое выбранного элемента.
<code>decompose</code>		Метод удаляет определенный тег из структуры документа и уничтожает его.

Более подробно можно ознакомиться с документацией BeautifulSoup на сайте <https://www.crummy.com/software/BeautifulSoup/bs4/doc.ru/bs4ru.html>.

Библиотека Requests дает вам возможность посылать HTTP/1.1-запросы, используя Python. С ее помощью вы можете добавлять контент, например, заголовки, формы, многокомпонентные файлы и параметры, используя только простые библиотеки Python. Также вы можете и получать доступ к данным.

Для получения веб-страницы вам нужно написать:

```
r = requests.get("Ссылка сайта")
```

Исходя из полученного в ответ класса, мы можем ориентироваться в дальнейших действиях.

Выделяют пять классов ответов:

- 1xx – информационные коды. Они отвечают за процесс передачи данных. Это временные коды, они информируют о том, что запрос принят и обработка будет продолжаться.
- 2xx – успешная обработка. Запрос был получен и успешно обработан сервером.
- 3xx – перенаправление (редирект). Эти ответы гласят, что нужно предпринять дальнейшие действия для выполнения запроса.
- 4xx – ошибка пользователя. Это значит, что запрос не может быть выполнен по его вине.

- 5xx – ошибка сервера. Эти коды возникают из-за ошибки на стороне сервера. В данном случае пользователь всё сделал правильно, но сервер не может выполнить запрос.

Более подробно можно ознакомиться с библиотекой requests на сайте <https://requests.readthedocs.io/en/master/>.

1.5 Пример парсинга сайта Metacritic.com

Для извлечения данных будет использован сайт www.metacritic.com. Пусть мы хотим выгрузить отсортированные данные по фильмам: <https://www.metacritic.com/browse/movies/score/metascore/all/filtered?sort=desc&view=detailed>.

Используя данные с сайта можно создать, например, хранилище имен авторов, название произведений, их оценки. Но сначала их нужно выгрузить. Сперва нужно изучить исходный код страницы (рис 1.4).

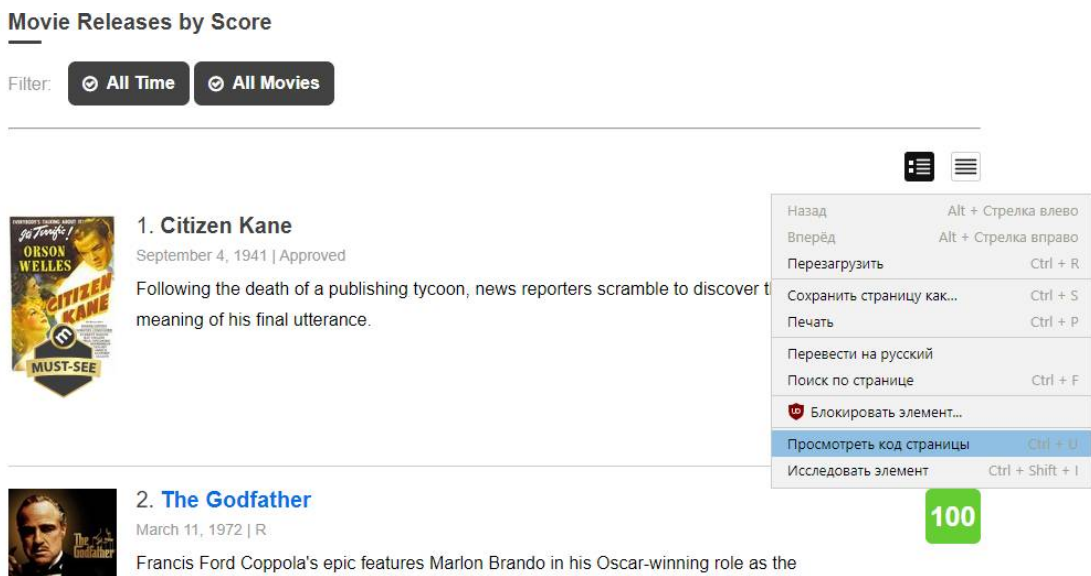


Рисунок 1.4– Страница www.metacritic.com для анализа

После нажатия «Просмотр код страницы» На экране будет выведена HTML-разметка страница. Например, такая (рис 1.5).

На этом примере можно увидеть, что разметка включает массу на первый взгляд перемешанных данных. Задача веб-скрапинга — получение доступа к тем частям страницы, которые нужны. Многие разработчики

используют регулярные выражения для этого, но библиотека Beautiful Soup в Python — более дружелюбный способ извлечения необходимой информации.



Рисунок 1.5. Код страницы

Прохождение по структуре HTML

HTML — это HyperText Markup Language («язык гипертекстовой разметки»), который работает за счет распространения элементов документа со специальными тегами. В HTML есть много разнообразных тегов, но стандартный шаблон включает три основных: `html`, `head` и `body`. Они организуют весь документ. В случае со скрапингом интерес представляет только тег `body`.

Написанный скрипт уже получает данные о разметке из указанного адреса. Дальше нужно сосредоточиться на конкретных интересующих данных.

Если в браузере воспользоваться инструментом «Inspect» (CTRL+SHIFT+I), то можно достаточно просто увидеть, какая из частей разметки отвечает за тот или иной элемент страницы. Достаточно навести мышью на определенный тег `div`, как он подсветит соответствующую информацию на странице. Таким образом и происходит дешифровка данных, которые требуется получить (рис 1.6). Скрапинг же позволяет извлекать все

похожие разделы HTML-документа. И это все, что нужно знать об HTML для скрапинга.

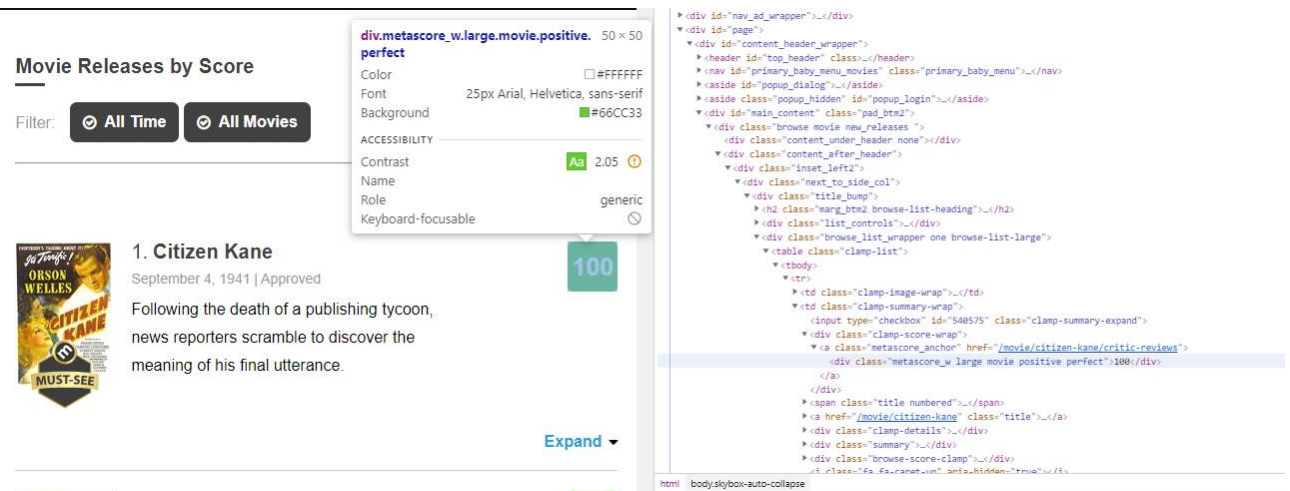


Рисунок 1.6. Исследование страницы

Селекторы сайта metacritic.com:

- `input.clamp-summary-expand` – id;
- `a.title` – название фильма;
- `div.metascore_w large movie positive` – score;
- `div.clamp-details>span:nth-of-type(1)` – дата выхода фильма;
- `div.clamp-details>span:nth-of-type(2)` – рейтинг фильма;
- `div.summary` – описание фильма.

Пример загрузки данных с сохранением в файл `film.csv` в `metacritic1.ipynb` и в приложении Б. Выгруженный файл имеет вид (рис. 1.7).

1.6. Порядок выполнения работы

1. Используя сайт <https://www.themoviedb.org/person?language=en-US> предложите ER диаграмму (https://studme.org/93800/informatika/er-diagramy_notatsii_chena), которая будет включать как минимум следующие сущности: Movie (фильмы), Genre (жанры), TV Series (сериалы), Seasons (сезоны), Episodes(эпизоды), Movie Peoples (актеры), Actor's Role (роль которую играл актер в фильме/сериале. Нарисуйте диаграмму в любом графическом редакторе).

2. Выгрузите реальные данные с сайта <https://www.themoviedb.org/> используя библиотеки `requests` и `beautifulsoup` или API (<https://developers.themoviedb.org/3/getting-started/introduction>) в некоторую промежуточную модель данных (например, в объектную или реляционную) в соответствии с предложенной вами схемой (Задание 1).

3. Затем из промежуточной модели данных сохраните каждую таблицу в виде отдельного csv файла.

Id	Name	Metascore	Rate	Description
0,540575	Citizen Kane	100	Not Rated	"Following the death of a publishing tycoon, news reporters scramble to discover the meaning of his final utterance."
1,522553	The Godfather	100	Not Rated	"Francis Ford Coppola's epic features Marlon Brando in his Oscar-winning role as the patriarch of the Corleone family. Director Coppola paints a...
2,548059	Rear Window	100	Not Rated	A wheelchair-bound photographer spies on his neighbours from his apartment window and becomes convinced one of them has committed mur...
3,546633	Casablanca	100	Not Rated	"A Casablanca, Morocco casino owner in 1941 shelters his former lover and her husband, a Czechoslovakian freedom fighter, from the Nazis."
4,542176	Boyhood	100	Not Rated	"Filmed over 12 years with the same cast, Richard Linklater's Boyhood is a groundbreaking story of growing up as seen through the eyes of a child nam...
5,545507	Three Colors: Red	100	Not Rated	"Krzysztof Kieslowski closes his Three Colors trilogy in grand fashion, with an incandescent meditation on fate and chance, starring Irne Jacob...
6,540585	Vertigo	100	Not Rated	"Vertigo creates a dizzying web of mistaken identity, passion and murder after an acrophobic detective rescues a mysterious blonde from the bay. [Uni...
7,549306	Notorious	100	Not Rated	A woman is asked to spy on a group of Nazi friends in South America. How far will she have to go to ingratiate herself with them?
8,547319	Singin' in the Rain	99	Not Rated	A silent film production company and cast make a difficult transition to sound.
9,542045	City Lights	99	Not Rated	The Tramp (Charlie Chaplin) struggles to help a blind flower girl he has fallen in love with.
10,545441	Moonlight	99	Not Rated	"Moonlight is the tender, heartbreaking story of a young man's struggle to find himself, told across three defining chapters in his life as he exper...
11,541955	Intolerance	99	Not Rated	"The story of a poor young woman, separated by prejudice from her husband and baby, is interwoven with tales of intolerance from throughout his...
12,549592	Pinocchio	99	Not Rated	"A living puppet, with the help of a cricket as his conscience, must prove himself worthy to become a real boy."
13,549144	Touch of Evil	99	Not Rated	"This film noir portrait of corruption and morally-compromised obsessions stars Welles as Hank Quinlan, a crooked police chief who frames a Mex...
14,547931	The Treasure of the Sierra Madre	98	Not Rated	"Fred Dobbs and Bob Curtin, two Americans searching for work in Mexico, convince an old prospector to help them mine for g...
15,500064	Pan's Labyrinth	98	Not Rated	"Following a bloody civil war, young Ofelia enters a world of unimaginable cruelty when she moves in with her new stepfather, a tyrannical mil...
16,547335	Some Like It Hot	98	Not Rated	"When two male musicians witness a mob hit, they flee the state in an all-female band disguised as women, but further complications set in."
17,548020	North by Northwest	98	Not Rated	"A hapless New York advertising executive is mistaken for a government agent by a group of foreign spies, and is pursued across the count...
18,551317	Rashomon	98	Not Rated	"The rape of a bride and the murder of her samurai husband are recalled from the perspectives of a bandit, the bride, the samurai's ghost and a woo...
19,547873	All About Eve	98	Not Rated	An ingenue insinuates herself into the company of an established but aging stage actress and her circle of theater friends.
20,542732	Hoop Dreams	98	Not Rated	Two inner-city Chicago boys with hopes of becoming professional basketball players struggle to become college players.
21,542392	Jules and Jim	97	Not Rated	Decades of a love triangle concerning two friends and an impulsive woman.
22,549434	The Wild Bunch	97	Not Rated	"An aging group of outlaws look for one last big score as the "traditional" American West is disappearing around them."
23,546649	My Left Foot	97	Not Rated	"True story of cerebral palsied Christy Brown, who overcame his illness and poverty to become an accomplished artist, poet and writer."
24,549176	The Third Man	97	Not Rated	"Pulp novelist Holly Martins travels to shadowy, postwar Vienna, only to find himself investigating the mysterious death of an old friend, Harry L...
25,550187	Nomadland	97	Not Rated	"Following the economic collapse of a company town in rural Nevada, Fern (Frances McDormand) packs her van and sets off on the road exploring a...
26,522445	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb	97	Not Rated	"Through a series of military and political accidents, a psychotic general - U.S. Air Force C...
27,546631	Gone with the Wind	97	Not Rated	"A Southern belle struggles with the devastation of the Civil War and Reconstruction. This classic won 8 Academy Awards, including Best Pi...
28,500049	"4 Months, 3 Weeks and 2 Days"	97	Not Rated	"During the final days of communism in Romania, two college roommates Otilia and Gabita are busy preparing for a night away...
29,546720	Psycho	97	Not Rated	"A Phoenix secretary embezzles \$40,000 from her employer's client, goes on the run, and checks into a remote motel run by a young man, Norman Bates...
30,553271	Battleship Potemkin	97	Not Rated	"In the midst of the Russian Revolution of 1905, the crew of the battleship Potemkin mutiny against the brutal, tyrannical regime of the ve...
31,549141	A Streetcar Named Desire	97	Not Rated	Disturbed Blanche DuBois (Vivien Leigh) moves in with her sister in New Orleans and is tormented by her brutish brother-in-law (Ma...
32,547411	American Graffiti	97	Not Rated	A couple of high school grads spend one final night cruising the strip with their buddies before they go off to college.
33,549563	Dumbo	96	Not Rated	"This simple animated tale is set in a circus and spotlights a baby elephant, Dumbo, who is mocked and ridiculed because his ears are too big, only to be...

Рисунок 1.7 – Содержание CSV файла, выгруженных данных

1.7. Содержание отчета

1. Титульный лист.
2. Задание.
3. ER диаграмма в нотации Чена.
4. Описание библиотек `requests` и `beautifulsoup` или API, в зависимости от реализации.
3. Листинг программы.
4. Результаты работы программы.

Лабораторная работа №2

Тема: нейронные сети для работы с языковыми задачами.

Цель работы: изучение методов и алгоритмов обработки текста в Tensorflow и Keras.

2.1. Глубокое обучение сетей для обработки естественного языка

Моделями глубокого обучения, пригодными для обработки текста (который можно интерпретировать как последовательности слов или символов), являются рекуррентные нейронные сети и одномерные сверточные нейронные сети.

В число прикладных применений этих алгоритмов попадают:

1. Классификация текстов.
2. Сравнение документов.
3. Автоматический перевод.
4. Анализ эмоциональной окраски.
5. Аннотирование.
6. Генерация текста.
7. Чат-боты.

Обработка текста включает в себя следующие этапы:

1. Анализ входных данных. Нейронная сеть работает только с числами.

Поэтому любой текст должен пройти процесс Векторизации.

2. Токенизация. Токенизация текста – разбиение текста на элементы (токены). В качестве токенов могут выступать буквы, слова, предложения. Для такой трансформации используются специальные модели, наиболее популярными из которых являются:

- bag of words («сумка слов») – детальная репрезентативная модель для упрощения обработки текстового содержания. Она не учитывает грамматику или порядок слов и нужна для определения количества вхождений отдельных слов в анализируемый текст: создается вектор длиной в словарь, для каждого

слова считается количество вхождений в текст и это число подставляется на соответствующую позицию в векторе. Однако, при этом теряется порядок слов в тексте, а значит, после векторизации предложения.

– n-граммы – комбинации из n последовательных терминов для упрощения распознавания текстового содержания. Эта модель определяет и сохраняет смежные последовательности слов в тексте.

Например, предложение «The cat sat on the mat» можно разложить на следующий набор 2-грамм:

{"The", "The cat", "cat", "cat sat", "sat", "sat on", "on", "on the", "the", "the mat", "mat"}

Также его можно разложить на такой набор 3-грамм:

{"The", "The cat", "cat", "cat sat", "The cat sat", "sat", "sat on", "on", "cat sat on", "on the", "the", "sat on the", "the mat", "mat", "on the mat"}

Такие наборы называют мешком биграмм или мешком триграмм соответственно. Термин мешок в данном случае отражает тот факт, что вы имеете дело с множеством токенов, а не со списком или последовательностью.

Механизм токенизации приведён на рис. 2.1:



Рисунок 2.1 – Преобразование текста в токены и затем в векторы

3. Векторизация. Векторизация – замена токенов на цифры, сопоставление цифровых векторов с полученными ранее токенами.

Существуют несколько методов кодирования:

One-Hot Encoding – это метод кодирования категориальных переменных в числовые данные, с которыми могут работать алгоритмы машинного обучения. Может применяться как кодировщик каждого токена (по одному символу на токен) или кодировать только соответствующие токену элементы, остальные приравнивать к нулю.

Embedding – числовые векторы, которые получены из слов или других языковых сущностей. Каждому токену сопоставляется вектор, размерность которого ниже чем у One-Hot Encoding.

Прямое кодирование (one-hot encoding)

Унитарный код – двоичный код фиксированной длины, содержащий только одну 1 (прямой унитарный код) или только один 0 (обратный (инверсный) унитарный код).

Длина кода определяется количеством кодируемых объектов, то есть каждому объекту соответствует отдельный разряд кода, а значение кода положением 1 или 0 в кодовом слове.

Возьмем Толковый словарь Ушакова и пронумеруем все слова с первого до последнего. Так слово «абажур» будет стоять на 7 месте. Всего слов в словаре 85 289 слов. Числовой вектор слова будет иметь 85288 нулей на всех позициях, кроме седьмой, где будет единица.

На рис. 2.2 представлен пример вектора one-hot encoding.

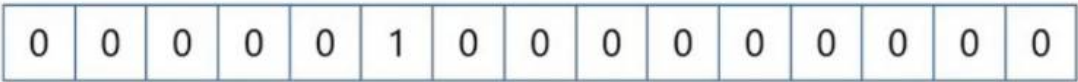


Рисунок 2.2 – Вектор one-hot encoding

Прямое кодирование на уровне слов изображено на рис. 2.3

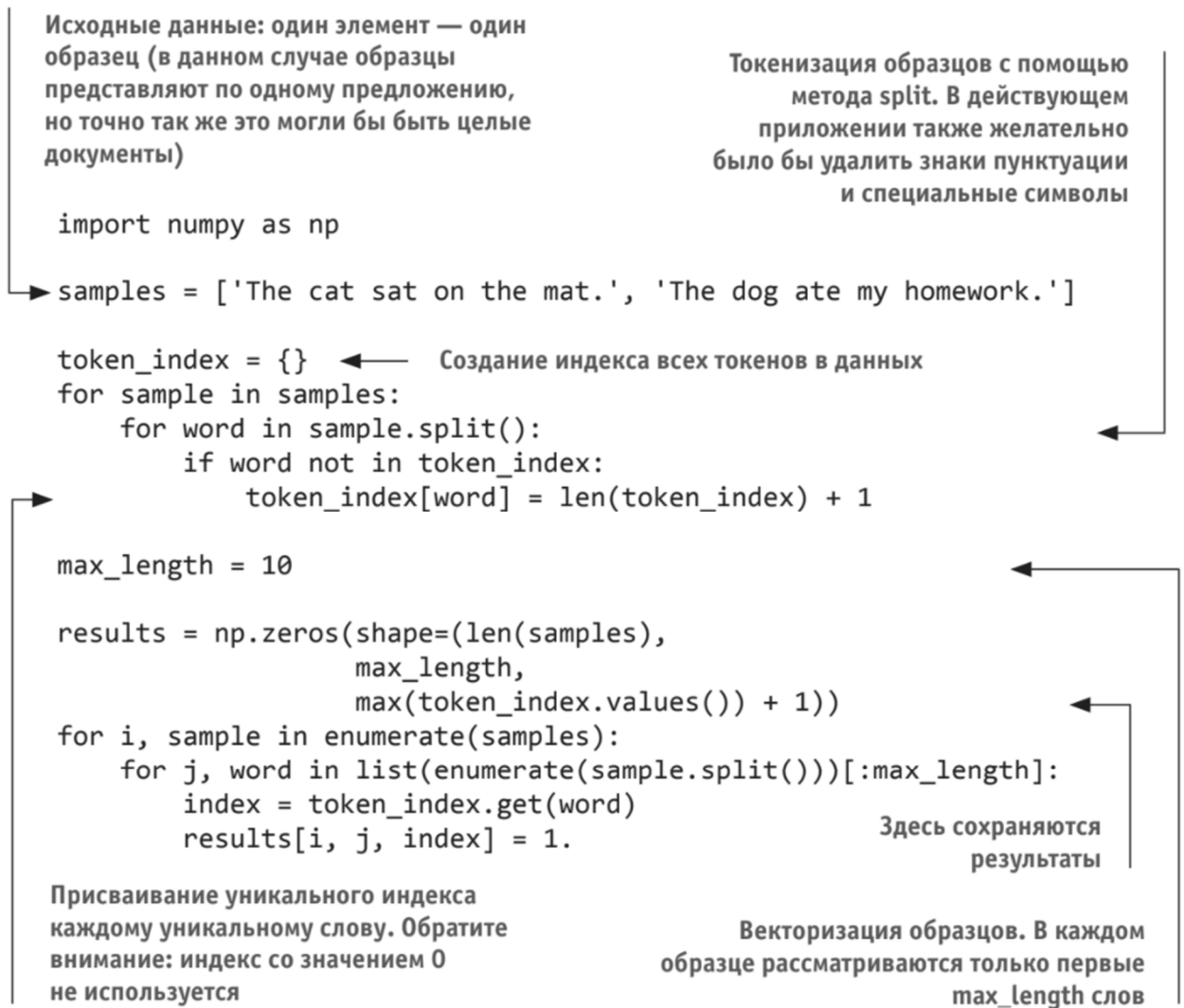


Рисунок 2.3 – Прямое кодирование на уровне слов.

Прямое кодирование можно также выполнить на уровне символов (рис. 2.4).

```
import string

samples = ['The cat sat on the mat.', 'The dog ate my homework.']
characters = string.printable
token_index = dict(zip(characters, range(1, len(characters) + 1)))

max_length = 50
results = np.zeros((len(samples), max_length, max(token_index.keys()) + 1))
for i, sample in enumerate(samples):
    for j, character in enumerate(sample):
        index = token_index.get(character)
        results[i, j, index] = 1.
```

Все отображаемые символы ASCII

Рисунок 2.4 – Прямое кодирование на уровне символов.

В фреймворке Keras имеются встроенные утилиты для прямого кодирования простых текстовых данных на уровне слов и символов. Они обладают рядом важных свойств, таких как удаление из строк специальных символов и возможность принимать в расчет только N слов, наиболее часто встречающихся в наборе данных. На рис. 2.5 изображен пример использования Keras для прямого кодирования слов.

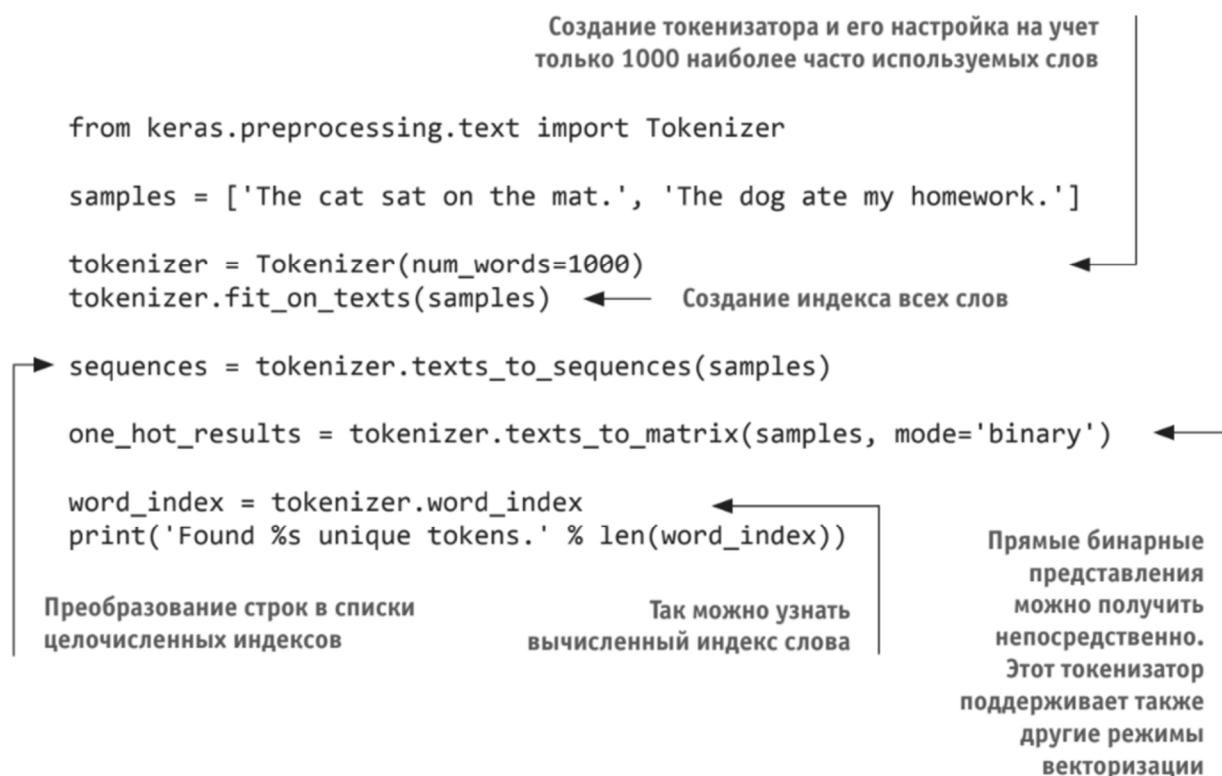


Рисунок 2.5 – Прямое кодирование символов с Keras.

Прием прямого кодирования имеет разновидность — так называемое прямое хеширование признаков (one-hot hashing trick), которое можно использовать, когда словарь содержит слишком большое количество токенов, чтобы его можно было использовать явно. Вместо явного присваивания индекса каждому слову и сохранения ссылок на эти индексы в словаре можно хешировать слова в векторы фиксированного размера. Главное достоинство этого метода — отсутствие необходимости хранить индексы слов, что позволяет сэкономить память и кодировать данные по мере необходимости. Единственный недостаток — этот метод восприимчив к хеш-коллизиям: два

разных слова могут получить одинаковые хеш-значения, и впоследствии любая модель машинного обучения не сможет различить эти слова. Вероятность хеш-коллизий снижается, когда размер пространства хеширования намного больше общего количества уникальных токенов, подвергаемых хешированию (рис. 2.6).

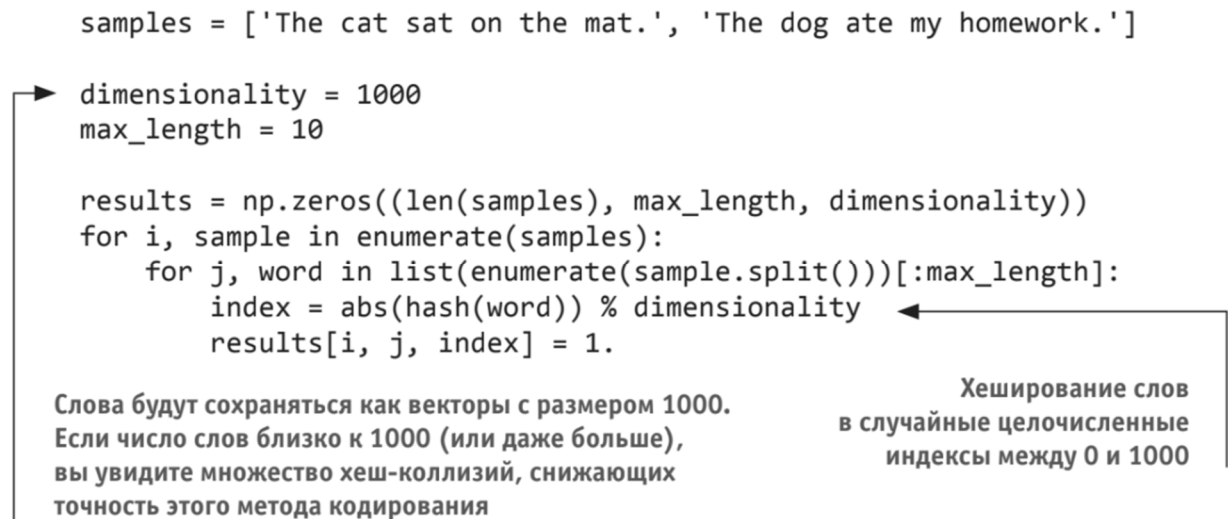


Рисунок 2.6 – Прямое кодирование на уровне слов с использованием хеширования.

Векторное представление слов (embedding)

Векторные представления слов являются малоразмерными векторами вещественных чисел. Числовой вектор размерности k – это список из k чисел, в котором порядок чисел строго определен. Например, трехмерным вектором можно считать (2.3, 1.0, 7.35).

В самой примитивной форме эмбединги слов получают простой нумерацией слов в некотором достаточно обширном словаре и установкой значения единицы в длинном векторе размерности, равной числу слов в словаре.

Получить векторные представления можно двумя способами:

1. Конструировать векторные представления в процессе решения основной задачи (изначально создаются случайные векторы слов, которые затем обучаются)

2. Загрузить в модель векторные представления, полученные в результате другого машинного обучения.

В нейронных сетях плотное векторное представление слов определяется в процессе обучения. На первом этапе в модель загружаются обученные векторные представления, которые обучаются с помощью метода обратного распространения ошибки.

Метод обратного распространения ошибки – метод вычисления градиента, который используется при обновлении весов многослойного перцептрона.

Нейрон обучается с помощью тренировочного множества $(x_1, x_2, t)(x_1, x_2, t)$ где x_1 и x_2 – это входные данные сети и t – правильный ответ.

Общепринятый метод вычисления несоответствия между ожидаемым t и получившимся y ответом – квадратичная функция потерь:

$$E = (t - y)^2, E = (t - y)^2, \text{ где } E \text{ ошибка.}$$

Следовательно, задача преобразования входных значений в выходные может быть сведена к задаче оптимизации, заключающейся в поиске функции, которая даст минимальную ошибку.

$$Y = x_1 w_1 + x_2 w_2,$$

где w_1 и w_2 – веса на ребрах, соединяющих входные вершины с выходной. Следовательно, ошибка зависит от весов ребер, входящих в нейрон. И именно это нужно менять в процессе обучения.

В результате, слова, имеющие схожие значения должны стоять рядом (рис. 2.7). Чем больше антагонизма в значении слов, тем дальше друг от друга их расположение (рис. 2.8-2.9).



Рисунок 2.7 – Расположение близких по значению слов

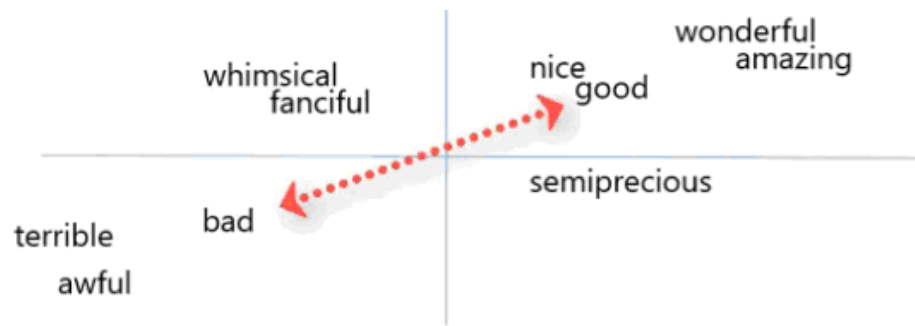


Рисунок 2.8 – Расположение противоположных по значению слов нейтральной эмоциональной окраски

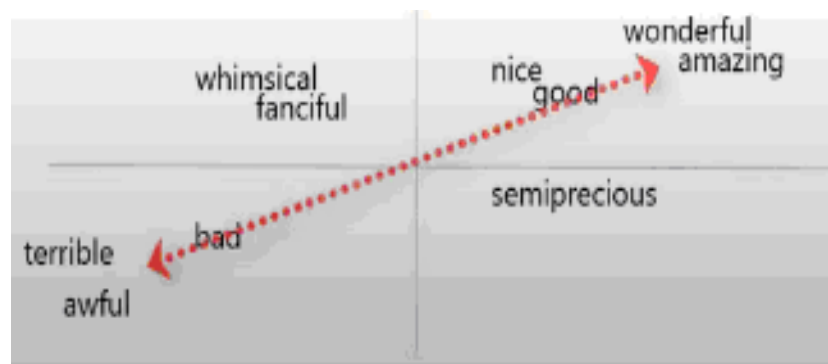


Рисунок 2.9 – Расположение противоположных по значению слов сильной эмоциональной окраски

2.2. Анализ тотальности текста в Tensorflow и Keras

Будем использовать набор отзывов на фильмы сайта *mdb*. Это набор достаточно популярный при обучении нейронным сетям и другим методам машинного обучения. Он включает в себя 50 тысяч отзывов на фильмы. Набор состоит из двух частей: для обучения 25 тысяч отзывов и набор для тестирования тоже 25 тысяч отзывов.

При подготовке этого набора данных использовались следующие служебные коды:

- 0 – используются для символа-заполнителя;
- 1 – для представления начала последовательностей;
- 2 – представления неизвестных слов.

Составители определяли положительные отзывы или отрицательные по количеству звезд, который поставили фильму. Положительные – с оценкой больше или равно 7 (им присваивалась 1) и отрицательные – меньше или равно 4 (им присваивался 0). Таким образом, с точки зрения машинного обучения эта задача бинарной классификации.

Нам необходимы следующие библиотеки:

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import utils
from tensorflow.keras.preprocessing.sequence import
pad_sequences
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Давайте мы посмотрим на какую-нибудь рецензию, мы увидим, что вместо текста отзывов у нас список чисел, которые идут друг за другом. Каждое число соответствует токenu слова. Если посмотреть на формат правильного ответа - единица или ноль.

В начале нашего кода (Приложение А) мы указывали, сколько слов будет использоваться для анализа (10000 слов).

```
Max_words = 10000
(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words = Max_words)
```

Соответственно слова, которые не входят в эти 10000 слов будут заменены на число 2. Все слова приведены к нижнему регистру, удалены знаки препинания.

В наборе данных mdb используются частотное кодирование слов. Это означает, что в тексте каждое слово заменяется на частоту его появления.

Мы можем скачать словарь python который использовался для такого кодирования:

```
word_index = imdb.get_word_index()
```

Мы можем сделать реверсивный словарь, который будет по числу определять слово.

```
reverse_word_index = dict()
for key, value in word_index.items():
    reverse_word_index[value] = key
```

Подготовим данных для работы с НС. Выбираем длину, например 200 слов и обрезаем рецензии, которые имеют длину больше, и добавляем символы заполнителя – если меньше. В данном наборе в качестве такого символа используется 0.

```
maxlen = 200
x_train = pad_sequences(x_train, maxlen=maxlen,
padding='post')
x_test = pad_sequences(x_test, maxlen=maxlen,
padding='post')
```

Создание нейронной сети для классификации текста:

```
model = Sequential()
model.add(Dense(128, activation='relu',
input_shape=(maxlen,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Обучение нейронной сети:

```
history = model.fit(x_train, y_train, epochs=25,
batch_size=128, validation_split=0.1)
```

Проверяем работу сети на тестовом наборе данных:

```
scores = model.evaluate(x_test, y_test, verbose=1)
```

В примере выше использовалось частотное кодирование слов. Это не единственный вариант кодирования.

Представление текста в формате One-hot encoding:

```
def vectorize_sequences(sequences,
dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
x_train = vectorize_sequences(x_train, max_words)
x_test = vectorize_sequences(x_test, max_words)
```

Представление текста в формате embedding для определения тональности текста предполагает использование в сети слоя embedding.

Создание сети:

```
model = Sequential()
model.add(Embedding(max_words, 2,
input_length=maxlen))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])
```

Можно получить матрицу плотных представлений:

```
embedding_matrix = model.layers[0].get_weights()[0]
```

Загрузить словарь с номерами слов:

```
word_index_org = imdb.get_word_index()
```

Дополнить словарь служебными символами:

```
word_index = dict()
```

```

for word,number in word_index_org.items():
word_index[word] = number + 3
word_index["<Заполнитель>"] = 0
word_index["<Начало последовательности>"] = 1
word_index["<Неизвестное слово>"] = 2
word_index["<Не используется>"] = 3

```

Проверить на примере какого-нибудь слова:

```

word = 'good'
word_number = word_index[word]
print('Номер слова', word_number)
print('Вектор для слова',
embedding_matrix[word_number])

```

Выберем коды слов, по которым можно выделить эмоциональную характеристику и посмотрим на их векторное представление.

```

review = ['brilliant', 'fantastic', 'amazing',
'good', 'bad', 'awful','crap', 'terrible', 'trash']
enc_review = []
for word in review:
    enc_review.append(word_index[word])
enc_review
plt.scatter(review_vectors[:,0],
review_vectors[:,1])
for i, txt in enumerate(review):
    plt.annotate(txt, (review_vectors[i,0],
review_vectors[i,1]))

```

В приложениях В, Д и Е приведены примеры реализации определения тональности отзывов на фильмы из IMDb при использовании различного представления слов для определения тональности текста.

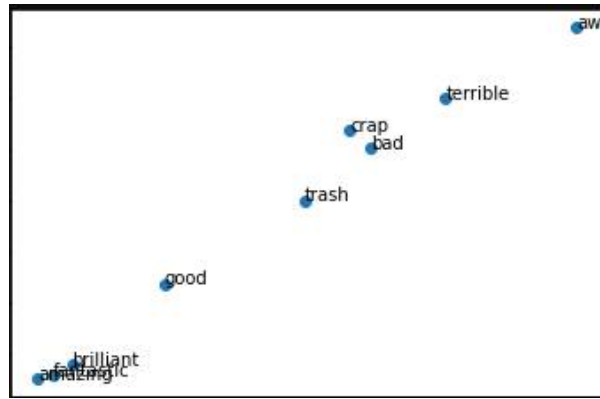


Рисунок 2.11 – Эмоциональная характеристика

2.3. Порядок выполнения работы

1. Выбрать данные для анализа. <https://tproger.ru/translations/the-best-datasets-for-machine-learning-and-data-science/>.
2. Выполнить постановку задачи (в зависимости от выбранных данных).
3. Выполнить предварительную обработку текста.
4. Разработать нейросетевую модель для решения вашей задачи. Для четных вариантов – использовать рекуррентную НС, для нечетных – сверточную.
5. Протестировать НС на данных, которые не использовались для получения модели, сделать выводы.

2.4. Содержание отчета

1. Титульный лист.
2. Описание предметной области и постановка задачи.
3. Описание выполнения предварительной работы с текстом
4. Описание используемых функций из библиотек Tensorflow и Keras.
5. Нейросетевая модель.
3. Листинг программы.
4. Результаты тестирования.

Лабораторная работа №3

Тема: решения задачи оптимизации одномерной функции простым генетическим алгоритмом.

Цель: изучение основных принципов ГА; научиться использовать библиотеку `dear` для оптимизации одномерной функции.

3.1. Основы генетических алгоритмов

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:

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 (фитнесс) функции имеют большие шансы выжить и репродуцировать. Формализация этого принципа, как мы увидим далее, дает оператор репродукции.

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).

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.3.1.

ГА берет множество параметров оптимизационной проблемы и кодирует их последовательностями конечной длины в некотором конечном алфавите (в простейшем случае двоичный алфавит «0» и «1»).

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

- оператор репродукции (ОР);
- оператор скрещивания или кроссинговера (ОК);
- 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.

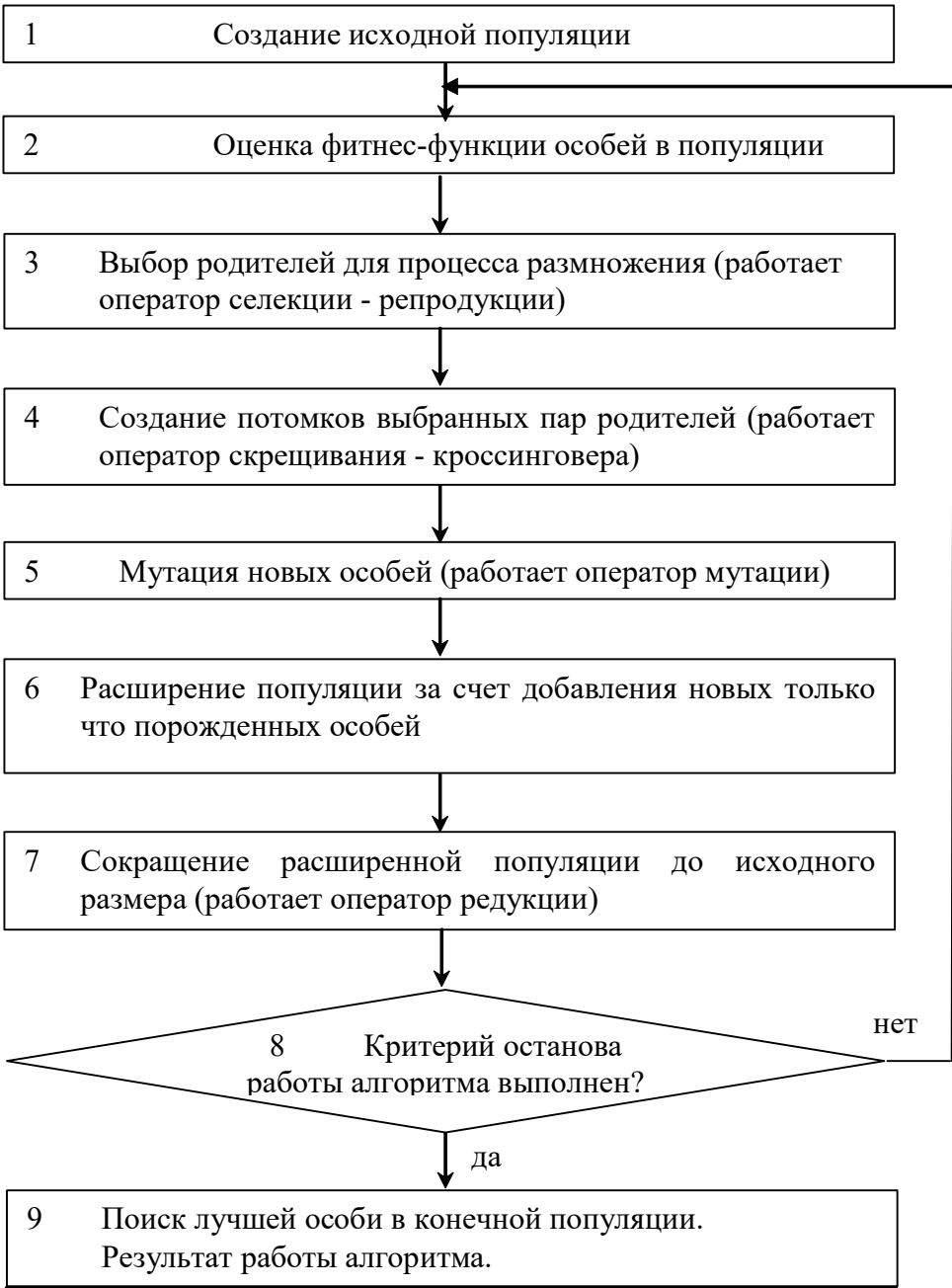


Рис.3.1. Простой генетический алгоритм

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.

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.

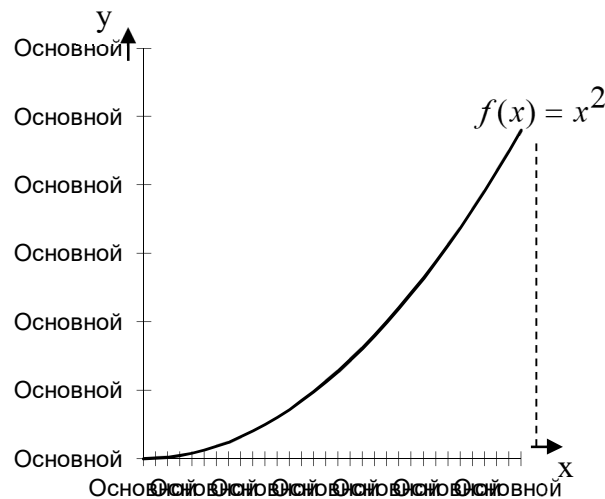


Рис.3.2. Пример функции

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.

Начальный этап работы ГА для данного примера приведен в верхней таблице (репродукция) рис.3.3.

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.

Репродукция

№ хромосомы	Начальная популяция особей	Десятичное значение x	Значение f(x)	$\frac{f(x_i)}{\sum f(x_j)}$	Среднее значение $\overline{f(x)}$	Максимально е 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 $\overline{f(x)}$	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 $\overline{f(x)}$	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.3.3. Эволюция популяции

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.

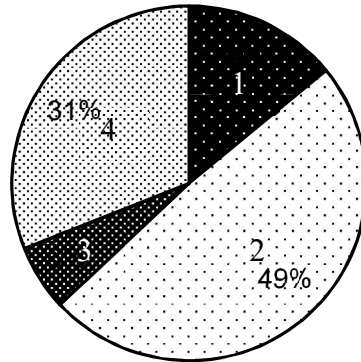


Рис.3.4. Колесо рулетки

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 -ой хромосомы определяется $M=P(x_i)*N$, 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 не попадает, 4-я - в одном экземпляре. Полученная промежуточная популяция является исходной для дальнейшего выполнения операторов кроссинговера и мутации.

Оператор кроссинговера. 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 подстрок 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:

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

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

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

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

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

Следует отметить, что ОК 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 частями 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 максимальное значения ЦФ) .

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 $A=a_1a_2...a_L$ случайно выбирается k -ая 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.

3.2. Применение DEAP для реализации простого генетического алгоритма

Distributed Evolutionary Algorithms in Python (DEAP) – распределенные эволюционные алгоритмы на Python) поддерживает быструю разработку решений с применением генетических алгоритмов и других методов эволюционных вычислений. DEAP предлагает различные структуры данных и инструменты, необходимые для реализации самых разных решений на основе генетических алгоритмов.

Начнем с рассмотрения модуля DEAP creator. Он используется как метафабрика и позволяет расширять существующие классы, добавляя в них новые атрибуты. Пусть, например, имеется класс Employee. С помощью модуля creator мы можем создать из него класс Developer следующим образом:

```
from deap import creator

creator.create("Developer", Employee, position =
"Developer", programmingLanguages = set)
```

Первым аргументом функции create() передается имя нового класса, вторым – существующий класс, подлежащий расширению. Все последующие аргументы определяют атрибуты нового класса. Если значением аргумента является класс (например, dict или set), то он будет добавлен в новый класс как атрибут экземпляра, инициализируемый в конструкторе. Если же это не класс (а, например, литерал), то он добавляется как атрибут класса (статический).

Таким образом, созданный класс Developer расширяет класс Employee и имеет атрибут класса position, равный строке Developer, и атрибут экземпляра programmingLanguages типа set, который инициализируется в конструкторе.

Следовательно, новый класс эквивалентен такому:

```
class Developer(Employee):
    position = "Developer"
    def __init__(self):
        self.programmingLanguages = set()
```

Этот новый класс существует в контексте модуля `creator`, поэтому ссылаться на него следует по имени `creator.Developer`.

При работе с DEAP модуль `creator` обычно служит для создания классов `Fitness` и `Individual`, используемых в генетических алгоритмах.

Создание класса `Fitness`. При работе с DEAP значения приспособленности инкапсулированы в классе `Fitness`. DEAP позволяет распределять приспособленность по нескольким компонентам (называемым целями), у каждого из которых есть свой вес. Комбинация весов определяет поведение, или стратегию приспособления в конкретной задаче.

Определение стратегии приспособления. Для определения стратегии в состав DEAP входит абстрактный класс `base.Fitness`, который содержит кортеж `weights`. Этому кортежу необходимо присвоить значения, чтобы определить стратегию и сделать класс пригодным для использования. Для этого мы расширяем базовый класс `Fitness` с помощью модуля `creator` так же, как делали это ранее с классом `Developer`:

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

Получается класс `creator.FitnessMax`, расширяющий класс `base.Fitness`, в котором атрибут класса `weights` инициализирован значением `(1.0,)`.

Обратите внимание на запятую в конце определения `weights`, которая присутствует, хотя задан всего один вес. Она необходима, потому что `weights` – кортеж.

Стратегия этого класса `FitnessMax` – максимизировать приспособленность индивидуумов с единственной целью. Если бы нам нужно было минимизировать приспособленность в задаче с одной целью, то для задания соответствующей стратегии можно было бы определить такой класс:


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

Можно также определить класс для оптимизации сразу нескольких целей различной важности:

```
creator.create("FitnessCompound", base.Fitness,
weights=(1.0, 0.2, -0.5))
```

В классе `creator.FitnessCompound` используется три компоненты приспособленности с весами 1.0, 0.2 и -0.5. Это означает, что первая и вторая компоненты (цели) максимизируются, а третья минимизируется, причем первая компонента самая важная, следующей по важности является третья, а последней – вторая.

Хранение значения приспособленности. Если кортеж `weights` определяет стратегию приспособления, то кортеж `values` используется для хранения самих значений функции приспособленности в базовом классе `base.Fitness`. Эти значения дает отдельно определяемая функция, которую обычно называют `evaluate()`. Как и `weights`, кортеж `values` содержит по одному значению для каждой компоненты приспособленности (цели). Третий кортеж, `wvalues`, содержит взвешенные значения, полученные перемножением элементов кортежа `values` и соответственных элементов кортежа `weights`. Всякий раз, как устанавливаются значения приспособленности, соответствующие взвешенные значения вычисляются и сохраняются в `wvalues`. Они используются при сравнении индивидуумов. Взвешенные приспособленности можно сравнивать лексикографически с помощью следующих операторов: `>`, `<`, `>=`, `<=`, `==`, `!=`

Созданный класс `Fitness` мы будем использовать в определении класса `Individual`.

Создание класса `Individual`. Второе типичное применение модуля `creator` – определение индивидуумов, образующих популяцию в генетическом алгоритме. В DEAP класс `Individual` создается путем расширения базового класса, представляющего хромосому. Кроме того, каждый экземпляр класса `Individual` должен содержать функцию приспособленности в качестве

атрибута. Чтобы удовлетворить обоим требованиям, мы воспользуемся модулем `creator` для создания класса `creator.Individual`:

```
creator.create("Individual", list,
fitness=creator.FitnessMax)
```

Результат работы этой строки:

- созданный класс `Individual` расширяет встроенный класс Python `list`. Это означает, что все хромосомы имеют тип `list`;
- в каждом экземпляре класса `Individual` имеется атрибут `fitness` созданного ранее класса `FitnessMax`.

Использование класса `Toolbox`. Второй механизм, предлагаемый каркасом DEAP, – класс `base.Toolbox`. Он используется как контейнер для функций (или операторов) и позволяет создавать новые операторы путем назначения псевдонимов или настройки существующих функций.

Пусть, например, имеется следующая функция `sumOfTwo()`:

```
def sumOfTwo(a, b):
    return a + b
```

Воспользовавшись модулем `toolbox`, мы сможем создать новый оператор `incrementByFive()` на основе функции `sumOfTwo()`:

```
from deap import base
toolbox = base.Toolbox()
toolbox.register("incrementByFive", sumOfTwo, b=5)
```

Первым аргументом функции `register()` передается имя нового оператора (или псевдоним существующего), вторым – настраиваемая функция. Все остальные (необязательные) аргументы передаются этой функции при вызове нового оператора. Рассмотрим, к примеру, следующее определение:

```
toolbox.incrementByFive(10)
```

Этот вызов эквивалентен такому:

```
sumOfTwo(10, 5)
```

поскольку аргументу `b` было присвоено фиксированное значение 5 в определении оператора `incrementByFive`.

Создание генетических операторов. Во многих случаях класс Toolbox используется для настройки существующих функций из модуля tools, который содержит ряд полезных функций, относящихся к генетическим операциям отбора, скрещивания и мутации, а также утилиты для инициализации.

Например, в коде ниже определены три псевдонима, которые впоследствии будут использованы как генетические операторы:

```
from deap import tools
toolbox.register("select",      tools.selTournament,
tournsize=3)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate",      tools.mutFlipBit,
indpb=0.02)
```

Опишем, что здесь было сделано.

1. Имя select зарегистрировано как псевдоним существующей в модуле tools функции selTournament() с аргументом tournsize = 3. В результате создается оператор toolbox.select, который выполняет турнирный отбор с размером турнира 3.

2. Имя mate зарегистрировано как псевдоним существующей в модуле tools функции cxTwoPoint(). В результате создается оператор toolbox.mate, который выполняет двухточечное скрещивание.

3. Имя mutate зарегистрировано как псевдоним существующей в модуле tools функции mutFlipBit с аргументом indpb = 0.02. В результате создается оператор toolbox.mutate, который выполняет мутацию инвертированием бита с вероятностью 0.02.

Модуль tools предоставляет реализации различных генетических операторов.

Функции отбора находятся в файле selection.py. Перечислим некоторые из них:

- selRoulette() – отбор по правилу рулетки;

- `selStochasticUniversalSampling()` – стохастическая универсальная выборка;
- `selTournament()` – турнирный отбор.

Функции скрещивания находятся в файле `crossover.py`:

- `cxOnePoint()` – однотоочечное скрещивание;
- `cxUniform()` – равномерное скрещивание;
- `cxOrdered()` – упорядоченное скрещивание (OX1);
- `cxPartiallyMatched()` – скрещивание с частичным сопоставлением (partially matched crossover – PMX).

В файле `mutation.py` находятся две функции мутации:

- `mutFlipBit()` – мутация инвертированием бита;
- `mutGaussian()` – нормально распределенная мутация.

Создание популяции. Файл `init.py` модуля `tools` содержит несколько функций, полезных для создания и инициализации популяции. Особенно полезна функция `initRepeat()`, принимающая три аргумента:

- тип контейнера результирующих объектов;
- функция, генерирующая объекты, которые помещаются в контейнер;
- сколько объектов генерировать.

Например, следующая строка создает список из 30 случайных чисел от 0 до 1:

```
randomList = tools.initRepeat(list, random.random,
30)
```

В этом примере `list` – тип, выступающий в роли заполняемого контейнера, `random.random` – порождающая функция, а 30 – количество вызовов этой функции, необходимых для заполнения контейнера.

Если мы захотим заполнить список случайными целыми числами, равными 0 или 1 можно было бы создать функцию, которая вызывает `random.randint()` для генерации одного случайного числа, равного 0 или 1, а затем

использовать ее в качестве порождающей функции в `initRepeat()`, как показано ниже:

```
def zeroOrOne():
    return random.randint(0, 1)

randomList = tools.initRepeat(list, zeroOrOne, 30)
```

Или можно воспользоваться модулем `toolbox`:

```
toolbox.register("zeroOrOne", random.randint, 0, 1)

randomList = tools.initRepeat(list,
    toolbox.zeroOrOne, 30)
```

Здесь вместо того чтобы явно определять функцию `zeroOrOne()`, мы создали оператор (или псевдоним) `zeroOrOne`, который вызывает `random.randint()` с фиксированными параметрами 0 и 1.

Вычисление приспособленности. Как было отмечено выше, в классе `Fitness` задаются веса, определяющие стратегию приспособления (например, максимизация или минимизация), а сами значения приспособленности возвращает отдельно определенная функция. Эта функция обычно регистрируется в модуле `toolbox` под псевдонимом `evaluate`, как показано ниже:

```
def someFitnessCalculationFunction(individual):
    return _some_calculation_of_the_fitness

toolbox.register("evaluate", someFitnessCalculationFunction)
```

В данном примере функция `someFitnessCalculationFunction()` вычисляет приспособленность заданного индивидуума, а имя `evaluate` зарегистрировано в качестве ее псевдонима.

3.3. Решение задачи OneMax с помощью DEAP

Рассмотрим решение задачи OneMax с помощью DEAP. Задача OneMax – это простая задача оптимизации, которая заключается в том, чтобы найти двоичную строку заданной длины, для которой сумма составляющих ее цифр

максимальна. Например, при решении задачи OneMax длины 5 будут рассматриваться такие кандидаты:

10010 (сумма цифр = 2);

01110 (сумма цифр = 3);

11111 (сумма цифр = 5).

Очевидно (нам), что решением всегда является строка, состоящая из одних единиц. Но генетический алгоритм не обладает таким знанием, поэтому должен слепо искать решение, пользуясь генетическими операторами. Если алгоритм справится с работой, то найдет решение (или приближение к нему) за разумное время.

1. Выбор хромосомы. Поскольку в задаче OneMax мы имеем дело с двоичными строками, с выбором хромосом проблем не возникает – каждый индивидуум представлен двоичной строкой, которая непосредственно соответствует потенциальному решению. На языке Python это реализуется в виде списка, содержащего числа 0 или 1. Длина хромосомы совпадает с размером задачи OneMax. Например, в задаче OneMax размера 5 индивидуум 10010 будет представлен списком [1, 0, 0, 1, 0].

2. Вычисление приспособленности

Поскольку мы хотим найти индивидуума с наибольшей суммой цифр, следует использовать стратегию FitnessMax. А поскольку каждый индивидуум представлен списком целых чисел, равных 0 или 1, то приспособленность вычисляется просто как сумма элементов списка, например: `sum([1, 0, 0, 1, 0]) = 2`.

3. Выбор генетических операторов

Теперь нужно решить, какие реализации генетических операторов отбора, скрещивания и мутации использовать.

В качестве оператора отбора можно для начала взять турнир, потому что его реализация проста и эффективна.

Для скрещивания подойдет одноточечный или двухточечный оператор, поскольку в результате скрещивания двух двоичных строк этими методами получается допустимая двоичная строка.

В качестве операции мутации можно взять простое инвертирование бита, этот метод хорошо работает для двоичных строк.

Задание условия остановки

Всегда имеет смысл ограничивать количество поколений, чтобы алгоритм не работал вечно. Тем самым мы получаем одно условие остановки. Кроме того, так уж получилось, что мы знаем наилучшее решение в задаче OneMax – двоичная строка из одних единиц со значением приспособленности, равным количеству единиц, – поэтому можно взять это в качестве второго условия остановки. В реальной задаче такая априорная информация обычно неизвестна.

Если хотя бы одно условие выполнено – достигнут предел количества поколений или найдено наилучшее решение, то алгоритм останавливается.

4. Реализация средствами DEAP.

Сначала импортируем необходимые модули из DEAP и еще несколько вспомогательных библиотек:

```
from deap import base
from deap import creator
from deap import tools
import random
import matplotlib.pyplot as plt
```

Затем объявляем несколько констант, содержащих значения параметров самой задачи и генетического алгоритма:

```
# константы задачи
ONE_MAX_LENGTH = 100 # длина подлежащей оптимизации
                        битовой строки

# константы генетического алгоритма
```

```

POPULATION_SIZE = 200 # количество индивидуумов в
популяции
P_CROSSOVER = 0.9 # вероятность скрещивания
P_MUTATION = 0.1 # вероятность мутации индивидуума
MAX_GENERATIONS = 50 # максимальное количество
поколений

```

Важный аспект генетического алгоритма – его вероятностный характер, поэтому в алгоритм нужно внести элемент случайности. Однако на этапе экспериментирования требуется, чтобы результаты были воспроизводимы. Для этого мы задаем какое-нибудь фиксированное начальное значение генератора случайных чисел:

```

RANDOM_SEED = 42
random.seed(RANDOM_SEED)

```

Впоследствии эти строки нужно будет удалить, чтобы при разных прогонах получались разные результаты.

Одним из основных компонентов каркаса DEAP является класс `Toolbox`, который позволяет регистрировать новые функции (или операторы), настраивая поведение существующих функций. В данном случае мы воспользуемся им, чтобы определить оператор `zeroOrOne` путем специализации функции `random.randint(a, b)`. Эта функция возвращает случайное целое число N такое, что $a \leq N \leq b$. Если задать в качестве a и b фиксированные значения 0 и 1, то оператор `zeroOrOne` будет случайным образом возвращать 0 или 1. Во фрагменте ниже мы определяем переменную `toolbox`, а затем используем ее для регистрации оператора `zeroOrOne`:

```

toolbox = base.Toolbox()
toolbox.register("zeroOrOne", random.randint, 0, 1)

```

Далее следует создать класс `Fitness`. Поскольку у нас всего одна цель – сумма цифр, а наша задача – максимизировать ее, то выбираем стратегию `FitnessMax`, задав в кортеже `weights` всего один положительный вес:


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

По соглашению, в DEAP для представления индивидуумов используется класс с именем `Individual`, для создания которого применяется модуль `creator`. В нашем случае базовым классом является `list`, т. е. хромосома представляется списком. Дополнительно в класс добавляется атрибут `fitness`, инициализируемый экземпляром определенного ранее класса `FitnessMax`:

```
creator.create("Individual", list,
fitness=creator.FitnessMax)
```

Следующий шаг нашей программы – регистрация оператора `individualCreator`, который создает экземпляр класса `Individual`, заполненный случайными значениями 0 или 1. Для этого мы настроим ранее определенный оператор `zeroOrOne`. В качестве базового класса используется вышеупомянутый оператор `initRepeat`, специализированный следующими аргументами:

- класс `Individual` в качестве типа контейнера, в который помещаются созданные объекты;
- оператор `zeroOrOne` в качестве функции генерации объектов;
- константа `ONE_MAX_LENGTH` в качестве количества генерируемых объектов (сейчас она равна 100).

Поскольку оператор `zeroOrOne` создает объекты, принимающие случайное значение 0 или 1, то получающийся в результате оператор `individualCreator` заполняет экземпляр `Individual` 100 случайными значениями 0 или 1:

```
toolbox.register("individualCreator",
tools.initRepeat, creator.Individual, toolbox.zeroOrOne,
ONE_MAX_LENGTH)
```

Регистрируем оператор `populationCreator`, создающий список индивидуумов. В его определении также используется оператор `initRepeat` со следующими аргументами:

- класс `list` в качестве типа контейнера;
- оператор `individualCreator`, определенный ранее в качестве функции, генерирующей объекты в списке.

Последний аргумент `initRepeat` – количество генерируемых объектов – здесь не задан. Это означает, что при использовании оператора `populationCreator` мы должны будем указать этот аргумент, т. е. задать размер популяции:

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

Для вычисления приспособленности мы сначала определим свободную функцию, которая принимает экземпляр класса `Individual` и возвращает его приспособленность. В данном случае мы назвали функцию, вычисляющую количество единиц в индивидууме, `oneMaxFitness`. Поскольку индивидуум представляет собой не что иное, как список значений 0 и 1, то на поставленный вопрос в точности отвечает встроенная функция Python `sum()`:

```
def oneMaxFitness(individual):
    return sum(individual), # вернуть кортеж
```

Как уже было сказано, значения приспособленности в DEAP представлены кортежами, поэтому если возвращается всего одно значение, то после него нужно поставить запятую.

Теперь определим оператор `evaluate` – псевдоним только что определенной функции `oneMaxfitness()`. Ниже мы узнаем, что использование псевдонима `evaluate` для вычисления приспособленности – принятое в DEAP соглашение:

```
toolbox.register("evaluate", oneMaxFitness)
```

Генетические операторы обычно создаются как псевдонимы существующих функций из модуля `tools` с конкретными значениями аргументов. В данном случае аргументы будут такими:

- турнирный отбор с размером турнира 3;
- одноточечное скрещивание;

- мутация инвертированием бита.

Обратите внимание на параметр `indpb` функции `mutFlipBit`. Эта функция обходит все атрибуты индивидуума – в нашем случае список значений 0 и 1 – и для каждого атрибута использует значение данного аргумента как вероятность инвертирования (применения логического оператора НЕ) значения атрибута. Это значение не зависит от вероятности мутации, которая задается константой `P_MUTATION`, – мы определили ее выше, но пока не использовали. Вероятность мутации нужна при решении о том, вызывать ли функцию `mutFlipBit` для данного индивидуума в популяции:

```
toolbox.register("select",      tools.selTournament,
tournamentsize=3)

toolbox.register("mate", tools.cxOnePoint)

toolbox.register("mutate",      tools.mutFlipBit,
indpb=1.0/ONE_MAX_LENGTH)
```

Итак, подготовка завершена, все операторы определены, и мы готовы реализовать генетический алгоритм.

Генетический алгоритм может быть реализован следующим образом:

1. Создаем начальную популяцию оператором `populationCreator`, задавая размер популяции `POPULATION_SIZE`. Также инициализируем переменную `generationCounter`, которая понадобится нам позже:

```
population =
toolbox.populationCreator(n=POPULATION_SIZE)

generationCounter = 0
```

2. Для вычисления приспособленности каждого индивидуума в начальной популяции воспользуемся функцией Python `map()`, которая применяет оператор `evaluate` к каждому элементу популяции. Поскольку оператор `evaluate` – это псевдоним функции `oneMaxFitness()`, получающийся итерируемый объект содержит вычисленные значения приспособленности каждого индивидуума. Затем мы преобразуем его в список кортежей:

```
fitnessValues = list(map(toolbox.evaluate,
population))
```

3. Поскольку элементы списка `fitnessValues` взаимно однозначно соответствуют элементам популяции (представляющей собой список индивидуумов), мы можем воспользоваться функцией `zip()`, чтобы объединить их попарно, сопоставив каждому индивидууму его приспособленность:

```
for individual, fitnessValue in zip(population,
fitnessValues):
    individual.fitness.values = fitnessValue
```

4. Далее, так как в нашем случае имеет место приспособляемость всего с одной целью, то извлекаем первое значение из каждого кортежа приспособленности для сбора статистики:

```
fitnessValues = [individual.fitness.values[0] for
individual in population]
```

5. В качестве статистики мы собираем максимальное и среднее значение приспособленности в каждом поколении. Для этого нам понадобятся два списка, создадим их:

```
maxFitnessValues = []
meanFitnessValues = []
```

6. Теперь мы готовы написать главный цикл алгоритма. В самом начале цикла проверяются условия остановки. Одно из них – ограничение на количество поколений, второе – проверка на лучшее возможное решение (двоичная строка из одних единиц):

```
while max(fitnessValues) < ONE_MAX_LENGTH and
generationCounter < MAX_GENERATIONS:
```

7. Затем обновляется счетчик поколений. Он используется в условии остановки и в последующих предложениях печати:

```
generationCounter = generationCounter + 1
```

8. Сердце алгоритма – генетические операторы, которые применяются на следующем шаге. Сначала – оператор отбора `toolbox.select`, который мы

выше определили как турнирный отбор. Поскольку размер турнира был задан в определении оператора, сейчас нам осталось передать только популяцию и ее размер:

```
offspring          =          toolbox.select(population,
len(population))
```

9. Далее отобранные индивидуумы, которые находятся в списке `offspring`, клонируются, чтобы можно было применить к ним следующие генетические операторы, не затрагивая исходную популяцию:

```
offspring = list(map(toolbox.clone, offspring))
```

Заметим, что, несмотря на имя `offspring`, это пока еще клоны индивидуумов из предыдущего поколения, и к ним еще только предстоит применить оператор скрещивания для создания потомков.

10. Следующий генетический оператор – скрещивание. Ранее мы определили его в атрибуте `toolbox.mate` как псевдоним одноточечного скрещивания. Мы воспользуемся встроенной в Python операцией среза, чтобы объединить в пары каждый элемент списка `offspring` с четным индексом со следующим за ним элементом с нечетным индексом. Затем с помощью функции `random()` мы «подбросим монету» с вероятностью, заданной константой `P_CROSSOVER`, и тем самым решим, применять к паре индивидуумов скрещивание или оставить их как есть. И наконец, удалим значения приспособленности потомков, потому что они были модифицированы и старые значения уже не актуальны:

```
for child1, child2 in zip(offspring[::2],
offspring[1::2]):
    if random.random() < P_CROSSOVER:
        toolbox.mate(child1, child2)
        del child1.fitness.values
        del child2.fitness.values
```

Отметим, что функция `mate` принимает двух индивидуумов и модифицирует их на месте, т. е. присваивать им новые значения не нужно.

11. Последний генетический оператор – мутация, ранее мы определили его в атрибуте `toolbox.mutate` как псевдоним инвертирования бита. Мы должны обойти всех потомков и применить оператор мутации с вероятностью `P_MUTATION`. Если индивидиум подвергся мутации, то нужно удалить значение его приспособленности (если оно существует), поскольку оно могло быть перенесено из предыдущего поколения, а после мутации уже не актуально:

```
for mutant in offspring:
    if random.random() < P_MUTATION:
        toolbox.mutate(mutant)
        del mutant.fitness.values
```

12. Те индивидиумы, к которым не применялось ни скрещивание, ни мутация, остались неизменными, поэтому их приспособленности, вычисленные в предыдущем поколении, не нужно заново пересчитывать. В остальных индивидиумах значение приспособленности будет пустым. Мы находим этих индивидиумов, проверяя свойство `valid` класса `Fitness`, после чего вычисляем новое значение приспособленности так же, как делали это ранее:

```
freshIndividuals = [ind for ind in offspring if not
ind.fitness.valid]
freshFitnessValues = list(map(toolbox.evaluate,
freshIndividuals))
for individual, fitnessValue in
zip(freshIndividuals, freshFitnessValues):
    individual.fitness.values = fitnessValue
```

13. После того как все генетические операторы применены, нужно заменить старую популяцию новой:

```
population[:] = offspring
```

14. Прежде чем переходить к следующей итерации, учтем в статистике текущие значения приспособленности. Поскольку приспособленность представлена кортежем (из одного элемента), необходимо указать индекс [0]:

```
fitnessValues = [ind.fitness.values[0] for ind in
population]
```

15. Далее мы вычисляем максимальное и среднее значения, помещаем их в накопители и печатаем сводную информацию:

```
maxFitness = max(fitnessValues)
meanFitness = sum(fitnessValues) / len(population)
maxFitnessValues.append(maxFitness)
meanFitnessValues.append(meanFitness)
print("- Поколение {}: Макс приспособ. = {}, Средняя
приспособ. = {}".format(generationCounter, maxFitness,
meanFitness))
```

16. Дополнительно мы находим индекс (первого) лучшего индивидуума, пользуясь только что найденным значением приспособленности, и распечатываем этого индивидуума:

```
best_index =
fitnessValues.index(max(fitnessValues))
print("Best Individual = ", *population[best_index],
"\n")
```

17. После срабатывания условий остановки накопители статистики можно использовать для построения двух графиков с помощью библиотеки `matplotlib`. Во фрагменте кода ниже рисуется график изменения лучшей и средней приспособленности:

```
plt.plot(maxFitnessValues, color='red')
plt.plot(meanFitnessValues, color='green')
plt.xlabel('Поколение')
plt.ylabel('Макс/средняя приспособленность')
```

```
plt.title('Зависимость максимальной и средней
приспособленности от поколения')
plt.show()
```

Теперь мы можем испытать свой первый генетический алгоритм – запустим его и найдем решение задачи OneMax.

Выполнение программы. В результате выполнения написанной выше программы получается такая распечатка:

```
- Поколение 1: Макс приспособ. = 65.0, Средняя
приспособ. = 53.575
```

```
Лучший индивидуум = 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 0
1 0 0 1 0 1 0 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1 1 1 0 0
1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0 1 0 1 0
1 1 1 0 1 1 0 0 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0
...
```

```
- Поколение 40: Макс приспособ. = 100.0, Средняя
приспособ. = 98.29
```

```
Лучший индивидуум = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Как видим, после смены 40 поколений было найдено решение, содержащее только единицы, для которого приспособленность равна 100. После этого алгоритм остановился. Начальное значение средней приспособленности было равно примерно 53, конечное – 100.

Ниже показан график (рис. 3.5), построенный с помощью matplotlib/

Полный листинг программы в приложении Ж (Ж.1. Листинг программы решение задачи OneMax с помощью DEAP).

А теперь рассмотрим использование встроенных алгоритмов DEAP. Каркас DEAP включает несколько встроенных эволюционных алгоритмов, находящихся в модуле algorithms. Один из них, eaSimple, реализует общую

структуру генетического алгоритма и может заменить большую часть написанного нами кода. Для сбора и печати статистики можно использовать другие полезные объекты DEAP, Statistics и logbook.

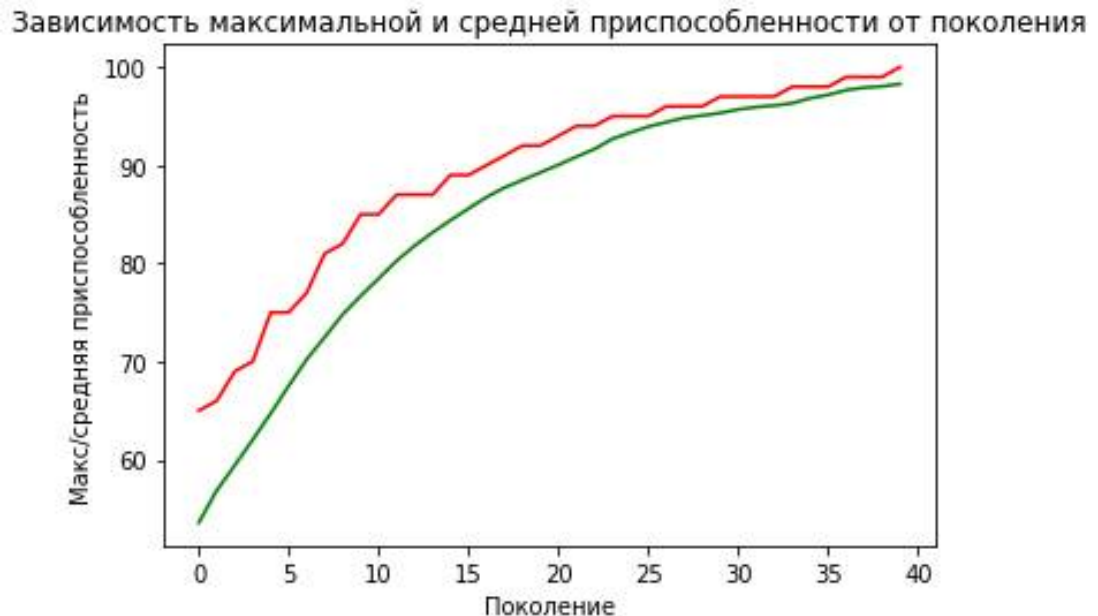


Рис. 3.5. Зависимость максимальной и средней приспособленности от поколения.

Рассмотрим программу, которая реализует то же решение задачи OneMax, но с меньшим объемом кода.

Объект Statistics. Первое изменение относится к способу сбора статистики. Для этой цели мы воспользуемся классом tools.Statistics, предоставляемым DEAP. Он позволяет собирать статистику, задавая функцию, применяемую к данным, для которых вычисляется статистика.

1. Поскольку в нашем случае данными является популяция, зададим функцию, которая извлекает приспособленность каждого индивидуума:

```
stats = tools.Statistics(lambda ind:
ind.fitness.values)
```

2. Теперь можно зарегистрировать функции, применяемые к этим значениям на каждом шаге. В нашем примере это функции NumPy max и mean, но можно регистрировать и другие функции (например, min и std):

```
import numpy
stats.register("max", numpy.max)
```

```
stats.register("avg", numpy.mean)
```

Как мы скоро увидим, собранная статистика возвращается в объекте `logbook` в конце работы программы.

Алгоритм

Теперь можно приступить к основной работе. Для этого нужно всего одно обращение к методу `algorithms.eaSimple`, одному из встроенных в DEAP эволюционных алгоритмов. Этому методу передаются популяция, `toolbox`, объект статистики и другие параметры:

```
population, logbook =
algorithms.eaSimple(population, toolbox,
cxpb=P_CROSSOVER, mutpb=P_MUTATION,
ngen=MAX_GENERATIONS, stats=stats, verbose=True)
```

Метод `algorithms.eaSimple` предполагает, что в `toolbox` уже зарегистрированы операторы `evaluate`, `select`, `mate` и `mutate`, – и мы действительно сделали это в первоначальной версии программы. Условие остановки задается с помощью параметра `ngen` – максимального количества поколений.

Объект `logbook`. Метод `algorithms.eaSimple` возвращает два объекта – конечную популяцию и объект `logbook`, содержащий собранную статистику. Интересующую нас статистику можно извлечь методом `select()` и использовать для построения графиков, как и раньше:

```
maxFitnessValues, meanFitnessValues =
logbook.select("max", "avg")
plt.plot(maxFitnessValues, color='red')
plt.plot(meanFitnessValues, color='green')
plt.xlabel('Generation')
plt.ylabel('Max / Average Fitness')
plt.title('Max and Average Fitness over
Generations')
plt.show()
```

Теперь можно запустить эту сильно похудевшую версию программы.

Выполнение программы

При запуске программы с теми же параметрами, что и раньше, выдается такая распечатка:

```
gen nevals max avg
0 200 61 49.695
1 193 65 53.575
...
39 192 99 98.04
40 173 100 98.29
...
49 187 100 99.83
50 184 100 99.89
```

Она автоматически генерируется методом `algorithms.eaSimple` в соответствии с переданным ему объектом статистики, если аргумент `verbose` равен `True`.

Численно результаты похожи на те, что печатались в предыдущей версии программы, с двумя отличиями:

- в распечатке имеется строка для поколения 0, раньше мы ее не включали;
- алгоритм работает на протяжении всех 50 поколений, поскольку это было единственное условие остановки, тогда как в первоначальной программе имелось дополнительное условие – обнаружение наилучшего решения (известного заранее), – благодаря которому цикл остановился на 40-м поколении.

На графике наблюдается то же поведение, что и раньше. От предыдущего графика новый отличается тем, что продолжается до 50-го поколения, хотя наилучший результат получен уже в 40-м поколении.

Начиная с 40-го поколения максимальное значение приспособленности перестает изменяться, а среднее продолжает расти, пока в конце концов не

станет почти равным максимальному. Это означает, что в конце прогона почти все индивидуумы стали равны лучшему.

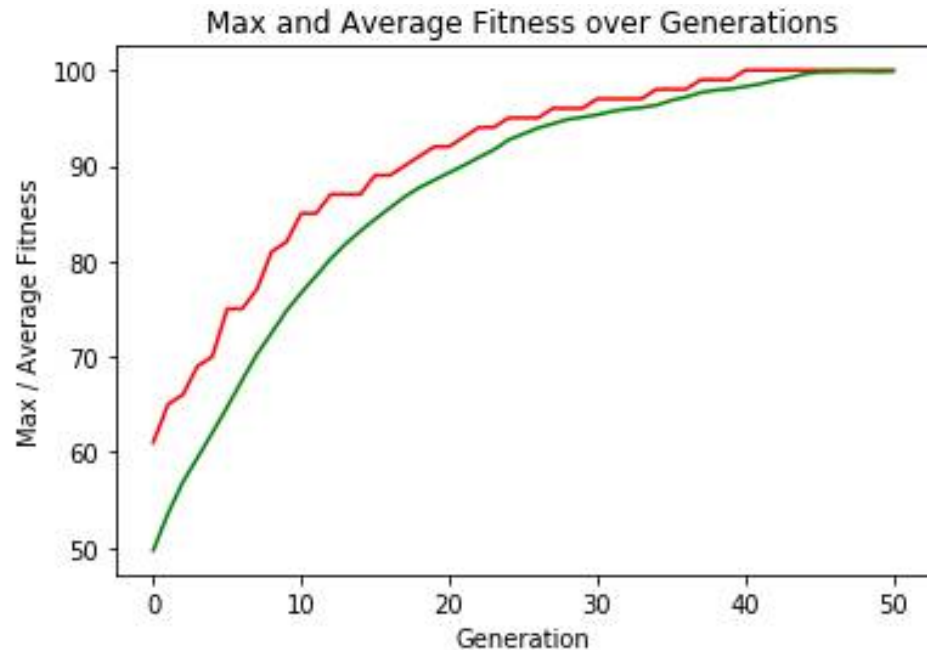


Рис. 3.6. Зависимость максимальной и средней приспособленности от поколения

Зал славы. У встроенного метода `algorithms.eaSimple` есть еще одна возможность – зал славы (hall of fame, сокращенно hof). Класс `HallOfFame`, находящийся в модуле `tools`, позволяет сохранить лучших индивидуумов, встретившихся в процессе эволюции, даже если вследствие отбора, скрещивания и мутации они были в какой-то момент утрачены. Зал славы поддерживается в отсортированном состоянии, так что первым элементом всегда является индивидуум с наилучшим встретившимся значением приспособленности.

Чтобы добавить зал славы, в написанный выше код нужно внести следующие изменения.

1. Определим константу, равную количеству индивидуумов, которых мы хотим хранить в зале славы:

```
HALL_OF_FAME_SIZE = 10
```

2. Прежде чем вызывать алгоритм `eaSimple`, создадим объект `HallOfFame` такого размера:

```
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)
```

3. Объект HallOfFame передается алгоритму eaSimple, который самостоятельно обновляет его в процессе выполнения генетического алгоритма:

```

        population,                                logbook                                =
algorithms.eaSimple(population,                                toolbox,
cxpb=P_CROSSOVER,                                mutpb=P_MUTATION,
ngen=MAX_GENERATIONS,      stats=stats,      halloffame=hof,
verbose=True)

```

4. По завершении алгоритма атрибут items объекта HallOfFame можно использовать для доступа к списку помещенных в зал славы индивидуумов:

```

print("Индивидуумы в зале славы = ", *hof.items,
sep="\n")

print("Лучший индивидуум = ", hof.items[0])

```

Результаты

Индивидуумы в зале славы =

```

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

```

...
...

```

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

```

```

Лучший индивидуум = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

```

Полный листинг программы в приложении Ж. (Ж.2. файл ГА_1_2.ipynb)

3.4. Представление вещественных решений в двоичной форме

В первом примере (нахождения максимального значения функции на отрезке) мы рассматривали только целочисленные решения. Обобщим ГА на случай вещественных чисел на следующем примере, в котором для функции $f(x) = (1,85 - x) \cdot \cos(3,5x - 0,5)$, представленной на рис.1.7 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 $x \in [-10, +10]$.

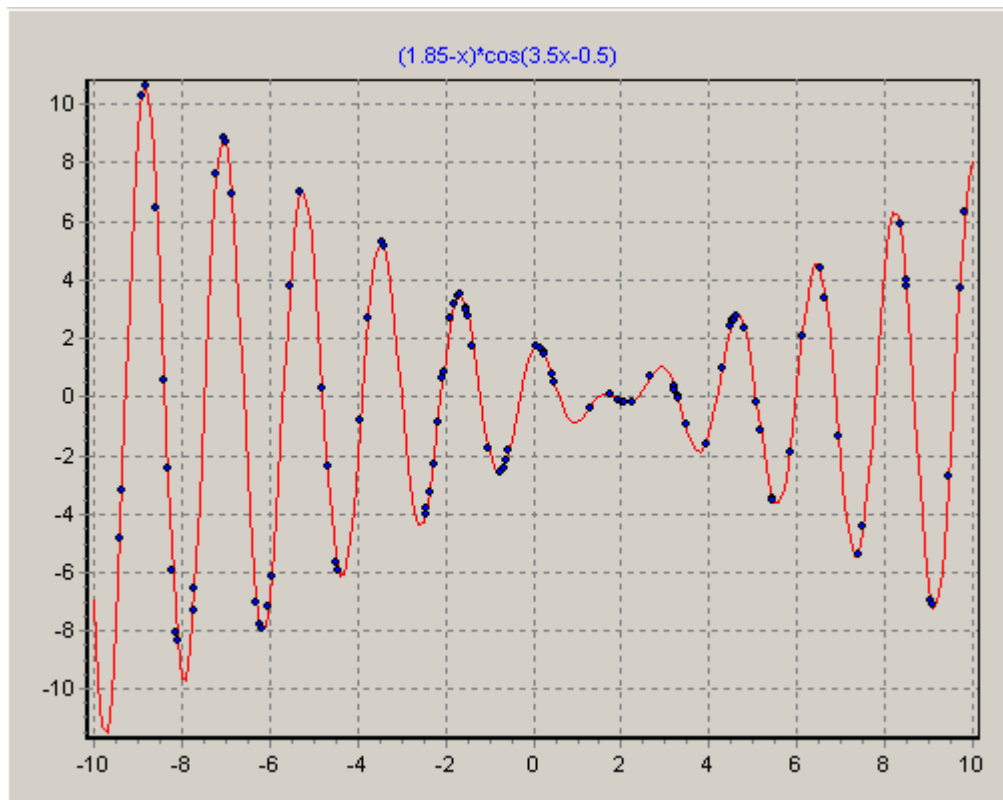


Рис.1.7. Пример функции с популяцией особей в начале эволюции

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 32768 частей и обеспечить необходимую точность. Отображение из двоичного представления $(b_{14}b_{13}...b_0)$ ($b_i \in \{0,1\}$) в вещественное число из отрезка $[-10, +10]$ выполняется в два шага.

1) Перевод двоичного числа в десятичное:

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

2) Вычисление соответствующего вещественного числа x :

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

где -10 левая граница области решения.

Естественно хромосомы (0000000000000000) и (1111111111111111) представляют границы отрезка -10 и $+10$ соответственно.

Очевидно, при данном двоичном представлении вещественных чисел можно использовать классический простой ГА. На рис. 3.7–3.10 представлено расположение особей - потенциальных решений на различных этапах ГА в процессе поиска решения.

На рисунке 3.7 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.

3.5. Решение задачи оптимизации одномерной функции с помощью DEAP

В отличие от задачи OneMax, нам необходимо вычислять размер особи, в зависимости от диапазона изменения функции и точности вычислений.

Ведем переменные:

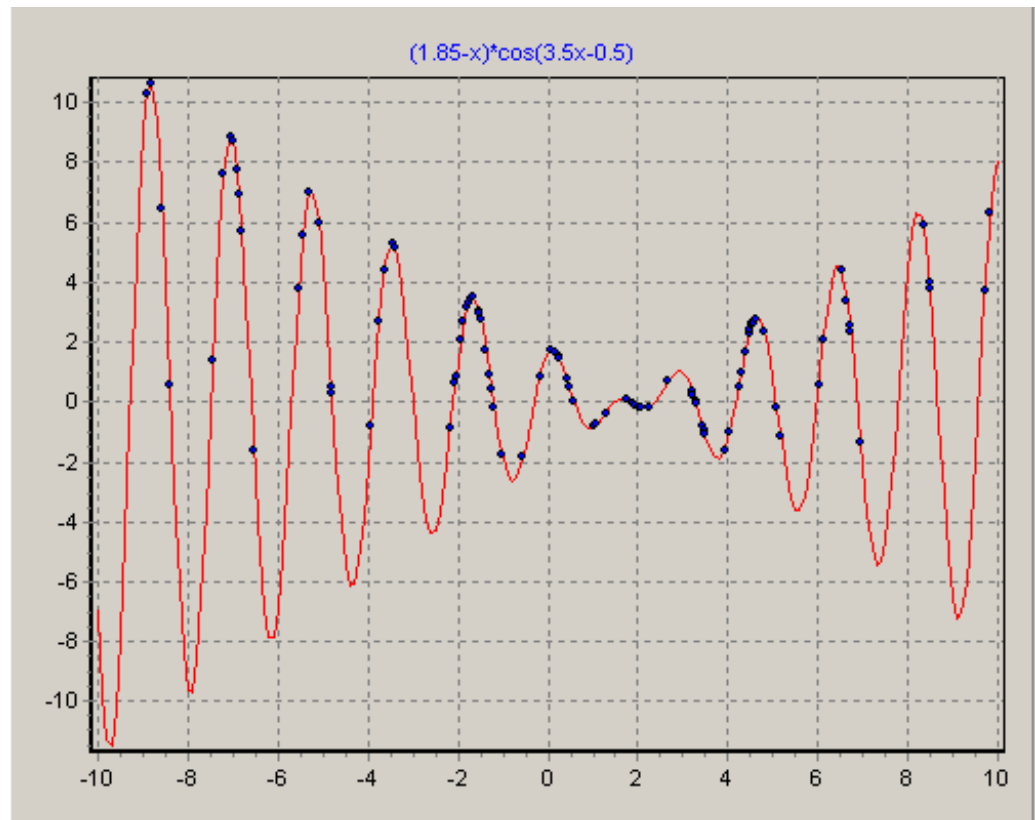


Рис.1.8. Начальная «конденсация» особей популяции в окрестностях экстремумов

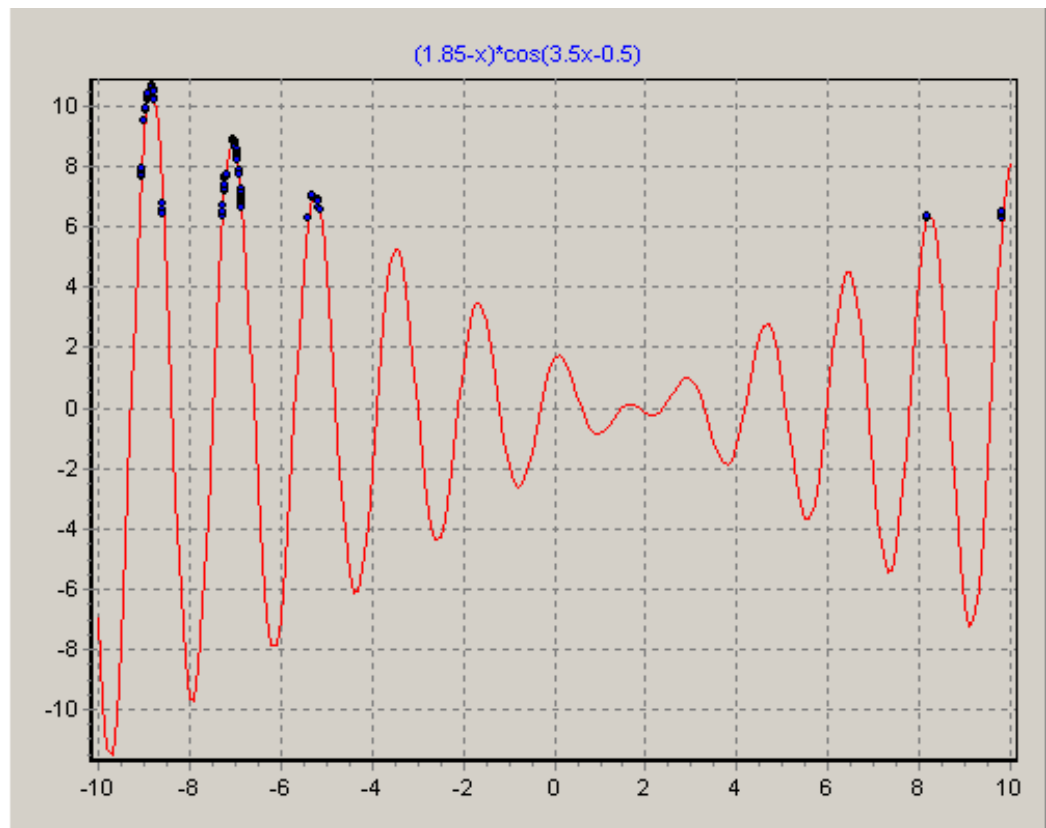


Рис.1.9. «Конденсация» особей в окрестностях экстремумов

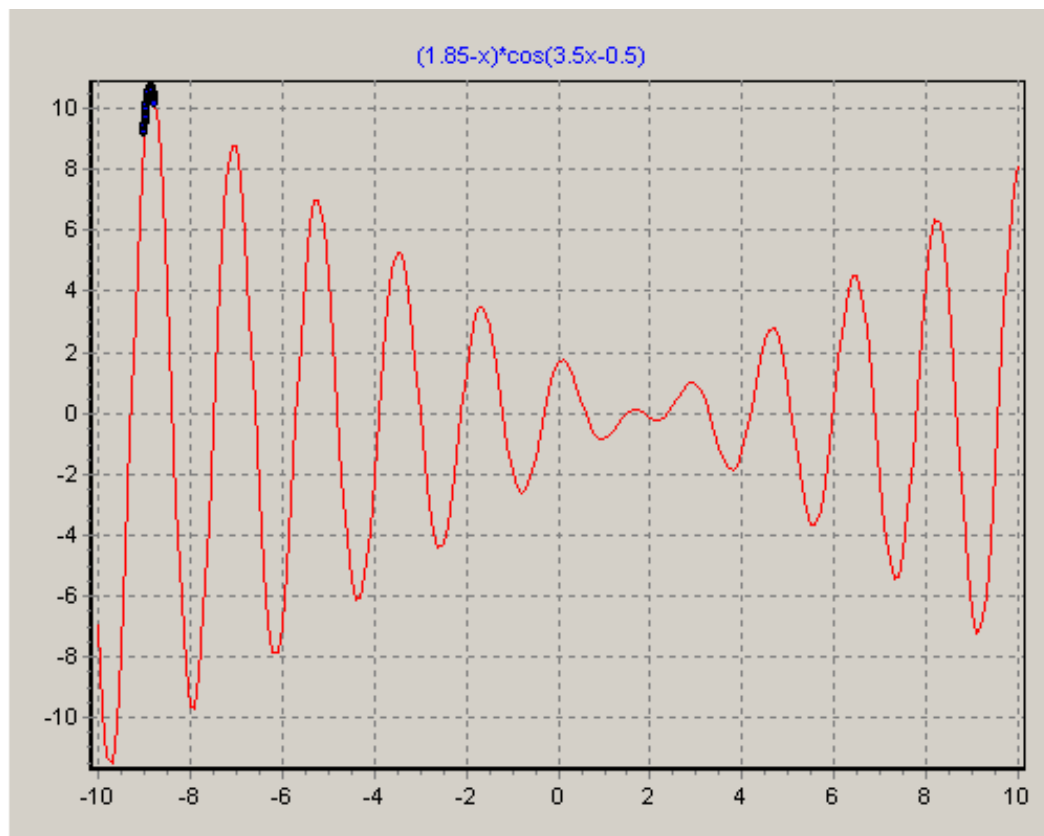


Рис.1.10. Положение особей популяции в конце эволюции

```
BOUND_LOW, BOUND_UP = -10.0, 10.0 # границы
```

```
EPS = 0.001 # точность
```

Напишем функцию вычисления степени 2 целого числа:

```
def power_of_two(n):
    sqr=1
    r=0
    while(sqr<=n):
        #print(r, ' ', sqr, ' ')
        sqr=sqr*2
        r+=1
    return(r)
```

Вычислим необходимую длину особи и обозначим ее переменной ONE_MAX_LENGTH:

```
def MAX_LENGTH(low,up,eps):
    N=(BOUND_UP-BOUND_LOW)/EPS
```

```

    return power_of_two(N)

ONE_MAX_LENGTH = MAX_LENGTH(BOUND_LOW, BOUND_UP, EPS)

```

В качестве функции оптимизации возьмем функцию из примера на рисунке 3.7. Соответственно функция расчета функции приспособленности будет иметь вид:

```

def FF(individual, low, up, eps):
    x =
    int(str(individual).replace(',', '').replace('[' , '').replace('] ', '').replace(' ', ''), 2)
    h= (up-low) / (2**ONE_MAX_LENGTH-1)
    x1=low+x*h
    f=(1.85-x1)*np.cos(3.5*x1-0.5)
    return f, # вернуть кортеж

```

При расчете значения функции, сначала выполняется перевод двоичного числа в десятичное (особь генерируется последовательностью 0 и 1). Затем выполняется смещение относительно нижней границы диапазона с учетом поправки на реальное значение величины равной разнице между двумя последовательными значениями особей в двоичном представлении. После выполняется расчет функции, от полученного значения особи.

Другие части кода из примера Ж.2 в приложении Ж можно оставить без изменений.

3.6. Порядок выполнения лабораторной работы

1. Изучить теоретический материал.
2. Ознакомиться с вариантами кодирования хромосомы.
3. Рассмотреть способы выполнения операторов репродукции, кроссинговера и мутации.
4. Разработать простой генетический алгоритм для нахождения оптимума заданной по варианту функции одной переменной (таб. 3.1). Вид экстремума: для четных – минимум, для не четных – максимум.

5. Выполнить программную реализацию разработанного алгоритма на языке высокого уровня. Рекомендованный язык Python с применением DEAP. Предусмотреть возможность просмотра процесса поиска решения.

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

- число особей в популяции;
- вероятность кроссинговера, мутации.

7. Сравнить найденное ГА решение с действительным.

Таблица 3.1. Индивидуальные задания.

	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	$\cos(3x-15) \cdot x$	$x \in [-9.6, 9.1]$
9	$\sin(x) / (1 + \exp(-x))$	$x \in [0.5, 10]$
10	$\cos(x) / (1 + \exp(-x))$	$x \in [0.5, 10]$
11	$(\exp(x) - \exp(-x)) \cos(x) / (\exp(x) + \exp(-x))$	$x \in [-5, 5]$
12	$(\exp(-x) - \exp(x)) \cos(x) / (\exp(x) + \exp(-x))$	$x \in [-5, 5]$
13	$\cos(x-0.5) / \text{abs}(x)$	$x \in [-10, 0), (0, 10]$, min
14	$\cos(2x) / \text{abs}(x-2)$	$x \in [-10, 2), (2, 10]$, max

3.7. 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 (выборочно графиков);
6. Диаграммы исследованных зависимостей параметров генетического алгоритма на сходимость.

Лабораторная работа №4

Тема: решения задачи оптимизации многомерной функции генетическим алгоритмом.

Цель: изучение основных принципов ГА; научиться использовать библиотеку `dear` для оптимизации многомерной функции.

4.1. 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 `real-coded` 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.

Появление новых особей в популяции ГА обеспечивают несколько биологических операторов: отбор, скрещивание и мутация. В качестве операторов отбора особей в родительскую пару здесь подходят любые известные: 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.

4.2. 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 $C_1=(c_{11},c_{21},\dots,c_{n1})$ и $C_2=(c_{12},c_{22},\dots,c_{n2})$ – две хромосомы, выбранные оператором селекции для скрещивания. После формулы для некоторых кроссоверов приводится рисунок – геометрическая интерпретация его работы. Предполагается, что $c_{k1} \leq c_{k2}$ и $f(C_1) \geq f(C_2)$.

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

Простейший кроссовер (simple crossover): случайным образом выбирается число k из интервала $\{1,2,\dots,n-1\}$ и генерируются два потомка $H_1=(c_{11},c_{21},\dots,c_{k1},c_{k+12},\dots,c_{n2})$ и $H_2=(c_{12},c_{22},\dots,c_{k2},c_{k+11},\dots,c_{n1})$.

Арифметический кроссовер (arithmetical crossover): создаются два потомка $H_1=(h_{11},\dots,h_{n1})$, $H_2=(h_{12},\dots,h_{n2})$, где $h_{k1}=w \cdot c_{k1} + (1-w) \cdot c_{k2}$, $h_{k2}=w \cdot c_{k2} + (1-w) \cdot c_{k1}$, $k=1, \dots, n$, w либо константа (равномерный арифметический кроссовер) из интервала $[0;1]$, либо изменяется с увеличением эпох (неравномерный арифметический кроссовер).

Геометрический кроссовер (geometrical crossover): создаются два потомка $H_1=(h_{11},\dots,h_{n1})$, $H_2=(h_{12},\dots,h_{n2})$, где $h_{k1}=(c_{k1})w \cdot (c_{k2})(1-w)$, $(c_{k2})w \cdot (c_{k1})(1-w)$, где w – случайное число из интервала $[0;1]$.

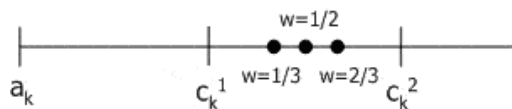


Рис. 4.1. Арифметический кроссовер

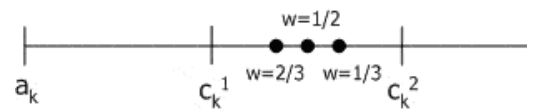


Рис. 4.2. Геометрический кроссовер

Смешанный кроссовер (blend, BLX-alpha crossover): генерируется один потомок $H=(h_1,\dots,h_k,\dots,h_n)$, где h_k – случайное число из интервала $[c_{\min}-I \cdot \alpha, c_{\max}+I \cdot \alpha]$, $c_{\min}=\min(c_{k1},c_{k2})$, $c_{\max}=\max(c_{k1},c_{k2})$, $I=c_{\max}-c_{\min}$. BLX-0.0 кроссовер превращается в плоский.

Линейный кроссовер (linear crossover): создаются три потомка $H_q=(h_{1q},\dots,h_{kq},\dots,h_{nq})$, $q=1,2,3$, где $h_{k1}=0.5 \cdot c_{k1} + 0.5 \cdot c_{k2}$, $h_{k2}=1.5 \cdot c_{k1} - 0.5 \cdot c_{k2}$,

$h_{k3} = -0.5 \cdot c_{k1} + 1.5 \cdot c_{k2}$. На этапе селекции в этом кроссовере отбираются два наиболее сильных потомка.

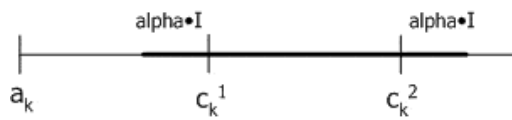


Рис. 4.3. Смешанный кроссовер.

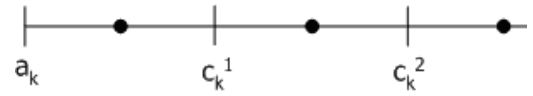


Рис. 4.4. Линейный кроссовер.

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

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

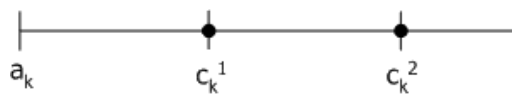


Рис. 4.5. Дискретный кроссовер

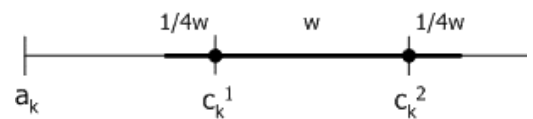


Рис. 4.6. Расширенный линейный кроссовер.

Эвристический кроссовер (Wright's heuristic crossover). Пусть C_1 – один из двух родителей с лучшей приспособленностью. Тогда $h_k = w \cdot (c_{k1} - c_{k2}) + c_{k1}$, w – случайное число из интервала $[0; 1]$.

Нечеткий кроссовер (fuzzy recombination, FR-d crossover): создаются два потомка $H_1 = (h_{11}, \dots, h_{n1})$, $H_2 = (h_{12}, \dots, h_{n2})$. Вероятность того, что в i -том гене появится число v_i , задается распределением $p(v_i) \in \{F(c_{k1}), F(c_{k2})\}$, где $F(c_{k1}), F(c_{k2})$ – распределения вероятностей треугольной формы (треугольные нечеткие функции принадлежности) со следующими свойствами ($c_{k1} \leq c_{k2}$ и $I = |c_{k1} - c_{k2}|$) табл. 4.1.

Таблица 4.1.

Распределение вероятностей	Минимум	Центр	Максимум
$F(c_k^1)$	$c_k^1 - d \cdot I$	c_k^1	$c_k^1 + d \cdot I$
$F(c_k^2)$	$c_k^2 - d \cdot I$	c_k^2	$c_k^2 + d \cdot I$

Параметр d определяет степень перекрытия треугольных функций принадлежности, по умолчанию $d=0.5$.

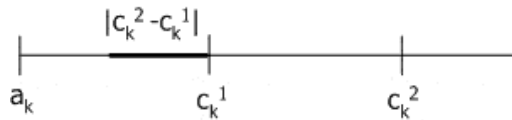


Рис. 4.7. Эвристический кроссовер.

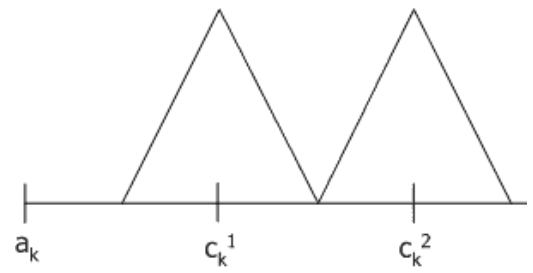


Рис. 4.8. Нечеткий кроссовер.

Мин-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:

$$C_1^{t+1} = \alpha \cdot C_w^t + (1 - \alpha) \cdot C_v^t \quad (4.1)$$

$$C_2^{t+1} = \alpha \cdot C_v^t + (1 - \alpha) \cdot C_w^t \quad (4.2)$$

$$C_3^{t+1} = \min\{c_k, c'_k\} \quad (4.3)$$

$$C_4^{t+1} = \max\{c_k, c'_k\} \quad (4.4)$$

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 (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 вероятностей 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 β :

$$P(\beta) = \begin{cases} 0.5(n+1)\beta^n, & \text{при } \beta \leq 1, \\ 0.5(n+1)\beta^{-(n+2)}, & \text{при } \beta > 1. \end{cases} \quad (4.5)$$

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:

$$\beta(u) = \begin{cases} (2u)^{\frac{1}{n+1}}, & \text{при } u(0,1) \leq 0.5, \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{n+1}}, & \text{при } u(0,1) > 0.5. \end{cases} \quad (4.6)$$

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

На рисунке 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.

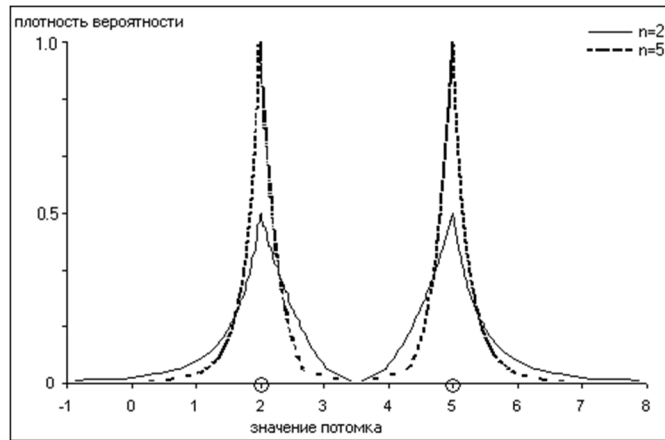


Рис. 4.9. SBX кроссовер.

4.3. 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

$$c'_k = \begin{cases} c_k + \Delta(t, c_{kr} - c_k), & \text{при } a = 0, \\ c_k - \Delta(t, c_k - c_{kl}), & \text{при } a = 1, \end{cases} \quad (4.7)$$

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:

$$\Delta(t, y) = y(1 - r^{(1 - \frac{t}{T})^b}) \quad (4.8)$$

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.

4.4. Применение DEAP для реализации генетического алгоритма с вещественным представлением особи

DEAP предлагает несколько готовых реализаций в модулях `crossover` и `mutation` для вещественных генетических операторов:

- `cxBlend()` – реализация скрещивания смешением, коэффициент α передается в параметре `alpha`;
- `cxSimulatedBinary()` – реализация имитации двоичного скрещивания, величина η передается в параметре `eta`;
- `mutGaussian()` – реализация нормально распределенной мутации, среднее и стандартное отклонение передаются в аргументах `mu` и `sigma`.

Кроме того, поскольку непрерывные функции обычно оптимизируются в ограниченной области, а не во всем пространстве, DEAP предоставляет два оператора, которые принимают параметры, описывающие границу, и гарантируют, что результирующие индивидуумы не выйдут за эти границы:

- `cxSimulatedBinaryBounded()` – ограниченный вариант оператора `cxSimulatedBinary()`, принимающий аргументы `low` и `up` – нижнюю и верхнюю границы области поиска соответственно;
- `mutPolynomialBounded()` – ограниченный оператор мутации, в котором распределение вероятностей задается полиномиальной (а не гауссовой) функцией. Этот оператор также принимает аргументы `low` и `up` – нижнюю и верхнюю границы области поиска. Кроме того, он принимает коэффициент скученности (параметр `eta`) – чем он больше, тем ближе мутант к исходному значению. Соответственно, при малых значениях `eta` мутант будет сильно отличаться от оригинала.

Рассмотрим на примере оптимизации функции `Eggholder`. Функция `Eggholder` (она называется так потому, что ее форма напоминает подставку для яиц) часто используется для тестирования алгоритмов оптимизации.

Нахождение единственного глобального минимума этой функции считается трудной задачей из-за большого количества локальных минимумов.

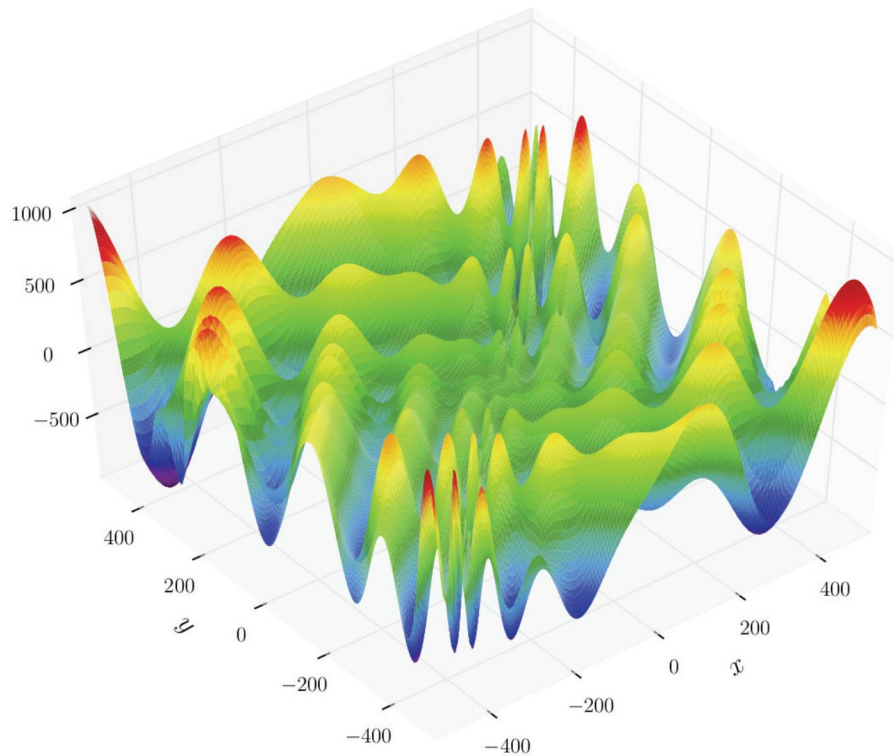


Рис. 4.10. Функция Eggholder.

Математически функция описывается выражением:

$$f(x, y) = -(y + 47) \cdot \sin \sqrt{\left| \frac{x}{2} + (y + 47) \right|} - x \cdot \sin \sqrt{|x - (y + 47)|}$$

и обычно вычисляется в области поиска, ограниченной интервалом $[-512, 512]$ по каждому измерению.

Известно, что глобального минимума эта функция достигает в точке $x=512$, $y = 404.2319$, в которой равна -959.6407 .

Для оптимизации функции Eggholder с помощью генетического алгоритма выполним следующие основные шаги программы.

1. Сначала определяются константы, а именно: количество измерений 2 (т. к. функция определена на плоскости xy) и границы области:

```
DIMENSIONS = 2 # количество измерений
BOUND_LOW,  BOUND_UP  = -512.0,  512.0 # границы,
одинаковые для всех измерений
```

2. Поскольку мы работаем с вещественными числами, ограниченными некоторой областью, далее определим вспомогательную функцию, порождающую числа с плавающей точкой, равномерно распределенные в этой области.

В этой функции предполагается, что верхняя и нижняя границы по всем измерениям одинаковы.

```
def randomFloat(low, up):
    return [random.uniform(l, u) for l, u in zip([low]
*DIMENSIONS, [up] * DIMENSIONS)]
```

3. Определяем переменную toolbox. Создаем класс Fitness. Поскольку наша задача – минимизация, то выбираем стратегию FitnessMin:

```
toolbox = base.Toolbox()
# define a single objective, minimizing fitness
strategy:
    creator.create("FitnessMin",          base.Fitness,
weights=(-1.0,))
# create the Individual class based on list:
    creator.create("Individual",          list,
fitness=creator.FitnessMin)
```

Далее определим оператор attrFloat. В нем только что определенная функция используется для порождения одного случайного числа с плавающей точкой в заданном диапазоне. Этот оператор будет затем использован оператором individualCreator для создания случайных индивидуумов. А тот, в свою очередь, – оператором populationCreator, порождающим заданное количество индивидуумов.

```
toolbox.register("attrFloat",          randomFloat,
BOUND_LOW, BOUND_UP)
    toolbox.register("individualCreator",
tools.initIterate, creator.Individual,
toolbox.attrFloat)
```

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

4. Мы собираемся минимизировать функцию Eggholder, поэтому ее и будем использовать как функцию приспособленности. Поскольку индивидуум представлен списком чисел с плавающей точкой длины 2, выделим из него значения x и y и вычислим функцию:

```
def eggholder(individual):
    x = individual[0]
    y = individual[1]
    f = (-(y + 47.0) * np.sin(np.sqrt(abs(x/2.0 + (y
+ 47.0)))) - x * np.sin(np.sqrt(abs(x - (y + 47.0))))))
    return f, # вернуть кортеж
toolbox.register("evaluate", eggholder)
```

5. Далее разберемся с генетическими операторами. Поскольку оператор отбора не зависит от типа индивидуума, а у нас уже есть опыт использования турнирного отбора размера 2 в сочетании с элитизмом, то и будем продолжать в том же духе. С другой стороны, операторы скрещивания и мутации нужно специализировать для чисел с плавающей точкой в заданных границах, поэтому воспользуемся предоставленным DEAP операторами `cxSimulatedBinaryBounded` для скрещивания и `mutPolynomialBounded` для мутации, определив заранее вероятности выполнения этих операторов и другие параметры ГА:

```
# Генетические операторы
toolbox.register("select", tools.selTournament,
tournsize=2)

toolbox.register("mate",
tools.cxSimulatedBinaryBounded, low=BOUND_LOW,
up=BOUND_UP, eta=CROWDING_FACTOR)
```

```

    toolbox.register("mutate",
tools.mutPolynomialBounded, low=BOUND_LOW, up=BOUND_UP,
eta=CROWDING_FACTOR, indpb=1.0/DIMENSIONS)

```

6. Как и раньше, воспользуемся модифицированным вариантом простого алгоритма генетического поиска, встроенного в DEAP, в который мы добавили элитизм, – будем сохранять лучших индивидуумов (в зале славы) и копировать их в следующее поколение без изменения:

```

    # create initial population (generation 0):
    population =
toolbox.populationCreator(n=POPULATION_SIZE)

    # prepare the statistics object:
    stats = tools.Statistics(lambda ind:
ind.fitness.values)
    stats.register("min", np.min)
    stats.register("avg", np.mean)
    # define the hall-of-fame object:
    hof = tools.HallOfFame(HALL_OF_FAME_SIZE)
    population, logbook =
algorithms.eaSimple(population, toolbox,
cxpb=P_CROSSOVER, mutpb=P_MUTATION,
ngen=MAX_GENERATIONS, stats=stats, halloffame=hof,
verbose=True)

```

7. Начнем с задания параметров генетического алгоритма. Поскольку оптимизировать функцию Eggholder нелегко, зададим относительно большую популяцию, при небольшом числе измерений на это можно пойти:

```

# Параметры генетического алгоритма
POPULATION_SIZE = 300
P_CROSSOVER = 0.9
P_MUTATION = 0.1
MAX_GENERATIONS = 500

```

```
HALL_OF_FAME_SIZE = 30
```

8. В дополнение к предыдущим параметрам появляется один новый – коэффициент скученности (η), задействованный в операциях скрещивания и мутации:

```
CROWDING_FACTOR = 20.0
```

Можно также определить отдельные коэффициенты скученности для скрещивания и для мутации.

Вот теперь мы готовы запустить программу. Ниже показаны результаты, полученные с заданными параметрами:

```
-- Best Individual = [512.0, 404.231805757432]
```

```
-- Best Fitness = -959.6406627208505
```

Это означает, что мы нашли результат, который очень близок к глобальному минимуму.

Из показанных ниже графиков статистики видно, что алгоритм сразу же нашел какой-то локальный минимум, а затем производил небольшие улучшения, пока в конечном итоге не вышел на полученный результат.

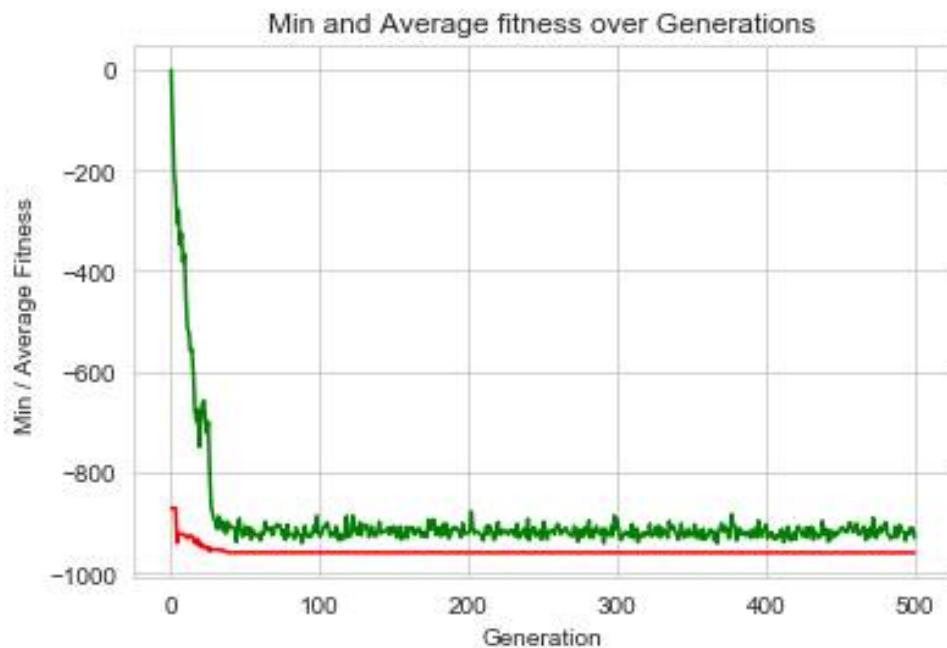


Рис. 4.11. Статистика оптимизации функции Eggholder.

4.5. Порядок выполнения лабораторной работы

1. Изучить теоретический материал.
2. Разработать генетический алгоритм нахождения оптимума функции согласно таблице вариантов, в приложении И.
3. Выполнить программную реализацию разработанного алгоритма на языке высокого уровня. Рекомендованный язык Python с применением DEAP. Предусмотреть возможность просмотра процесса поиска решения.
4. Для $n=2$ вывести на экран график данной функции с указанием найденного экстремума. Предусмотреть возможность просмотра процесса поиска решения.
5. Исследовать зависимость числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:
 - число особей в популяции;
 - вероятность кроссинговера, мутации.

Критерием остановки вычислений выбирать повторение лучшего результата заданное количество раз и / или достижение популяцией определенного возраста (например, 100 эпох).

4.6. 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 (выборочно графиков).
6. Диаграммы исследованных зависимостей.

Лабораторная работа №5

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

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

5.1. 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 $\bar{\sigma}$.

5.2. 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: $\{X_{1\min}, X_{1\max}\}, \{X_{2\min}, X_{2\max}\}, \dots, \{X_{P\min}, X_{P\max}\}$, установить номер поколения (итерации) $t=0$; задать стандартное отклонение σ_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 .

5.3. 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 (μ родителей производят 1 потомка);
- $(\mu+\lambda)$ – ЭС, где μ -родителей производят λ -потомков и отбор μ лучших представителей производится среди объединенного множества ($(\mu+\lambda)$ особей) родителей и потомков;
- (μ, λ) – ЭС, где μ -особей родителей порождает λ -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 σ не является постоянным, но изменяется по некоторому детерминированному алгоритму.

5.4. 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  $m + \lambda$ .

```

5.5. Порядок выполнения лабораторной работы

1. Разработать ЭС нахождения оптимума функции из лабораторной работы 2.
2. Выполнить программную реализацию разработанного алгоритма на языке высокого уровня. Рекомендованный язык Python с применением DEAP. 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. Сравнить результат с полученным в лабораторной работе №2.

5.6. 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.

Лабораторная работа №6

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

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

6.1. 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.

6.2. 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:

$$x_i(t + 1) = x_i(t) + v_i(t + 1) \quad (6.1)$$

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:

$$v_{ij}(t + 1) = v_{ij}(t) + c_1 r_1(t) [y_{ij}(t) - x_{ij}(t)] + c_2 r_2(t) [\hat{y}_j(t) - x_{ij}(t)] \quad (6.2)$$

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:

$$y_i = \begin{cases} y_i(t) & \text{if } f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & \text{if } f(x_i(t+1)) < f(y_i(t)) \end{cases} \quad (6.3)$$

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

$$\hat{y}_j(t) \in \{y_0(t), \dots, y_{n_s}(t)\} | f(\hat{y}_j(t)) = \min \{f(y_0(t)), \dots, f(y_{n_s}(t))\} \quad (6.4)$$

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.

6.3. Глобальный роевой алгоритм

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  $i=1, \dots, ns$  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 6.1 – Эффект социальной компоненты.

6.4. 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.6.2.

На рис. 6.2 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 роу. 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.

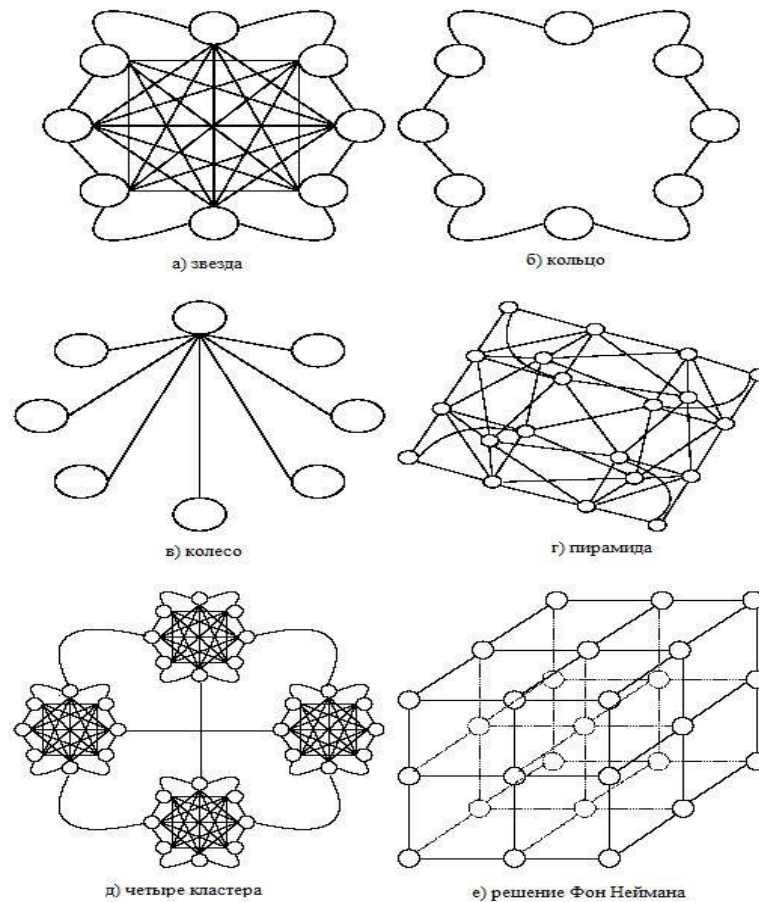


Рисунок 6.2 – Типовые сетевые структуры

На следующем рис. 6.6 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. 6.2 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. 6.2 г).

Далее на рис. 6.2 д) представлена четырех-кластерная социальная структура, в которой четыре кластера (клики) связаны друг с другом двумя соединениями.

Наконец, на рис. 6.2 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:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(y(t) - x_{ij}(t) + c_2 r_{2j}(t)[\widehat{y}_{lj}(t) - x_{ij}(t)] \quad (6.5)$$

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.

$$\widehat{y}_i(t+1) \in \{N_i | f(\widehat{y}_i(t+1)) = \min\{f(x)\}, \forall x \in N_i\} \quad (6.6)$$

где

$$N_i = \{y_{i-nN_i}(t), y_{i-nN_i+1}(t), \dots, y_{i-1}(t), y_i(t), y_{i+1}(t), \dots, y_{i+nN_i}(t)\} \quad (6.7)$$

при числе соседей nN_i .

Здесь локальная лучшая позиция относится к лучшей позиции соседнего окружения.

6.5. Локальный роевой алгоритм

Создание инициализация nx-мерного роя;

repeat

for каждой частицы i=1,...,ns do

//определить персональную лучшую позицию

If f(xi)<f(yi) then

yi=xi;

end

//определить лучшую позицию окружения

If f(yi) < f(yi) then f(yi) = yi;

end

end

for каждой частицы i=1,...,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.

6.6. 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.

6.7. Порядок выполнения лабораторной работы

1. Изучить теоретический материал.
2. Разработать роевый алгоритм для нахождения оптимума заданной по варианту функции в лабораторной работе №2. 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.

6.8. 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).

Лабораторная работа №7

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

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

7.1. 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.

7.2. 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. 7.1. С каждой дугой, соединяющей вершины (i,j) ассоциируем концентрацию феромона τ_{ij} .

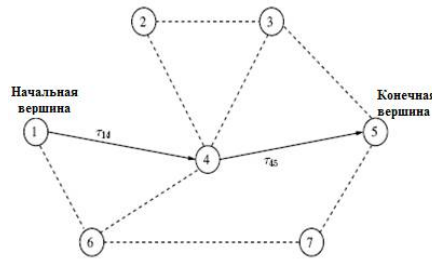


Рис. 7.1. Пример графа.

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:

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha(t)}{\sum_{j \in N_i^k} \tau_{ij}^\alpha(t)}, & \text{если } j \in N_i^k \\ 0, & \text{если } j \notin N_i^k \end{cases} \quad (7.1)$$

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) α - положительная константа, которая определяет влияние концентрации феромона. Очевидно большие значения α 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:

$$\Delta\tau_{ij}^k(t) \propto \frac{1}{L^k(t)} \quad (7.2)$$

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:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t) \quad (7.3)$$

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 (7.2) 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 (7.2) 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 (7.3);
  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 (7.4);
    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:

$$\tau_{ij}(t) \leftarrow (1 - \rho)\tau_{ij}(t) \quad (7.4)$$

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

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

7.3. Порядок выполнения лабораторной работы

1. Разработать МА для решение задачи поиска гамильтонова пути. Индивидуальное заданию выбирается по таблице К.1 в приложении К 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.

7.4. 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> .

Описание HTML тегов

Тег	Описание
<!--...-->	Используется для добавления комментариев.
<!DOCTYPE>	Объявляет тип документа и предоставляет основную информацию для браузера — его язык и версия.
<a>	Создаёт гипертекстовые ссылки.
<abbr>	Определяет текст как аббревиатуру или акроним. Поясняющий текст задаётся с помощью атрибута title.
<address>	Задаёт контактные данные автора/владельца документа или статьи. Отображается в браузере курсивом.
<area>	Представляет собой гиперссылку с текстом, соответствующей определенной области на карте-изображении или активную область внутри карты-изображения. Всегда вложен внутри тега <map>.
<article>	Раздел контента, который образует независимую часть документа или сайта, например, статья в журнале, запись в блоге, комментарий.
<aside>	Представляет контент страницы, который имеет косвенное отношение к основному контенту страницы/сайта.
<audio>	Загружает звуковой контент на веб-страницу.
	Задаёт полужирное начертание отрывка текста, не придавая акцент или важность выделенному.
<base>	Задаёт базовый адрес (URL), относительно которого вычисляются все относительные адреса. Это поможет избежать проблем при переносе страницы в другое место, так как все ссылки будут работать, как и прежде.
<bdi>	Изолирует отрывок текста, написанный на языке, в котором чтение текста происходит справа налево, от остального текста.
<bdo>	Отображает текст в направлении, указанном в атрибуте dir, переопределяя текущее направление написания текста.
<blockquote>	Выделяет текст как цитату, применяется для описания больших цитат.
<body>	Представляет тело документа (содержимое, не относящееся к метаданным документа).
 	Перенос текста на новую строку.
<button>	Создаёт интерактивную кнопку. Внутри тега можно поместить содержимое — текст или изображение.
<canvas>	Холст-контейнер для динамического отображения изображений, таких как простые изображения, диаграммы, графики и т.п. Для рисования используется скриптовый язык JavaScript.

<code><caption></code>	Добавляет подпись к таблице. Вставляется сразу после тега <code><table></code> .
<code><cite></code>	Используется для указания источника цитирования. Отображается курсивом.
<code><code></code>	Представляет фрагмент программного кода, отображается шрифтом семейства monospace.
<code><col></code>	Выбирает для форматирования один или несколько столбцов таблицы, не содержащих информацию одного типа.
<code><colgroup></code>	Создает структурную группу столбцов, выделяющую множество логически однородных ячеек.
<code><data></code>	Элемент используется для связывания значения атрибута value, которое представлено в машиночитаемом формате и может быть обработано компьютером, с содержимым тега.
<code><datalist></code>	Элемент-контейнер для выпадающего списка элемента <code><input></code> . Варианты значений помещаются в элементы <code><option></code> .
<code><dd></code>	Используется для описания термина из тега <code><dt></code> .
<code></code>	Помечает текст как удаленный, перечёркивая его.
<code><details></code>	Создаёт интерактивный виджет, который пользователь может открыть или закрыть. Представляет собой контейнер для контента, видимый заголовок виджета помещается в тег <code><summary></code> .
<code><dfn></code>	Определяет слово как термин, выделяя его курсивом. Текст, идущий следом, должен содержать расшифровку этого термина.
<code><dialog></code>	Интерактивный элемент, с которым взаимодействует пользователь для выполнения задачи, например, диалоговое окно, инспектор или окно. Без атрибута open не виден для пользователя.
<code><div></code>	Тег-контейнер для разделов HTML-документа. Используется для группировки блочных элементов с целью форматирования стилями.
<code><dl></code>	Тег-контейнер, внутри которого находятся термин и его описание.
<code><dt></code>	Используется для задания термина.
<code></code>	Выделяет важные фрагменты текста, отображая их курсивом.
<code><embed></code>	Тег-контейнер для встраивания внешнего интерактивного контента или плагина.
<code><fieldset></code>	Группирует связанные элементы в форме, рисуя рамку вокруг них.
<code><figcaption></code>	Заголовок/подпись для элемента <code><figure></code> .

<code><figure></code>	Самодостаточный тег-контейнер для такого контента как иллюстрации, диаграммы, фотографии, примеры кода, обычно с подписью.
<code><footer></code>	Определяет завершающую область (нижний колонтитул) документа или раздела.
<code><form></code>	Форма для сбора и отправки на сервер информации от пользователей. Не работает без атрибута action.
<code><h1-h6></code>	Создают заголовки шести уровней для связанных с ними разделов.
<code><head></code>	Элемент-контейнер для метаданных HTML-документа, таких как <code><title></code> , <code><meta></code> , <code><script></code> , <code><link></code> , <code><style></code> .
<code><header></code>	Секция для вводной информации сайта или группы навигационных ссылок. Может содержать один или несколько заголовков, логотип, информацию об авторе.
<code><hr></code>	Горизонтальная линия для тематического разделения параграфов.
<code><html></code>	Корневой элемент HTML-документа. Сообщает браузеру, что это HTML-документ. Является контейнером для всех остальных html-элементов.
<code><i></code>	Выделяет отрывок текста курсивом, не придавая ему дополнительный акцент.
<code><iframe></code>	Создает встроенный фрейм, загружая в текущий HTML-документ другой документ.
<code></code>	Встраивает изображения в HTML-документ с помощью атрибута <code>src</code> , значением которого является адрес встраиваемого изображения.
<code><input></code>	Создает многофункциональные поля формы, в которые пользователь может вводить данные.
<code><ins></code>	Выделяет текст подчеркиванием. Применяется для выделения изменений, вносимых в документ.
<code><kbd></code>	Выделяет текст, который должен быть введен пользователем с клавиатуры, шрифтом семейства monospace.
<code><label></code>	Добавляет текстовую метку для элемента <code><input></code> .
<code><legend></code>	Заголовок элементов формы, сгруппированных с помощью элемента <code><fieldset></code> .
<code></code>	Элемент маркированного или нумерованного списка.
<code><link></code>	Определяет отношения между документом и внешним ресурсом. Также используется для подключения внешних таблиц стилей.
<code><main></code>	Контейнер для основного уникального содержимого документа. На одной странице должно быть не более одного элемента <code><main></code> .
<code><map></code>	Создаёт активные области на карте-изображении. Является контейнером для элементов <code><area></code> .

<code><mark></code>	Выделяет фрагменты текста, помечая их желтым фоном.
<code><meta></code>	Используется для хранения дополнительной информации о странице. Эту информацию используют браузеры для обработки страницы, а поисковые системы — для ее индексации. В блоке <code><head></code> может быть несколько тегов <code><meta></code> , так как в зависимости от используемых атрибутов они несут разную информацию.
<code><meter></code>	Индикатор измерения в заданном диапазоне.
<code><nav></code>	Раздел документа, содержащий навигационные ссылки по сайту.
<code><noscript></code>	Определяет секцию, не поддерживающую сценарий (скрипт).
<code><object></code>	Контейнер для встраивания мультимедиа (например, аудио, видео, Java-апплеты, ActiveX, PDF и Flash). Также можно вставить другую веб-страницу в текущий HTML-документ. Для передачи параметров встраиваемого плагина используется тег <code><param></code> .
<code></code>	Упорядоченный нумерованный список. Нумерация может быть числовая или алфавитная.
<code><optgroup></code>	Контейнер с заголовком для группы элементов <code><option></code> .
<code><option></code>	Определяет вариант/опцию для выбора в раскрывающемся списке <code><select></code> , <code><optgroup></code> или <code><datalist></code> .
<code><output></code>	Поле для вывода результата вычисления, рассчитанного с помощью скрипта.
<code><p></code>	Параграфы в тексте.
<code><param></code>	Определяет параметры для плагинов, встраиваемых с помощью элемента <code><object></code> .
<code><picture></code>	Элемент-контейнер, содержащий один элемент <code></code> и ноль или несколько элементов <code><source></code> . Сам по себе ничего не отображает. Дает возможность браузеру выбирать наиболее подходящее изображение.
<code><pre></code>	Выводит текст без форматирования, с сохранением пробелов и переносов текста. Может быть использован для отображения компьютерного кода, сообщения электронной почты и т.д.
<code><progress></code>	Индикатор выполнения задачи любого рода.
<code><q></code>	Определяет краткую цитату.
<code><ruby></code>	Контейнер для Восточно-Азиатских символов и их расшифровки.
<code><rb></code>	Определяет вложенный в него текст как базовый компонент аннотации.
<code><rt></code>	Добавляет краткую характеристику сверху или снизу от символов, заключенных в элементе <code><ruby></code> , выводится уменьшенным шрифтом.

<rtc>	Отмечает вложенный в него текст как дополнительную аннотацию.
<rp>	Выводит альтернативный текст в случае если браузер не поддерживает элемент <ruby>.
<s>	Отображает текст, не являющийся актуальным, перечеркнутым.
<samp>	Используется для вывода текста, представляющего результат выполнения программного кода или скрипта, а также системные сообщения. Отображается моноширинным шрифтом.
<script>	Используется для определения сценария на стороне клиента (обычно JavaScript). Содержит либо текст скрипта, либо указывает на внешний файл сценария с помощью атрибута src.
<section>	Определяет логическую область (раздел) страницы, обычно с заголовком.
<select>	Элемент управления, позволяющий выбирать значения из предложенного множества. Варианты значений помещаются в <option>.
<small>	Отображает текст шрифтом меньшего размера.
<source>	Указывает местоположение и тип альтернативных медиаресурсов для элементов <picture>, <video>, <audio>.
	Контейнер для строчных элементов. Можно использовать для форматирования отрывков текста, например, выделения цветом отдельных слов.
	Расставляет акценты в тексте, выделяя полужирным.
<style>	Подключает встраиваемые таблицы стилей.
<sub>	Задаёт подстрочное написание символов, например, индекса элемента в химических формулах.
<summary>	Создаёт видимый заголовок для тега <details>. Отображается с закрашенным треугольником, кликнув по которому можно просмотреть подробности заголовка.
<sup>	Задаёт надстрочное написание символов.
<table>	Тег для создания таблицы.
<tbody>	Определяет тело таблицы.
<td>	Создаёт ячейку таблицы.
<template>	Используется для объявления фрагментов HTML-кода, которые могут быть клонированы и вставлены в документ скриптом. Содержимое тега не является его дочерним элементом.
<textarea>	Создаёт большие поля для ввода текста.
<tfoot>	Определяет нижний колонтитул таблицы.
<th>	Создаёт заголовок ячейки таблицы.
<thead>	Определяет заголовок таблицы.

<time>	Определяет дату/время.
<title>	Заголовок HTML-документа, отображаемый в верхней части строки заголовка браузера. Также может отображаться в результатах поиска, поэтому это следует принимать во внимание предоставление названия.
<tr>	Создает строку таблицы.
<track>	Добавляет субтитры для элементов <audio> и <video>.
<u>	Выделяет отрывок текста подчёркиванием, без дополнительного акцента.
	Создает маркированный список.
<var>	Выделяет переменные из программ, отображая их курсивом.
<video>	Добавляет на страницу видео-файлы. Поддерживает 3 видео формата: MP4, WebM, Ogg.
<wbr>	Указывает браузеру возможное место разрыва длинной строки.

Листинг программы metacritic.ipynb

```

In [11]: from bs4 import BeautifulSoup
import requests as req
import re

In [12]: # используем библиотеку requests для вывода html в отдельный файл
rq_header = {'User-Agent' : 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3626.121 Safari/537.36'}
response=req.get('https://www.metacritic.com/browse/movies/score/metacore/all/filtered?sort=desc&view=detailed', headers=rq_header)
with open('metacritic.html', 'wb') as output_file:
    output_file.write(response.text.encode('cp1251', 'ignore'))

In [13]: # используем библиотеку BeautifulSoup для парсинга
soup=BeautifulSoup(open('metacritic.html'),'html.parser')
tags=soup.find("div",{"class":"title_bump"})
#удаляем лишние элементы
for el in tags.find_all("div",{"class":"browse-score-clamp"}):
    el.decompose()
for el in tags.find_all("tr",{"class":"spacer"}):
    el.decompose()
#получение всей информации о каждой позиции
items=tags.find_all('tr')

In [14]: results=[]
#обход всех позиций
for item in items:
    #id фильма
    id_str=item.find('input',{'class':'clamp-summary-expand'}).get('id')

    #название фильма
    name_str=item.find('a',{'class':'title'}).text

    #получаем score
    score=item.find('div',{'class':'metascore_w large movie positive'})
    if score is None:
        score=item.find('div',{'class':'metascore_w large movie positive perfect'})
    score = score.text

    #дата фильма
    film_data=item.select_one('div.clamp-details>span:nth-of-type(1)').text

    #описание фильма
    description=item.select_one('div.summary').text.lstrip().rstrip()

    #рейтинг фильма
    rate_data=item.select_one('div.clamp-details>span:nth-of-type(2)')
    if rate_data is not None:
        rate_data = rate_data.text.replace(' | ', '')
    else: rate_data = 'Not Rated';

    # получаем год фильма
    if film_data == "TBA":
        year = '-';
    else:
        year_num=film_data.split(' ')
        year=int(year_num[1])

    df=(id_str, name_str, score, film_data, year, rate_data, description)
    results.append(df)

In [15]: import pandas as pd
df = pd.DataFrame(results)
df.columns=['Id', 'Name', 'Metascore', 'Date', 'Year', 'Rate', 'Description']

In [16]: df

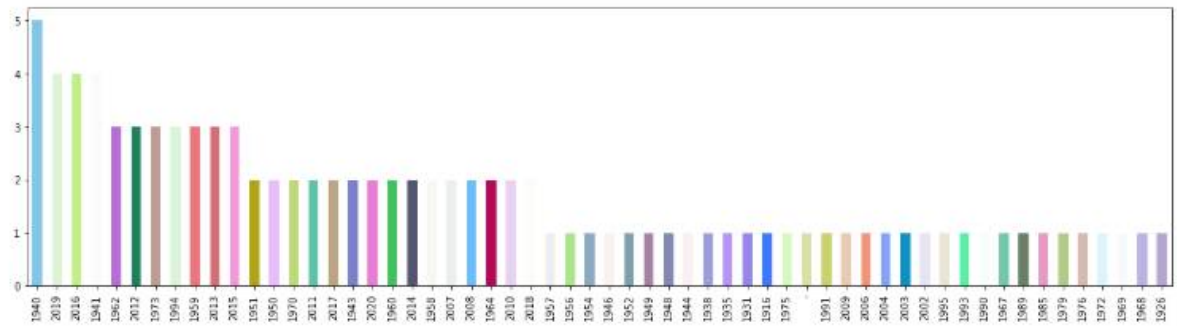
```

Out[16]:

	Id	Name	Metascore	Date	Year	Rate	Description
0	540575	Citizen Kane	100	September 4, 1941	1941	Approved	Following the death of a publishing tycoon, ne...
1	522553	The Godfather	100	March 11, 1972	1972	R	Francis Ford Coppola's epic features Marlon Br...
2	548059	Rear Window	100	September 1, 1954	1954	TV-G	A wheelchair-bound photographer spies on his n...
3	549833	Casablanca	100	January 23, 1943	1943	TV-PG	A Casablanca, Morocco casino owner in 1941 she...
4	542178	Boyhood	100	July 11, 2014	2014	R	Filmed over 12 years with the same cast, Richa...

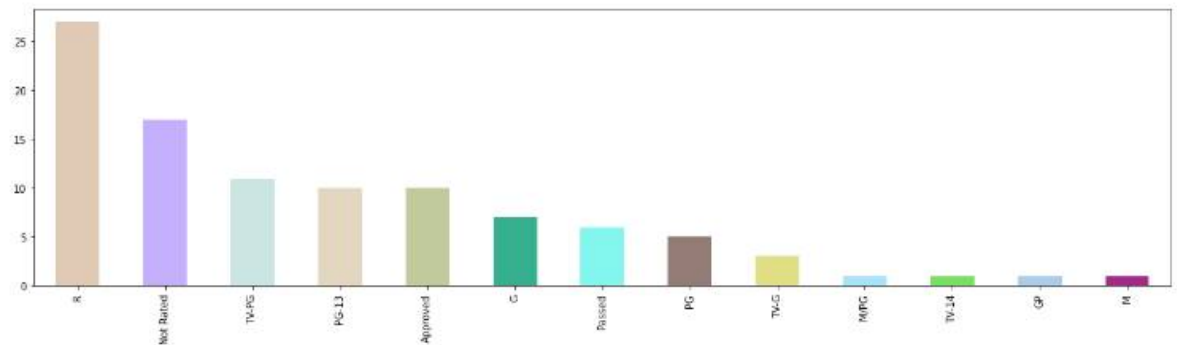
```
In [17]: import matplotlib.pyplot as plt
import numpy as np
df['Year'].value_counts().plot.bar(figsize = (20,5), color=np.random.rand(256, 4))
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1e1d3d74790>



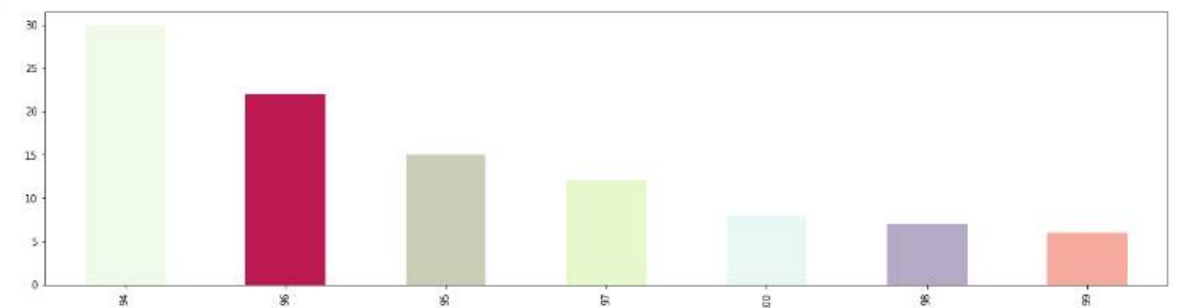
```
In [18]: df['Rate'].value_counts().plot.bar(figsize = (20,5), color=np.random.rand(256, 4) )
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1e1d3e39250>



```
In [19]: df['Metascore'].value_counts().plot.bar(figsize = (20,5), color=np.random.rand(256, 4))
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1e1d24f39d0>



```
In [20]: import csv
import pandas as pd
df.to_csv("films.csv", sep=',')
```

Листинг программы

```

from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import utils
from tensorflow.keras.preprocessing.sequence import
pad_sequences
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline

"""## Загружаем данные"""

max_words=10000

(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=max_words)

"""## Просмотр данных

Рецензия
"""

x_train[3]

"""Правильный ответ"""

y_train[3]

"""## Раскодируем текст рецензии

В наборе данных IMDB используется частотное кодирование
слов. Загрузим словарь, который использовался для
кодирования.
"""

word_index = imdb.get_word_index()

word_index

"""Преобразуем словарь, чтобы по номеру получать слово"""

reverse_word_index = dict()
for key, value in word_index.items():
    reverse_word_index[value] = key

"""Печатаем 20 самых частых слов"""

for i in range(1, 21):

```

```

        print(i, '->', reverse_word_index[i])

    """Раскодируем сообщения

    Служебные коды:
    0 - символ заполнитель
    1 - начало последовательности
    2 - неизвестное слово
    """

    index = 3
    message = ''
    for code in x_train[index]:
        word = reverse_word_index.get(code - 3, '?')
        message += word + ' '
    message

    y_train[index]

    """

    ## Подготовка данных для обучения
    """

    maxlen = 200

    x_train      =      pad_sequences(x_train,      maxlen=maxlen,
    padding='post')
    x_test       =      pad_sequences(x_test,       maxlen=maxlen,
    padding='post')

    x_train[3]

    y_train[3]

    """## Создание нейронной сети"""

    model = Sequential()
    model.add(Dense(128,                                activation='relu',
    input_shape=(maxlen,)))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    """## Обучаем нейронную сеть"""

    history = model.fit(x_train,
                        y_train,

```

```
        epochs=25,
        batch_size=128,
        validation_split=0.1)

plt.plot(history.history['accuracy'],
         label='Доля верных ответов на обучающем наборе')
plt.plot(history.history['val_accuracy'],
         label='Доля верных ответов на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Доля верных ответов')
plt.legend()
plt.show()

"""## Проверяем работу сети на тестовом наборе данных"""

scores = model.evaluate(x_test, y_test, verbose=1)

print("Доля верных ответов на тестовых данных, в
процентах:", round(scores[1] * 100, 4))
```

Листинг программы

```

from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline

"""## Загружаем данные"""

max_words=10000

(x_train,      y_train),      (x_test,      y_test)      =
imdb.load_data(num_words=max_words)

"""## Подготовка данных для обучения"""

x_train[0]

"""Функция для кодирования one hot encoding"""

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(x_train, max_words)
x_test = vectorize_sequences(x_test, max_words)

x_train[0][:50]

len(x_train[0])

y_train[0]

"""## Создание нейронной сети"""

model = Sequential()
model.add(Dense(128,                                activation='relu',
input_shape=(max_words,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

"""## Обучаем нейронную сеть"""

```

```
history = model.fit(x_train,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.1)

plt.plot(history.history['accuracy'],
         label='Доля верных ответов на обучающем наборе')
plt.plot(history.history['val_accuracy'],
         label='Доля верных ответов на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Доля верных ответов')
plt.legend()
plt.show()

"""## Проверяем работу сети на тестовом наборе данных"""

scores = model.evaluate(x_test, y_test, verbose=1)

print("Доля верных ответов на тестовых данных, в
процентах:", round(scores[1] * 100, 4))
```


Листинг программы

```

from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding,
Flatten, Dropout
from tensorflow.keras import utils
from tensorflow.keras.preprocessing.sequence import
pad_sequences
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
# %matplotlib inline

"""## Загружаем данные"""

max_words=10000

(x_train,      y_train),      (x_test,      y_test)      =
imdb.load_data(num_words=max_words)

"""## Просмотр данных

Рецензия
"""

x_train[3]

"""Правильный ответ"""

y_train[3]

"""## Подготовка данных для обучения"""

maxlen = 200

x_train      =      pad_sequences(x_train,      maxlen=maxlen,
padding='post')
x_test       =      pad_sequences(x_test,      maxlen=maxlen,
padding='post')

x_train[3]

y_train[3]

"""## Создание нейронной сети"""

model = Sequential()
model.add(Embedding(max_words, 2, input_length=maxlen))
model.add(Dropout(0.25))

```

```

model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

"""## Обучаем нейронную сеть"""

history = model.fit(x_train,
                   y_train,
                   epochs=15,
                   batch_size=128,
                   validation_split=0.1)

plt.plot(history.history['accuracy'],
         label='Доля верных ответов на обучающем наборе')
plt.plot(history.history['val_accuracy'],
         label='Доля верных ответов на проверочном
наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Доля верных ответов')
plt.legend()
plt.show()

"""## Проверяем работу сети на тестовом наборе данных"""

scores = model.evaluate(x_test, y_test, verbose=1)

"""## Исследуем обученное плотное векторное представление
слов

**Получаем матрицу плотных векторных представлений слов**
"""

embedding_matrix = model.layers[0].get_weights()[0]

embedding_matrix[:5]

"""**Загружаем словарь с номерами слов**"""

word_index_org = imdb.get_word_index()

"""Дополняем словарь служебными символами"""

word_index = dict()
for word, number in word_index_org.items():
    word_index[word] = number + 3
word_index["<Заполнитель>"] = 0
word_index["<Начало последовательности>"] = 1
word_index["<Неизвестное слово>"] = 2
word_index["<Не используется>"] = 3

```

```

"""**Ищем векторы для слов**"""

word = 'good'
word_number = word_index[word]
print('Номер слова', word_number)
print('Вектор для слова', embedding_matrix[word_number])

"""## Сохраняем обученные плотные векторные представления
в файл

**Составляем реверсивный словарь токенов (слов)**
"""

reverse_word_index = dict()
for key, value in word_index.items():
    reverse_word_index[value] = key

"""**Записываем плотные векторные представления в
файл**"""

filename = 'imdb_embeddings.csv'

with open(filename, 'w') as f:
    for word_num in range(max_words):
        word = reverse_word_index[word_num]
        vec = embedding_matrix[word_num]
        f.write(word + ",")
        f.write(','.join([str(x) for x in vec]) + "\n")

!head -n 20 $filename

"""**Сохраняем файл на локальный компьютер**"""

files.download('imdb_embeddings.csv')

"""## Визуализация плотных векторных представлений слов"""

plt.scatter(embedding_matrix[:,0], embedding_matrix[:,1])

"""Выбираем коды слов, по которым можно определить
тональность отзыва"""

review = ['brilliant', 'fantastic', 'amazing', 'good',
          'bad', 'awful', 'crap', 'terrible', 'trash']
enc_review = []
for word in review:
    enc_review.append(word_index[word])
enc_review

"""Получаем векторное представление интересующих нас
слов"""

review_vectors = embedding_matrix[enc_review]

```

```
review_vectors
```

```
"""Визуализация обученного плотного векторного  
представления слов, по которым можно определить  
эмоциональную окраску текста"""
```

```
plt.scatter(review_vectors[:,0], review_vectors[:,1])  
for i, txt in enumerate(review):  
    plt.annotate(txt, (review_vectors[i,0],  
review_vectors[i,1]))
```

Ж.1. Листинг программы «Решение задачи OneMax с помощью DEAP»

```
pip install deap

from deap import base
from deap import creator
from deap import tools
import random
import matplotlib.pyplot as plt

# константы задачи
ONE_MAX_LENGTH = 100 # длина подлежащей оптимизации
битовой строки

# константы генетического алгоритма
POPULATION_SIZE = 200 # количество индивидуумов в
популяции

P_CROSSOVER = 0.9 # вероятность скрещивания
P_MUTATION = 0.1 # вероятность мутации индивидуума
MAX_GENERATIONS = 50 # максимальное количество
поколений

RANDOM_SEED = 42
random.seed(RANDOM_SEED)

toolbox = base.Toolbox()
toolbox.register("zeroOrOne", random.randint, 0, 1)
creator.create("FitnessMax", base.Fitness,
weights=(1.0,))

creator.create("Individual", list,
fitness=creator.FitnessMax)
```

```

    toolbox.register("individualCreator",
tools.initRepeat,creator.Individual,  toolbox.zeroOrOne,
ONE_MAX_LENGTH)

    toolbox.register("populationCreator",
tools.initRepeat,list, toolbox.individualCreator)
    def oneMaxFitness(individual):
        return sum(individual), # вернуть кортеж

    toolbox.register("evaluate", oneMaxFitness)

    toolbox.register("select",      tools.selTournament,
tournamentsize=3)
    toolbox.register("mate", tools.cxOnePoint)
    toolbox.register("mutate",      tools.mutFlipBit,
indpb=1.0/ONE_MAX_LENGTH)

    # Генетический алгоритм

    population =
toolbox.populationCreator(n=POPULATION_SIZE)
    generationCounter = 0

    fitnessValues = list(map(toolbox.evaluate,
population))

    for individual, fitnessValue in zip(population,
fitnessValues):
        individual.fitness.values = fitnessValue

```

```

    fitnessValues = [individual.fitness.values[0] for
individual in population]

    maxFitnessValues = []
    meanFitnessValues = []

    while max(fitnessValues) < ONE_MAX_LENGTH and
generationCounter < MAX_GENERATIONS:
        generationCounter = generationCounter + 1
        offspring = toolbox.select(population,
len(population))
        offspring = list(map(toolbox.clone, offspring))
        for child1, child2 in zip(offspring[::2],
offspring[1::2]):
            if random.random() < P_CROSSOVER:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values
        for mutant in offspring:
            if random.random() < P_MUTATION:
                toolbox.mutate(mutant)
                del mutant.fitness.values
        freshIndividuals = [ind for ind in offspring if
not ind.fitness.valid]
        freshFitnessValues = list(map(toolbox.evaluate,
freshIndividuals))
        for individual, fitnessValue in
zip(freshIndividuals, freshFitnessValues):
            individual.fitness.values = fitnessValue
        population[:] = offspring

```

```

        fitnessValues = [ind.fitness.values[0] for ind
in population]
        maxFitness = max(fitnessValues)
        meanFitness = sum(fitnessValues) /
len(population)
        maxFitnessValues.append(maxFitness)
        meanFitnessValues.append(meanFitness)
        print("- Поколение {}: Макс приспособ. = {},
Средняя приспособ. = {}".format(generationCounter,
maxFitness, meanFitness))
        best_index =
fitnessValues.index(max(fitnessValues))
        print("Best Individual = ",
*population[best_index], "\n")
        plt.plot(maxFitnessValues, color='red')
        plt.plot(meanFitnessValues, color='green')
        plt.xlabel('Поколение')
        plt.ylabel('Макс/средняя приспособленность')
        plt.title('Зависимость максимальной и средней
приспособленности от поколения')
        plt.show()

```


Ж.2. Листинг программы «Решение задачи OneMax с помощью DEAP»

```

from deap import base
from deap import creator
from deap import tools
from deap import algorithms
import random
import matplotlib.pyplot as plt
import numpy as np

# константы задачи
ONE_MAX_LENGTH = 100 # длина подлежащей оптимизации
битовой строки

# константы генетического алгоритма
POPULATION_SIZE = 200 # количество индивидуумов в
популяции

P_CROSSOVER = 0.9 # вероятность скрещивания
P_MUTATION = 0.1 # вероятность мутации индивидуума
MAX_GENERATIONS = 50 # максимальное количество
поколений

HALL_OF_FAME_SIZE = 10 # Зал славы

toolbox = base.Toolbox()
toolbox.register("zeroOrOne", random.randint, 0, 1)
creator.create("FitnessMax", base.Fitness,
weights=(1.0,))
creator.create("Individual", list,
fitness=creator.FitnessMax)
toolbox.register("individualCreator",
tools.initRepeat, creator.Individual, toolbox.zeroOrOne,
ONE_MAX_LENGTH)

```

```

    toolbox.register("populationCreator",
tools.initRepeat,list, toolbox.individualCreator)

    def oneMaxFitness(individual):
        return sum(individual), # вернуть корт

    toolbox.register("evaluate", oneMaxFitness)
    toolbox.register("select",      tools.selTournament,
tournsize=3)
    toolbox.register("mate", tools.cxOnePoint)
    toolbox.register("mutate",      tools.mutFlipBit,
indpb=1.0/ONE_MAX_LENGTH)
    population =
toolbox.populationCreator(n=POPULATION_SIZE)
    generationCounter = 0
    fitnessValues = list(map(toolbox.evaluate,
population))

    for individual, fitnessValue in zip(population,
fitnessValues):
        individual.fitness.values = fitnessValue

    fitnessValues = [individual.fitness.values[0] for
individual in population]
    maxFitnessValues = []
    meanFitnessValues = []
    stats = tools.Statistics(lambda ind:
ind.fitness.values)
    stats.register("max", np.max)
    stats.register("avg", np.mean)

```

```

        population,                                logbook                                =
algorithms.eaSimple(population,                                toolbox,
cxpb=P_CROSSOVER,                                mutpb=P_MUTATION,
ngen=MAX_GENERATIONS, stats=stats, verbose=True)
        maxFitnessValues,                        meanFitnessValues                        =
logbook.select("max", "avg")
        plt.plot(maxFitnessValues, color='red')
        plt.plot(meanFitnessValues, color='green')
        plt.xlabel('Generation')
        plt.ylabel('Max / Average Fitness')
        plt.title('Max    and    Average    Fitness    over
Generations')
        plt.show()

        #    Зал    славы,    позволяет    сохранить    лучших
индивидуумов,    встретившихся    в    процессе    эволюции,    даже
если    вследствие    отбора,    скрещивания    и    мутации    они    были    в
какой-то    момент    утрачены.    Зал    славы    поддерживается    в
отсортированном    состоянии,    так    что    первым    элементом
всегда    является    индивидуум    с    наилучшим    встретившимся
значением    приспособленности.

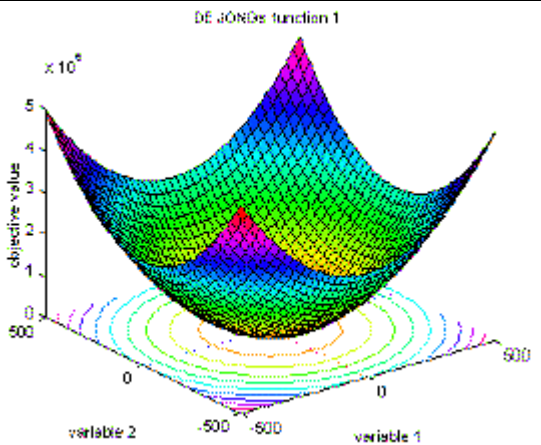
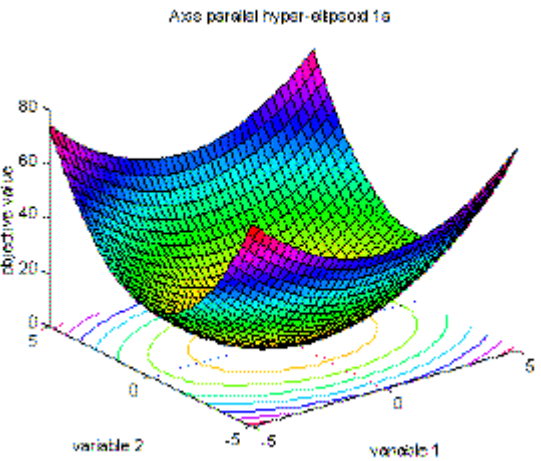
        hof = tools.HallOfFame(HALL_OF_FAME_SIZE)
        population,                                logbook                                =
algorithms.eaSimple(population,                                toolbox,
cxpb=P_CROSSOVER,                                mutpb=P_MUTATION,
ngen=MAX_GENERATIONS, stats=stats, halloffame=hof,
verbose=True)

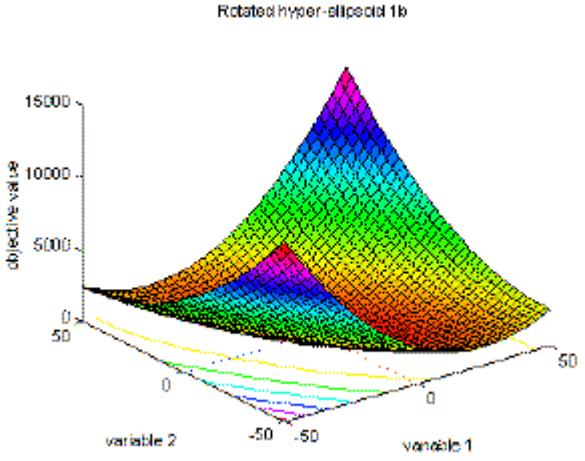
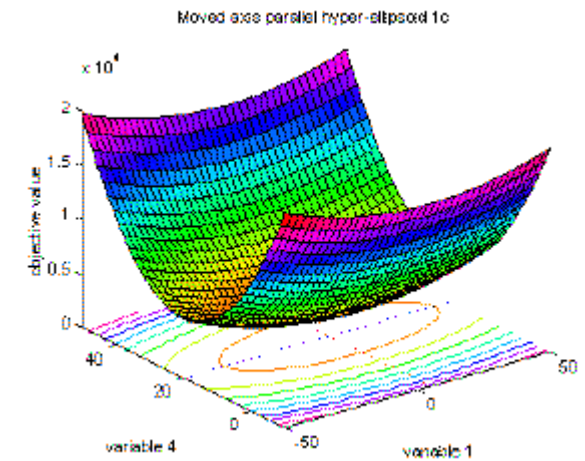
        print("Индивидуумы    в    зале    славы    =    ", *hof.items,
sep="\n")

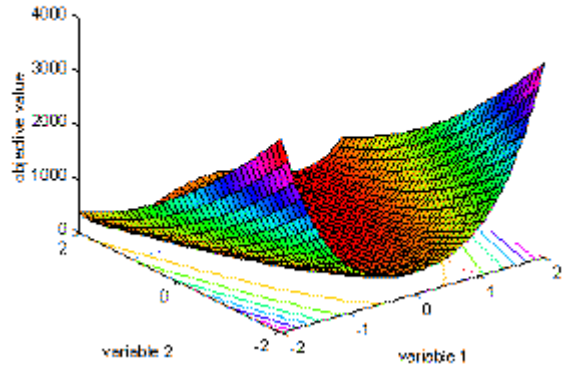
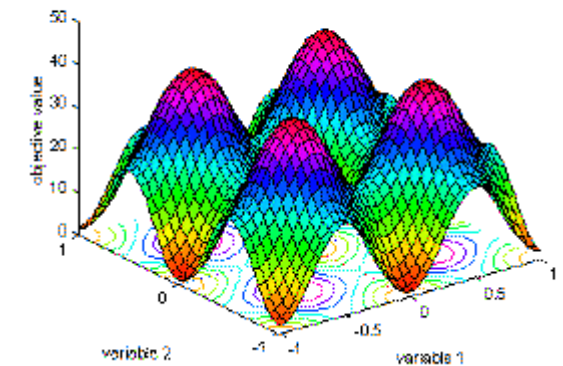
        print("Лучший    индивидуум    =    ", hof.items[0])

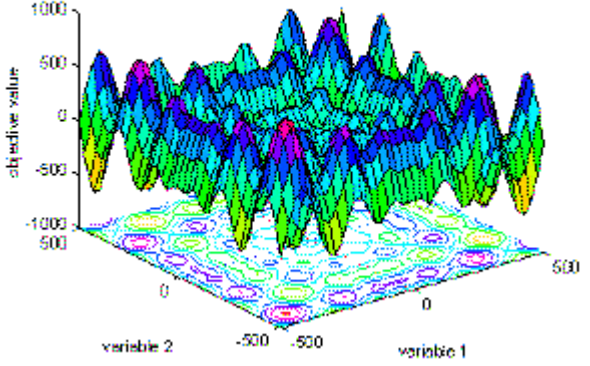
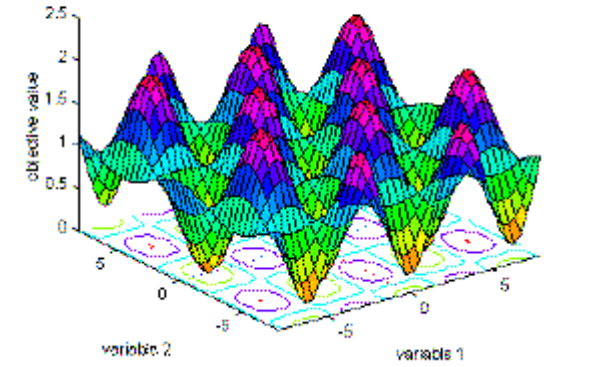
```

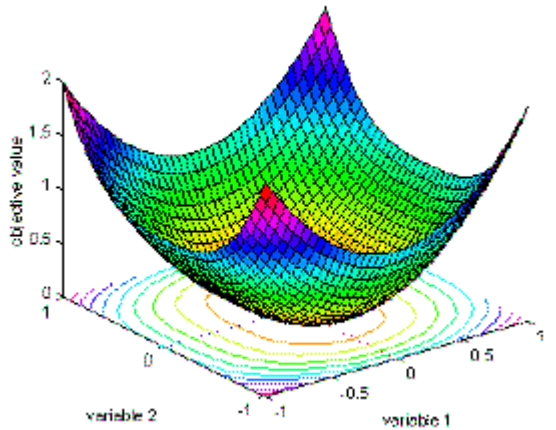
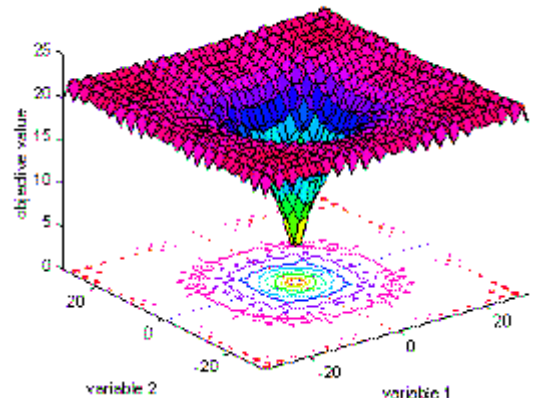
Таблица И.1. Индивидуальные задания на лабораторную работу №2

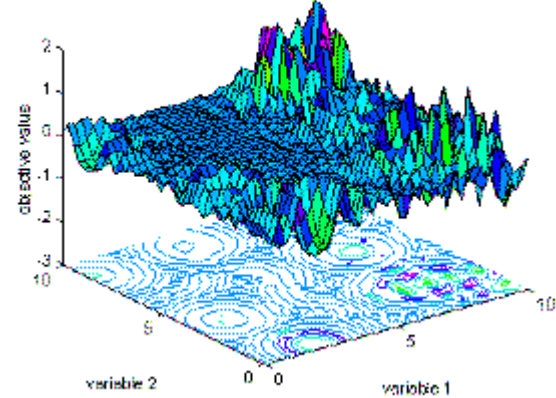
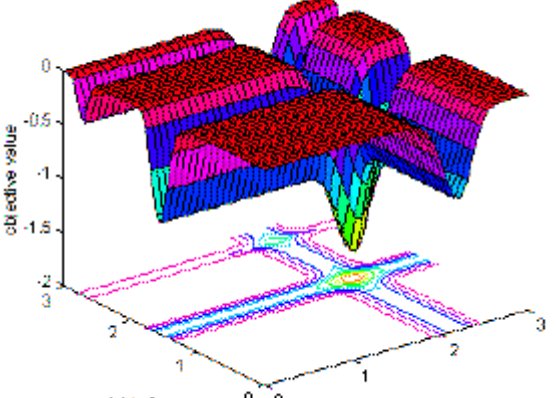
№ вв.	Название	Оптимум	Вид функции	График функции
1	De Jong's function 1	global minimum $f(x)=0$; $x(i)=0$, $i=1:n$.	$f_1(x) = \sum_{i=1}^n x_i^2 \quad -5.12 \leq x_i \leq 5.12$ $f_1(x) = \sum_{i=1}^n (x(i)^2),$ $i=1:n;$	
2	Axis parallel hyper-ellipsoid function	global minimum $f(x)=0$; $x(i)=0$, $i=1:n$.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

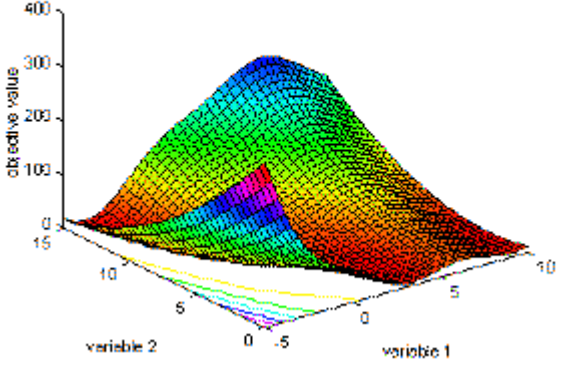
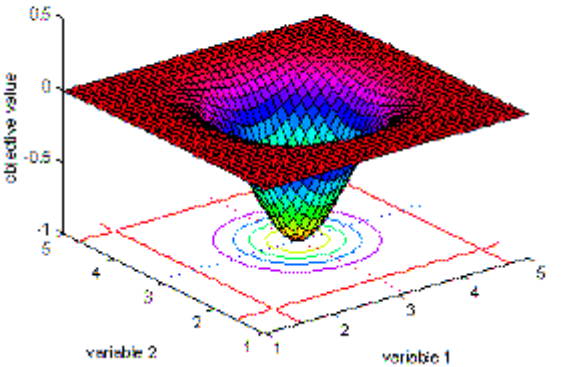
3	Rotated hyper-ellipsoid function	global minimum $f(x)=0$; $x(i)=0$, $i=1:n$	$f_{1b}(x) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2 \quad -65.536 \leq x_i \leq 65.536$ $f1b(x) = \text{sum}(\text{sum}(x(j)^2, j=1:i), i=1:n);$	<p>Rotated hyper-ellipsoid 1b</p> 
4	Moved axis parallel hyper-ellipsoid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

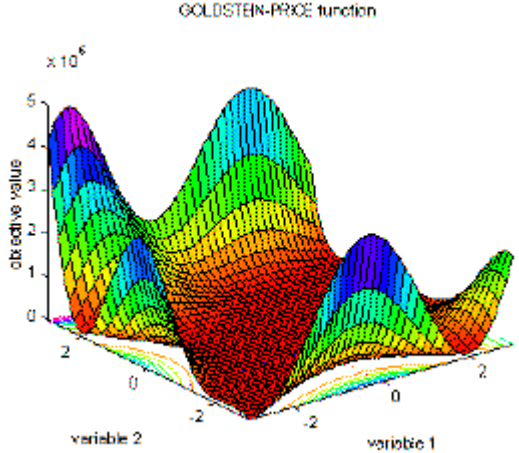
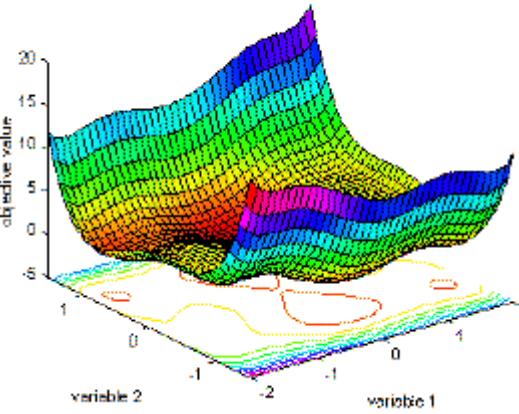
5	Rosenbrock's valley (De Jong's function 2)	global minimum $f(x)=0$; $x(i)=1$, $i=1:n$.	$f_2(x) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \quad -2.048 \leq x_i \leq 2.048$ $f2(x) = \text{sum}(100 \cdot (x(i+1) - x(i)^2)^2 + (1 - x(i))^2),$ $i=1:n-1;$	<p>ROSENBROCK's function 2</p> 
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 $f(x) = n \cdot 418.9829$; $x(i) = 420.9687$, $i = 1:n$.	$f_7(x) = \sum_{i=1}^n -x_i \cdot \sin\left(\sqrt{ x_i }\right) \quad -500 \leq x_i \leq 500$ $f7(x) = \sum_{i=1}^n (-x(i) \cdot \sin(\sqrt{\text{abs}(x(i))})),$ $i = 1:n;$	<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 $f(x)=0$; $x(i)=0$, $i=1:n$.	$f_{10}(x) = -a \cdot e^{-b \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} - e^{\frac{\sum_{i=1}^n \cos(c \cdot x_i)}{n}} + a + e^1 \quad -1 \leq x_i \leq 1$ $f10(x) = -a \cdot \exp(-b \cdot \sqrt{1/n \cdot \sum (x(i)^2)}) -$ $\exp(1/n \cdot \sum (\cos(c \cdot x(i)))) + a + \exp(1);$ $a=20; b=0.2; c=2 \cdot \pi; i=1:n;$	<p>ACKLEY's PATH function 10</p> 

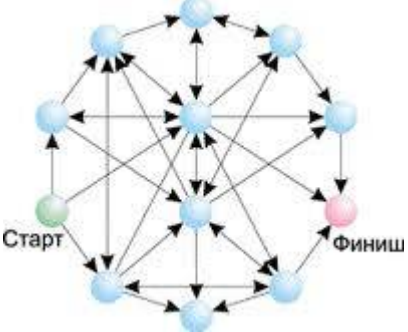
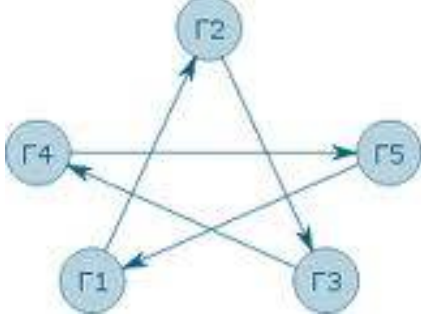
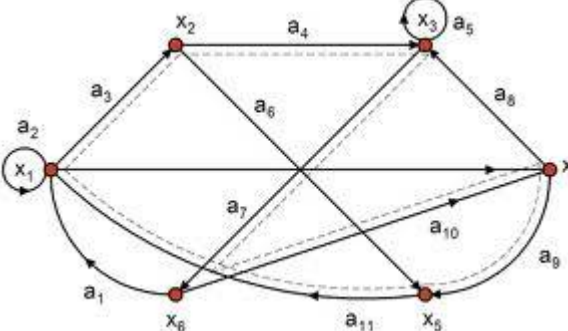
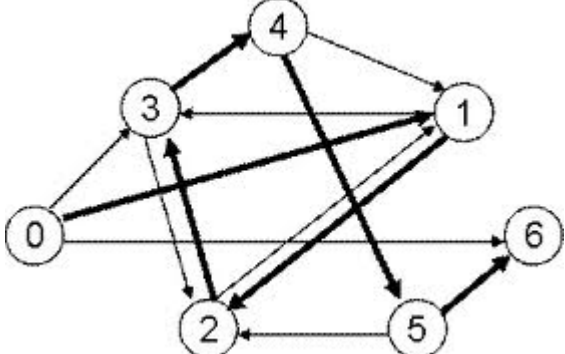
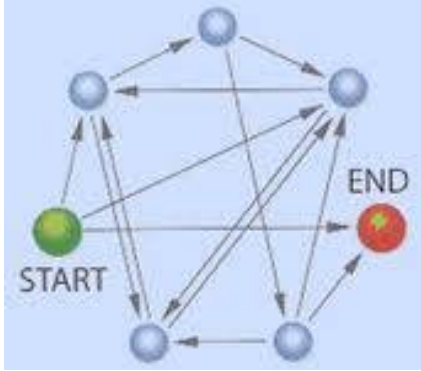
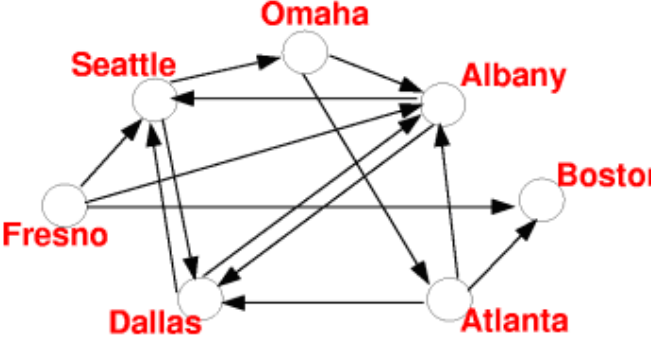
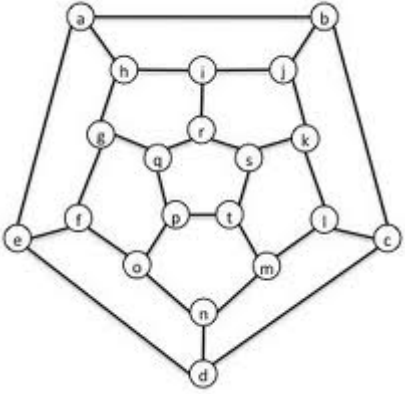
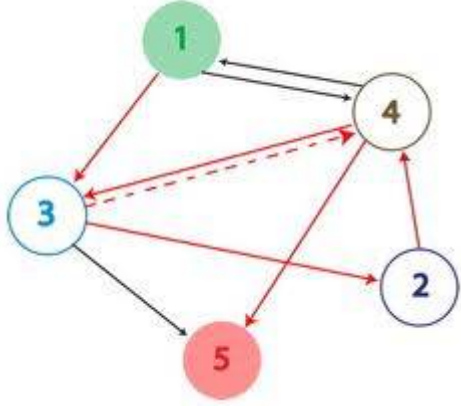
11	Langermann's function 11	<p>global minimum $f(x) = -1.4$ (for $m=5$); $x(i) = ???$, $i=1:n$.</p>	$f_{11}(x) = -\sum_{i=1}^m c_i \left(e^{-\frac{\ x-A(i)\ ^2}{\pi}} \cdot \cos(\pi \cdot \ x-A(i)\ ^2) \right) \quad i = 1:m, 2 \leq m \leq 10, 0 \leq x_i \leq 10$ <p> $f11(x) = -\sum(c(i) \cdot (\exp(-1/\pi \cdot \sum((x-A(i))^2)) \cdot \cos(\pi \cdot \sum((x-A(i))^2))))$, $i=1:m, m=5; A(i), C(i) > 0, m=5$ </p>	<p>LANGERMANN's function 11</p> 
12	Michalewicz's function 12	<p>global minimum $f(x) = -4.687$ $(n=5)$; $x(i) = ???$, $i=1:n$. $f(x) = -9.66$ $(n=10)$; $x(i) = ???$, $i=1:n$.</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> $f12(x) = -\sum(\sin(x(i)) \cdot (\sin(i \cdot x(i)^2/\pi))^{(2 \cdot m)})$, $i=1:n, m=10$; проверить для $n=5, 10$ </p>	<p>MICHALEWICZ's function 12</p> 

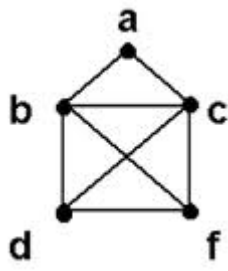
13	Branin's rcos function	<p>global minimum $f(x_1, x_2) = 0.397887$; $(x_1, x_2) = (-\pi, 12.275)$, $(\pi, 2.275)$, $(9.42478, 2.475)$.</p>	$f_{\text{Bran}}(x_1, x_2) = a \cdot (x_2 - b \cdot x_1^2 + c \cdot x_1 - d)^2 + e \cdot (1 - f) \cdot \cos(x_1) + e \quad -5 \leq x_1 \leq 10, 0 \leq x_2 \leq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

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

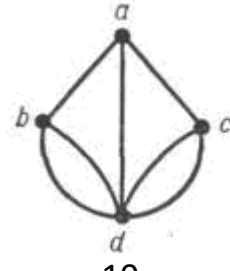
Для остальных вариантов берется строчка, соответствующая остатку от деления номера варианта на 16 (для 17-1, 18-2 и т.д.)

Таблица К.1. Индивидуальные задания на лабораторную работу №7.

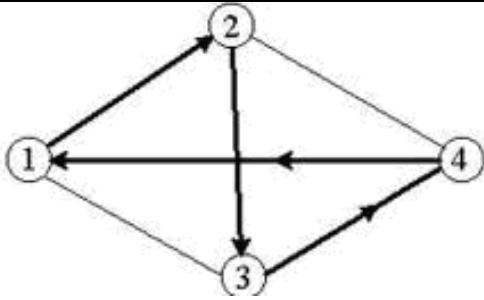
 <p>1</p>	 <p>2</p>
 <p>3</p>	 <p>4</p>
 <p>5</p>	 <p>6</p>
 <p>7</p>	 <p>8</p>



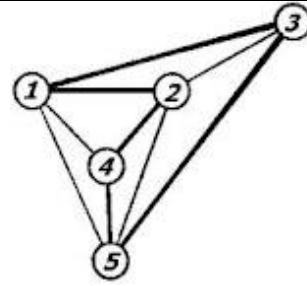
9



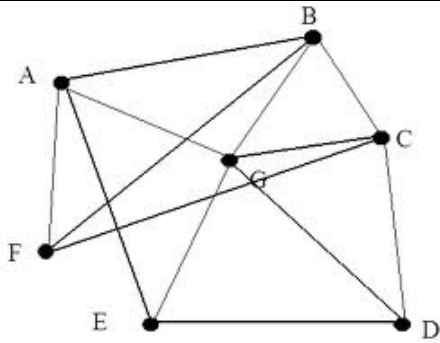
10



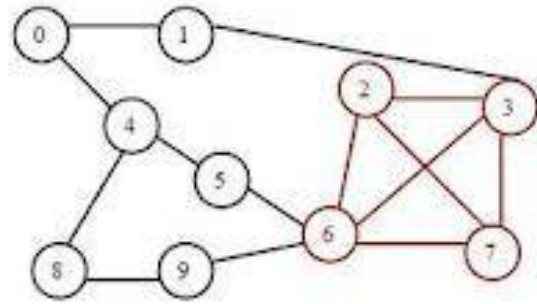
11



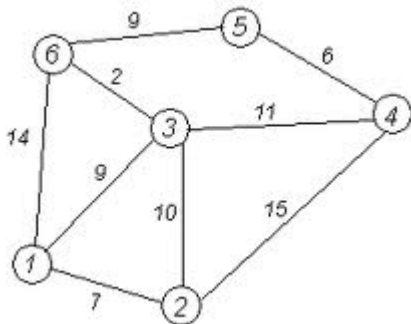
12



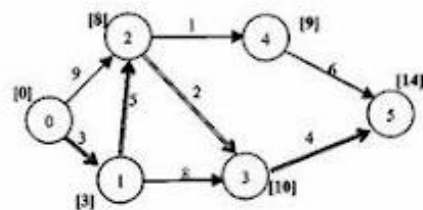
13



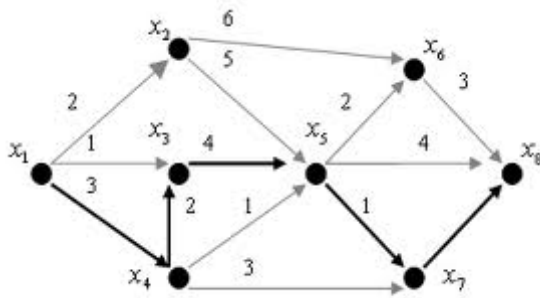
14



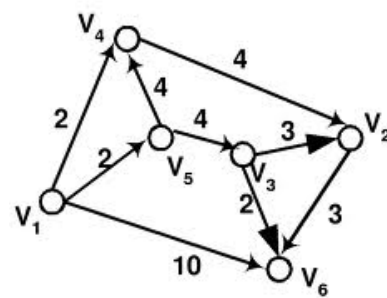
15



16



17



18

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
для проведения лабораторных 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»