|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |
| Институт информационных технологий | | |
| Кафедра вычислительной техники | | |

|  |
| --- |
|  |

**КУРСОВАЯ РАБОТА**

по дисциплине «Системный анализ данных в системах поддержки принятия решений»

(наименование дисциплины)

**Тема курсовой работы**\_\_\_\_\_\_\_\_\_«Материнские платы»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Студент группы**\_\_\_ ИКБО-14-20\_Вежновец Ф.Ю. \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(учебная группа, фамилия, имя отчество, студента) (подпись студента)

**Руководитель курсовой работы** \_к.т.н., доцент Сорокин А.Б. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, звание, ученая степень) (подпись руководителя)

**Рецензент** (при наличии) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, звание, ученая степень) (подпись рецензента)

Работа представлена к защите «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г.

Допущен к защите «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г.

Москва 2022 г.

|  |  |  |
| --- | --- | --- |
|  | Изображение выглядит как текст  Автоматически созданное описание |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |
| Институт информационных технологий | | |
| Кафедра вычислительной техники | | |

|  |  |
| --- | --- |
| **Утверждаю** | |
| Заведующий кафедрой |  |
|  | *Подпись* |
| Платонова О.В. | |
| *ФИО* | |
| «23» \_сентябрь\_ 2022г. | |

**ЗАДАНИЕ**

**на выполнение курсовой работы** по дисциплине

«Системный анализ данных в системах поддержки принятия решений»

Студент \_\_\_\_\_ Вежновец Ф.Ю. \_\_\_\_\_\_ Группа\_\_\_\_\_\_\_ИКБО-14-20\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| **Тема:** | «Материнские платы» |
|  |  |

**Исходные данные:** Набор данных Pima Indians Diabetes Database, программа Protege, MatLab\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:** реализовать консольное приложение: 1. Для онтологии по предметной области 2. Алгоритма имитации отжига, 3. Алгоритма роя частиц, 4. Алгоритма пчелиной колонии. 5. Муравьиный алгоритм. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Срок представления к защите курсовой работы: до «30»\_декабрь\_2022г.**

**Задание на курсовую работу выдал** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_Сорокин А.Б.\_)

*Подпись руководителя Ф.И.О. руководителя*

**Задание на курсовую работу получил** «23» \_сентябрь\_ 2022г

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Вежновец Ф.Ю.)

*Подпись обучающегося* Ф.И.О. исполнителя

Москва 2022г.

**ОТЗЫВ**

**на курсовую работу**

**по дисциплине «****Системный анализ данных в системах поддержки принятия решений»**

**Студент** Вежновец Ф.Ю. ИКБО-14-20

(ФИО студента) (Группа)

Характеристика курсовой работы

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Да | Нет | Не полностью |
| 1. Соответствие содержания курсовой работы указанной теме |  |  |  |
| 2. Соответствие курсовой работы заданию |  |  |  |
| 3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр. |  |  |  |
| 4. Полнота выполнения всех пунктов задания |  |  |  |
| 5. Логичность и системность содержания курсовой работы |  |  |  |
| 6. Отсутствие фактических грубых ошибок |  |  |  |

Замечаний:

Рекомендуемая оценка:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Сорокин А.Б.\_\_

Подпись руководителя ФИО руководителя

Содержание

[Содержание 4](#_Toc123010906)

[ВВЕДЕНИЕ 5](#_Toc123010907)

[1 ОНТОЛОГИЯ 6](#_Toc123010908)

[1.1 Постановка задачи 6](#_Toc123010909)

[1.2 Описание предметной области 6](#_Toc123010910)

[1.3 Создание сущностей 11](#_Toc123010911)

[1.4 Создание запросов 15](#_Toc123010912)

[1.5 Программная реализация 17](#_Toc123010913)

[2 Алгоритм оптимизации Отжигом 20](#_Toc123010914)

[2.1 Постановка задачи 21](#_Toc123010915)

[2.2 Расчетная часть 21](#_Toc123010916)

[2.3 Программная реализация 22](#_Toc123010917)

[3 АЛГОРИТМ ОПТИМИЗАЦИИ РОЯ 25](#_Toc123010918)

[3.1 Постановка задачи 27](#_Toc123010919)

[3.2 Расчётная часть 27](#_Toc123010920)

[3.3 Программная реализация 28](#_Toc123010921)

[4 МУРАВЬИНЫЙ АЛГОРИТМ 32](#_Toc123010922)

[4.1 Постановка задачи 34](#_Toc123010923)

[4.2 Расчётная часть 34](#_Toc123010924)

[4.3 Программная реализация 36](#_Toc123010925)

[5 ТАСУЮЩИЙ АЛГОРИТМ ПРЫГАЮЩИХ ЛЯГУШЕК 38](#_Toc123010926)

[5.1 Постановка задачи 39](#_Toc123010927)

[5.2 Расчётная часть 39](#_Toc123010928)

[5.3 Программная реализация 41](#_Toc123010929)

[ЗАключение 45](#_Toc123010930)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 46](#_Toc123010931)

[ПРИЛОЖЕНИЯ 47](#_Toc123010932)

[Приложение А 48](#_Toc123010933)

[Приложение Б 53](#_Toc123010934)

[Приложение В 57](#_Toc123010935)

[Приложение Г 60](#_Toc123010936)

[Приложение Д 64](#_Toc123010937)

ВВЕДЕНИЕ

Системный анализ – прикладное направление теории систем, применяемое при решении сложных слабоформализуемых проблем. Однако, четкого определения для определения не существует, в разных публикациях он трактуется по-разному.

В основном системный анализ используется в тех задачах, которые не могу быть сразу решены формальных математических методов, поэтому уделяет внимание процессу постановки задачи и использует не только формальные методы, но и методы качественного анализа. Так системный анализ опирается на понятия теории систем, философские концепции, помогает организовать процесс коллективного принятия решения, объединяя специалистов различных областей знаний [1]. Одним из представлений знаний для разных областей, выступает онтология. В информационных технологиях и интеллектуальных системах этот термин используется для обозначения представлений о моделях мира.

Системный анализ не существует в виде строгой методологической концепции и тесно связан с различными направлениями современной науки. Поэтому, в зависимости от того какое определение будет взято за первооснову, зависит решение вопроса о специфических признаках исследования системного объекта [1].

1. ОНТОЛОГИЯ

В информационных технологиях и интеллектуальных системах онтология была принята как модель представления мира, например сущностей, причинных связей и т. п.

Формальную модель онтологии можно представить в виде:

,

|  |  |
| --- | --- |
| где | – конечное множество концептов (понятий, терминов) предметной области, которую представляет онтология;  – конечное множество отношений между концептами (понятиями, терминами) заданной предметной области;  – конечное множество функций интерпретации (аксиоматизации), заданных на концептах и/или отношениях онтологии. |

Естественным ограничением, накладываемым на множество , является его конечность и не пустота. Иначе дело обстоит с компонентами и в определении онтологии. Очевидно, что и должны быть конечными множествами [1].

* 1. Постановка задачи

Придумать предметную область и на основе выбранной предметной области реализовать онтологию в программной среде Protégé 3.2 и реализовать в виде программы на выбранном языке программирования.

* 1. Описание предметной области

В качестве предметной области была выбрана «Материнские платы» (Рисунок 1.2.1).

От абстрактного класса «Материнские платы» наследуем два класса «База плат» и «Типы плат». От класса «База плат» наследуется два класса «INTEL» и «AMD». От класса «Типы плат» наследуется два класса «Z чипсет», «B чипсет» «H чипсет» и «X чипсет». Создадим данную структуру в программе Protégé 3.2 (Рисунок 1.2.2).

Класс «База плат» содержит слоты «Цена», «Процессор». Создадим класс в программе Protégé 3.2 (Рисунок 1.2.3).

Класс «Типы плат» содержит слот «Чипсет». Создадим класс в программе Protégé 3.2 (Рисунок 1.2.6).

Классы «INTEL», «AMD» имеют одинаковый набор слотов. Данные классы содержат слоты «Производитель», «Поколение», «Модель» и «Чипсет», a также содержит слоты, наследуемые от класса «База плат». Создадим класс в программе Protégé 3.2 (Рисунок 1.2.4 и 1.2.5).

Классы «Z чипсет», «B чипсет» «H чипсет» и «X чипсет» имеют одинаковый набор слотов. Данные классы содержат слоты «Производитель», «Поколение», «Цена», «Модель», «Чипсет», «Процессор», a также содержит слоты, наследуемые от класса «База плат». Создадим класс в программе Protégé 3.2 (Рисунок 1.2.7 и 1.2.10).

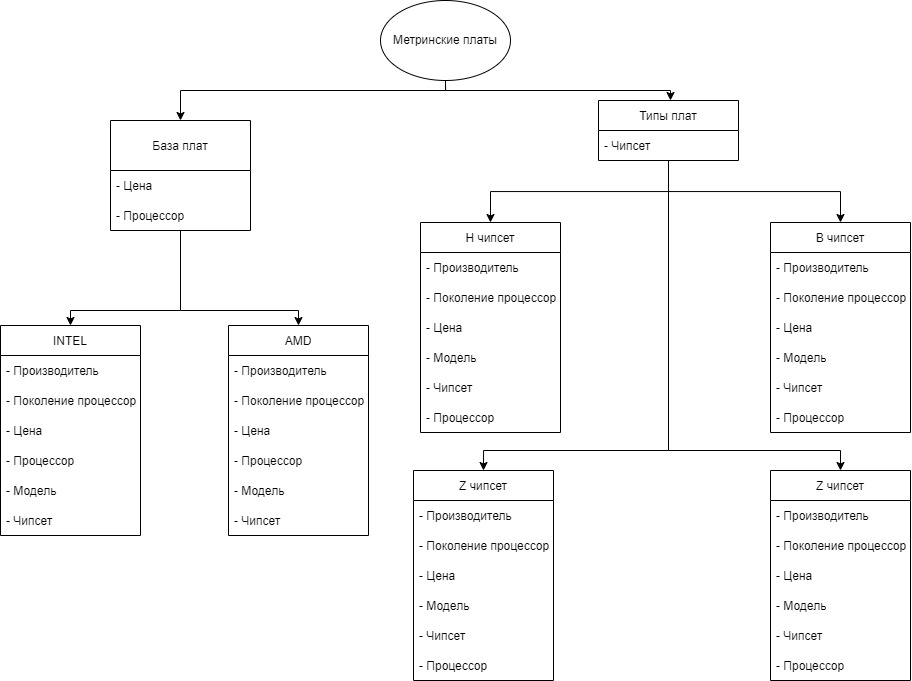


Рисунок 1.2.1 – Модель онтологии

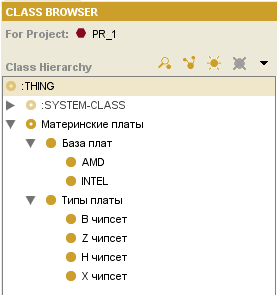


Рисунок 1.2.2 – Иерархия классов онтологии

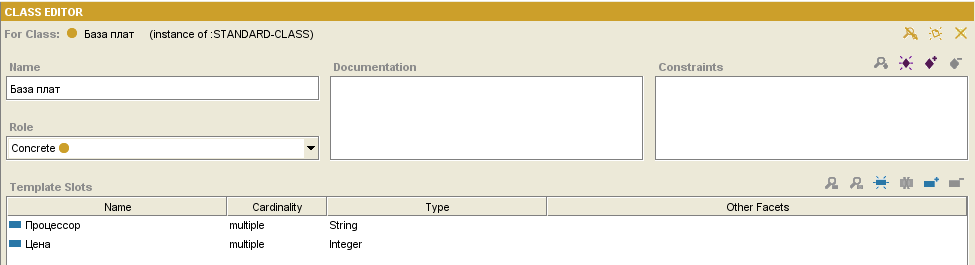


Рисунок 1.2.3 – Класс «База плат»

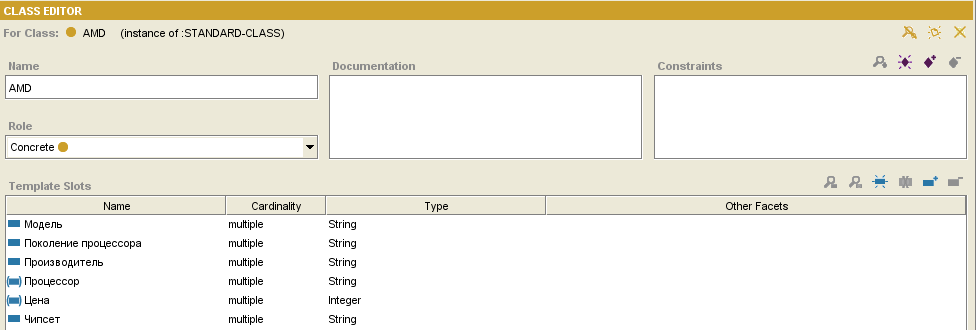


Рисунок 1.2.4 – Класс «AMD»

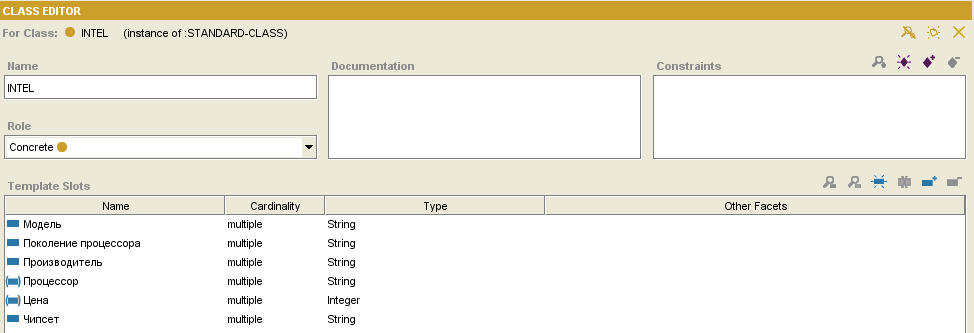


Рисунок 1.2.5 – Класс «INTEL»

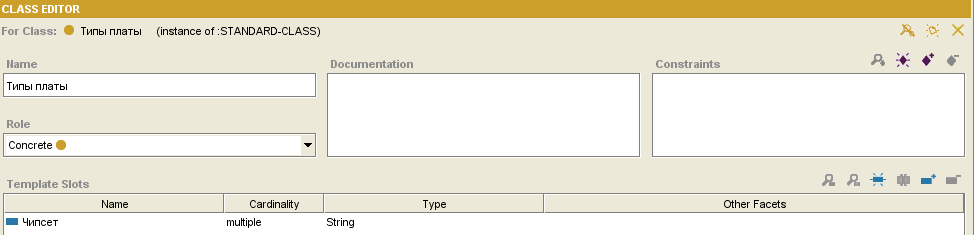


Рисунок 1.2.6 – Класс «Типы плат»

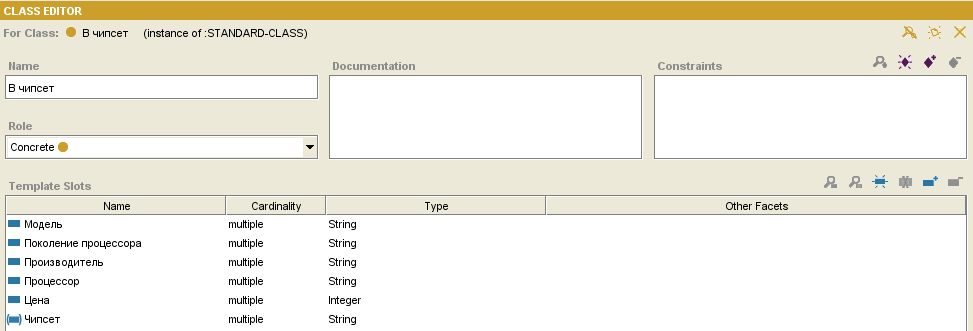


Рисунок 1.2.7 – Класс «B чипсет»

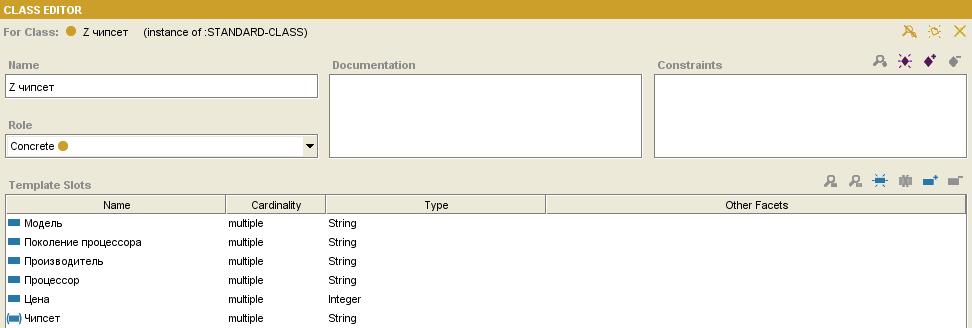


Рисунок 1.2.8 – Класс «Z чипсет»

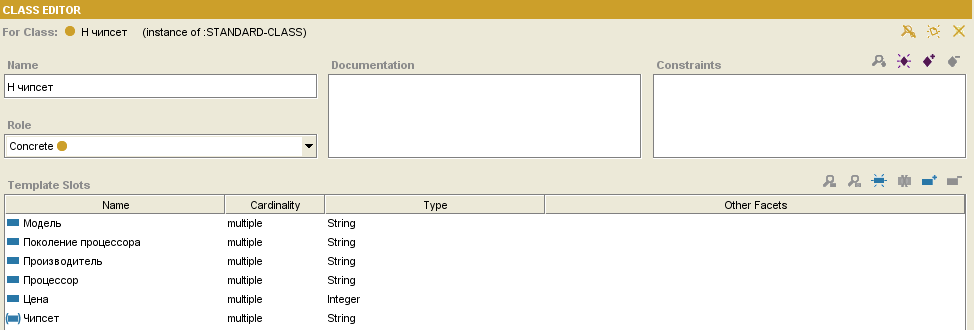


Рисунок 1.2.9 – Класс «H чипсет»

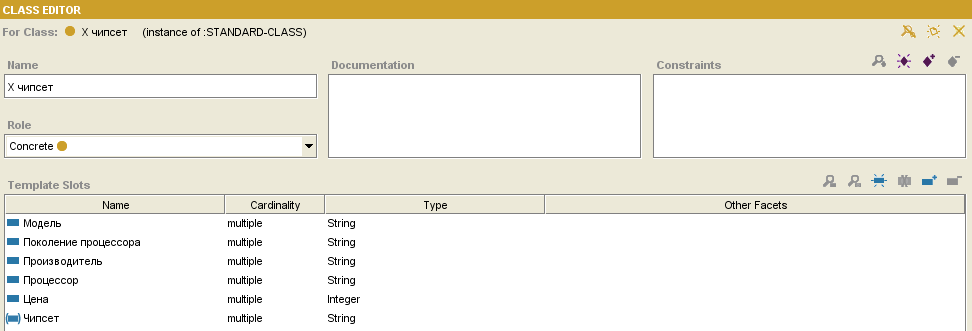


Рисунок 1.2.10 – Класс «B чипсет»

* 1. Создание сущностей

Создадим в каждом классе несколько объектов (Рисунок 1.3.1–1.3.5).

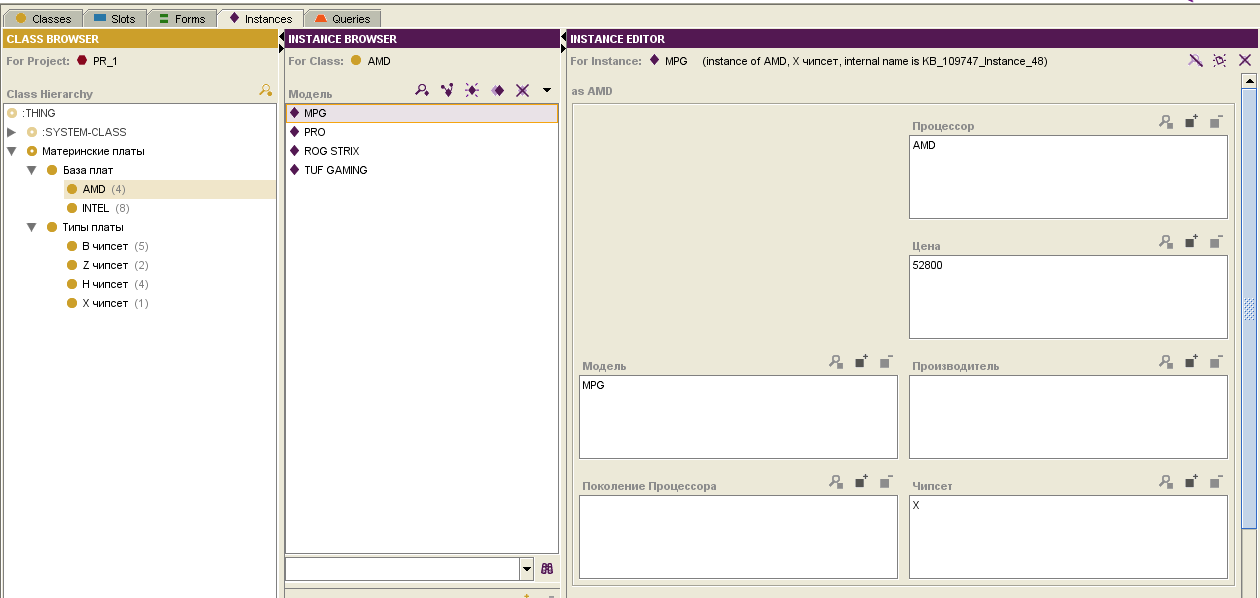


Рисунок 1.3.1 – Объект «MPG» класса «AMD» и класса «X чипсет»

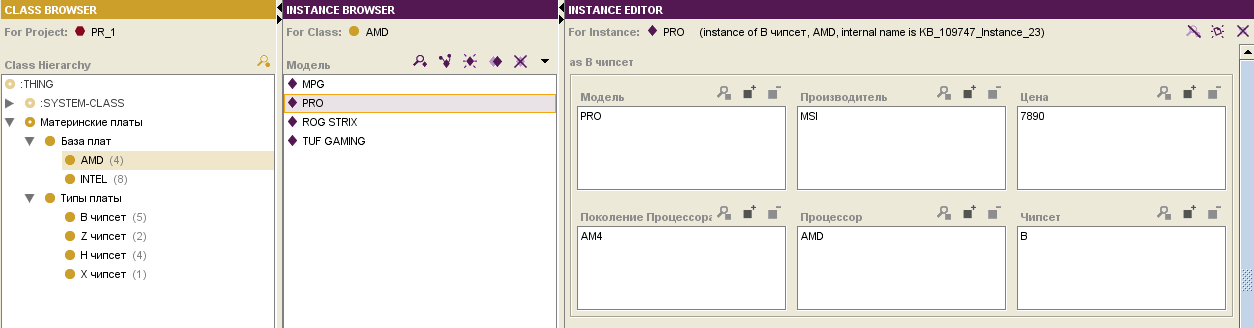


Рисунок 1.3.2 – Объект «PRO» класса «AMD» и класса «B чипсет»

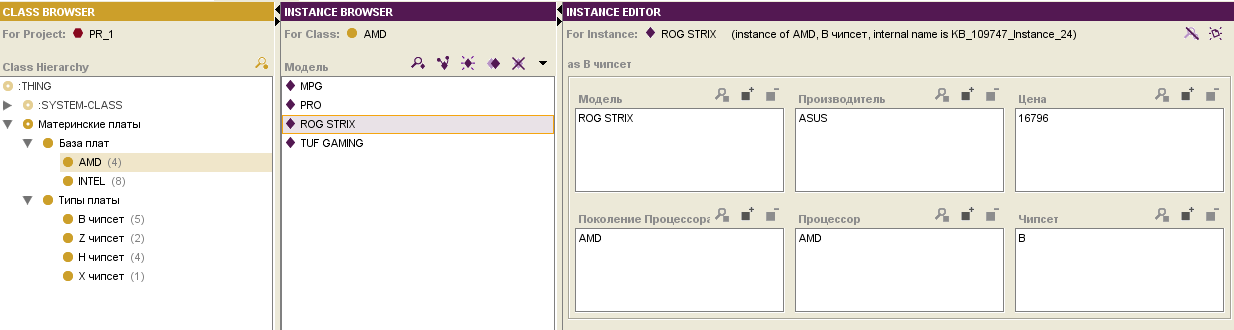


Рисунок 1.3.3 – Объект «ROG STRIX» класса «AMD» и класса «B чипсет»

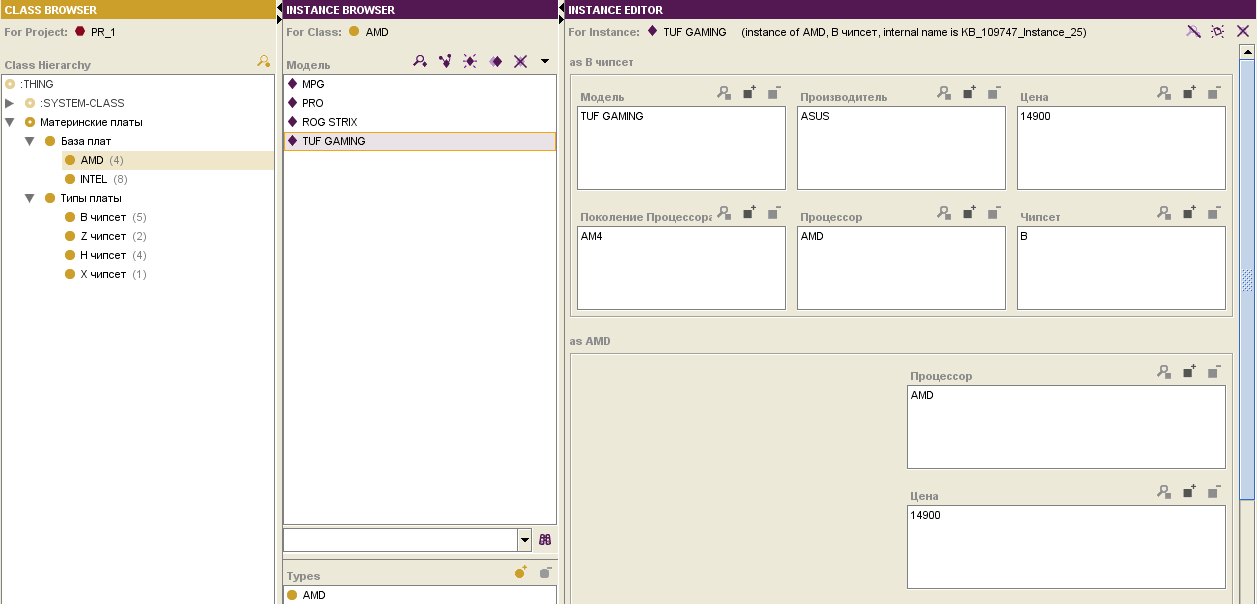


Рисунок 1.3.4 – Объект «TUF GAMING» класса «AMD» и класса «B чипсет»

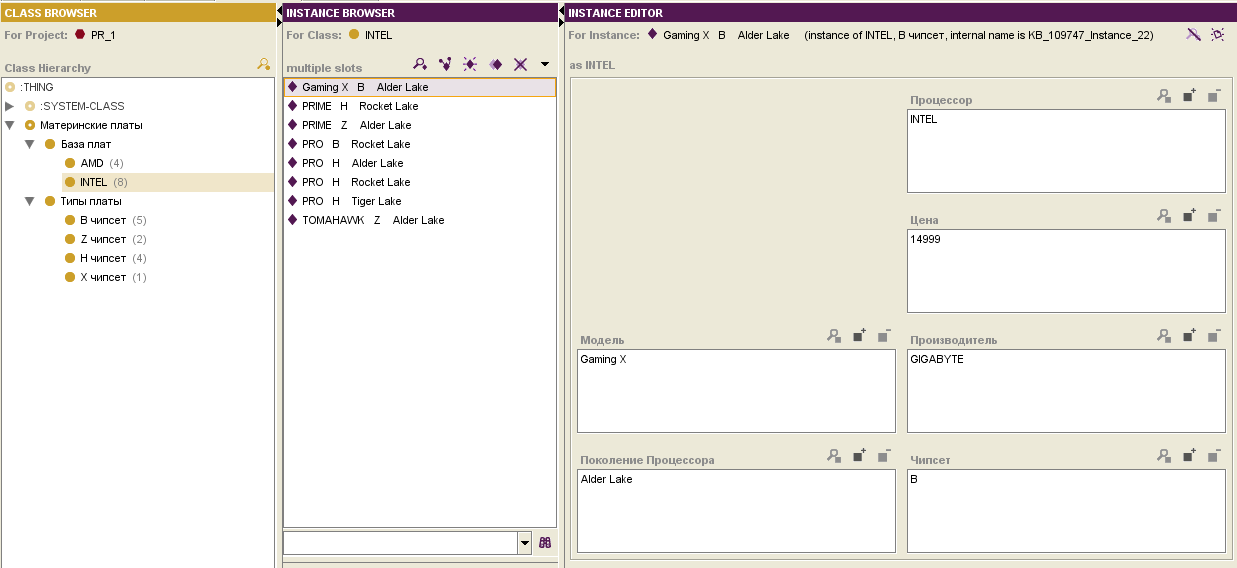


Рисунок 1.3.5 – Объект «Gaming X B Alder Lake» класса «INTEL» и класса «B чипсет»

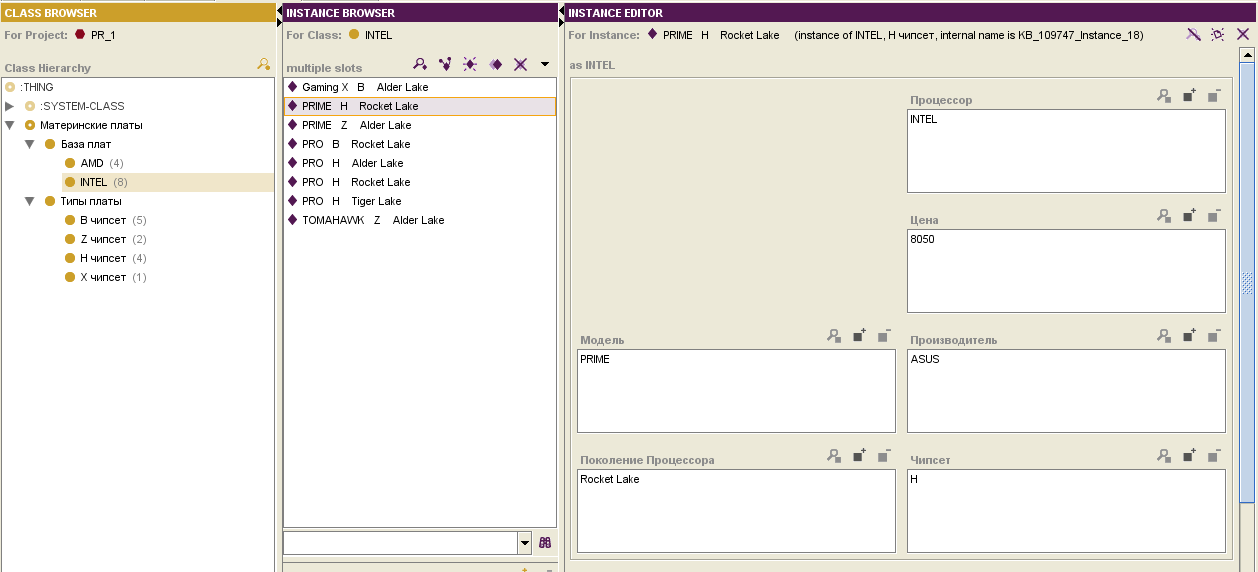


Рисунок 1.3.6 – Объект «Prime H Rocket Lake» класса «INTEL» и класса «H чипсет»

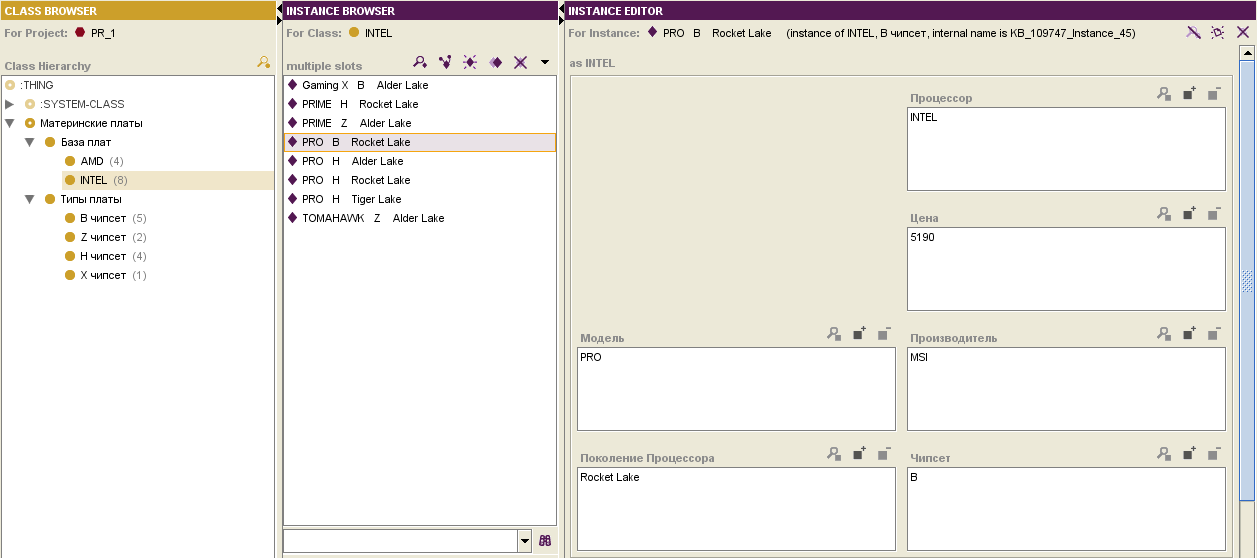


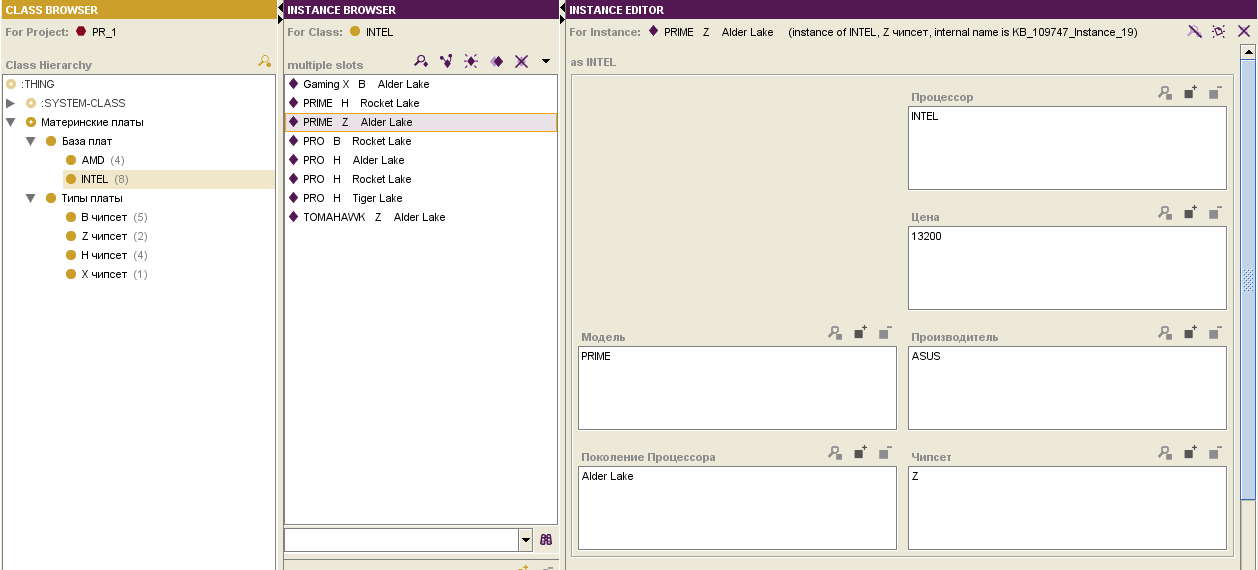
Рисунок 1.3.7– Объект «Prime H Rocket Lake» класса «INTEL» и класса «H чипсет»

Рисунок 1.3.8 – Объект «Prime Z Alder Lake» класса «INTEL» и класса «Z чипсет»

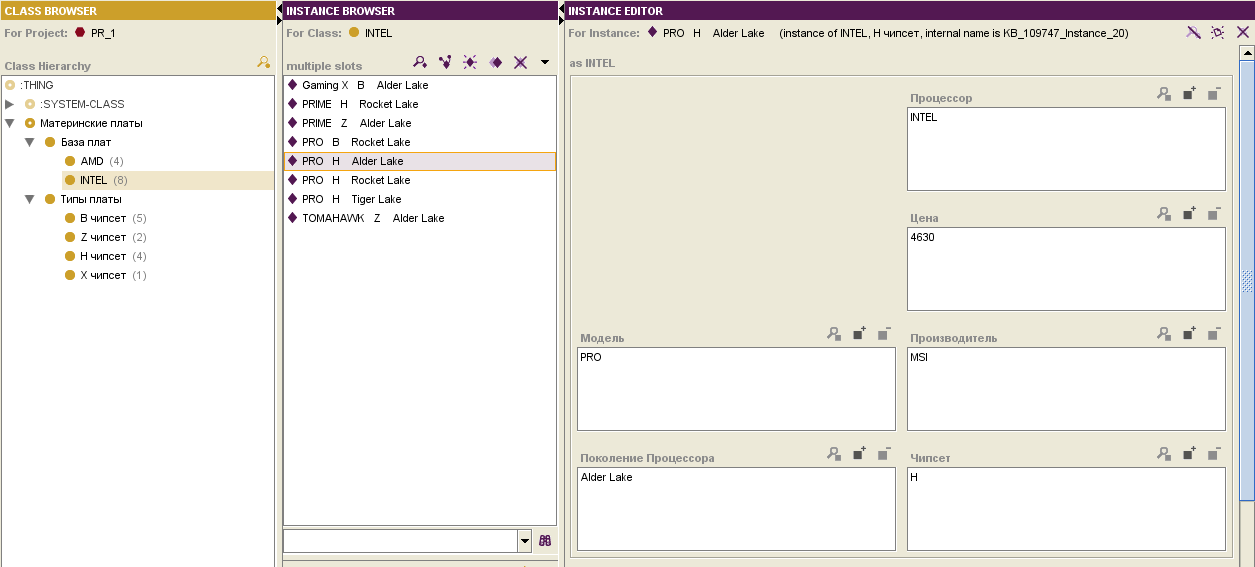


Рисунок 1.3.9 – Объект «PRO H Alder Lake» класса «INTEL» и класса «H чипсет»

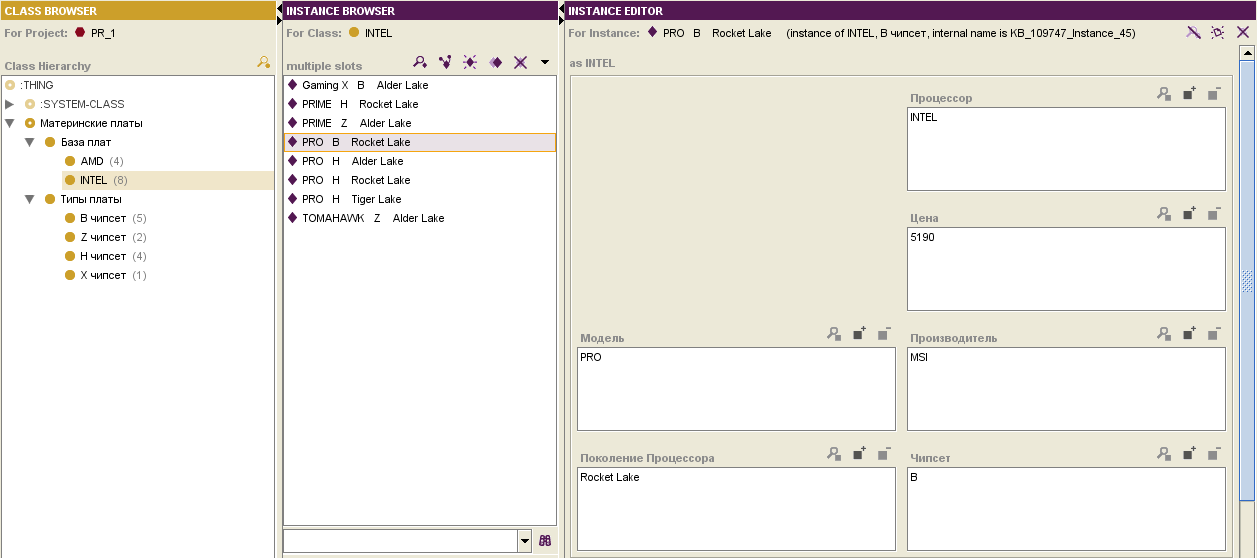


Рисунок 1.3.10 – Объект «PRO B Rocket Lake» класса «INTEL» и класса «B чипсет»

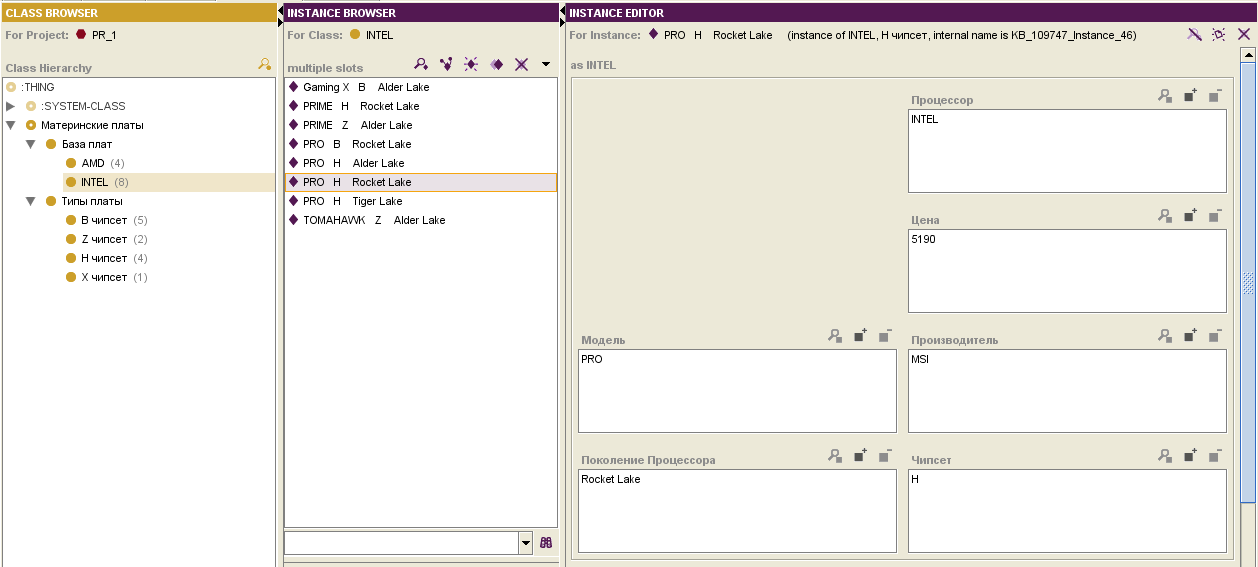


Рисунок 1.3.11 – Объект «PRO H Rocket Lake» класса «INTEL» и класса «H чипсет»

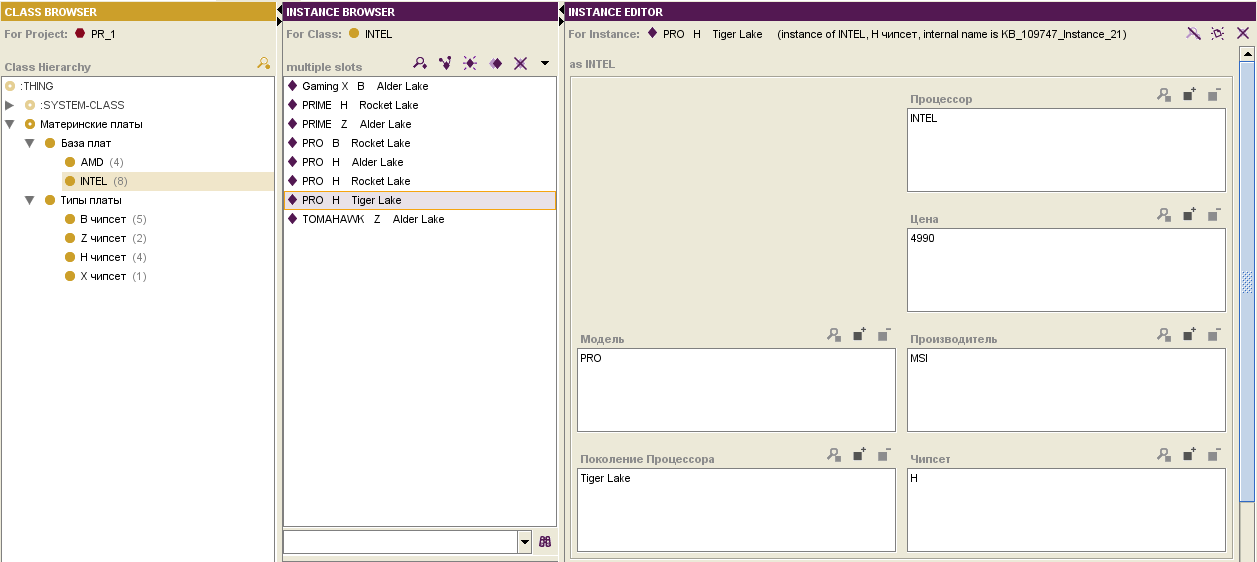


Рисунок 1.3.12 – Объект «PRO H Triger Lake» класса «INTEL» и класса «H чипсет»

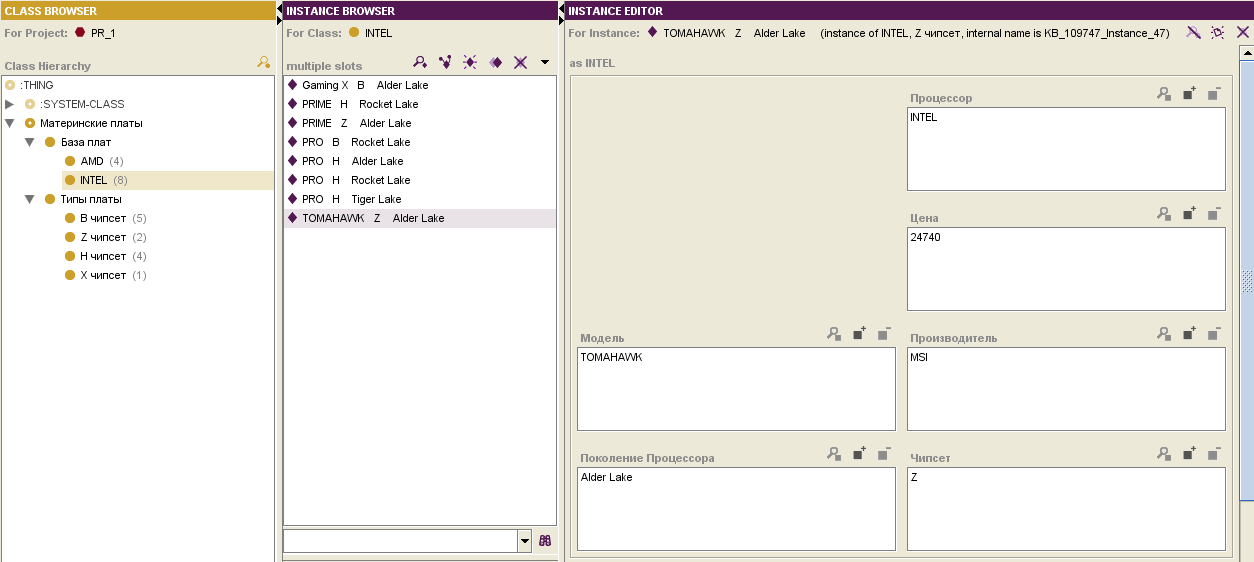


Рисунок 1.3.13 – Объект «TOMAHAWK Z Alder Lake» класса «INTEL» и класса «Z чипсет»

* 1. Создание запросов

Создадим запросы между объектами классов онтологии.

Первый запрос «Поиск по Базе плат». Найдем все совместимые материнские платы с «INTEL» (Рисунок 1.4.1).

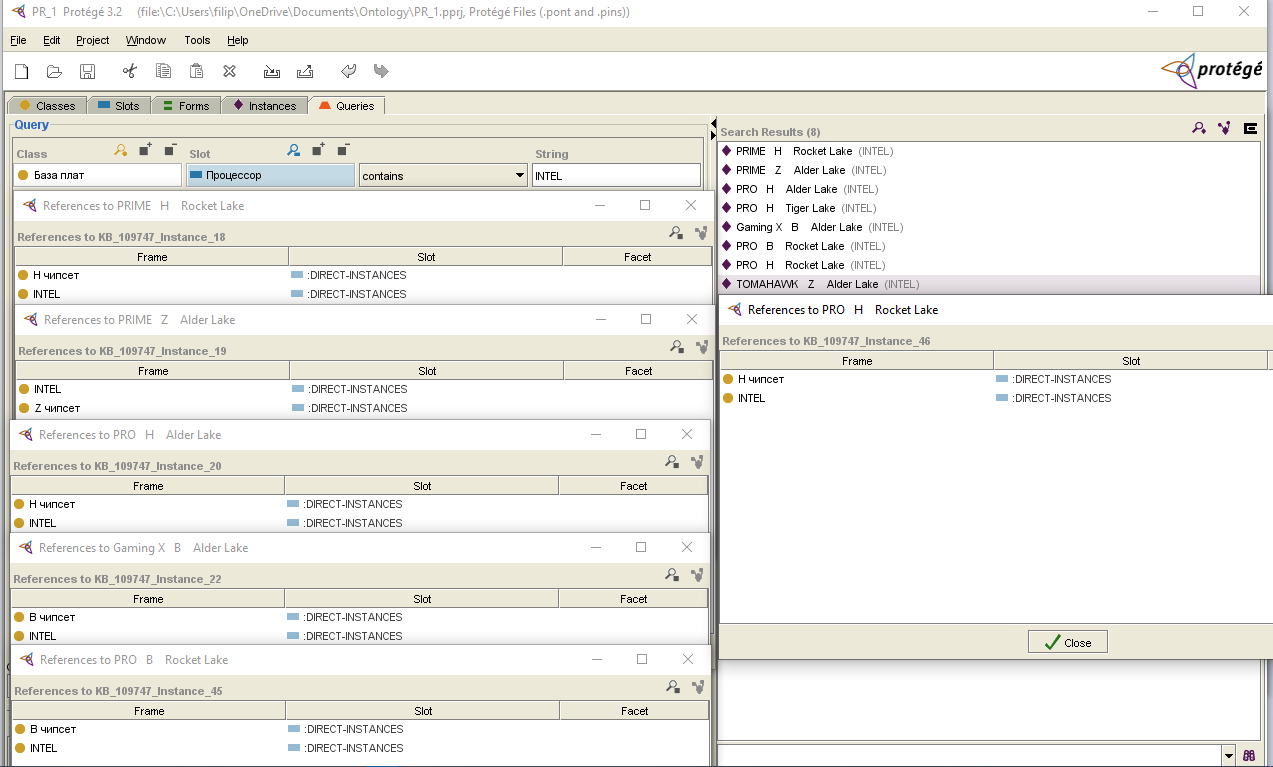


Рисунок 1.4.1 – Запрос «Найти совместимые материнские платы с INTEL»

Вторым запросом будет «Поиск по B чипсет». Найдем все материнские платы, которые дороже чем 15000 рублей (Рисунок 1.4.2).

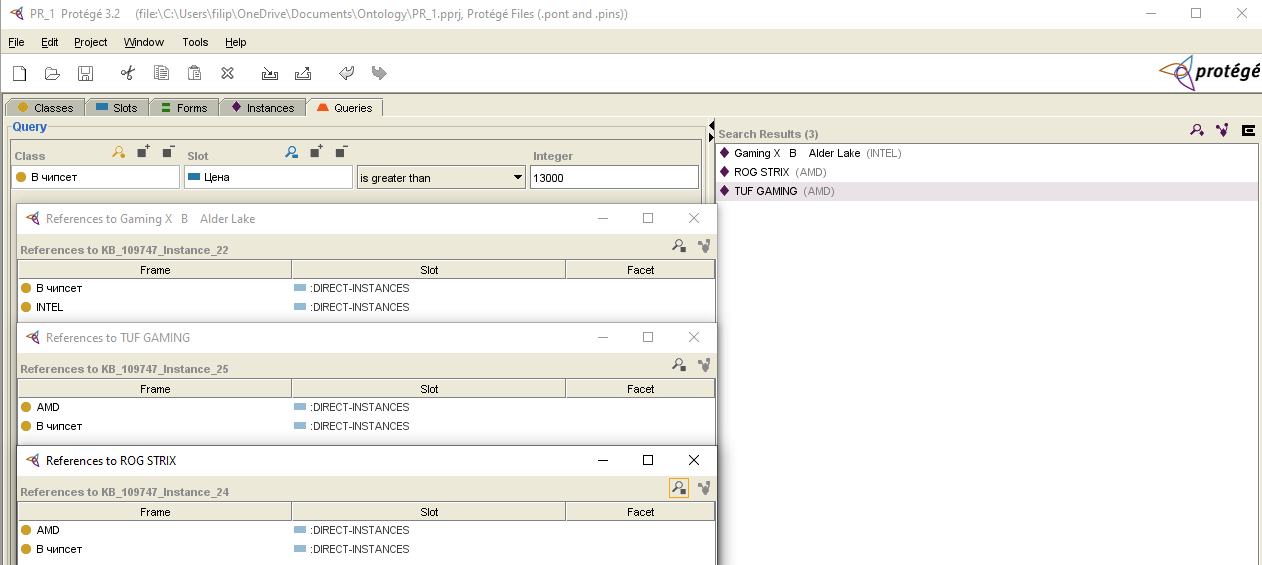


Рисунок 1.4.2 – Запрос «Материнские платы по цене»

Третий запрос «Материнскую плату по производителю». Найдем материнские платы по производителю «ASUS» (Рисунок 1.4.3).

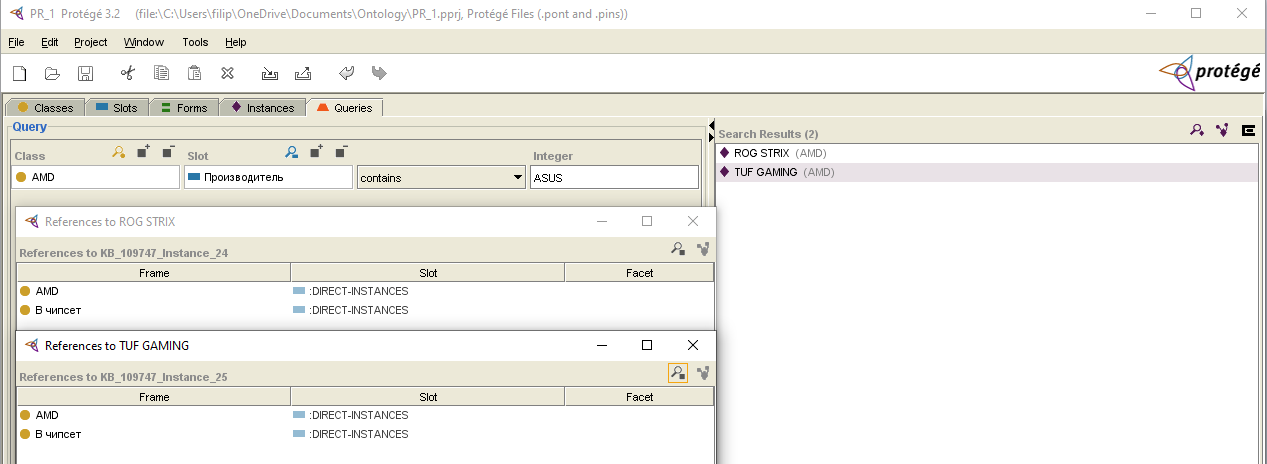


Рисунок 1.4.3 – Запрос «Материнскую плату по производителю»

* 1. Программная реализация

Создадим онтологию в программной реализации на языке высокого уровня Python [2] и выполним запросы.

Первым запросом найдем все совместимые материнские платы с «INTEL».

Вторым запросом найдем все материнские платы, которые дороже чем 15000 рублей.

Найдем материнские платы по производителю «ASUS».

Данные запросы продемонстрированы на Рисунках 1.5.1 – 1.5.3. Листинг кода приведен в Приложении А.

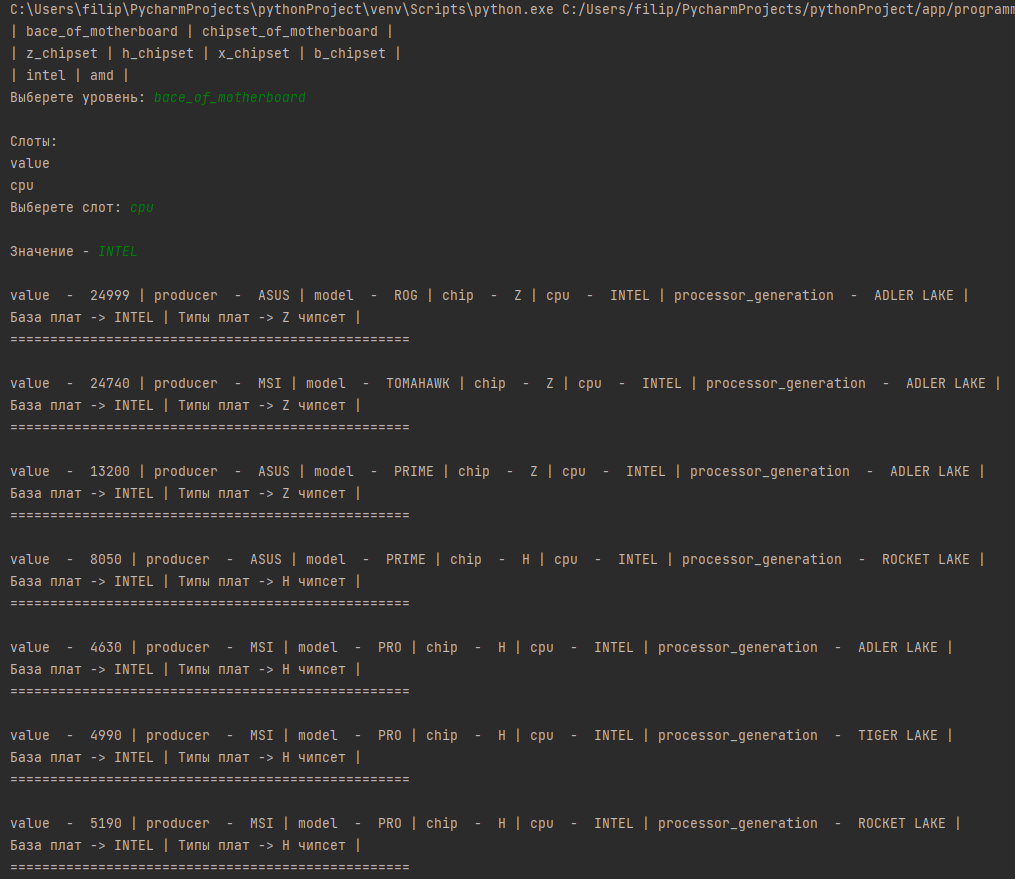


Рисунок 1.5.1 – Первый запрос в программной реализации

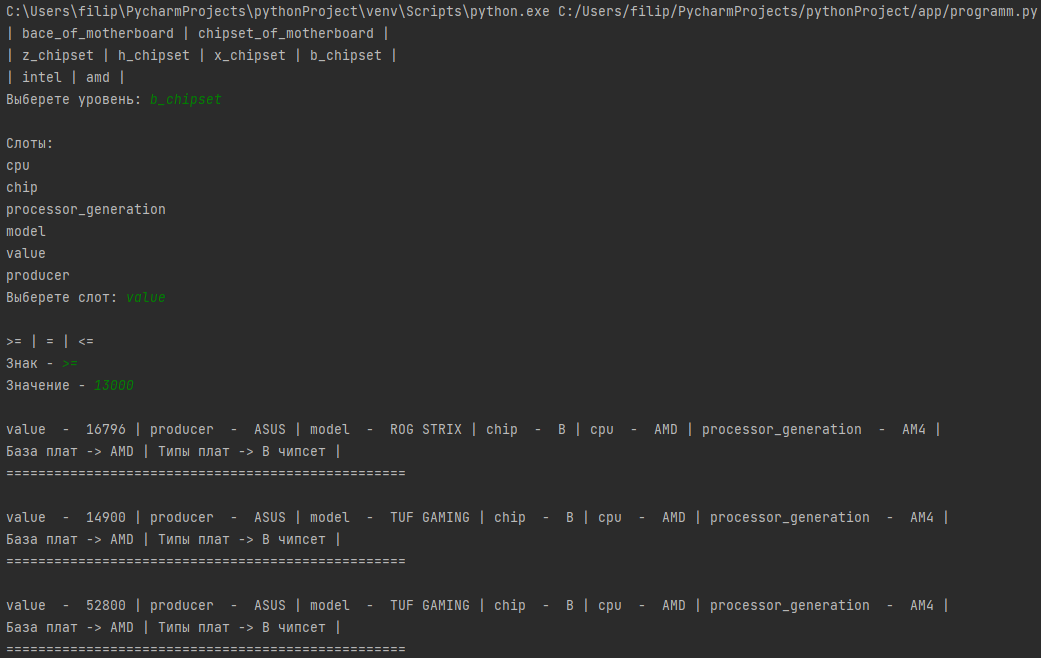


Рисунок 1.5.2 – Второй запрос в программной реализации



Рисунок 1.5.3 – Третий запрос в программной реализации

1. Алгоритм оптимизации Отжигом

Метод отжига служит для нахождения глобального минимума некоторой функции . Элементы непрерывного или дискретного множества () представляют состояние системы физической системы, а значения функции в этих точках – энергия системы. В каждый момент времени температура системы уменьшается и с каждым разом системе сложнее переходить в новое состояния , где – семейство вероятностных распределений [1].

Описание алгоритма:

Метод отжига служит для поиска глобального минимума некоторой функции 𝑓(𝑥), заданной для 𝑥 некоторого пространства 𝑆, дискретного или непрерывного. Элементы множества 𝑆 представляют собой состояние воображаемой физической системы («энергетические уровни), а значения функции 𝑓 в этих точках используется как энергия системы 𝐸=𝑓(𝑥). В каждый момент предполагается заданная температура системы 𝑇, как правило, уменьшающая с течением времени. После попадания в состояние 𝑥 при температуре 𝑇, следующее состояние системы выбирается в соответствии с заданным порождающим семейством вероятностных распределений 𝒢(𝑥,𝑇), которое при фиксированных 𝑥 и 𝑇 задает случайный элемент 𝐺(𝑥,𝑇) со значениями в пространстве 𝑆. После генерации нового состояния 𝑥′=𝐺(𝑥,𝑇), система с вероятностью ℎ(Δ𝐸,𝑇) переходит к следующему состоянию 𝑥′, в противном случае процесс генерации 𝑥′ повторяется. Здесь Δ𝐸 обозначает приращение функции 𝑓(𝑥′)−𝑓(𝑥). Величина ℎ(Δ𝐸, 𝑇) называется вероятностью принятия нового состояния.

Как правило, в качестве функции ℎ(Δ𝐸,𝑇) выбирается либо точное значение соответствующей физической величины

либо приближенное значение

Вторая формула используется наиболее часто. При ее использованииКонкретная схема метода отжига задается следующими параметрами:

Конкретная схема метода отжига задается следующими параметрами:

* Выбором закона изменения температуры 𝑇(𝑘), где 𝑘 – номер шага.
* Выбором порождающего семейства распределений 𝒢(𝑥,𝑇).
* Выбором функции вероятности принятия ℎ(Δ𝐸,𝑇).
  1. Постановка задачи

Пусть существует некоторое количество пунктов в 3-мерном пространстве. Коммивояжеру необходимо посетить все эти пункты.

* 1. Расчетная часть

Рассчитаем одну итерацию алгоритма.

1. Создадим 6 точек для графа и обозначаем их величины:

* 1 – 2 длиной 52,
* 1 – 3 длиной 73,
* 1 – 4 длиной 87,
* 1 – 5 длиной 66,
* 1 – 6 длиной 89,
* 2 – 3 длиной 60,
* 2 – 4 длиной 59,
* 2 – 5 длиной 54,
* 2 – 6 длиной 90,
* 3 – 4 длиной 100,
* 3 – 5 длиной 79,
* 3 – 6 длиной 79,
* 4 – 5 длиной 58,
* 4 – 6 длиной 93,
* 5 – 6 длиной 69,

1. Обозначим первоначальный путь:
2. Найдем длину первоначального пути: 499
3. Сгенерируем возможный путь, через перестановку прохождения некоторых точек: и длину пути полученной системы: 476
4. Т. к. энергия полученной системы лучше, уменьшаем температуру по Больцмановскомуотжигу.
5. Принимаем новый путь: , а энергия системы: .
   1. Программная реализация

Данный алгоритм реализован на языке Python [2], а для визуализации данных использовался сгенерированный скрипт для Matplotlib [3].

Визуализация решения представлена на Рисунке 2.3.1. Визуализация начального графа алгоритма представлена на Рисунке 2.3.2

На Рисунке 2.3.3 предоставлена визуализация конечного пути после работы алгоритма.

На Рисунке 2.3.4 предоставлен график снижения энергии к количеству итераций.

Листинг программы приведен в Приложении Б.

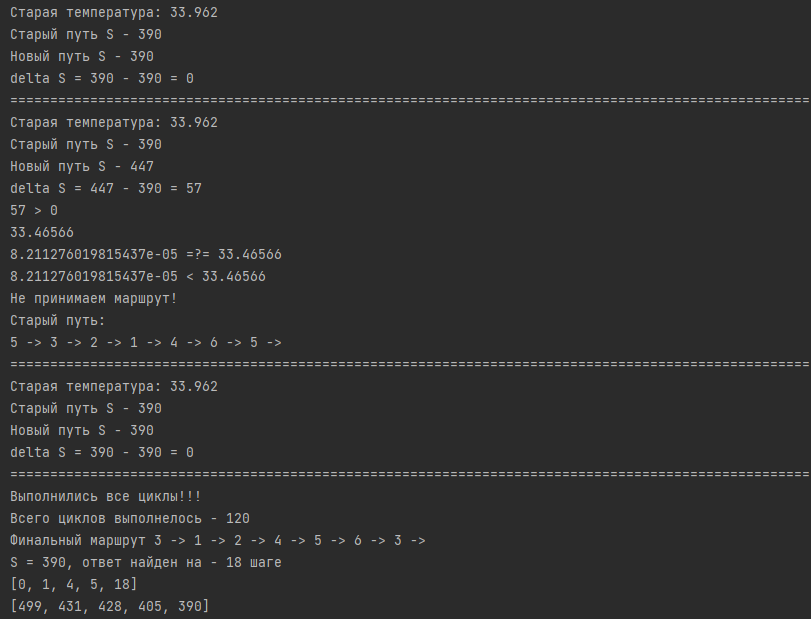


Рисунок 2.3.1 – Визуализация решения представлена

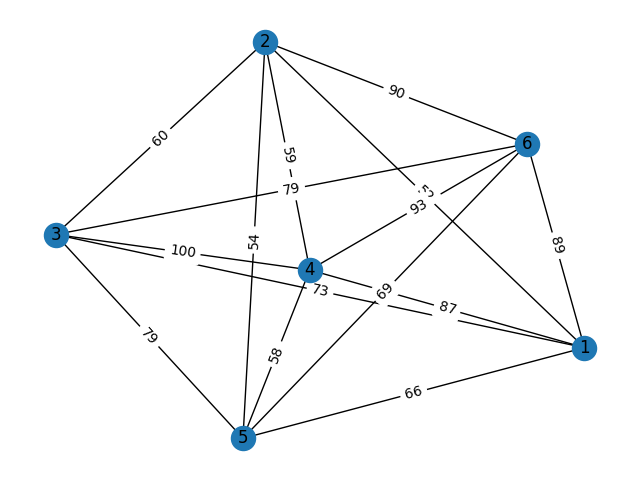


Рисунок 2.3.2 – Визуализация начального графа алгоритма

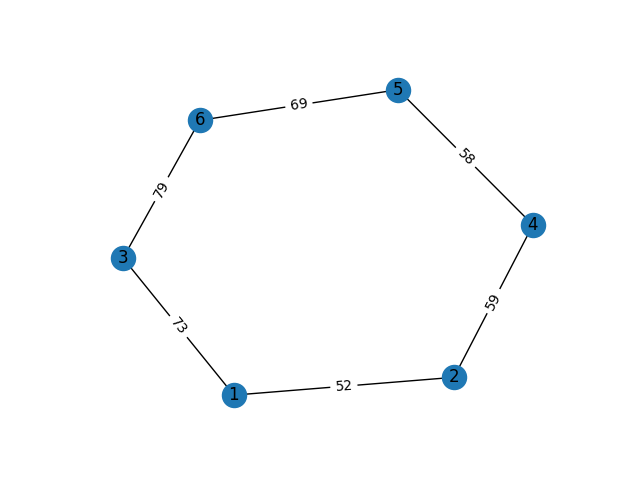


Рисунок 2.3.3 – Визуализация конечного пути после работы алгоритма

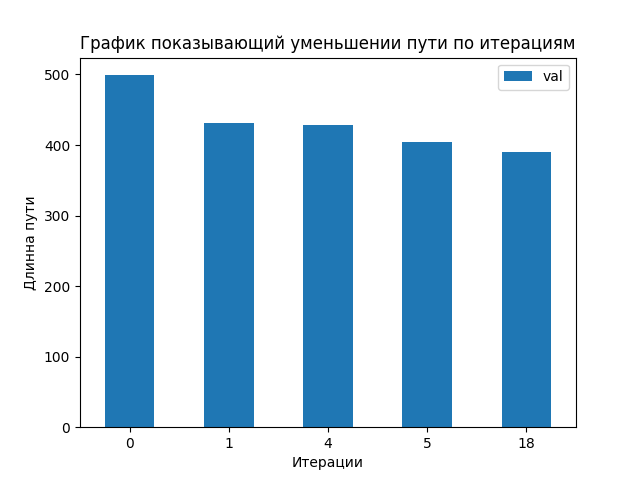


Рисунок 2.3.4 – Визуализация конечного пути после работы алгоритма

1. АЛГОРИТМ ОПТИМИЗАЦИИ РОЯ

Роевой алгоритм использует рой частиц, где каждая частица представляет собой потенциальное решение. Положение частицы в пространстве поиска определяет в соответствии со свои опытом и опытом других частиц. В процессе оптимизации частицы обмениваются между собой информацией о лучших найденных решениях и изменяют свои позиции и скорости в соответствии с этими параметрами. При нахождении глобального оптимального решения все частицы сразу узнают об этом [1].

Роевой алгоритм 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 𝒚𝒊, которая отражает когнитивную компоненту. Аналогично значение глобального оптимума, достигнутого частицами роя, будем обозначать 𝒚𝒊̂, которое отражает социальную компоненту. Таким образом, каждая частица роя подчиняется достаточно простым правилам поведения (изложенным ниже формально), которые учитывают локальный успех каждой особи и глобальный оптимум всех особей (или некоторого множества соседей) роя.

Каждая i-я частица характеризуется в момент времени t своей позицией (𝑡) в гиперпространстве и скоростью движения 𝑣𝑖(𝑡). Позиция частицы изменяется в соответствии со следующей формулой:

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

(𝑡+1) =(𝑡)+(𝑡)[(𝑡)−(𝑡)]+ (𝑡)[(𝑡)−(𝑡)] (9.2)

Здесь 𝑣𝑖(𝑡) - 𝑗-ая компонента скорости (𝑗=1,… ,𝑛𝑥) 𝑖-ой частицы в момент времени 𝑡, (𝑡) - 𝑗-я координата позиции 𝑖 -й частицы, и – положительные

коэффициенты ускорения (часто полагаемые 2), регулирующие вклад когнитивной и социальной компонент, (𝑡) и (𝑡) ~ (0,1) - случайные числа из диапазона [0,1], которые генерируются в соответствии с нормальным распределением и вносят элемент случайности в процесс поиска. Кроме этого, ( (𝑡) - персональная лучшая позиция по 𝑗-й координате 𝑖-ой частицы, а 𝑦𝑗̂(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:

где 𝑓: →𝑅 фитнесс-функция. Как и в эволюционных алгоритмах фитнесс-функция измеряет близость текущего решения к оптимуму.

Существует два основных подхода в оптимизации роя частиц, под названиями lbest и gbest, отличающиеся топологией соседства, используемой для обмена опытом между частицами. Для модели gbest лучшая частица определяется из всего роя. Глобальная лучшая позиция (gbest) (𝑡) в момент 𝑡 определяется в соответствии с

(𝑡)∈{(𝑡),…, (𝑡)}|𝑓((𝑡))=𝑚𝑖𝑛{(𝑡),…, (𝑡)} (9.4)

где 𝑛𝑠 – общее число частиц в рое.

В процессе поиска решения описанные действия выполняются для каждой частицы роя.

* 1. Постановка задачи

Пусть задана функция . Известно, что у данной функции есть единственный минимум , в точке . Найдем минимум заданной функции методом Роя.

* 1. Расчётная часть

Опишем одну итерацию алгоритма для 2-х частиц.

Инициализируем частицы в случайные позиции на плоскости и найдем значения функции в заданных функциях .

Отсюда видно, что глобальный минимум на данном этапе точке .

Рассчитаем нового положения точек в пространстве, на основе полученного решения.

Рассчитаем новое положение для точки :

Рассчитаем новое положение точки :

Найдем новый глобальный минимум:

Новый глобальный минимум равен .

* 1. Программная реализация

Данный алгоритм реализован на языке Python [2], а для визуализации данных использовался скрипт для Matplotlib [3].

Создадим рой на 100 частиц и 1000 итераций, результат вычислений приведен на Рисунке 3.3.1.

Визуализация начального положения роя приведена на Рисунке 3.3.2.

Визуализация конечного положения роя приведена на Рисунке 3.3.3.

Визуализация конечного положения роя, в другой интерпретации приведена на Рисунке 3.3.4.

Листинг решения приведен в Приложении В.

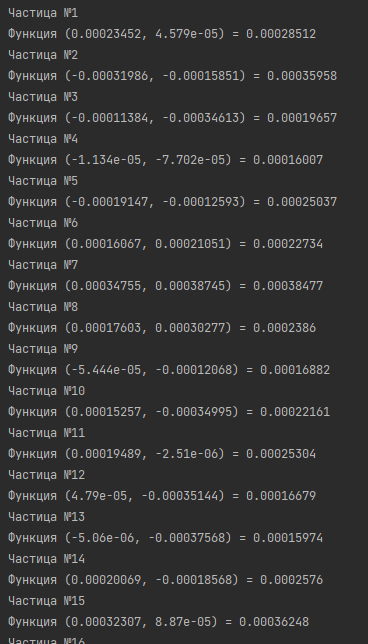


Рисунок 3.3.1 – Результат выполнения алгоритма

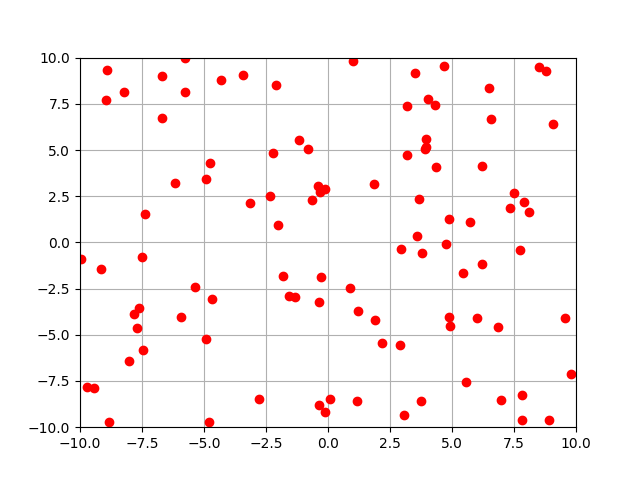


Рисунок 3.3.32– Визуализация начального положения роя

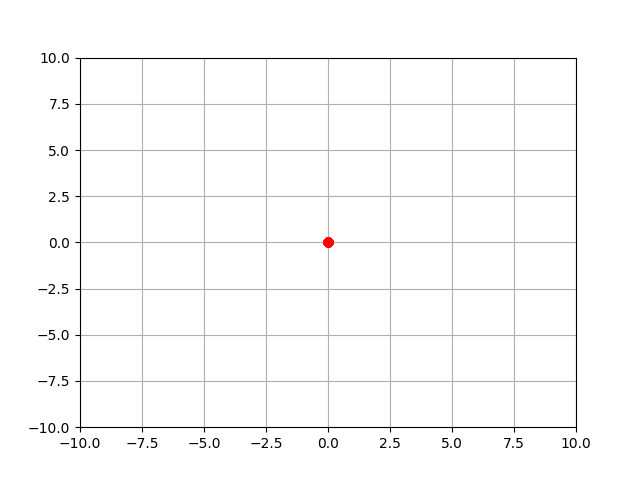


Рисунок 3.3.3 – Визуализация конечного положения роя

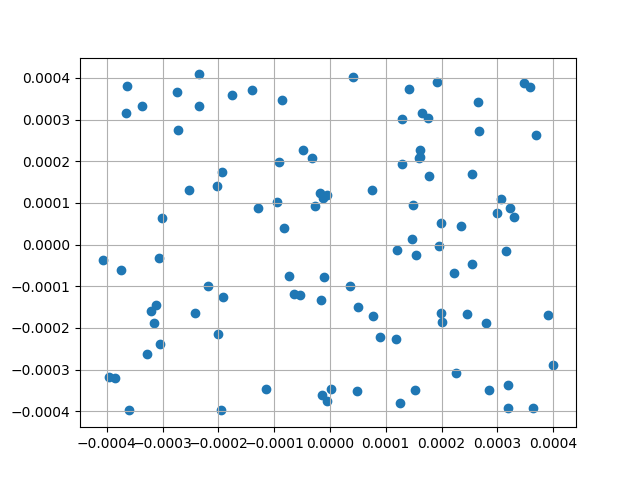


Рисунок 3.3.4 – Визуализация конечного положения роя

1. МУРАВЬИНЫЙ АЛГОРИТМ

Муравьиные алгоритмы основаны на использовании популяции потенциальных решений и разработаны для решения задач комбинаторной оптимизации, прежде всего, поиска различных путей на графах. 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 [1].

Первые муравьиные алгоритмы, разработанные в , относятся по современной классификации к «муравьиным системам» (antsystems), которые будут изложены ниже.

Сначала мы рассмотрим (исключительно в учебных целях) 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 𝑛𝐺 = |𝑉| - число узлов в графе. Обозначим 𝐿𝑘 - 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.10.7. С каждой дугой, соединяющей вершины (𝑖, 𝑗), ассоциируем концентрацию феромона 𝜏𝑖𝑗.

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

Муравей выбирает следующую дугу пути случайным образом в фактически в соответствии с алгоритмом 10.1 следующим образом. Множество муравьев k={1,…,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 -й муравей находится в 𝑖-ой вершине,то он выбирает следующую вершину 𝑗 ∈ на основе вероятностей перехода

Здесь представляет множество возможных вершин, связанных с -й вершиной, для 𝑘-го муравья. Если для любого -го узла и 𝑘-го муравья =∅, тогда предшественник узла 𝑖 включается в . В этом случае в пути возможны петли. Эти петли удаляются при достижении конечного города пути. В (4.2) 𝛼 - положительная константа, которая определяет влияние концентрации феромона. Очевидно большие значения ∝ 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

Здесь – длина пути, построенного 𝑘-м муравьем в момент времени 𝑡.

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

где - число муравьев. Общая концентрация феромона для данной дуги пропорциональна «качеству» путей, в которые входит эта дуга, поскольку откладываемое количество феромона 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 (𝑡) обозначает решение в момент 𝑡, и некоторая функция 𝑓((𝑡)) выражает качество решения. Если Δ не пропорционально качеству решения и все муравьи откладывают одинаковое количество феромона Δ=Δ=⋯=Δ, 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 мы имеем явную оценку качества решения, которая ведет к тому, что дуги, входящие в длинные пути, становятся менее привлекательными для окончательных решений.

* 1. Постановка задачи

Дано 6 точек, между которыми существуют связи необходимо найти кротчайший путь от пункта 1 до пункта 6, используя муравьиный алгоритм.

* 1. Расчётная часть

Опишем одну итерацию для 2-х муравьев .

Вычислим, вероятность перехода из пункта 1 в пункт 2.

Выберем случайное число , . Т. к. , следовательно выберет путь .

Аналогично считаем остальные переходы.

Пусть , тогда , следовательно выбираем путь .

Пусть , тогда , следовательно выбираем путь .

Пусть , тогда , следовательно выбираем путь .

Таким образом для мы вычислили путь:

Выполним те же действия для .

Пусть , тогда , следовательно путь не выбирается.

Т. к. путь не подошел ищем следующий вариант.

Пусть , тогда , следовательно выбираем путь .

Пусть , тогда , следовательно путь не выбирается.

Т. к. путь не подошел ищем следующий вариант.

Пусть , тогда , следовательно выбираем путь .

Итак, для мы вычислили путь:

Вычислим пройденные дистанции и изменения феромона на пройденном пути: , , , .

Таким образом видно, что путь был лучше на данной итерации и на пути, который прошел , увеличение числа феромона больше.

* 1. Программная реализация

Данный алгоритм реализован на языке Python [2], а для визуализации данных использовался скрипт для Matplotlib [3].

Создадим колонию на 6 муравьев и 500 итераций, результат вычислений приведен на Рисунке 4.3.1.

Визуализация решения приведена на Рисунках 4.3.2 ­– 4.3.3.

Листинг решения приведен в Приложении Г.

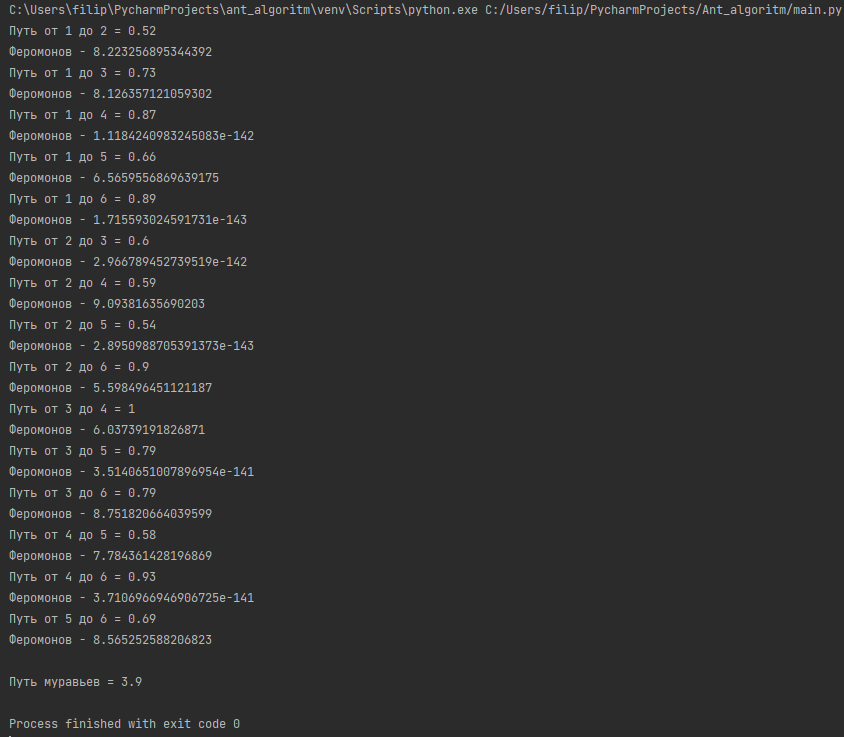


Рисунок 4.3.1 – Результат выполнения алгоритма

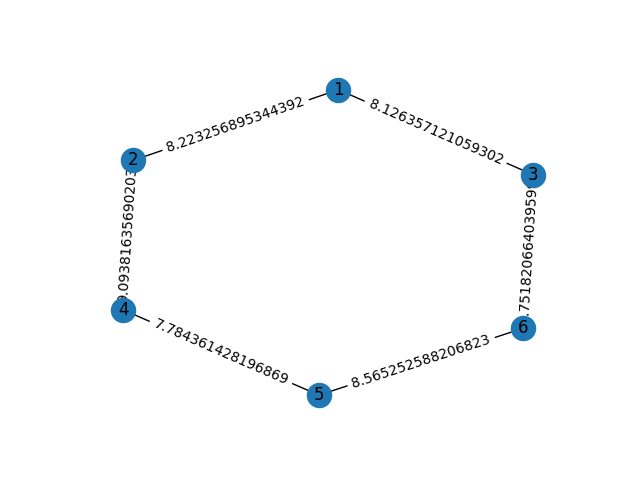


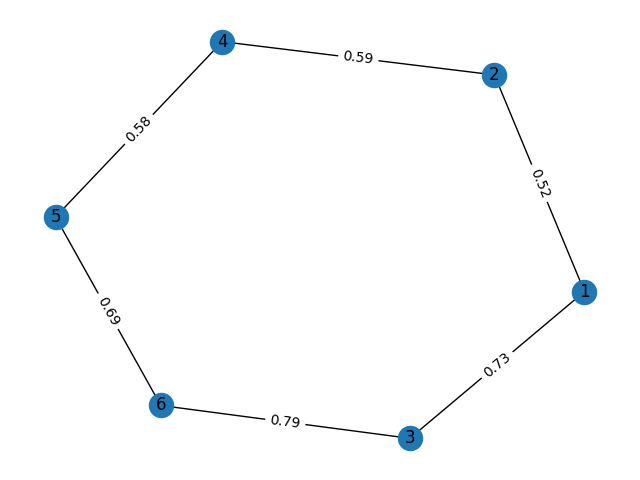
Рисунок 4.3.2 – Визуализация решения, на гранях отображено количество феромонов  


Рисунок 4.3.2 – Визуализация решения, на гранях отображено длина пути между городами

1. ТАСУЮЩИЙ АЛГОРИТМ ПРЫГАЮЩИХ ЛЯГУШЕК

По сути, алгоритм представляет собой гибридизацию меметического алгоритма и алгоритма роя частиц. Алгоритм вдохновлен поведением группы лягушек в процессе поиска пищи. [3].

Рассматриваем задачу глобальной условной максимизации. Алгоритм SFL включает в себя следующие основные шаги:

1. Инициализация популяции.
2. Сортировка лягушек по значению функции.
3. Разделение лягушек по группам (мемеплексам), разделение выполняем по следующему правилу соседства: агента относим к мемеплексу , агента - к мемемплексу и т. д. до мемеплекса , к которому относим агента ; агента относим к мемеплексу , агента , - к мемеплексу и так далее до посреднего агента , который должен быть отнесен к мемеплексу . Полагаем, что величины |S], |S7 кратны.
4. Для каждого из мемеплексов выполним процедура меметрической эволюции. В каждом из мемеплексов находим лучшего агента по целевой функции. Пытаемся улучшить положение агента в направлении лучшего агента в мемеплексе:

где Rand число от 0 до 1, – значение переменной лучшего агента в мемеплексе, а – значение переменной агента чьё положение мы хотим улучшить, – новое значение агента.

1. Проверяем улучшилось ли положение агента, если нет, то пытаемся улучшить положение агента в направлении лучшего агента в популяции:

где Rand число от 0 до 1, – значение переменной лучшего агента в популяции, а – значение переменной агента чьё положение мы хотим улучшить, – новое значение агента.

1. Проверяем улучшилось ли положение агента, если нет, то создаём нового агента вместо улучшаемого в популяции.
2. После чего нужно повторить операции по меметрической эволюции для заданного количества агентов в мемеплексах.
3. Из всех мемеплексов объединяем всех агентов в популяцию.
4. Если условие завершения итераций не выполнено, то возвращаемся к шагу 2.
   1. Постановка задачи

Пусть задана функция . Известно, что у данной функции есть единственный минимум , в точке . Найдем минимум заданной функции методом прыгающих лягушек.

Рассмотрим работу алгоритма.

Первый шаг это иници

* 1. Расчётная часть

Укажем входные параметры системы:

* Количество лягушек в популяции: 4;
* Количество итераций: ;
* Количество групп: ;

Выберем случайные места нахождения наших лягушек в диапазоне от -10 до +10 и вычислим значения функции в них:

Согласно, значениям функции, отсортируем наших лягушек.

Распределим их по группам заполняя каждую группу по 1 лягушке от самого большого значения функции к наименьшему:

Группа 1:

Группа 2:

В первой группе направим первую лягушку в направлении лучшей по функции лягушку группы:

Рассчитаем новое значение функции;

Новое значение лучше, чем предыдущее, присваиваем его лягушке.

Проводим аналогичные вычисления для второй группы:

Рассчитаем новое значение функции;

После выполнения итерации получаем координаты и значения функций наших лягушек:

Так же поступаем и со вторым лучшим участком, а затем аналогично и с выбранными участками. После чего среди всех новых точек снова отмечаются лучшие и выбранные, а процесс повторяется заново.

* 1. Программная реализация

Данный алгоритм реализован на языке Python [2], а для визуализации данных использовался скрипт для Matplotlib [3].

Создадим популяцию лягушек:

* Количество лягушек в популяции: 50;
* Количество итераций: ;
* Количество групп: ;

Результат выполнения алгоритма приведен на Рисунке 5.3.1.

Визуализация начального положения популяции лягушек приведена на Рисунке 5.3.2.

Визуализация конечного популяции лягушек приведена на Рисунке 5.3.3.

Визуализация конечного популяции лягушек, в альтернативном варианте приведена на Рисунке 5.3.4.

Листинг решения приведен в Приложении Д.

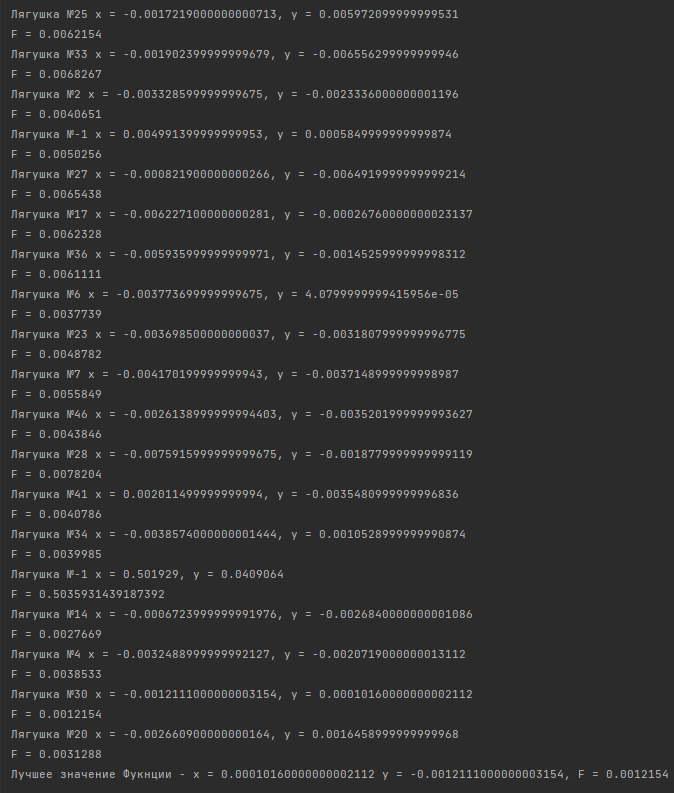


Рисунок 5.3.1 – Результат выполнения алгоритма

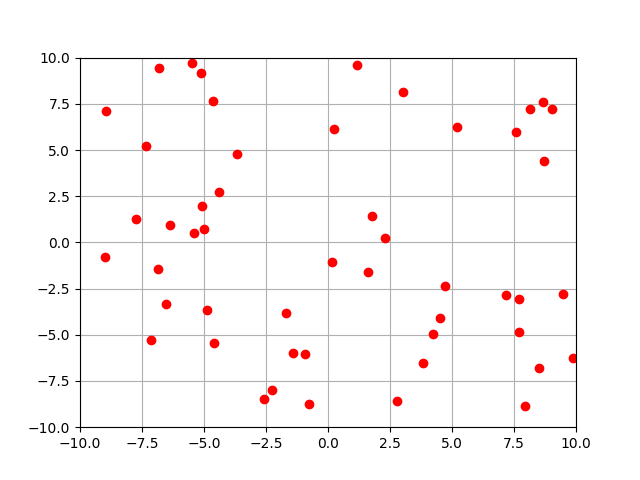


Рисунок 5.3.2 – Визуализация начального положения популяции лягушек

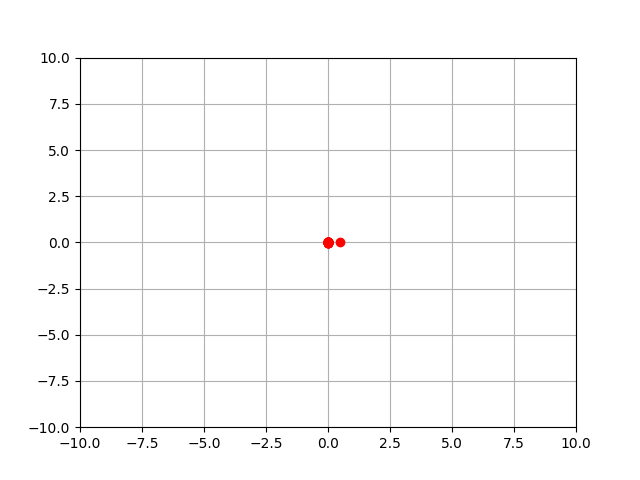


Рисунок 5.3.2 – Визуализация конечного популяции лягушек

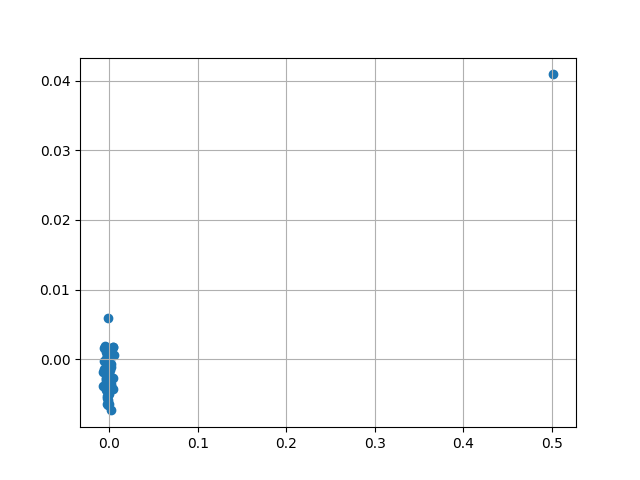


Рисунок 5.3.2 – Визуализация конечного популяции лягушек, в альтернативном варианте

ЗАключение

В ходе выполнения курсовой работы были получены навыки в разработке онтологии в программе Protégé и последующая ее реализация на языке Python. Так же изучены различные алгоритмы оптимизации, такие как, алгоритм отжига, роевой алгоритм, муравьиный алгоритм и тасующий алгоритм прыгающих лягушек. Все изученные алгоритмы реализованы на языке Python и выполнена визуализация на скриптовом языке Matplotlib.

Муравьиный алгоритм, а также алгоритм отжига были использованы для нахождения кротчайшего пути в графе.

Роевой алгоритм, тасующий алгоритм прыгающих лягушек были протестированы на одной функции, в результате чего можно сделать вывод о том, что тасующий алгоритм прыгающих лягушек быстрее и точнее чем роевой алгоритм, роевой – работает достаточно быстро, но с менее точен, бактериальный – имеет высокую точно, но работает намного медленнее роевого и пчелиной колонии.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сорокин А. Б. Лекции и методические материалы — РТУ МИРЭА, 2022.
2. Python documentation — URL: https://www.python.org/ (Дата обращения: 20.12.2022)
3. Matplotlib — Visualization with Python — URL: https://matplotlib.org/ (Дата обращения: 21.12.2022)
4. Информационные технологии, № 7, 2012. Теоретический и прикладной научно-технический журнал. — М.: Новые технологии, 2012. — 80 с.
5. Настройка весовых матриц ЗСУР регулятора с помощью биоинспирированных алгоритмов оптимизации. Вестник РГРТУ. 2016. № 55. — Рязань: РГРТУ. — С. 131–139.
6. Алгоритм роя частиц. Описание и реализации на языках Python и C# URL: https://jenyay.net/Programming/ParticleSwarm (Дата обращения: 15.12.2022)

ПРИЛОЖЕНИЯ

Приложение А – Листинг кода для онтологии

Приложение Б – Листинг кода для алгоритма отжига

Приложение В – Листинг кода для роевого алгоритма

Приложение Г – Листинг кода для муравьиного алгоритма

Приложение Д – Листинг кода для тасующий алгоритм прыгающих лягушек.

Приложение А

Листинг кода для онтологии

Листинг А.1 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':

out()

Листинг А.2 – Файл json со всеми объектами в онтологии

[

{

"value": 7890,

"producer": "MSI",

"model": "PRO",

"chip": "B",

"cpu": "AMD",

"processor\_generation": "AM4"

},

{

"value": 24999,

"producer": "ASUS",

"model": "ROG",

"chip": "Z",

"cpu": "INTEL",

"processor\_generation": "ADLER LAKE"

},

{

"value": 16796,

"producer": "ASUS",

"model": "ROG STRIX",

"chip": "B",

"cpu": "AMD",

"processor\_generation": "AM4"

},

{

"value": 14900,

"producer": "ASUS",

"model": "TUF GAMING",

"chip": "B",

"cpu": "AMD",

"processor\_generation": "AM4"

},

{

"value": 24740,

"producer": "MSI",

"model": "TOMAHAWK",

"chip": "Z",

"cpu": "INTEL",

"processor\_generation": "ADLER LAKE"

},

{

"value": 13200,

"producer": "ASUS",

"model": "PRIME",

"chip": "Z",

"cpu": "INTEL",

"processor\_generation": "ADLER LAKE"

},

Продолжение Листинга А.2

{

"value": 8050,

"producer": "ASUS",

"model": "PRIME",

"chip": "H",

"cpu": "INTEL",

"processor\_generation": "ROCKET LAKE"

},

{

"value": 4630,

"producer": "MSI",

"model": "PRO",

"chip": "H",

"cpu": "INTEL",

"processor\_generation": "ADLER LAKE"

},

{

"value": 4990,

"producer": "MSI",

"model": "PRO",

"chip": "H",

"cpu": "INTEL",

"processor\_generation": "TIGER LAKE"

},

{

"value": 5190,

"producer": "MSI",

"model": "PRO",

"chip": "H",

"cpu": "INTEL",

"processor\_generation": "ROCKET LAKE"

},

{

"value": 52800,

"producer": "MSI",

"model": "MPG",

"chip": "X",

"cpu": "AMD",

"processor\_generation": "AM4"

}

]

Листинг А.3 – Файл json для определение принадлежности к классу

[

{

"value": 7890,

"producer": "MSI",

"model": "PRO",

"chip": "B",

"cpu": "AMD",

"processor\_generation": "AM4"

},

{

"value": 24999,

"producer": "ASUS",

"model": "ROG",

"chip": "Z",

"cpu": "INTEL",

"processor\_generation": "ADLER LAKE"

},

{

"value": 16796,

Продолжение Листинга А.3

"producer": "ASUS",

"model": "ROG STRIX",

"chip": "B",

"cpu": "AMD",

"processor\_generation": "AM4"

},

{

"value": 14900,

"producer": "ASUS",

"model": "TUF GAMING",

"chip": "B",

"cpu": "AMD",

"processor\_generation": "AM4"

},

{

"value": 24740,

"producer": "MSI",

"model": "TOMAHAWK",

"chip": "Z",

"cpu": "INTEL",

"processor\_generation": "ADLER LAKE"

},

{

"value": 13200,

"producer": "ASUS",

"model": "PRIME",

"chip": "Z",

"cpu": "INTEL",

"processor\_generation": "ADLER LAKE"

},

{

"value": 8050,

"producer": "ASUS",

"model": "PRIME",

"chip": "H",

"cpu": "INTEL",

"processor\_generation": "ROCKET LAKE"

},

{

"value": 4630,

"producer": "MSI",

"model": "PRO",

"chip": "H",

"cpu": "INTEL",

"processor\_generation": "ADLER LAKE"

},

{

"value": 4990,

"producer": "MSI",

"model": "PRO",

"chip": "H",

"cpu": "INTEL",

"processor\_generation": "TIGER LAKE"

},

{

"value": 5190,

"producer": "MSI",

"model": "PRO",

"chip": "H",

"cpu": "INTEL",

"processor\_generation": "ROCKET LAKE"

},

Продолжение Листинга А.3

{

"value": 52800,

"producer": "MSI",

"model": "MPG",

"chip": "X",

"cpu": "AMD",

"processor\_generation": "AM4"

}

]

Листинг А.4 – Функция выбора класса, слота и значения из онтологии

def out():

print("| bace\_of\_motherboard | chipset\_of\_motherboard "

"|\n| z\_chipset | h\_chipset | x\_chipset | b\_chipset |\n| intel | amd |")

level = input("Выберете уровень: ")

print("\nСлоты:")

data = read\_json(file\_data)[level][0]

for i in data:

if data[i] != "False":

print(data[i])

slot = input("Выберете слот: ")

print()

if slot == "value":

print(">= | = | <=")

sign = input("Знак - ")

else:

sign = '='

meaning = input("Значение - ")

if level == "chipset\_of\_motherboard" or level == "bace\_of\_motherboard":

serpch\_in\_main\_class(level, slot, sign, meaning)

else:

serpch\_in\_inherited\_class(level, slot, sign, meaning)

Листинг А.5 – Функция выбора числовых значений

def serpch\_in\_main\_class(level, slot, sign, meaning):

database = read\_json(file\_bace)

if sign == '=':

for i in database:

if slot == 'value':

if i[slot] == int(meaning):

where(i, level, slot, meaning)

elif i[slot] == meaning:

where(i, level, slot, meaning)

elif sign == '>=' and slot == 'value':

for i in database:

if i[slot] >= int(meaning):

where(i, level, slot, meaning)

elif sign == '<=' and slot == 'value':

for i in database:

if i[slot] <= int(meaning):

where(i, level, slot, meaning)

Листинг А.6 – Функция выбора строковых значений

def serpch\_in\_inherited\_class(level, slot, sign, meaning):

level = transformations(level)

database = read\_json(file\_bace)

if sign == '=':

for i in database:

if slot == 'value':

if i[level[0]] == level[1] and i[slot] == int(meaning):

where(i, level, slot, meaning)

elif i[level[0]] == level[1] and i[slot] == meaning:

where(i, level, slot, meaning)

elif sign == '>=' and slot == 'value':

for i in database:

if i[level[0]] == level[1] and i[slot] >= int(meaning):

where(i, level, slot, meaning)

elif sign == '<=' and slot == 'value':

for i in database:

if i[level[0]] == level[1] and i[slot] <= int(meaning):

where(i, level, slot, meaning)

Листинг А.7 – Функция вывода принадлежности классу на экран

def where(data, level, slot, meaning):

print()

for i in data:

print(i, ' - ', data[i], end=' | ')

print()

base = []

if data["cpu"] == 'INTEL':

base.append("База плат -> INTEL")

if data["cpu"] == 'AMD':

base.append("База плат -> AMD")

if data["chip"] == 'X':

base.append("Типы плат -> X чипсет")

if data["chip"] == 'B':

base.append("Типы плат -> B чипсет")

if data["chip"] == 'Z':

base.append("Типы плат -> Z чипсет")

if data["chip"] == 'H':

base.append("Типы плат -> H чипсет")

for i in base:

print(i, end=' | ')

print()

print("=" \* 50)

Приложение Б

Листинг кода для алгоритма отжига

Листинг Б.1 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':

count\_of\_vertices = 6

k\_iteration\_max = 150

minimum\_temperature = 25

graph\_construction(k\_iteration\_max, count\_of\_vertices, minimum\_temperature)

Листинг Б.2 – Основная функция для выполнения алгоритма, и хранения данных

def graph\_construction(k\_iteration\_max, count\_of\_vertices, minimum\_temperature):

graph = nx.Graph()

mass\_iteration = [0]

mass\_all\_trail\_weight = []

mass\_all\_trail = []

mass\_temperature = [100]

mass\_changes = []

mass\_iteration\_count = [0]

table\_graph = len\_weight\_write\_in\_graph(graph, count\_of\_vertices)

trail\_calculation(count\_of\_vertices, mass\_all\_trail)

calculation\_length\_trail(mass\_all\_trail[-1], table\_graph, mass\_all\_trail\_weight)

pos = nx.spring\_layout(graph)

nx.draw(graph, pos, with\_labels=1)

edge\_labels = nx.get\_edge\_attributes(graph, 'weight')

nx.draw\_networkx\_edge\_labels(graph, pos, edge\_labels=edge\_labels)

plt.savefig('../static/1.png')

while mass\_temperature[-1] > minimum\_temperature and k\_iteration\_max + 1 > len(mass\_iteration):

mass\_all\_trail.append(random\_change\_road(mass\_all\_trail, count\_of\_vertices))

length\_trail\_new = calculation\_length\_trail\_new(mass\_all\_trail[-1], table\_graph)

annealing(mass\_all\_trail\_weight, length\_trail\_new, mass\_iteration,

mass\_temperature, mass\_iteration\_count, mass\_all\_trail)

if mass\_temperature[-1] <= minimum\_temperature:

print("Достижение минимальной температуры!!!")

else:

print("Выполнились все циклы!!!")

print(f'Всего циклов выполнелось - {mass\_iteration[-1]}')

print(

f'Финальный маршрут ', end='')

for i in mass\_all\_trail[-1]:

print(i, end=' -> ')

print(f'\nS = {mass\_all\_trail\_weight[-1]}, ответ найден на - {mass\_iteration\_count[-1]} шаге')

graph\_final\_data(count\_of\_vertices, mass\_all\_trail[-1], table\_graph)

create\_plots(mass\_iteration\_count, mass\_all\_trail\_weight)

print(f'{mass\_iteration\_count}\n{mass\_all\_trail\_weight}\n{mass\_all\_trail}')

Листинг Б.3 – функция для создания графа

def len\_weight\_write\_in\_graph(graph, count\_of\_vertices):

len\_weight = random\_len\_calculations(count\_of\_vertices)

for i in range(len(len\_weight)):

graph.add\_edge(len\_weight[i][0], len\_weight[i][1], weight=len\_weight[i][2])

return len\_weight

Листинг Б.4 – Функция, создающая граф и задающая его размерность

def random\_len\_calculations(count\_of\_vertices):

len\_weight = []

for i in range(1, count\_of\_vertices + 1):

for j in range(i + 1, count\_of\_vertices + 1):

len\_weight.append([i, j, random.randint(50, 101)])

return len\_weight

Листинг Б.5 – Метод создающий начальный маршрут

def trail\_calculation(count\_of\_vertices, mass\_all\_trail):

trail = []

while len(trail) != count\_of\_vertices:

cache = random.randint(1, count\_of\_vertices)

if cache not in trail:

trail.append(cache)

trail.append(trail[0])

mass\_all\_trail.append(trail)

Листинг Б.6 – Функция, возвращающая длину заданного пути

def calculation\_length\_trail\_new(last\_trail, table\_graph):

length\_original\_route = 0

for i in range(len(last\_trail) - 1):

for j in range(len(table\_graph)):

if last\_trail[i] == table\_graph[j][0] and last\_trail[i + 1] == table\_graph[j][1] or \

last\_trail[i + 1] == table\_graph[j][0] and last\_trail[i] == table\_graph[j][1]:

length\_original\_route += table\_graph[j][2]

return length\_original\_route

Листинг Б.7 – Метода рандомного изменения пути

def random\_change\_road(new\_mass\_all\_trail, count\_of\_vertices):

new\_trail\_new = cam(new\_mass\_all\_trail[-1])

close\_peaks = False

while True:

first\_number = round(random.randrange(1, count\_of\_vertices))

second\_number = round(random.randrange(1, count\_of\_vertices))

if first\_number != second\_number:

for i in range(1, len(new\_trail\_new) - 1):

if new\_trail\_new[i] == first\_number and new\_trail\_new[i + 1] == second\_number \

or new\_trail\_new[i + 1] == first\_number and new\_trail\_new[i] == second\_number:

close\_peaks = True

if close\_peaks is False:

for j in range(len(new\_trail\_new)):

if new\_trail\_new[j] == second\_number:

new\_trail\_new[j] = first\_number

elif new\_trail\_new[j] == first\_number:

new\_trail\_new[j] = second\_number

if new\_trail\_new not in new\_mass\_all\_trail:

return new\_trail\_new

close\_peaks = False

Листинг Б.8 – Метода для определения кратчайшего пути и изменения температуры

def annealing(mass\_all\_trail\_weight, new\_trail, mass\_iteration, mass\_temperature, mass\_iteration\_count, mass\_all\_trail):

mass\_iteration.append(mass\_iteration[-1] + 1)

print(f'Старая температура: {mass\_temperature[-1]}')

print(f'Старый путь S - {mass\_all\_trail\_weight[-1]}')

print(f'Новый путь S - {new\_trail}')

delta = new\_trail - mass\_all\_trail\_weight[-1]

print(f'delta S = {new\_trail}', '-', f'{mass\_all\_trail\_weight[-1]} = {delta}')

if delta < 0:

print(f'{delta} < 0')

print('Расчёт температуры:')

print(f'{mass\_temperature[0]} / {math.log(1 + mass\_iteration[-1])}')

new\_temp = round(mass\_temperature[0] / math.log(1 + mass\_iteration[-1]), 3)

mass\_temperature.append(new\_temp)

print(f'Новая температура - {mass\_temperature[-1]}')

mass\_all\_trail\_weight.append(new\_trail)

mass\_iteration\_count.append(mass\_iteration[-1])

print('Новый путь: ')

for i in mass\_all\_trail[-1]:

print(i, end=' -> ')

print()

elif delta > 0:

print(f'{delta} > 0')

# p = 100 \* math.e \*\* (-new\_trail/mass\_temperature[-1])

p = (mass\_temperature[0] \* math.e) \*\* (- delta / mass\_temperature[-1])

random\_p = round((random.uniform(1, mass\_temperature[-1])), 5)

print(random\_p)

print(f'{p} =?= {random\_p}')

if p >= random\_p:

print(f'{p} >= {random\_p}')

mass\_all\_trail\_weight.append(new\_trail)

print(mass\_all\_trail\_weight)

mass\_iteration\_count.append(mass\_iteration[-1])

print('Новый путь: ')

for i in mass\_all\_trail[-1]:

print(i, end=' -> ')

print()

else:

print(f'{p} < {random\_p}')

print('Не принимаем маршрут!')

print('Старый путь: ')

for i in mass\_all\_trail[mass\_iteration\_count[-1] - 1]:

print(i, end=' -> ')

print()

# mass\_all\_trail\_weight.append(mass\_all\_trail\_weight[-1])

print('=' \* 100)

Листинг Б.9 – Метод для создания и сохранение получившегося графа

def graph\_final\_data(count\_of\_vertices, mass\_all\_trail, table\_graph):

graph\_final = nx.Graph()

fig2, axs2 = plt.subplots(1)

for i in range(len(mass\_all\_trail) - 1):

for j in range(len(table\_graph)):

if mass\_all\_trail[i] == table\_graph[j][0] and mass\_all\_trail[i + 1] == table\_graph[j][1] or \

mass\_all\_trail[i + 1] == table\_graph[j][0] and mass\_all\_trail[i] == table\_graph[j][1]:

graph\_final.add\_edge(table\_graph[j][0], table\_graph[j][1], weight=table\_graph[j][2])

fig2 = nx.spring\_layout(graph\_final)

nx.draw(graph\_final, fig2, with\_labels=1)

edge\_labelss = nx.get\_edge\_attributes(graph\_final, 'weight')

nx.draw\_networkx\_edge\_labels(graph\_final, fig2, edge\_labels=edge\_labelss)

plt.savefig('../static/2.png')

Листинг Б.10 – Метод для создания и сохранение графа показывающего уменьшение длины пути по итерацияма

def create\_plots(mass\_iteration\_count, mass\_all\_trail\_weight):

fig3, axs3 = plt.subplots(1)

fig3 = pd.DataFrame({'voltage': mass\_iteration\_count, 'val': mass\_all\_trail\_weight})

# fig3 = plt.plot(mass\_iteration\_count, mass\_all\_trail\_weight)

axs3 = fig3.plot.bar(x='voltage', y='val', rot=0)

axs3.set\_xlabel('Итерации')

axs3.set\_ylabel('Длинна пути')

axs3.set\_title('График показывающий уменьшении пути по итерациям')

plt.savefig('../static/3.png')

Приложение В

Листинг кода для роевого алгоритма

Листинг В.1 – Функция main

from directory.programm import start

if \_\_name\_\_ == '\_\_main\_\_':

iteration = 1000

quantity = 100

start(iteration, quantity)

Листинг В.2 – Основной метод для определения переменных и вызова функций дальнейшей реализации

def start(iteration, quantity):

mass\_quantity = []

for i in range(1, quantity + 1):

mass\_quantity.append(

[i, round(random.uniform(-10.0, 10.0), 8), round(random.uniform(-10.0, 10.0), 8), 0, 0, 0, 0])

graph(mass\_quantity)

mass\_velosity = []

# 3 - значение функции на итерации

# 4 - лучшее значение функций

# 5 - лучшая x

# 6 - лучшая y

# 7 - с крость x

# 8 - с крость y

best\_value\_in\_all = [0, 0, 1000000]

for i in range(iteration):

print(mass\_quantity)

local\_position(mass\_quantity)

global\_position(mass\_quantity, best\_value\_in\_all)

print(mass\_quantity)

# print(mass\_best\_value\_in\_all)

velosity(mass\_quantity, best\_value\_in\_all)

print(mass\_quantity)

for i in range(len(mass\_quantity)):

print(f'Частица №{mass\_quantity[i][0]}\n'

f'Функция ({mass\_quantity[i][1]}, {mass\_quantity[i][2]})'

f' = {mass\_quantity[i][3]}')

graph\_final(mass\_quantity)

graph\_final\_v2(mass\_quantity)

Листинг В.3 – Метод создания и сохранения начального графика расположения роя

def graph(mass\_quantity):

plt.axis([-10, 10, -10, 10])

x = []

y = []

for i in range(len(mass\_quantity)):

x.append(mass\_quantity[i][1])

y.append(mass\_quantity[i][2])

plt.plot(x, y, 'ro')

plt.grid()

plt.savefig('static/start.png')

plt.close()

Листинг В.4 – Метод расчёт функции для элемента роя

def local\_position(mass\_quantity):

for i in range(len(mass\_quantity)):

new\_best\_value\_function = round((math.sqrt((mass\_quantity[i][1] \*\* 2 + mass\_quantity[1][2] \*\* 2))), 8)

mass\_quantity[i][3] = new\_best\_value\_function

if mass\_quantity[i][3] < mass\_quantity[i][4]:

mass\_quantity[i][4] = mass\_quantity[i][3]

mass\_quantity[i][5] = mass\_quantity[i][1]

mass\_quantity[i][6] = mass\_quantity[i][2]

Листинг В.5 – Метод сравнения локальной функции с глобальным минимумом

def global\_position(mass\_quantity, best\_value\_function):

for i in range(len(mass\_quantity)):

if mass\_quantity[i][4] < best\_value\_function[2]:

best\_value\_function = [mass\_quantity[i][5], mass\_quantity[i][6], mass\_quantity[i][4]]

Листинг В.6 – Метод изменяет позиция элемента в рое

def velosity(mass\_quantity, best\_value\_in\_all):

c\_1 = 0.01

c\_2 = 0.01

r\_1 = round(random.uniform(0.0, 1.0), 8)

r\_2 = round(random.uniform(0.0, 1.0), 8)

for i in range(len(mass\_quantity)):

# print()

new\_v\_x = round(mass\_quantity[i][-2] + c\_1 \* r\_1 \* \

(mass\_quantity[i][5] - mass\_quantity[i][1]) + \

c\_2 \* r\_2 \* (best\_value\_in\_all[0] - mass\_quantity[i][1]), 8)

new\_v\_y = round(mass\_quantity[i][-1] + c\_1 \* r\_1 \* \

(mass\_quantity[i][6] - mass\_quantity[i][2]) + \

c\_2 \* r\_2 \* (best\_value\_in\_all[1] - mass\_quantity[i][2]), 8)

print(f'{mass\_quantity[i][-2]} + {c\_1} \* {r\_1} \*'

f'({mass\_quantity[i][5]} - {mass\_quantity[i][1]}) + '

f'{c\_2} \* {r\_2} \* ({best\_value\_in\_all[0]} - {mass\_quantity[i][1]})')

print(f'{mass\_quantity[i][-1]} + {c\_1} \* {r\_1} \*'

f'({mass\_quantity[i][6]} - {mass\_quantity[i][2]}) + '

f'{c\_2} \* {r\_2} \* ({best\_value\_in\_all[0]} - {mass\_quantity[i][2]})')

print('Частица - ', i)

print(f'Новая позиция x = {mass\_quantity[i][1]} + {new\_v\_x} = {round(new\_v\_x + mass\_quantity[i][1], 8)}')

print(f'Новая позиция y = {mass\_quantity[i][2]} + {new\_v\_y} = {round(new\_v\_y + mass\_quantity[i][2], 8)}')

mass\_quantity[i][1] = round(new\_v\_x + mass\_quantity[i][1], 8)

mass\_quantity[i][2] = round(new\_v\_y + mass\_quantity[i][2], 8)

print()

Листинг В.7 – Метод создание и сохранения графа, финального расположения частиц в рое

def graph\_final(mass\_quantity):

plt.axis([-10, 10, -10, 10])

x = []

y = []

for i in range(len(mass\_quantity)):

x.append(mass\_quantity[i][1])

y.append(mass\_quantity[i][2])

plt.plot(x, y, 'ro')

plt.grid()

plt.savefig('static/final.png')

plt.close()

Листинг В.8 – Метод создание и сохранения альтернативного графа, финального расположения частиц в рое

def graph\_final\_v2(mass\_quantity):

fig = plt.figure()

x = []

y = []

ax = fig.add\_subplot()

for i in range(len(mass\_quantity)):

x.append(mass\_quantity[i][1])

y.append(mass\_quantity[i][2])

ax.scatter(x, y)

ax.grid()

plt.savefig('static/final\_v2.png')

plt.close()

Листинг В.9 – Метод создание и сохранения графа, начального расположения частиц в рое

def graph(mass\_quantity):

plt.axis([-10, 10, -10, 10])

x = []

y = []

for i in range(len(mass\_quantity)):

x.append(mass\_quantity[i][1])

y.append(mass\_quantity[i][2])

plt.plot(x, y, 'ro')

plt.grid()

plt.savefig('static/start.png')

plt.close()

Листинг В.9 – Метод вывода

def print\_first\_play(best\_value\_function, mass\_quantity, mass\_value\_function):

for i in range(len(mass\_quantity)):

print(f'F(x{mass\_quantity[i][0]} : {mass\_quantity[i][1]}, '

f'y{mass\_quantity[i][0]} : {mass\_quantity[i][2]}) = {mass\_value\_function[i]}')

for i in range(len(mass\_value\_function)):

if mass\_value\_function[i] == best\_value\_function:

print(f'Лучшее значение функции - F(x{i + 1}, y{i + 1})'

f' = {best\_value\_function}')

Приложение Г

Листинг кода для муравьиного алгоритма

Листинг Г.1 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':

alfa = 1 # 1

beta = 0.05 # 0.05

count\_iteration = 500 # 500

count\_ant = 6

count\_city = 6

evaporation\_of\_hormones = 0.255 # 0.30

ants = Ant(count\_ant, alfa, beta, count\_iteration, count\_city, evaporation\_of\_hormones)

ants.iteration\_calculation()

Листинг Г.2 – Класс муравья

class Ant:

def \_\_init\_\_(self, count\_ant, alfa, beta, count\_iteration, count\_city, evaporation\_of\_hormones):

self.count\_ant = count\_ant

self.alfa = alfa

self.beta = beta

self.count\_iteration = count\_iteration

self.count\_city = count\_city

self.evaporation\_of\_hormones = evaporation\_of\_hormones

Листинг Г.3 – Метод для просчёта нужного количества операций

def iteration\_calculation(self):

self.random\_len\_calculations() # задаём пути и феромоны

self.graph()

for count in range(self.count\_iteration):

self.mass\_trail\_ant\_go = []

for i in range(self.count\_ant):

self.mass\_ant\_path\_in\_ant = []

self.generate\_new\_city\_start()

for j in range(self.count\_city - 1):

self.calculation\_next\_city\_togo()

self.mass\_ant\_path\_in\_ant.append(self.mass\_ant\_path\_in\_ant[0])

self.mass\_trail\_ant\_go.append(self.mass\_ant\_path\_in\_ant)

self.calculation\_pheromone()

for i in range(len(self.len\_weight)):

print(f'Путь от {self.len\_weight[i][0]} до {self.len\_weight[i][1]} = {self.len\_weight[i][2]}\n'

f'Феромонов - {self.len\_weight[i][-1]}')

print()

self.graph\_final()

self.graph\_final\_w()

Листинг Г.4 – Метод для создания графа

def graph(self):  
 graph def random\_len\_calculations(self):  
 self.len\_weight = []  
 for i in range(1, self.count\_city + 1):  
 for j in range(i + 1, self.count\_city + 1):  
 self.len\_weight.append([i, j, random.uniform(0, 1), 0.2])

Листинг Г.5 – Метод для создания визуализации начального графа

def graph(self):  
 graph = nx.Graph()  
 for i in range(len(self.len\_weight)):  
 graph.add\_edge(self.len\_weight[i][0], self.len\_weight[i][1], weight=self.len\_weight[i][2])  
 pos = nx.spring\_layout(graph)  
 nx.draw(graph, pos, with\_labels=1)  
 edge\_labels = nx.get\_edge\_attributes(graph, 'weight')  
 nx.draw\_networkx\_edge\_labels(graph, pos, edge\_labels=edge\_labels)  
 plt.savefig('static/start.png')  
 plt.close()

Листинг Г.6 – Метод получения 1 города в пути муравья

def generate\_new\_city\_start(self):

cycle = True

while cycle:

new\_city\_start = random.randint(1, self.count\_city)

if len(self.mass\_trail\_ant\_go) == 0:

self.mass\_ant\_path\_in\_ant.append(new\_city\_start)

break

else:

mass\_first\_cyti = []

for i in range(len(self.mass\_trail\_ant\_go)):

mass\_first\_cyti.append(self.mass\_trail\_ant\_go[i][0])

if new\_city\_start not in mass\_first\_cyti:

self.mass\_ant\_path\_in\_ant.append(new\_city\_start)

cycle = False

break

Листинг Г.7 – Метод вычисления пути муравья в зависимости от длинны и феромонов

def calculation\_next\_city\_togo(self):

mass\_all\_possible\_city = []

for i in range(1, self.count\_city + 1):

if i not in self.mass\_ant\_path\_in\_ant:

mass\_all\_possible\_city.append([self.mass\_ant\_path\_in\_ant[-1], i])

# print('Массив пар составлен:', mass\_all\_possible\_city)

summ\_all\_trail = 0

for weight in range(len(self.len\_weight)): # расёт суммы свех дорог

for j in range(len(mass\_all\_possible\_city)):

if mass\_all\_possible\_city[j][0] == self.len\_weight[weight][0] and \

mass\_all\_possible\_city[j][1] == self.len\_weight[weight][1] and \

mass\_all\_possible\_city[j][1] not in self.mass\_ant\_path\_in\_ant:

mass\_all\_possible\_city[j].append(self.len\_weight[weight][-1] \*\* self.alfa \*

(self.len\_weight[weight][-2]) \*\* self.beta)

summ\_all\_trail += self.len\_weight[weight][-1] \*\* self.alfa \* \

(self.len\_weight[weight][-2]) \*\* self.beta

elif mass\_all\_possible\_city[j][1] == self.len\_weight[weight][0] and \

mass\_all\_possible\_city[j][0] == self.len\_weight[weight][1] and \

mass\_all\_possible\_city[j][1] not in self.mass\_ant\_path\_in\_ant:

mass\_all\_possible\_city[j].append(self.len\_weight[weight][-1] \*\* self.alfa \*

(self.len\_weight[weight][-2]) \*\* self.beta)

summ\_all\_trail += self.len\_weight[weight][-1] \*\* self.alfa \* \

(self.len\_weight[weight][-2]) \*\* self.beta

Продолжение Листинга Г.7

summ\_coefecents = 0

for i in range(len(mass\_all\_possible\_city)):

mass\_all\_possible\_city[i][-1] = round(mass\_all\_possible\_city[i][-1] / summ\_all\_trail, 5)

summ\_coefecents += mass\_all\_possible\_city[i][-1]

mass\_coefecents = []

for i in range(len(mass\_all\_possible\_city) + 1): # приведения к обево пути к 1

mass\_coefecents.append(0)

for j in range(i):

mass\_coefecents[-1] += mass\_all\_possible\_city[j][-1]

x = random.uniform(0, 1)

for i in range(len(mass\_coefecents)):

if mass\_coefecents[i] <= x <= mass\_coefecents[i + 1]:

self.mass\_ant\_path\_in\_ant.append(mass\_all\_possible\_city[i][1])

break

Листинг Г.8 – Метод, реализующий обновление феромонов в графе

def calculation\_pheromone(self):

for i in range(len(self.len\_weight)):

self.len\_weight[i][-1] \*= self.evaporation\_of\_hormones

for i in range(len(self.mass\_trail\_ant\_go)):

path = 0

for j in range(len(self.mass\_trail\_ant\_go[i]) - 1):

for k in range(len(self.len\_weight)):

if self.mass\_trail\_ant\_go[i][j] == self.len\_weight[k][0] and \

self.mass\_trail\_ant\_go[i][j + 1] == self.len\_weight[k][1]:

path += self.len\_weight[k][-2]

elif self.mass\_trail\_ant\_go[i][j + 1] == self.len\_weight[k][0] and \

self.mass\_trail\_ant\_go[i][j] == self.len\_weight[k][1]:

path += self.len\_weight[k][-2]

for j in range(len(self.mass\_trail\_ant\_go[i])-1):

for k in range(len(self.len\_weight)):

if self.mass\_trail\_ant\_go[i][j] == self.len\_weight[k][0] and \

self.mass\_trail\_ant\_go[i][j + 1] == self.len\_weight[k][1]:

self.len\_weight[k][-1] += self.count\_city / path

elif self.mass\_trail\_ant\_go[i][j + 1] == self.len\_weight[k][0] and \

self.mass\_trail\_ant\_go[i][j] == self.len\_weight[k][1]:

self.len\_weight[k][-1] += self.count\_city / path

Листинг Г.9 – Метод создания визуализации финального графа

def graph\_final(self):

graph\_final = nx.Graph()

fig2, axs2 = plt.subplots(1)

max\_firamone = self.len\_weight

new\_max\_fir = []

while len(new\_max\_fir) != self.count\_city:

minimum = [-1]

iteration = -1

for i in range(len(max\_firamone)):

if max\_firamone[i][-1] > minimum[-1]:

minimum = max\_firamone[i]

iteration = i

new\_max\_fir.append(minimum)

max\_firamone.pop(iteration)

for i in range(len(new\_max\_fir)):

graph\_final.add\_edge(new\_max\_fir[i][0], new\_max\_fir[i][1], color='r', weight=new\_max\_fir[i][-1])

Продолжение Листинга Г.9

min\_road = 0

for i in range(len(new\_max\_fir)):

min\_road += new\_max\_fir[i][-2]

self.new\_max\_fir = new\_max\_fir

print('Путь муравьев =', min\_road)

fig2 = nx.spring\_layout(graph\_final)

nx.draw(graph\_final, fig2, with\_labels=1)

edge\_labelss = nx.get\_edge\_attributes(graph\_final, 'weight')

nx.draw\_networkx\_edge\_labels(graph\_final, fig2, edge\_labels=edge\_labelss)

plt.savefig('static/final.png')

plt.close()

Листинг Г.10 – Метод создания визуализации финального альтернативного графа

def graph\_final\_w(self):

graph\_w = nx.Graph()

for i in range(len(self.new\_max\_fir)):

graph\_w.add\_edge(self.new\_max\_fir[i][0], self.new\_max\_fir[i][1], weight=self.new\_max\_fir[i][2])

fig3 = nx.spring\_layout(graph\_w)

nx.draw(graph\_w, fig3, with\_labels=1)

edge\_labelss = nx.get\_edge\_attributes(graph\_w, 'weight')

nx.draw\_networkx\_edge\_labels(graph\_w, fig3, edge\_labels=edge\_labelss)

plt.savefig('static/final\_w.png')

Приложение Д

Листинг кода для тасующий алгоритм прыгающих лягушек

Листинг Д.1 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':

f1 = Frog(50, 1, 25, 50) # count\_frog, iteration, count\_memplex, global\_iteration

population = f1.start()

graph\_final(population)

graph\_final\_v2(population)

Листинг Д.2 – Класс Frog

class Frog:

def \_\_init\_\_(self, count\_frog, iteration, count\_memplex, global\_iteration):

self.count\_frog = count\_frog

self.iteration = iteration

self.count\_memplex = count\_memplex

self.global\_iteration = global\_iteration

Листинг Д.3 – Метод, простирывающий все итерации

def start(self):

self.population = []

self.mass\_best\_value\_frog = []

for i in range(self.count\_frog):

x = round(random.uniform(-10, 10), 7)

y = round(random.uniform(-10, 10), 7)

self.population.append([i + 1, x, y, 0, 0, 0]) # number\_frog x y new\_x, new\_y fitness

self.calck\_fitnes\_funck()

self.graph()

for iteration in range(self.global\_iteration):

self.calck\_best\_frog\_f()

self.division\_into\_memeplexes()

self.new\_frog\_before\_iteration = self.improving\_position\_object\_memeplex()

self.population.clear()

self.population\_unification()

print('-------')

print(self.population)

print('-------')

print(self.mass\_best\_value\_frog)

for i in range(len(self.population)):

print(f'Лягушка №{self.population[i][0]} x = {self.population[i][1]}, y = {self.population[i][2]}\n'

f'F = {self.population[i][-1]}')

print(f'Лучшее значение Фукнции - x = {self.mass\_best\_value\_frog[-1][-1]} y = {self.mass\_best\_value\_frog[-1][-2]},'

f' F = {self.mass\_best\_value\_frog[-1][-3]}')

return self.population

Листинг Д.4 – Метод расчета локальных функций для каждой лагушки

def calck\_fitnes\_funck(self):

for i in range(len(self.population)):

self.population[i][-1] = round(math.sqrt(self.population[i][1] \*\* 2 + self.population[i][2] \*\* 2), 7)

Листинг Д.5 – Метод сортировка лягушек по целевой функции и нахождения глобального минимума

def calck\_best\_frog\_f(self):

self.best\_frog\_sorted\_population = []

while len(self.best\_frog\_sorted\_population) != self.count\_frog:

maximum = 0, 0

for i in range(len(self.population)):

if self.population[i][-1] > maximum[0]:

maximum = self.population[i][-1], i

self.best\_frog\_sorted\_population.append(self.population[maximum[1]])

self.population.pop(maximum[1])

print('Отобраный по масимуму функции лягушки', self.best\_frog\_sorted\_population)

for i in range(len(self.best\_frog\_sorted\_population)):

print(f'({round(self.best\_frog\_sorted\_population[i][1], 2)}, {round(self.best\_frog\_sorted\_population[i][1], 2)}) ='

f' {round(self.best\_frog\_sorted\_population[i][-1], 2)}')

self.best\_population\_value = [self.best\_frog\_sorted\_population[-1][-1],

self.best\_frog\_sorted\_population[-1][1], self.best\_frog\_sorted\_population[-1][2]]

print('Лучшее значение популяции', self.best\_population\_value[0], 'x -',

self.best\_population\_value[1], 'y -', self.best\_population\_value[2])

self.mass\_best\_value\_frog.append(self.best\_population\_value)

Листинг Д.6 – Метод разделение лягушек по группам

def division\_into\_memeplexes(self):

self.new\_memeplexes = []

for i in range(self.count\_memplex):

self.new\_memeplexes.append([])

self.count\_frogs\_into\_memeplexes = self.count\_frog // self.count\_memplex

print(self.count\_frogs\_into\_memeplexes)

for c in range(self.count\_frogs\_into\_memeplexes):

for i in range(len(self.new\_memeplexes)):

self.new\_memeplexes[i].append(self.best\_frog\_sorted\_population[0])

self.best\_frog\_sorted\_population.pop(0)

print(self.new\_memeplexes)

for i in range(len(self.new\_memeplexes)):

for j in range(len(self.new\_memeplexes[i])):

print(f'({round(self.new\_memeplexes[i][j][1], 2)}, {round(self.new\_memeplexes[i][j][1], 2)}) ='

f' {round(self.new\_memeplexes[i][j][-1], 2)}')

while

Листинг Д.7 – В группе выполняем процедуру меметрической эволюции

def improving\_position\_object\_memeplex(self):

for i in range(len(self.new\_memeplexes)):

new\_x = round(random.uniform(0, 1) \* (self.new\_memeplexes[i][-1][1] - self.new\_memeplexes[i][0][1]), 7)

new\_y = round(random.uniform(0, 1) \* (self.new\_memeplexes[i][-1][2] - self.new\_memeplexes[i][0][2]), 7)

self.new\_memeplexes[i][0][-3] = self.new\_memeplexes[i][0][1] + new\_x

self.new\_memeplexes[i][0][-2] = self.new\_memeplexes[i][0][2] + new\_y

print('f')

print(f'{random.uniform(0, 1)} \* ({self.new\_memeplexes[i][-1][1]} - {self.new\_memeplexes[i][0][1]})')

print(f'{random.uniform(0, 1)} \* ({self.new\_memeplexes[i][-1][2]} - {self.new\_memeplexes[i][0][2]})')

print(f'{self.new\_memeplexes[i][0][-3]} = {self.new\_memeplexes[i][0][1]} + {new\_x}')

Продолжение Листинга Д.7

print(f'{self.new\_memeplexes[i][0][-2]} = {self.new\_memeplexes[i][0][2]} + {new\_y}')

check\_value\_frog = self.new\_memeplexes[i][0]

new\_value\_f\_f = self.calck\_new\_fuck\_value(check\_value\_frog)

print(new\_value\_f\_f)

print('f')

print('Старое значение функции для лягушки', self.new\_memeplexes[i][0][-1])

print('Новое значение для лягушки после перемещения в своей группе', new\_value\_f\_f)

if new\_value\_f\_f > self.new\_memeplexes[i][0][-1]:

self.new\_memeplexes[i][0] = self.improving\_position\_object\_global(check\_value\_frog)

print()

print(f'({round(self.new\_memeplexes[i][0][1], 2)}, {round(self.new\_memeplexes[i][0][1], 2)}) ='

f' {round(self.new\_memeplexes[i][0][-1], 2)}')

print()

else:

check\_value\_frog[-1] = new\_value\_f\_f

check\_value\_frog[1] = self.new\_memeplexes[i][0][-3]

check\_value\_frog[2] = self.new\_memeplexes[i][0][-2]

self.new\_memeplexes[i][0] = check\_value\_frog

print()

print(f'({round(self.new\_memeplexes[i][0][1], 2)}, {round(self.new\_memeplexes[i][0][1], 2)}) ='

f' {round(self.new\_memeplexes[i][0][-1], 2)}')

print()

print(self.new\_memeplexes[i][0])

return self.new\_memeplexes

Листинг Д.8 – Метод улучшения положения лягушки с ориентиром на лучшую лягушку в популяции

def improving\_position\_object\_global(self, check\_value\_frog):

print(check\_value\_frog)

new\_x = round(random.uniform(0, 1) \* (self.best\_population\_value[1] - check\_value\_frog[1]), 7)

print(f'{random.uniform(0, 1)} \* ({self.best\_population\_value[1]} - {check\_value\_frog[1]})')

new\_y = round(random.uniform(0, 1) \* (self.best\_population\_value[2] - check\_value\_frog[2]), 7)

print(f'{random.uniform(0, 1)} \* ({self.best\_population\_value[2]} - {check\_value\_frog[2]})')

check\_value\_frog[-3] += check\_value\_frog[1] + new\_x

check\_value\_frog[-2] += check\_value\_frog[2] + new\_y

print(f'{check\_value\_frog[-3]} = {check\_value\_frog[1]} + {new\_x}')

print(f'{check\_value\_frog[-2]} = {check\_value\_frog[2]} + {new\_y}')

new\_value\_f\_f = self.calck\_new\_fuck\_value(check\_value\_frog)

print(new\_value\_f\_f)

print('Старое значение функции для лягушки', check\_value\_frog[-1])

print('Новое значение для лягушки после перемещения в популяции', new\_value\_f\_f)

if new\_value\_f\_f > check\_value\_frog[-1]:

check\_value\_frog = self.random\_value\_frog()

else:

check\_value\_frog[-1] = new\_value\_f\_f

check\_value\_frog[-1] = new\_value\_f\_f

check\_value\_frog[1] = check\_value\_frog[-3]

check\_value\_frog[2] = check\_value\_frog[-2]

return check\_value\_frog

Листинг Д.9 – Метод пересоздания лягушки

def random\_value\_frog(self):

x = round(random.uniform(-1, 1), 7)

y = round(random.uniform(-1, 1), 7)

f = math.sqrt(x \*\* 2 + y \*\* 2)

return [-1, x, y, 0, 0, f]

Листинг Д.10 – Метод объединения лягушек из гурупп в популяцию

def population\_unification(self):

for i in range(self.count\_frogs\_into\_memeplexes):

mass\_indexes = []

while len(mass\_indexes) != self.count\_memplex:

random\_index = random.randint(0, self.count\_memplex - 1)

if random\_index not in mass\_indexes:

mass\_indexes.append(random\_index)

for i in mass\_indexes:

j = random.randint(0, len(self.new\_frog\_before\_iteration[i]) - 1)

self.population.append(self.new\_frog\_before\_iteration[i][j])

self.new\_frog\_before\_iteration[i].pop(j

Листинг Д.11 – Метод создания визуализации начального графа

def graph(self):

plt.axis([-10, 10, -10, 10])

x = []

y = []

for i in range(len(self.population)):

x.append(self.population[i][1])

y.append(self.population[i][2])

plt.plot(x, y, 'ro')

plt.grid()

plt.savefig('static/start.png')

plt.close()

Листинг Д.12 – Метод создания визуализации конечного графа

def graph\_final(mass\_quantity):

plt.axis([-10, 10, -10, 10])

x = []

y = []

for i in range(len(mass\_quantity)):

x.append(mass\_quantity[i][1])

y.append(mass\_quantity[i][2])

plt.plot(x, y, 'ro')

plt.grid()

plt.savefig('static/final.png')

plt.close()

Листинг Д.13 – Метод создания альтернативного визуализации конечного графа

def graph\_final\_v2(mass\_quantity):

fig = plt.figure()

x = []

y = []

ax = fig.add\_subplot()

for i in range(len(mass\_quantity)):

x.append(mass\_quantity[i][1])

y.append(mass\_quantity[i][2])

ax.scatter(x, y)

ax.grid()

plt.savefig('static/final\_v2.png')

plt.close()