МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МОЭВМ

ОТЧЕТ

по учебной практической работе

Тема: Генетические алгоритмы

Самойлов З. А.
Кирильцев Д. А.
Березовская В. В.
Студенты гр. 0381 Соколов Д. В.

Преподаватель Жангиров Т. Р.

Санкт-Петербург

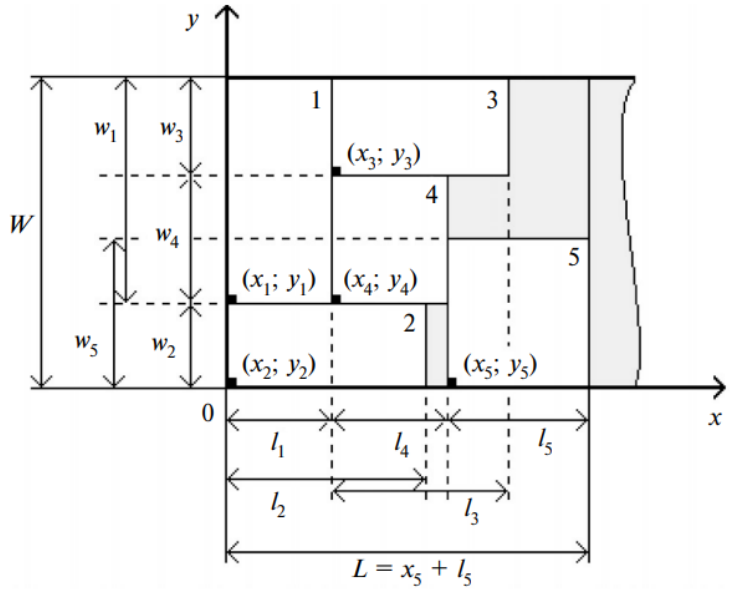
2022

Задание:

Задача раскроя

Дана полубесконечная лента ткани фиксированный ширины, из нее необходимо вырезать прямоугольные участки ткани заданных размеров. Необходимо разместить прямоугольники таким образом, чтобы минимизировать длину используемой ленты и количество отходов.

Графическая интерпретация:



Входные данные:

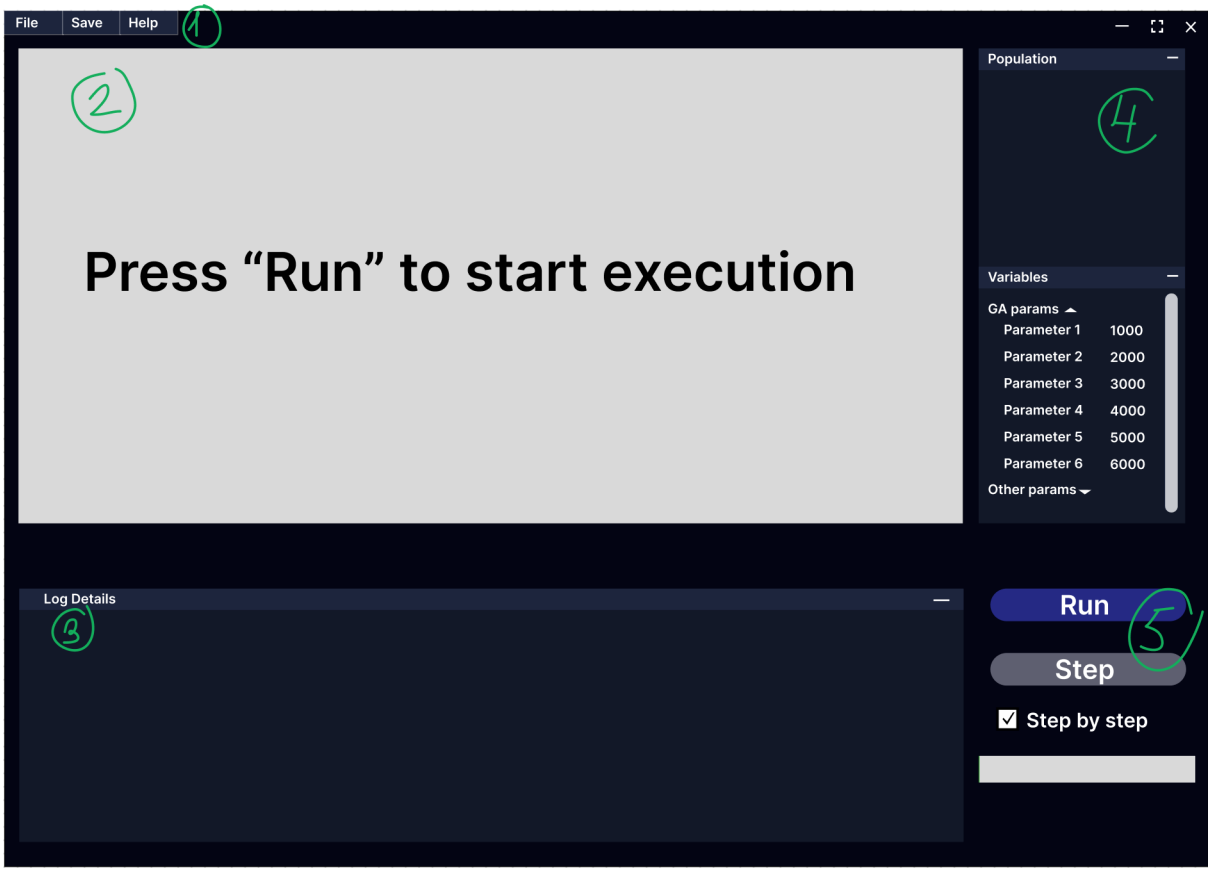
- Ширина ленты W

- Количество прямоугольников и их размеры l и w

GUI

Итерация 2

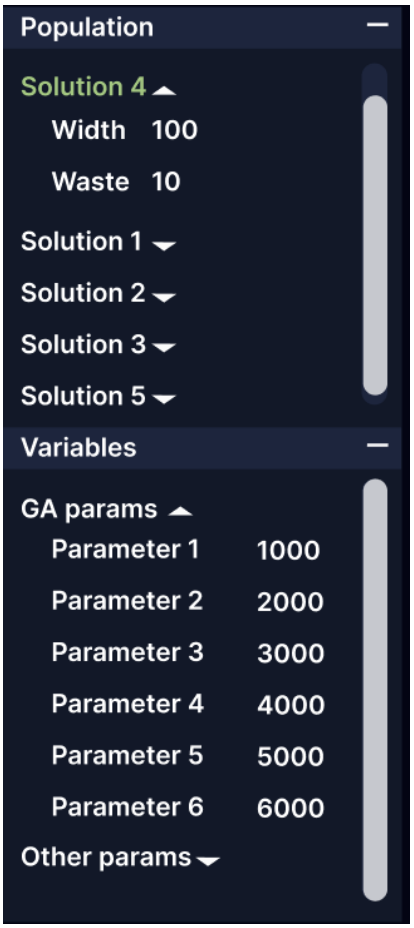
Скетч GUI, который планируется реализовать, можно детально рассмотреть на **figma.** В представленной схеме(находится на figma или можно найти на github в папке design) продемонстрировано поэтапное использование разработанного приложения, ниже подробно разобраны все компоненты GUI.



1. **Menu bar** – меню, состоящее из кнопок **file, save, help.** Первая позволяет создать новый или загрузить в программу файл, из которого будут считываться входные данные, вторая – сохранить результат работы, третья – выводит инструкцию по использованию приложения. Взаимодействие с данными объектами происходит по нажатию ЛКМ, после чего выпадает, например, список действий.

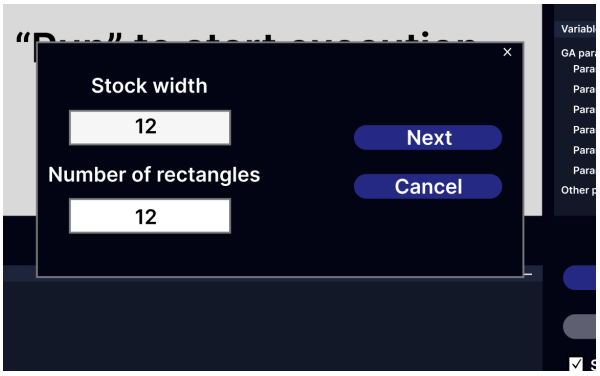
2. Блок демонстрации работы алгоритма.
3. Окно, выводящее текстовые логи с пояснениями к ГА.

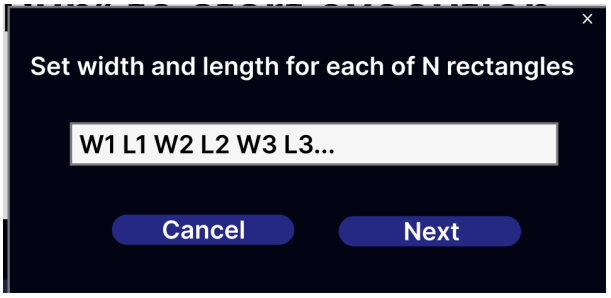
4. Блок **Population** с одновременным отображением особей популяции с выделением лучшей особи зелёным цветом, блок **Variables** – параметры, при которых в данный момент работает алгоритм.



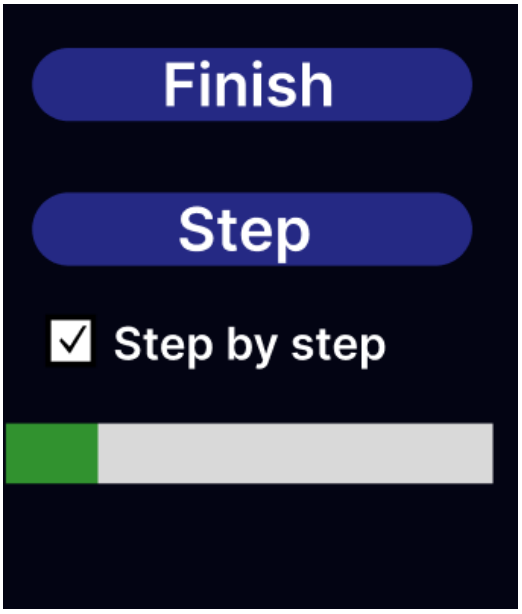
5. Кнопки **Run, Step** и индикатор выполнения.

   По нажатию кнопки запуска во всплывающем окне запрашиваются входные данные у пользователя, если они не были загружены через файл.





Затем можно запросить у программы самостоятельное выполнение поэтапно, если поставить галочку в **Step by step**, либо самостоятельно с помощью кнопки **Step** проделать это. Также есть возможность перейти сразу к завершению работы программы, нажав **Finish**(после запуска кнопка **Run** меняется на **Finish**)



Итерация 3

Созданы все основные поля: поле вывода результата работы(2) алгоритма(без визуализации), окно логов(3), кнопки **Run, Step, Step by step** и **status bar**, индикатор выполнения алгоритма(5), поле вывода переменных и особей популяций(4).

Итерация 4

Был добавлен **Menu Bar**, в которм реализована кнопка справки и импорта файла с данными. Реализована функция **drag’n’drop** окна, исправлен баг изменения позиции всплывающих окон(**HelpDialog, FirstDialog, SecondDialog**) при перемещении главного окна, теперь они на любой ОС отображаются по центру окна.

Взаимодействие с ГА:

Помимо обычного взаимодействия (запуск и пошаговое выполнение) добавлена возможность изменения параметров алгоритма в соответствующем списке. В ближайшее время будет добавлено отображение выбранной особи популяции.

Итерация 5

После подключения backend проекта были устранены основные ошибки. Так же добавлена “защита” от нестандартного поведения поведения пользователя: попытка добавить не текстовый файл для ввода данных; неправильный ввод данных, например, инициализация нулями; нажатие одной и той же кнопки несколько раз подряд и т.д.

Подключены логи приложения на каждое действие приложения и алгоритма.

Генетический алгоритм

Итерация 2

Нами было выбрано три генетических алгоритма:

1. Канонический ГА (+ Гибридная версия)
2. Genitor (+ Гибридная версия)
3. Модель диффузии (Островная модель)

Данный набор алгоритмов был выбран для дальнейшего сравнения и определения лучшего для решения задачи 2DCSP.

Канонический гибридный алгоритм, несмотря на свою простоту, представляет собой базу для модификаций, позволяя настраивать свои операторы в зависимости от задачи.

Genitor предлагает решение, которое на алгоритмическом уровне отличается от канонической модели, поэтому представляет собой интерес сравнение двух видов ГА между собой.

Island model позволяет ускорить вычисления за счет распараллеливания операторов ГА и обработки подпуляций одновременно, поэтому было бы интересно выяснить поможет ли нам такой подход быстрее решить 2DCSP или однопоточных алгоритмов будет достаточно.

Представление особи - список индексов прямоугольников по порядку их раскроя на ленте (декодирования) слева направо.

Используемые метрики качества:

1. Время сходимости. (в секундах)
2. Длина используемой ленты

Особенности мутации и кроссинговера:

1) Канонический (Гибридный) алгоритм
    a) Мутация перестановкой.
    b) Дискретная мутация с уникальной маской.
2) Genitor
    a) Мутация перестановкой.
    b) Дискретная мутация с уникальной маской.
3) Island model
    a) Canonical islands (Типы мутаций и кроссинговера на острове идентичны пункту 1)
    b) Genitor islands (Типы мутаций и кроссинговера на острове идентичны пункту 2)

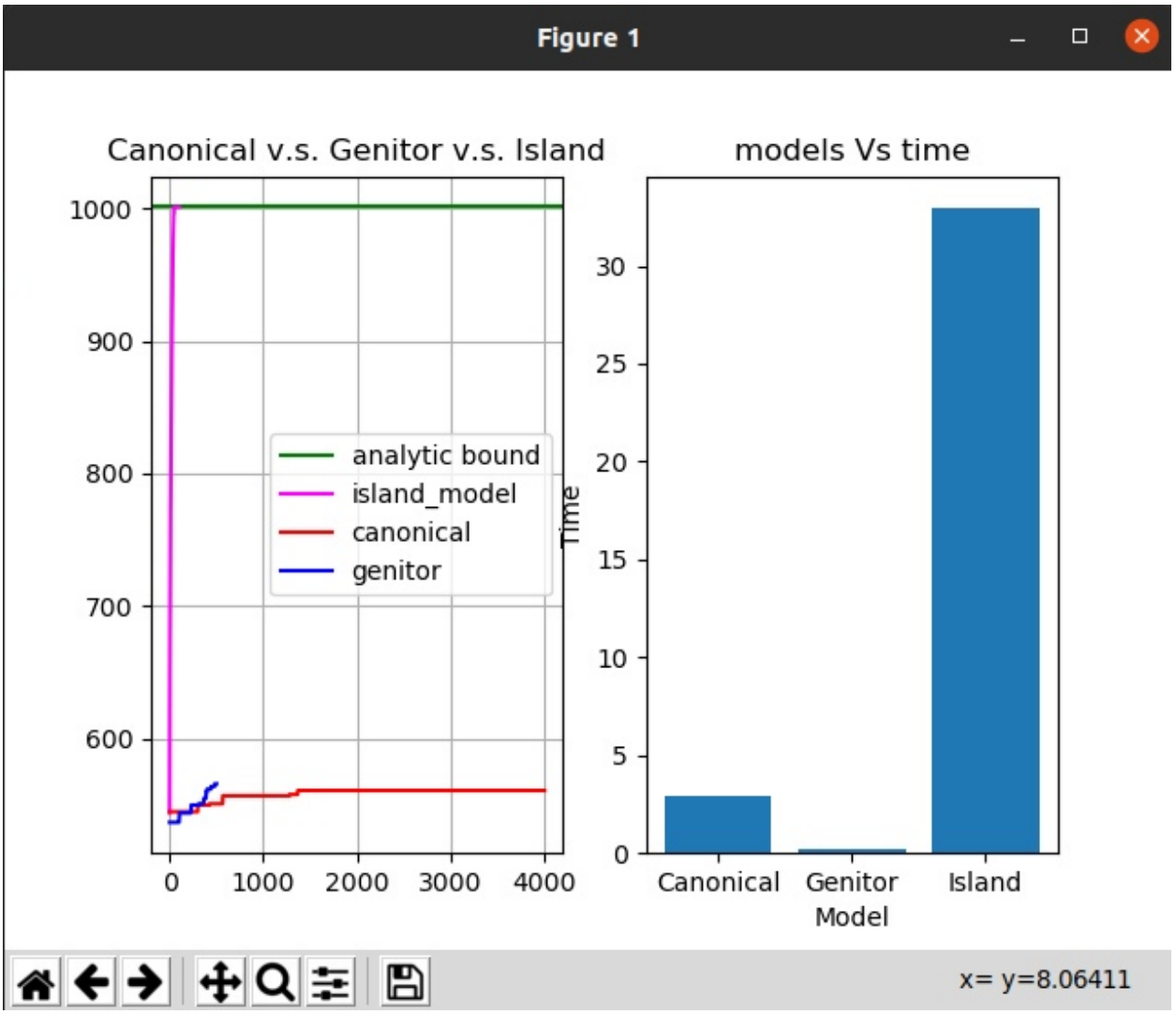
Итерация 3

Были реализованы основные функции библиотеки evpy, а также алгоритмы:

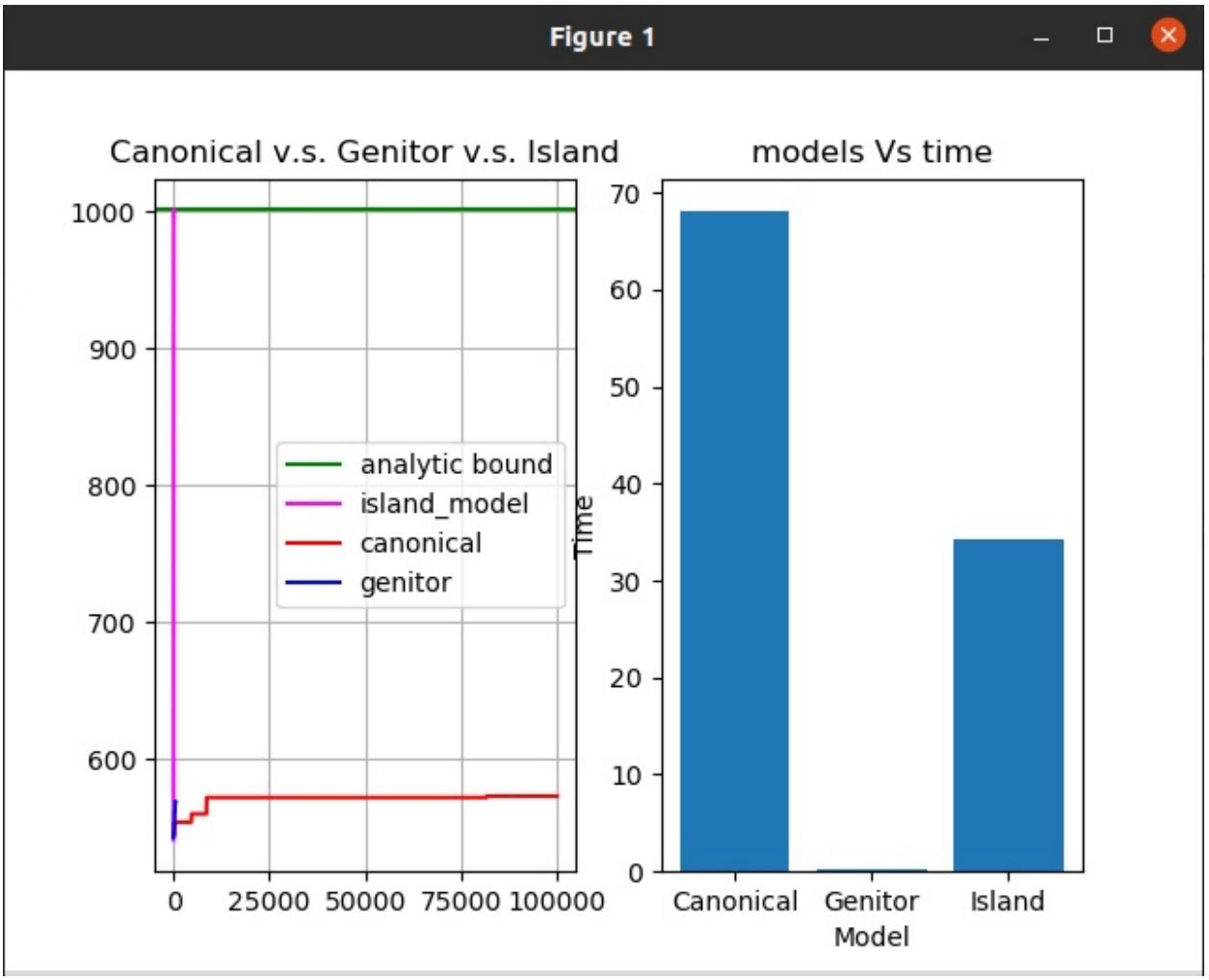
- Canonical
- Genitor
- Island Model

Ниже представлены результаты тестов (тестирование проводилось на отличной от заявленной задаче):

1) Достижение максимума островной моделью путем больших временных затрат (время в секундах):



2) С увеличением итераций наблюдается выигрыш во времени у островной модели (время в секундах):



Итерация 4

Был добавлен декодер для генотипа. Декодер выполняет расстановку прямоугольников в установленной генотипом последовательности.

Итерация 5

Добавлена возможность логирования работы алгоритма для родительского класса Algorithm. Были внесены изменения для обеспечения записи логов в алгоритм Solver, использующийся в Model.

Было решено использовать канонический алгоритм, т.к. он более стабильный по сравнению с Genitor, т.е. шанс найти ложный минимум меньше, и более быстрый, чем Island model, в рамках нашей задачи.

Архитектура программы

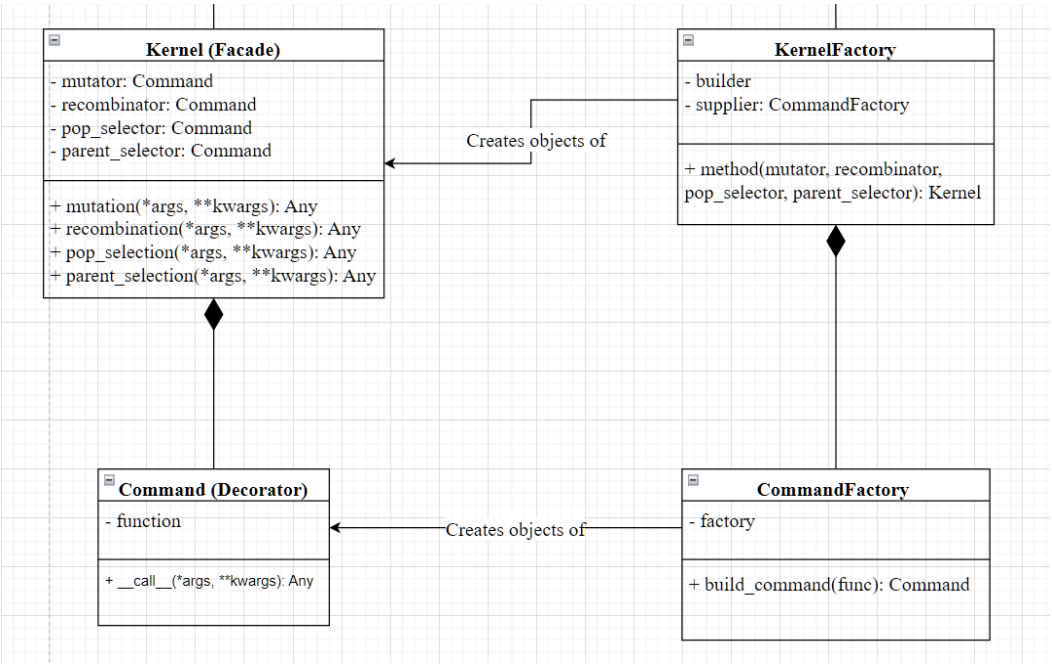
Итерация 3

Был проведен рефакторинг библиотеки, реализующей генетические алгоритмы evpy, в соответствии с парадигмой ООП. Новая архитектура полностью охватывает имеющийся на момент итерации функционал библиотеки (вплоть до классических алгоритмов).

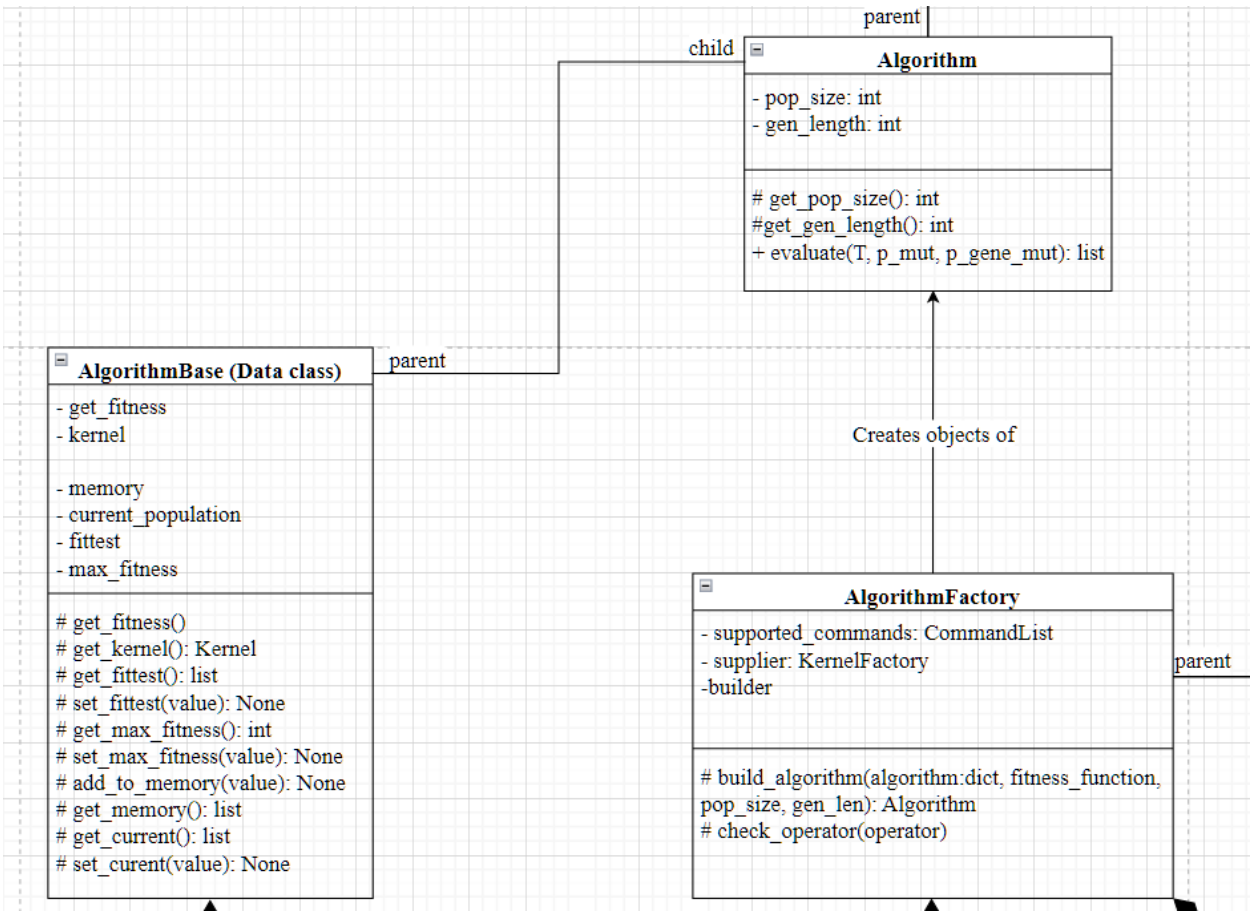
Описание архитектуры:
Библиотека логически разбита на 3 модуля:

- Алгоритмы (классические и параллельные)
- Генетически операторы (функции, использующиеся в алгоритмах)
- Декораторы (являются “оберткой” надо генетическими операторами, для обеспечения работы архитектуры библиотеки)

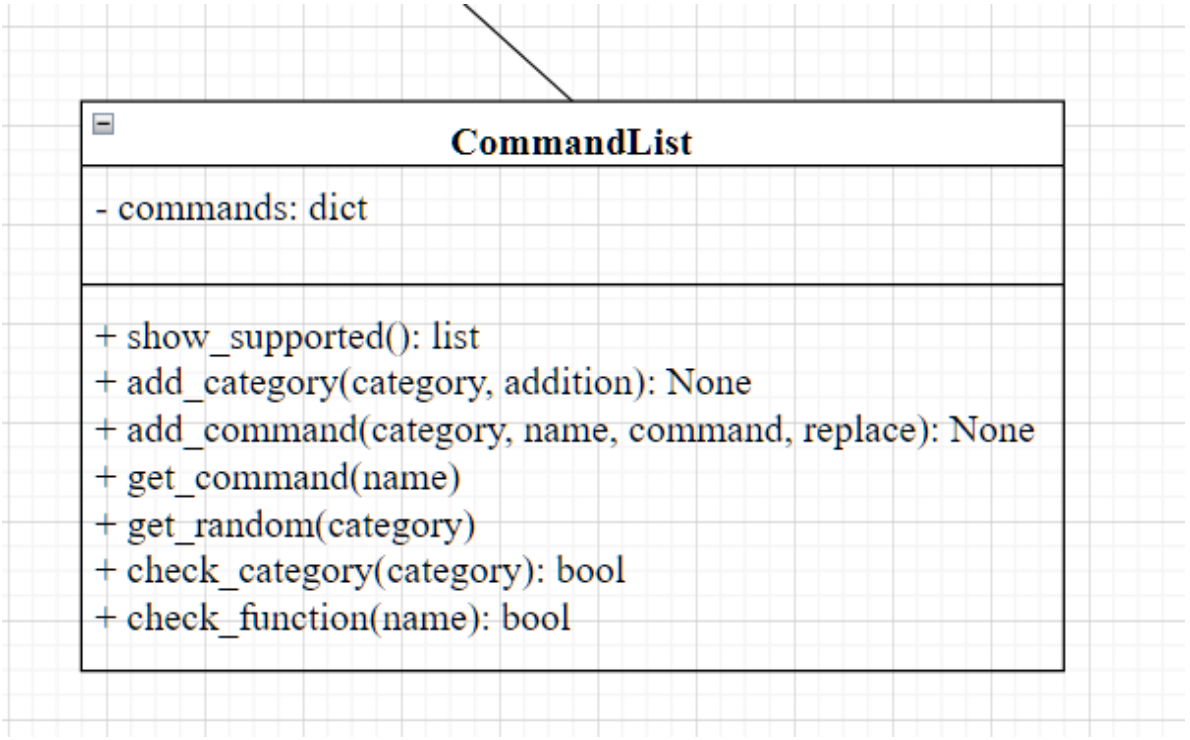
Уровень 1: Самый низкий “уровень” архитектуры работает с генетическими операторами (декораторы Command).
Уровень 2: На втором “уровне”, при помощи композиции происходит объединение классов-декораторов в своеобразный набор инструментов для генетического алгоритма (фасады Kernel). Для каждой такой единицы абстракции реализована своя фабрика. Каждая фабрика уровня выше является композицией фабрик с уровней ниже.



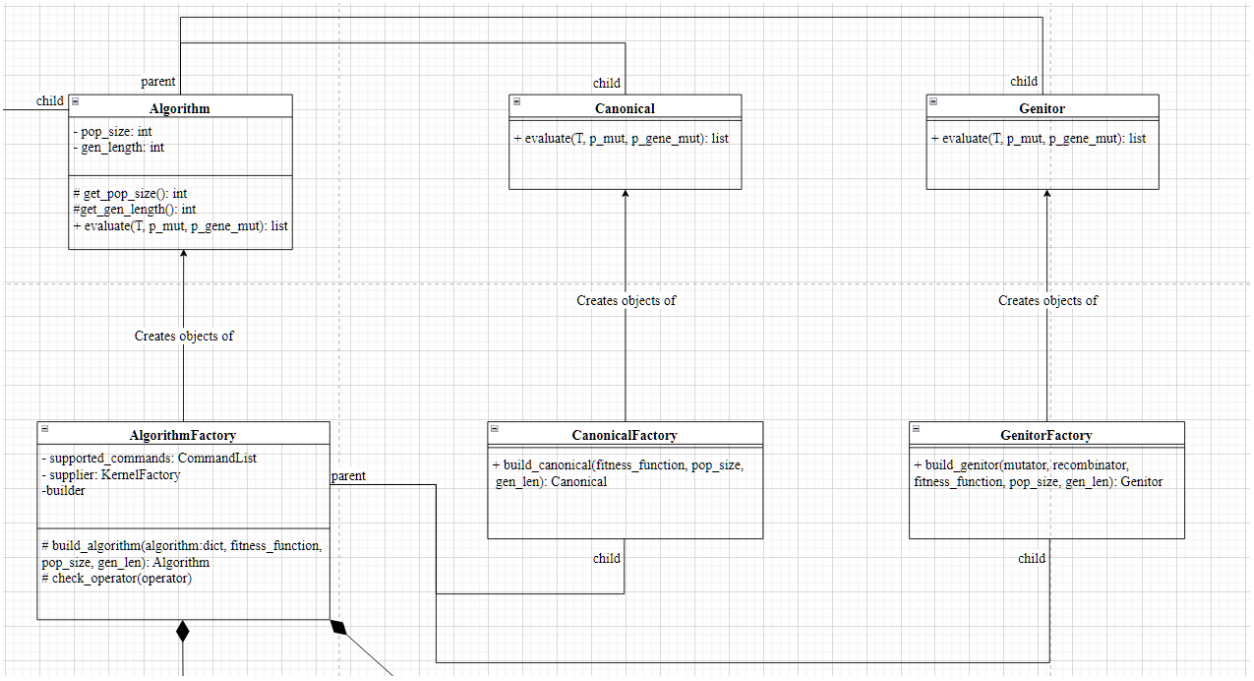
Уровень 3: На следующем “уровне” для более удобной расширяемости функционала библиотеки был реализован класс данных AlgorithmBase. Он хранит в себе минимальный набор данных, необходимых для работы генетического алгоритма, кроме самой реализации алгоритма. Таким образом пользователь может свободно определять функцию работы алгоритма, унаследовав пользовательский от класса AlgorithmBase. От него наследуется родительский класс для классических алгоритмов Algorithm.



Фабрика родительского класса AlgorithmFactory также включает в себя CommandList, который является списком всех поддерживаемых библиотекой функций.



Уровень 4: Классические генетические алгоритмы. Как уже говорилось ранее, от класса Algorithm наследуются Canonical и Genitor алгоритмы. Для каждого из этих классов реализована собственная фабрика.



Итерация 4

Были реализованы классы для решения поставленной проблематики: Model и Presenter. Первый решает непосредственно задачу раскроя, второй обеспечивает связь с GUI. Обеспечено возвращение решения из модели (в виде массива вспомогательных классов для удобного представления) в GUI. Под вспомогательные же классы было переопределено поведение алгоритма в Solver (библиотечные функции остались без изменений).

Итерация 5

Было добавлено логирование работы алгоритма через паттерн Observer. Исправлены незначительные баги и расширено взаимодействие Model с графическим интерфейсом.