МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по итерации №3

по дисциплине «Учебная практика»

Тема: Задача о покрытием окружностями

Зайцев Владислав
Аненков Иван

Студенты гр. 3383 Абдусаламов Рустам

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы.

Решение задачи о покрытии окружностями, с помощью генетического алгоритма, а также реализация графического интерфейса.

Задание.

Задача о покрытием окружностями

Дано N точек в двумерном евклидовом пространстве. Необходимо найти координаты центра и радиусы для M (задается пользователем) окружностей такие, чтобы окружности суммарно содержали максимальное количество точек внутри и не пересекались между собой.

Основные теоретические положения.

Задача генетического алгоритма (ГА) - найти оптимальное решение некоторой задачи, путем развития популяции потенциальных решений называемых индивидуумами. Решения итеративно оцениваются и используются для создания следующего поколения.

Генотип - описание одного индивидуума, набор генов, сгруппированных в одну хромосому.

Популяция - множество индивидуумов (потенциальных решений), которые хранит генетический алгоритм. Популяция отображает текущее поколение.

Функция приспособленности (целевая функция) - функция, которую необходимо оптимизировать в рамках решаемой задачи. На каждой итерации ГА рассчитывает приспособленность индивидуума для формирования нового поколения.

Отбор - формирование множества индивидуумов для использования следующего поколения.

Скрещивание - процесс создания пары новых индивидуумов (потомков), путем комбинации хромосом двух родителей из текущей популяции.

Мутация - процесс случайного изменения хромосомы для поиска неисследованных областей и поддержания разнообразия популяции

Выполнение работы

Распределение ролей:

Ответственный за генетический алгоритм — Зайцев Владислав

3383 Ответственный за GUI - Аненков Иван 3383

Ответственный за загрузку/выгрузку данных и помощь в разработке алгоритма и GUI - Абдусаламов Рустам 3383

План решения задачи

Для решения задачи при помощи генетического алгоритма необходимо определить, что является генотипом в данной задаче и какие функции приспособленности, скрещивания и мутации будут использоваться. В качестве генотипа будет выступать массив вещественных чисел длины 3 * M, где M — число окружностей для покрытия точек (задано пользователем). Каждые три элемента массива описывают одну окружность, а именно координаты её центра и радиус. В качестве функции приспособленности будет использоваться функция, которая вычисляет число точек, покрытых не пересекающимся окружностями в рамках одного генотипа. Отбор индивидуумов будет осуществляться по турнирной схеме, где размер турнира задаётся пользователем.

Для скрещивания был выбран метод одноточечного скрещивания, который позволяет составлять новые решения путём замены части окружностей в индивидууме. Поскольку существует чёткое разделение данных на окружности (каждые три элемента массива), то при выборе точки для скрещивания учитывается, что выбранная точка должна делить изначальный массив на два массива, в которых содержаться изначальные окружности. Это делается для того, чтобы не терять информацию об окружностях. Для мутации будет использоваться вещественная мутация, а именно изменение параметров окружностей при помощи Гауссовского шума с параметрами (mu, sigma). Параметры задаются пользователем. Данный метод был выбран поскольку при мутации обменом, обращением и перестановкой не происходит изменения

решения, потому что данные методы меняют порядок окружностей, который в данной задаче не важен.

Разработка генетического алгоритма

Для реализации генетического алгоритма создан класс GeneticAlgorithm, который при инициализации принимает набор координат точек, которые нужно покрыть, число окружностей для покрытия и набор параметров: pop_size — размер генерируемой популяции, attempts_limit — число попыток для генерации особи, num_generations — число шагов алгоритма, crossover_rate — вероятность скрещивания, mutation_rate — вероятность мутации, r_max — максимальный размер радиуса окружностей, sigma — параметр для Гауссовского шума, tournament_size — размер турнира). Поля класса: points — списко координат точек, params — набор парамметров, M — число окружностей для покрытия, bounding_box — массив из координат минимальных и максимальных координат точек, population — список особей, history массив для хранения поколений, current_generation — текущее поколение, stats — словарь с лучшим значением и средним, current_fitness — текущая оценка популяции.

Метод _has_intersections(self, individual) проверяет есть ли пересечения между любыми двумя окружностями в individual. Возвращает True, если какие-то окружности пересекаются, и False, если все окружности не пересекаются.

Метод initialize_population(self) генерирует популяцию из pop_size особей, каждая с M непересекающимися окружностями. Для ограничения числа попыток генерации популяции используется значение attempts_limit. При генерации используются случайные значения параметров окружностей. Выполняется дополнительная проверка, чтобы окружности не пересекались. Если не удалось собрать популяцию нужного размера, то увеличение происходит при помощи добавления дубликатов лучшей особи.

Метод evaluate_population(self) является целевой функцией, которая возвращает массив из числа покрытых точек для каждой особи. В этой же

функции выполняется поиск максимального значения и среднего значения среди популяции.

Метод _fitness_no_penalty(self, individual) используется для подсчёта числа покрытых точек для одной особи. Для этого выполняется обход всех заданных точек и проверка на их принадлежность какой либо окружности из individual. Данный метод применяется ко всем особям из популяции в методе evaluate_population(self)

Метод select(self) применяется для отбора особей для следующего поколения. Для этого выбирается k случайных индексов из популяции. Среди этих индексов находится и возвращается особь с максимальным значением current_fitness.

Метод crossover(self, p1, p2) выполняет одноточечное скрещивание особей p1 и p2. Скрещивание выполняется не более max_tries раз. Данное ограничение введено из-за возможности получения некорректных решений при скрещивании, а именно пересечения окружностей. Для скрещивания берётся случайная точка в диапазоне [1; M-1] умноженная на 3, для того чтобы разрез не содержал разделённых окружностей. При помощи функции make_children(pt1, pt2, point) выполняется изменение массивов pt1 и pt2 путём обмена частей, разделённых point. Если оба новых решения валидные, то метод возвращает их, если только одно валидное, то возвращает его и копию изначального. Если не получилось ни одного валидного решения за max_tries раз, то возвращаются копии исходных особей.

Метод mutate(self, individual) выполняет вещественное мутирование для individual. Для этого для каждой окружности с некоторой вероятностью меньшей mutation_rate выполняется изменение параметров при помощи Гауссовского шума с заданными параметрами mu и sigma. Полученные значения корректируются так, чтобы значения окружностей не выходили за пределы bounding_box. Если окружности в новом решении пересекаются, то выполняется откат к изначальному решению.

Метод step(self) выполняет один шаг алгоритма. Он оценивает текущею популяцию при помощи метода evaluate_population и создаёт новую, в которой содержится лучшая особь из прошлой. Для создания новой популяции выполняет отбор двух родителей при помощи метода select() и их скрещивание с заданной вероятностью. Если скрещивания не было, то передаются сами родители. Новые особи мутируют с заданной вероятностью и добавляются в новую популяцию.

Метод run_to_end(self) выполняет метод step() num_generations раз.

Метод reset(self) заново создаёт популяцию.

Модификация генетического алгоритма

При модификации часть параметров, вводимых пользователем, было решено перенести как гиперпарамметры, вводимые в поля класса GeneticAlgorithm.

в качестве модификации алгоритма было решено добавить другие варианты скрещивания и мутации. Для этого в метод crossover(self, p1, p2, type_flag) дополнительно передаётся type_flag, который указывает на метод скрещивания:

type_flag == 0 => Одноточечное скрещивание

type_flag == 1 => Двухточечное скрещивание

type_flag == 2 => Равномерное скрещивание

type_flag == 3 => Скрещивание смешением

При двухточечном скрещивании берутся две точки для разделения p1 и p2. Новые особи получаются путём замены частей между элементами point1 и point2. При равномерном скрещивании каждое значение у исходных значений меняется местами равновероятно.

При скрещивании смещением каждое значение особоей изменяется случайным образом в диапазоне [min_val - alpha * (max_val – min_val); max_val + alpha * (max_val – min_val)]

Различные методы мутации реализуются таким же образом, как и при скрещивании при помощи добавления флага type_flag:

type_flag == 0 => Вещественная мутация

type_flag == 1 => Мутация радиуса

type_flag == 2 => Мутация слиянием

type_flag == 3 => Мутация равномерным шумом

type_flag == 4 => Комбинированная мутация (Гауссовский шум для координат центра, равномерный для радиуса)

При мутации радиуса выполняется изменение только радиуса окружностей при помощи Гауссовского шума. Данный метод хорошо работает, когда точки образуют кластеры, в которых расстояние между соседними точками достаточно мало.

При мутации слиянием выполняется слияние ближайших окружностей и генерация новой случайной окружности.

При мутации равномерным шумом выполняется мутация аналогичная вещественной, но с использованием равномерного шума. Параметр delta для генерации шума задаётся пользователем.

При комбинированном шуме координаты центра изменяются при помощи Гауссовского шума, а радиус при помощи равномерного шума. В данном подходе координаты центра изменяются плавно, а радиус изменяется более значительно.

В рамках улучшения алгоритма было решено не отбрасывать невалидные решения, а штрафовать их. Для этого был реализован метод _penalty_intersections(self, individual), который увеличивает штраф для пересекающихся окружностей по формуле penalty += ((ri + rj) - dist) * self.penalty_coeff, где ri + rj сумма радиусов пересекающихся окружностей, dist — расстояние между центрами окружностей, а self.penalty_coeff коэффициент штрафования, задаваемый в конструкторе класса. Итоговый штраф дополнительно увеличивается в зависимости от числа пересекающихся окружностей. Поскольку теперь могут быть возможны пересечения метод initialize_population() был переработан. Теперь параметры окружностей генерируются без ограничения на пересечение. В функции приспособленности

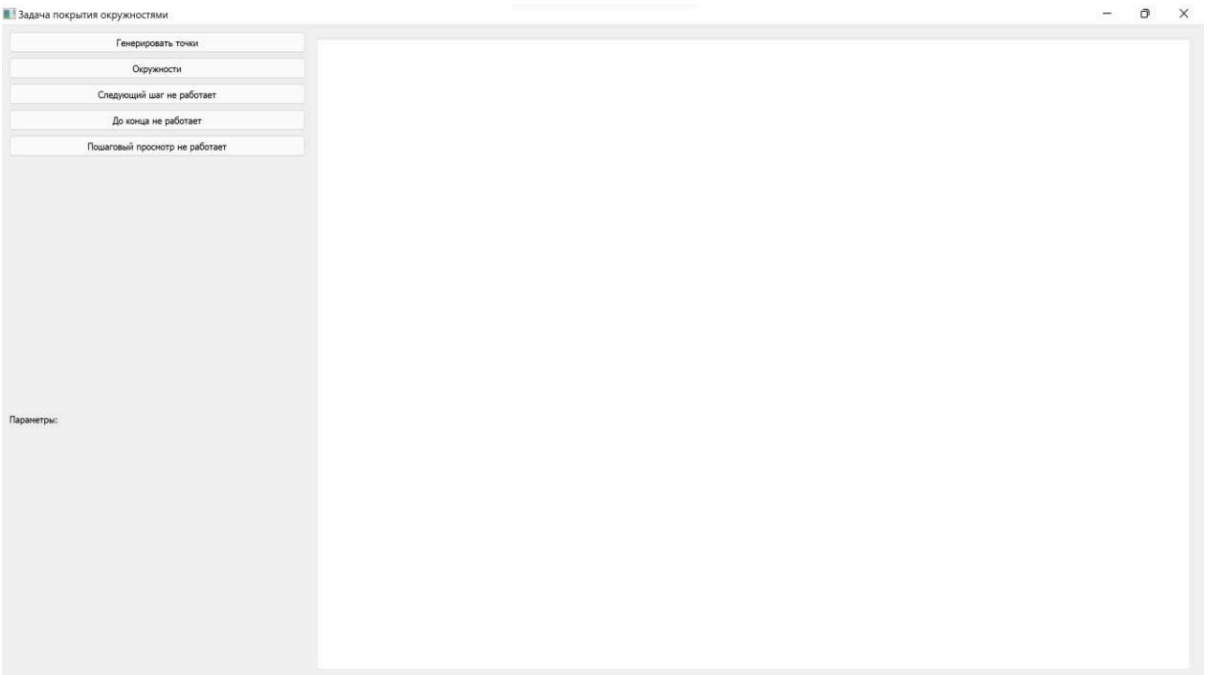
используется метод _fitness_with_penalty(), который возвращает значение разности покрытых точек и минимального значения между результатом _penalty_intersections() и числом покрытых точек умноженным на 0.8 для одной особи. Результаты работы изменённого алгоритма оказались несильно лучше версии с отбрасыванием решений.

Выполнение графического интерфейса (GUI).

Для реализации GUI используется фреймворк PyQt, а для построения графиков MatPlotLib.

в начале было реализовано главное окно графического интерфейса. За реализацию отвечает класс GAWindow. В конструкторе происходит инициализация главного окна, точек и окружностей, вызов метода init_ui() для построения интерфейса. Метод init_ui() предназначен для разметки и оформления интерфейса, создается основной виджет, в который будут вставлены остальные элементы. Также создается панель управления, на которой будут располагаться кнопки для управления. Создание кнопок происходит с помощью метода init_controls().

Рис. 1 – Главное окно.



Далее были реализованы методы, необходимые для построения точек (generate_points) и окружностей (init_circles). Также добавлена возможность изменения параметров, которые используются для работы генетического алгоритма и кнопка “очистки” приложения. Добавленные параметры: r_max – значения максимального радиуса окружностей, num_generations – количество поколений, circles_count – количество окружностей. Для генерации используются значения из равномерного распределения. Графическое отображение точек и окружностей происходит с помощью класса GAVisualizer и его методов. На данном этапе происходит только начальная отрисовка точек и окружностей. Для передачи новых значений используется метод set_data(self, points, circles), а для отрисовки графика метод - update_plot(self). Представлено на рисунках 2 и 3.

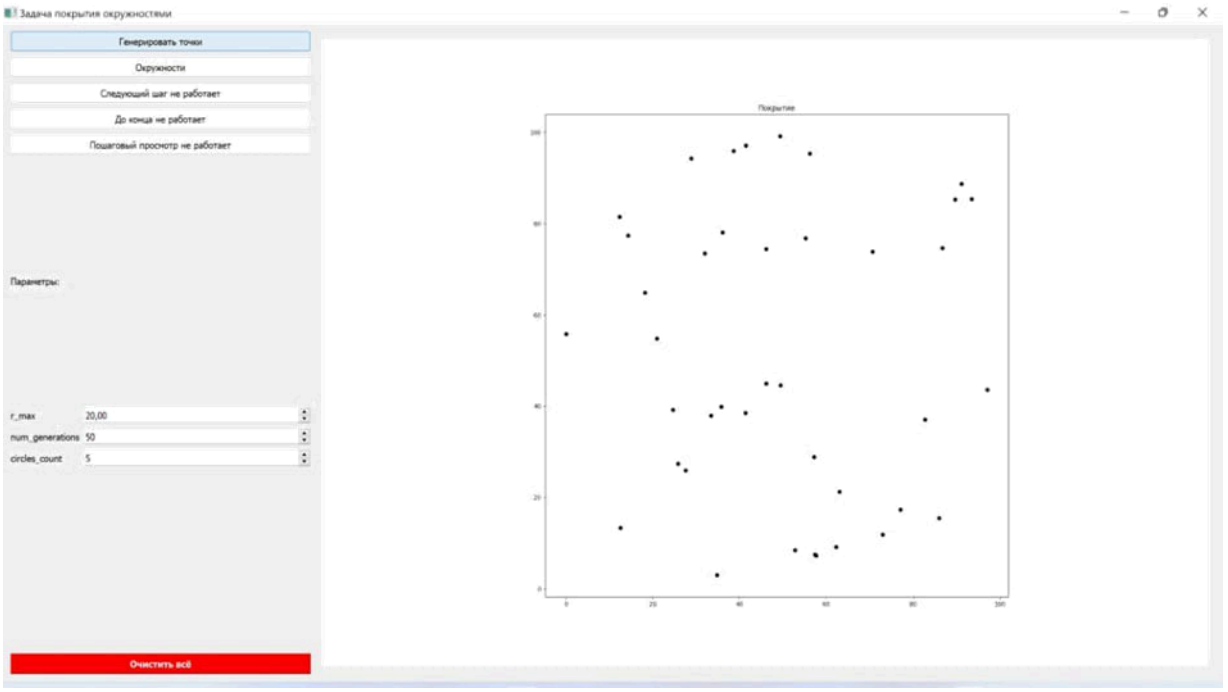


Рис.2 – Генерация точек.

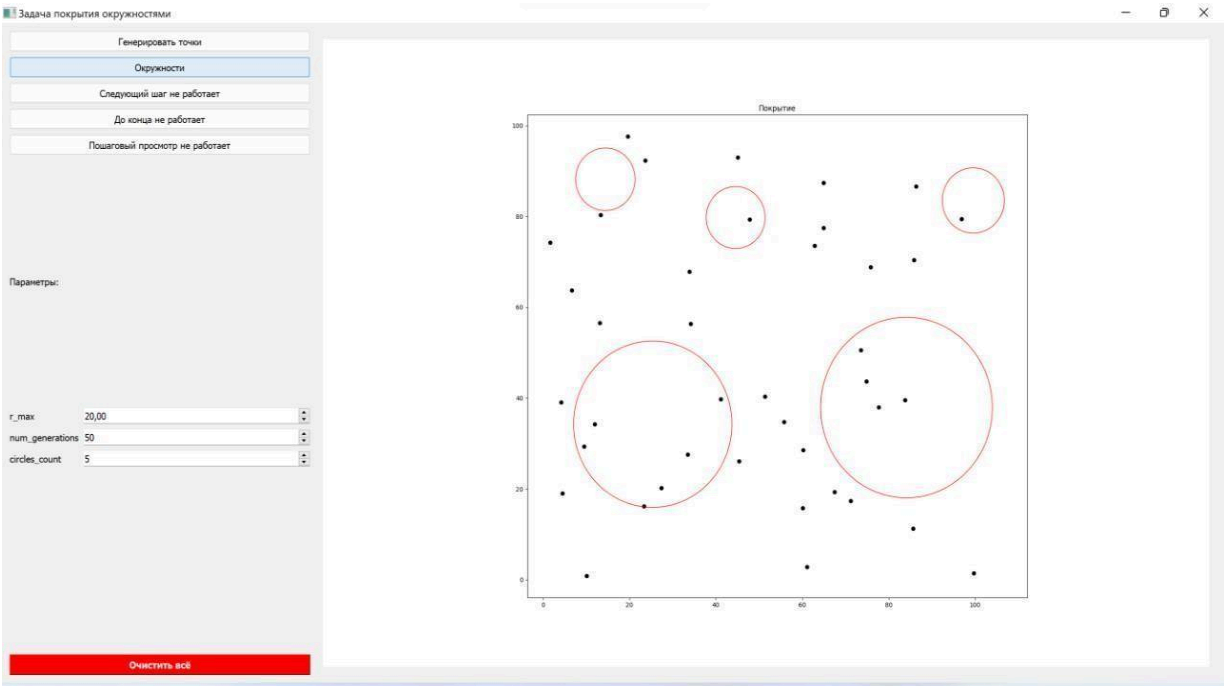


Рис. 3 – Генерация точек и окружностей

Добавлена возможность задания точек из файла. После того, как алгоритм успешно завершился, лучшее решение будет выгружено в файл.

Далее, по мере написания основного алгоритма, стали рабочими кнопки графического интерфейса, предназначенные для просмотра шагов алгоритма. Также появился график, который строится в ходе работы алгоритма. Метод step_ga(self) используется для выполнения одного шага алгоритма, происходит обновление графика и печать значений: номера поколения, лучшего и среднего значения. Метод run_to_end(self) предназначен для просмотра конечного решения, он устроен так, что метод одного шага вызывается пока установленное значение поколения меньше, чем заданное количество поколений (задается в графическом интерфейсе). Методы def step_through_all(self) и def step_through_tick(self) используются для пошагового просмотра решения алгоритма. Запуск одного шага через метод step_through_tick(), который вызывает сам себя с задержкой времени. В методе def step_through_all(self) происходит проверка на инициализацию окружностей и точек и вызов step_through_tick. Изменен метод update_plot(self): добавлена отрисовка лучшего и среднего покрытий. Представлено на рисунках 4 и 5.

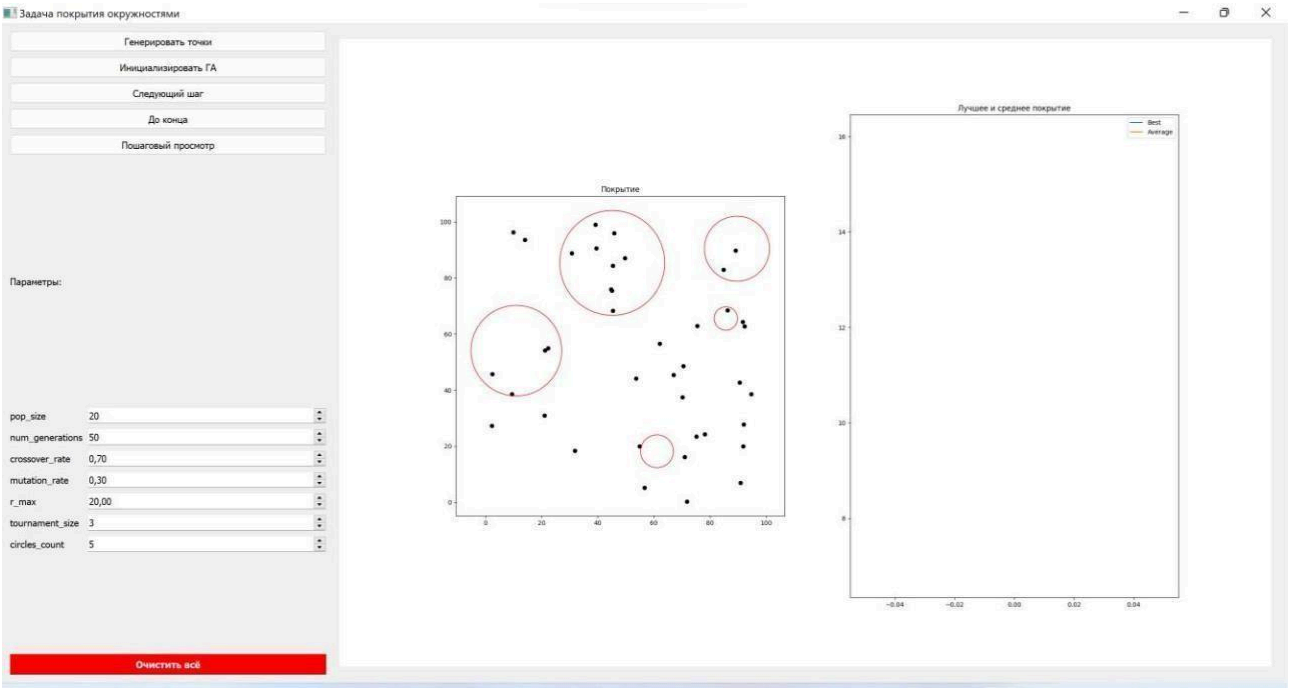


Рис.4 – Инициализация решения и холста с графиком.

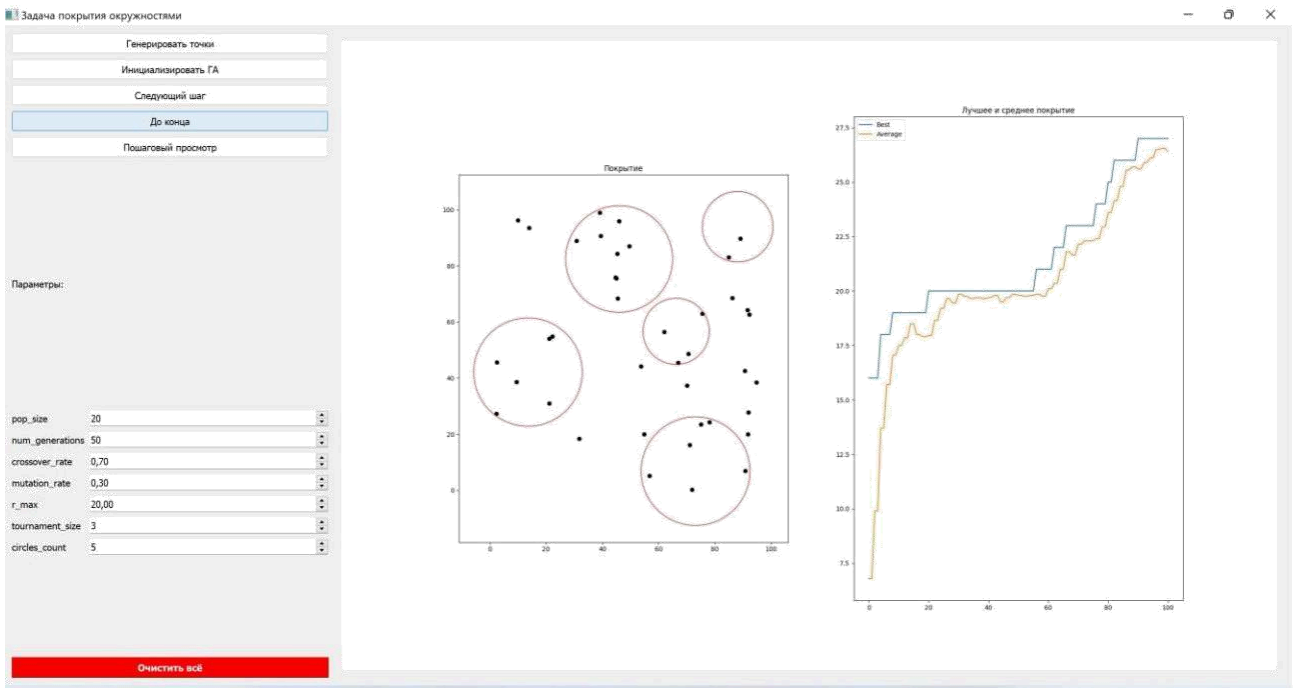


Рис. 5 – Конечные значения и их график.

Далее добавлена возможность сделать шаг назад в работе алгоритма – метод step_back(self): данный метод можно применить если текущее поколение не первое и история должна содержать как минимум два поколения (текущее и хотя бы одно предыдущее). Копируется предыдущее состояние популяции из истории, и история обрезается до текущего поколения. Добавлено сохранение графиков в файл и возможность очистки окружностей. Также были добавлены новые параметры, так как произошло расширение основного алгоритма.

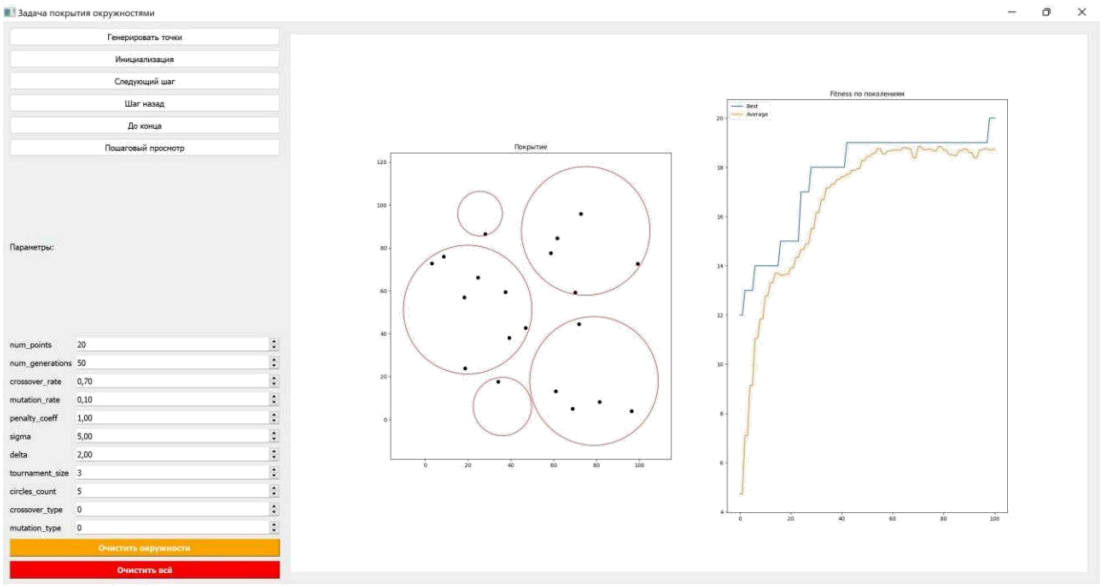


Рис. 6 – Расширение графического интерфейса

Изменения в GUI после первой итерации.

Изменено положение графиков на холсте, теперь график лучших и средних значений находится выше графика, на котором отображается покрытие, увеличен шрифт для подписи осей графиков. Также кнопки для различных видов расстановки точек были перенесены выше и теперь находятся в одном блоке, который называется “Расстановка точек” на панели управления. Блок “Параметры” перенесен выше, также перенесена кнопка очистки окружностей. Пример графического интерфейса представлен на рисунке 7.

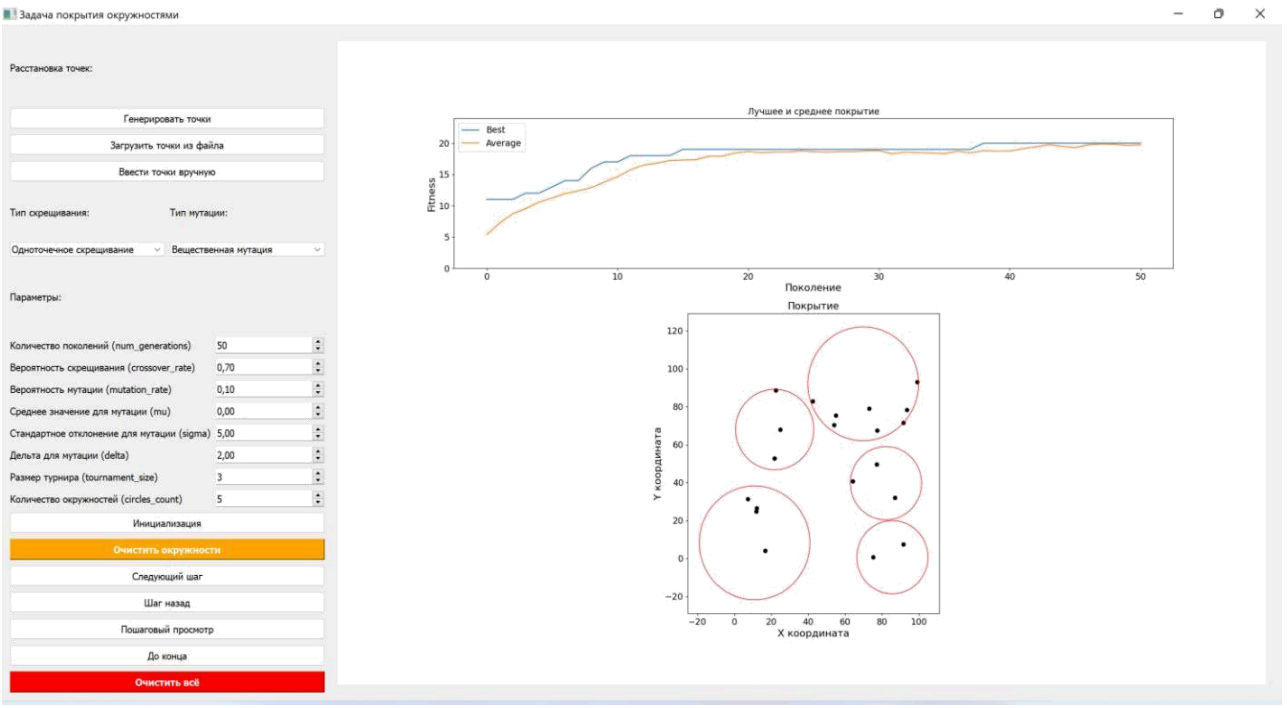


Рис. 7 – Обновление графического интерфейса.

Изменения в GUI после второй итерации.

Добавлена возможность изменения значения размера популяции внутри графического интерфейса, изменен график покрытия, теперь его размеры не изменяются в зависимости от шага алгоритма. Графический интерфейс выглядит следующим образом.

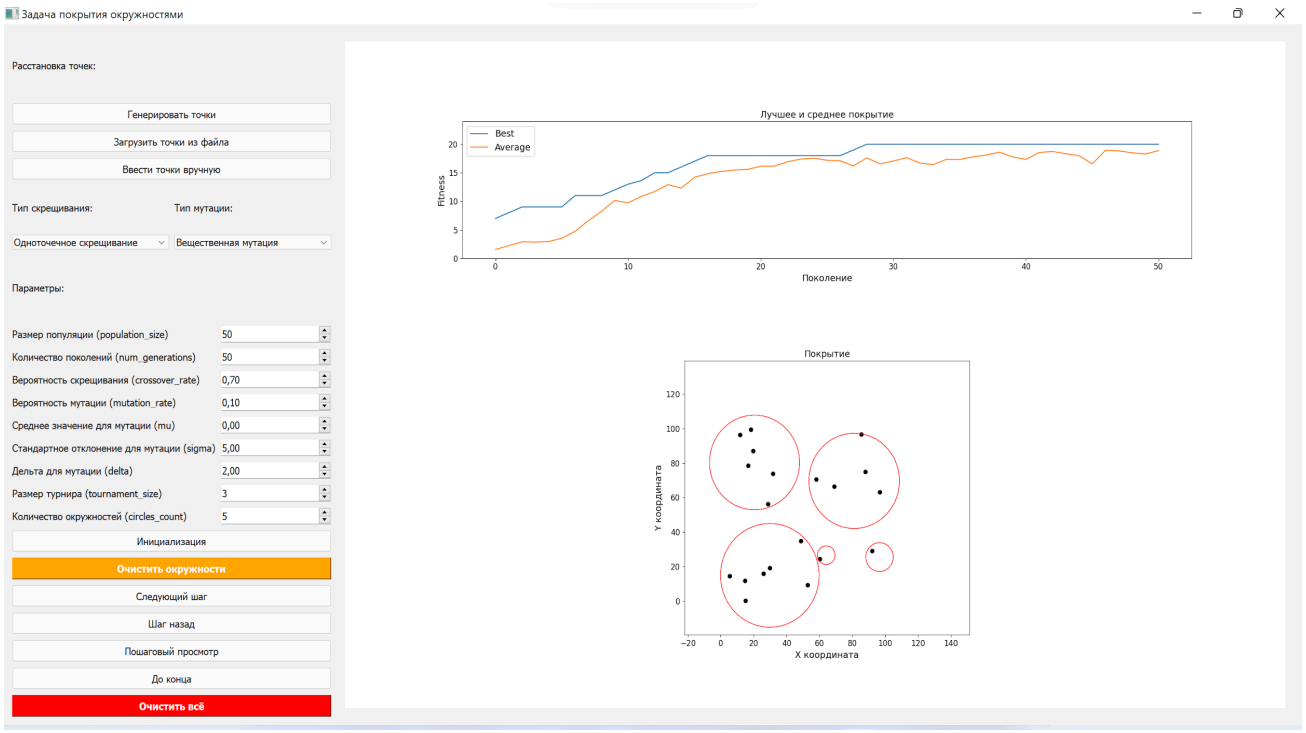


Рис. 8 – Изменение графического интерфейса.

Вывод

В рамках работы были изучены основные понятия и методы работы с генетическими алгоритмами. Была реализована программа, решающая задачу о покрытии точек конечным числом окружностей. Для программы написан графический интерфейс, позволяющий увидеть динамику изменения решений, полученных алгоритмом.