Цели и предпринятые действия.

Необходимо было переписать Классический генетический алгоритм, который находит решение для доски 8x8, чтобы он находил решение для доски 64x64. Затем модифицировать его, изменив фитнесс функцию, которая считает число противоположное количество уникальных пар ферзей под ударом на фитнесс функцию, которая считает число противоположное количеству пар ферзей, находящихся под ударом без перекрытия. Провести тесты для их сравнения.

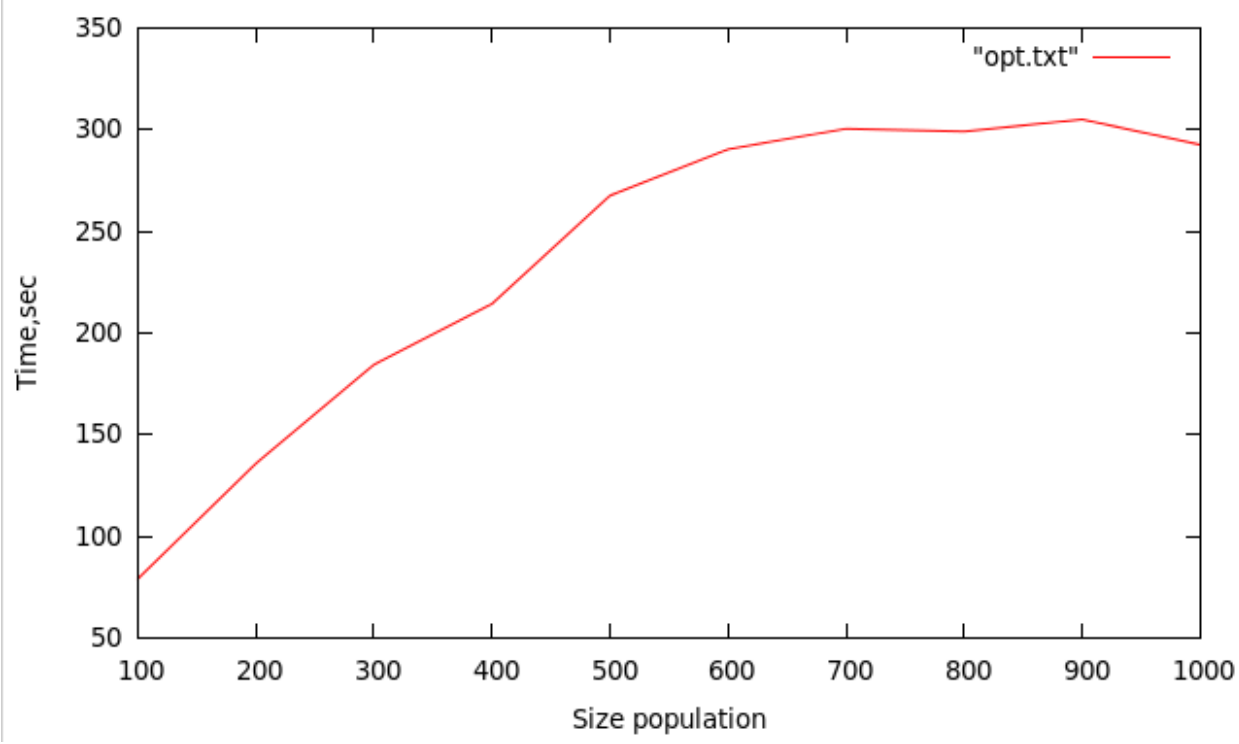
Для этого были сделаны следующие действия:

1) Модифицирован метод класса CBitArray getOctet.
2) Написан класс CChomosomeChessboard64x64 с фитнесс функцией, которая считает число противоположное количество уникальных пар ферзей под ударом.
3) Изменён метод applyCrossover в CPopulation, т.к. там был допущен целый ряд ошибок.
4) Изменён метод searchSolution в CPopulation.
5) Изменены методы selectSingle, applyMutation, т.к. при компиляции под Windows они работали некорректно. К тому же была исправлена ошибка с неравномерным распределением случайных значений.
6) На основе п.3 выбран оптимальный размер популяции.
7) Написан класс CChomosomeChessboard64x64Modify с фитнесс функцией, которая считает число противоположное количеству пар ферзей, находящихся под ударом без перекрытия.
8) В CPopulation добавлены методы runTournamentSelection и MaxFitness.
9) В CPopulation изменён метод runEpoch, чтобы можно было использовать его не только для Классического генетического алгоритма, но и для модифицированного. С этой же целью изменён конструктор CPopulation и updateRouletteWheel.
10) Переписан main.cpp, для предоставления удобного интерфейса. Добавлен man(инструкция).
11) Проведены замеры среднего времени нахождения решения с не более чем 5 конфликтами на шахматной доске для оптимальной популяции Классического и модифицированного генетического алгоритма.

Подробное описание действий.

1) **Модифицирован метод класса CBitArray getOctet.**
   А) int bitIndex = index * 3;
   исправлено на
   int bitIndex = index * 6;
   Б) return (array[cellIndex] >> bitOffset) & 7;
   исправлено на
   return (array[cellIndex] >> bitOffset) & 63;
2) **Написан класс CChomosomeChessboard64x64 с фитнесс функцией, которая считает число противоположное количество уникальных пар ферзей под ударом.**
   А) Смотри chessboard64.hpp
3) **Изменён метод applyCrossover в CPopulation, т.к. там был допущен целый ряд ошибок.**
   А) В скрещивании участвовала только последняя пара хромосом.
   Б) Менялась только первая хромосома, в неё копировалась часть другой хромосомы, а после эта же часть копировалась во вторую.
4) **Изменён метод searchSolution в CPopulation.**
   А) Добавлен критерий остановки алгоритма.
5) **Изменены методы selectSingle, applyMutation, т.к. при компиляции под Windows они работали некорректно.**
   А) Была исправлена ошибка с неравномерным распределением случайных значений.
   Выражение rand() % 100001 даёт неравномерное распределение, корректней писать не (rand() % 100001) <= (mutationProbability * 100000), а rand() <= (mutationProbability * RAND_MAX). К тому же программа начинает работать корректно под Windows. Объясняется это те, что rand() в Windows и Linux реализованы по разному и возвращают значения в разных диапазонах(Windows до 2^15-1, Linux до 2^31-1).
6) **На основе п.3 выбран оптимальный размер популяции.**
   А) Оптимальным размером популяции с параметрами вероятности скрещивания 0.5 и вероятности мутации 0.25 оказался размер равный 100.
   Замечание.

Начиная с 700 при подсчёте времени возникали ошибки, в виде переполнения, вследствие чего возникали отрицательные значения времени и среднее время было занижено. Были попытки исправить этот недочёт функциями time и difftime, но, т.к. работа выполнялась на виртуальной машине, и возникала необходимость погрузить её в “сон”, то от этой идеи пришлось отказаться.



7) **Написан класс CChomosomeChessboard64x64Modify с фитнесс функцией, которая считает число противоположное количеству пар ферзей, находящихся под ударом без перекрытия.**
   A) Смотри chessboard64modify.cpp
8) **В CPopulation добавлены методы runTournamentSelection и MaxFitness.**
   A) Функция runTournamentSelection выбирает 3 случайных представителя (не обязательно уникальных) и добавляет в родительский пул лучшего из них.
   Б) Функция MaxFitness вспомогательная. Она возвращает лучшую из 2-х хромосом.
9) **В CPopulation изменён метод runEpoch, чтобы можно было использовать его не только для Классического генетического алгоритма, но и для модифицированного. С этой же целью изменён конструктор CPopulation и updateRouletteWheel.**

А) В runEpoch добавлен флаг от которого завит будет выполняться турнирная селекция или метод колеса-рулетки.

Б) Из CPopulation исключена инициализация rouletteWheel и перенесена в updateRouletteWheel.

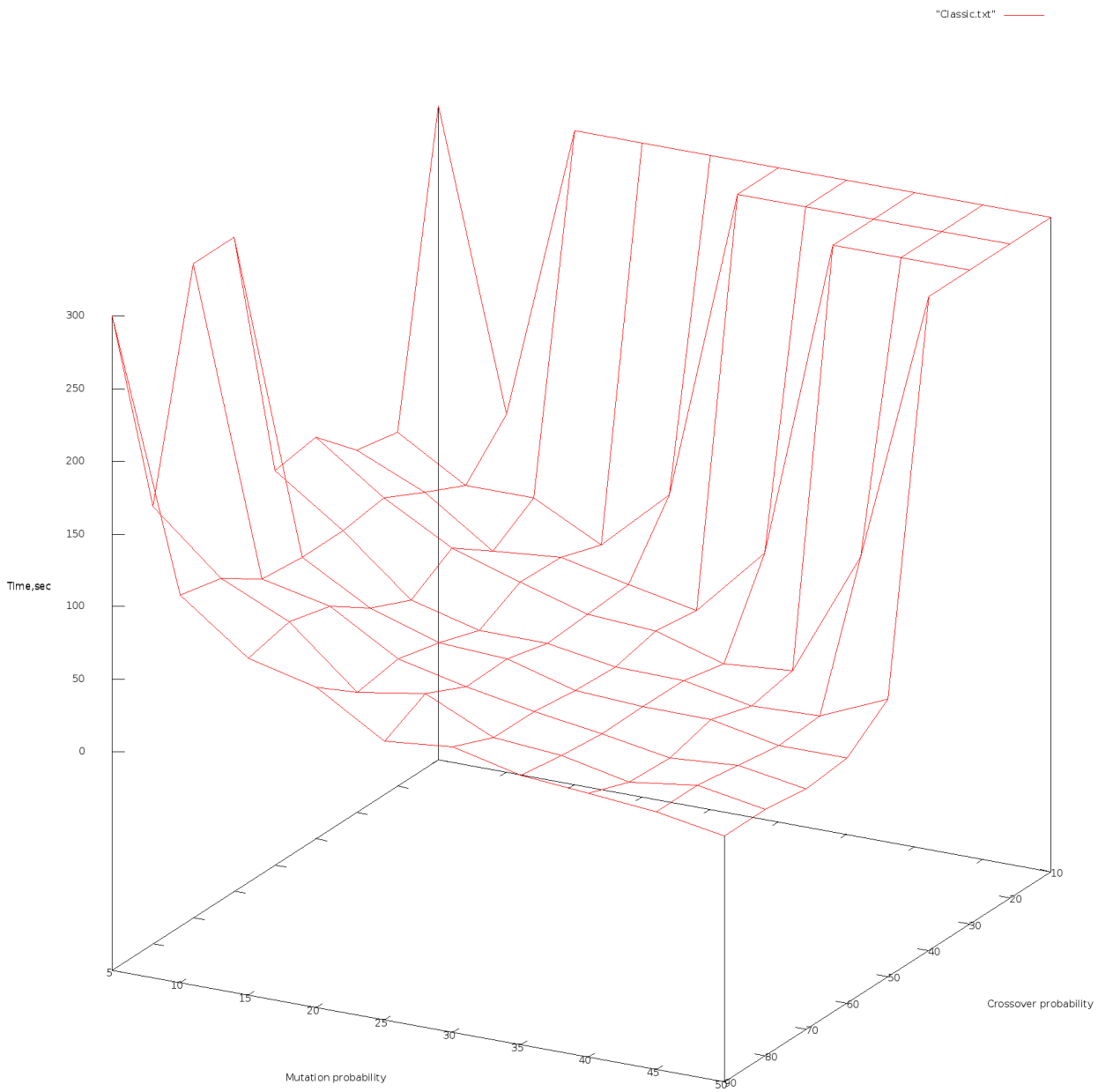
10) Переписан main.cpp, для предоставления удобного интерфейса. Добавлен man(инструкция).

А) Есть возможность с помощью аргументов командной строки запускать функцию подбора оптимального размера популяции, функцию тестирования классического и модифицированного алгоритма.

11) Проведены замеры среднего времени нахождения решения с не более чем 5 конфликтами на шахматной доске для оптимальной популяции Классического и модифицированного генетического алгоритма.

А) Результаты Классического генетического алгоритма.

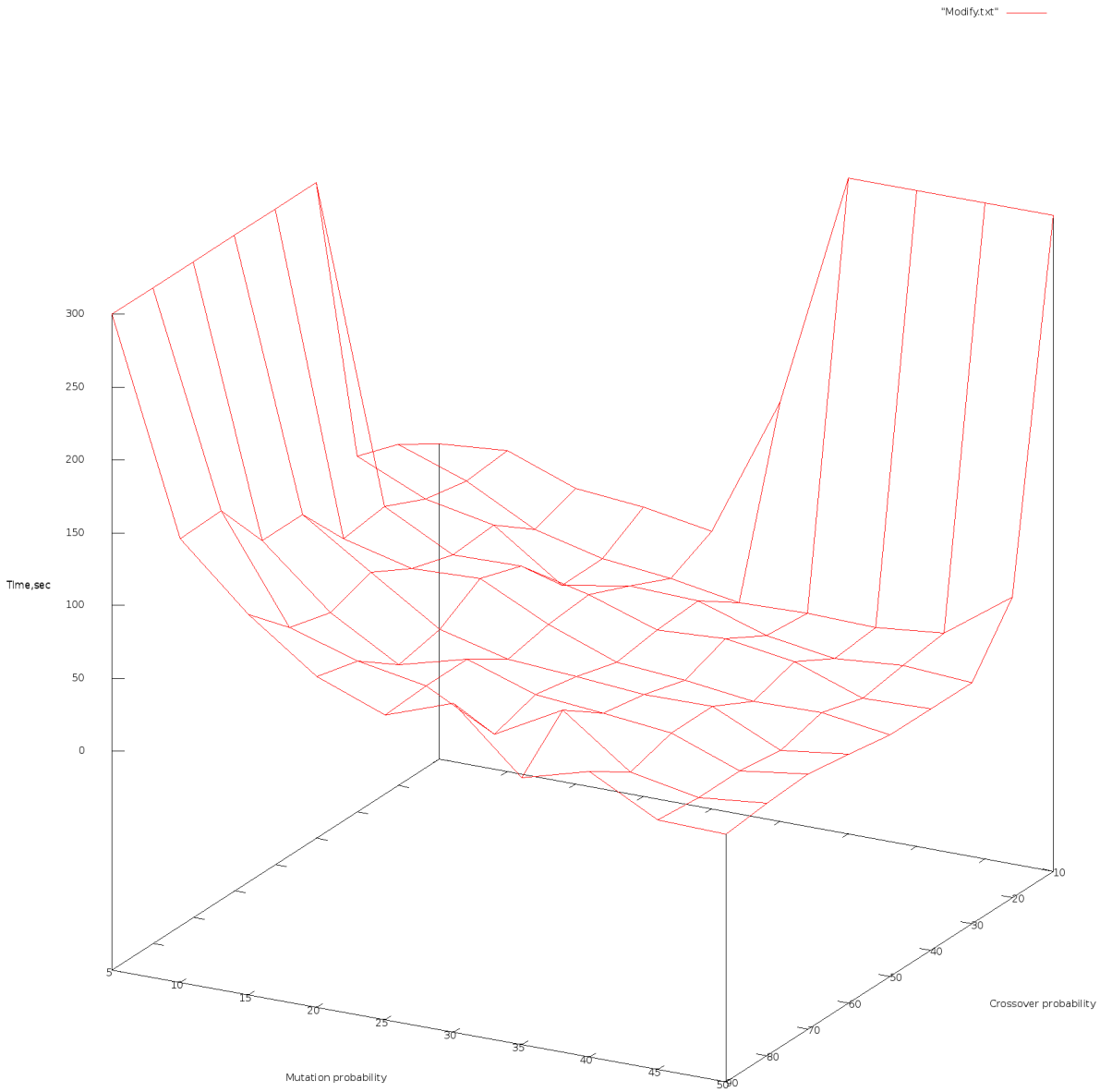
	5	10	15	20	25	30	35	40	45	50
10	300	96.36	300	300	300	300	300	300	300	300
20	93.76	65.49	41.90	84.67	300	300	300	300	300	300
30	99.58	79.09	47.06	51.36	41	31.303	79.91	300	300	300
40	126.18	93.06	67.39	52.30	38.45	35.45	21.33	25.18	112.69	300
50	121.45	88.75	49.18	36.72	36.54	28.78	27.87	19.06	21	40.72
60	300	88.57	61.54	46.51	43.78	30.90	28.33	28	18.42	18.42
70	300	91.78	80.90	53.48	42.69	34.30	27.48	19.54	22.72	15.27
80	151.03	110.06	88.54	48.48	56.12	34.51	30.54	21	27.42	19.42
90	300	116.30	81.60	70	41.54	46.15	35.27	31.57	27.21	18.96



Б) Результаты модифицированного генетического алгоритма

	5	10	15	20	25	30	35	40	45	50
10	66.66	70.60	52.87	48.75	40.93	138.06	300	300	300	300
20	84.27	67.51	43.15	31.60	26.39	17.81	19.30	18	23	56.21
30	94.39	73.63	64.15	31.39	39.42	37.54	21.93	14.78	18.57	15.27
40	300	86.54	61.60	62.51	51.36	35.63	38.15	30.66	14.36	15.66
50	300	82.42	70.42	72.15	48.93	31.42	27.60	21.57	22.63	15.72
60	300	117.09	85.96	54.81	43.06	39.51	35.78	36.51	14.84	20.81
70	300	117.33	76.09	48.84	61	45.24	41.15	36.15	18.87	25.30

80	300	155.87	83.84	69.45	60.72	36.33	61.57	27.48	18.36	22.96
90	300	154.84	111	76.93	58.75	75.66	33.15	45.69	21.06	19.96



Замечание.

Задание было выполнено не полностью. Требовалось провести замеры среднего времени работы модифицированного генетического алгоритма для подобранного оптимального размера популяции при 100 экспериментах, а приведённые выше результаты(для подобранного оптимального размера популяции) были получены при 33 экспериментах. К тому же при срабатывании критерия остановки более чем в 33% случаях считалось среднее время равным 300.

Выводы:

Турнирная селекция позволяет находить решения даже при неудачных выбранных вероятностях скрещивания и мутации, однако она замедляет поиск решения. Но благодаря тому, что была изменена функция приспособленности, которая вычисляется быстрее, модифицированный алгоритм по усреднённому среднему времени нахождения решения превзошёл Классический алгоритм (50,20 секунды против 53,62). Таким образом сочетание этих модификаций оказалось удачным.