**ОЛИМПИАДА ШКОЛЬНИКОВ «ШАГ В БУДУЩЕЕ»**

**НАУЧНО-ОБРАЗОВАТЕЛЬНОЕ СОРЕВНОВАНИЕ «ШАГ В БУДУЩЕЕ, МОСКВА»**

**Информатика, искусственный интеллект и системы управления**

**Программное обеспечение ЭВМ и информационные технологии**

**ИСПОЛЬЗОВАНИЕ РАСПРЕДЕЛЁННОЙ СЕТИ ПРИ РЕШЕНИИ ЗАДАЧ ОПТИМИЗАЦИИ С ПОМОЩЬЮ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ**

11 класс, ГБОУ Школа №1532,

Формаковский Александр Павлович

Регистрационный номер: 313

Руководитель: учитель информатики, ГБОУ Школа №1532,

Сергиенко Антон Борисович

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

Подпись научного руководителя

**Москва, 2022**

# Содержание

[Содержание 2](#_Toc98144593)

[Введение 3](#_Toc98144594)

[Цель и задачи работы 4](#_Toc98144595)

[Методика выполнения работы 4](#_Toc98144596)

[Результаты и обсуждение 12](#_Toc98144597)

[Выводы 17](#_Toc98144598)

[Список используемой литературы 18](#_Toc98144599)

# 

# Введение

При решении задач глобальной оптимизации с помощью генетического алгоритма часто возникает ситуация, когда вычисление значения целевой функции занимает сильно больше времени по сравнению с самим алгоритмом. Такой задачей может быть, например, обучение нейронной сети, или любая другая задача, в которой нельзя мгновенно получить или сохранить в кэш значение целевой функции. Как правило, при решении подобных задач с помощью одного последовательно выполняющего инструкции компьютера, алгоритм будет выполняться чересчур долго.

Например, стандартный генетический алгоритм, который решает тестовую задачу, с фиксированной длительностью выполнения целевой функции (0,1 секунда) (табл. 1):

|  |  |
| --- | --- |
| Тип Алгоритма | Классический генетический алгоритм |
| Параметры задаются для | Центральной машины |
| Количество индивидов | 60 |
| Количество генов | 200 |
| Режим мутации | WEAK |
| Количество поколений | 60 |
| Количество машин в сети | - |
| Количество тестов для E | 12 |
| Количество вычислений целевой функции | 3600 |
| Время(мс) | 4805812 |

Таблица 1 - Результаты классического генетического алгоритма с тестовой функцией.

Обработка всего 60 поколений генетического алгоритма заняла больше часа, а для того, чтобы найти оптимальное решение, часто может потребоваться и 480 поколений и даже больше, и это при том, что целевая функция вычисляется всего за 0,1 секунду.

Генетический алгоритм, о котором идет речь, является эвристическим алгоритмом поиска, который был вдохновлен процессом природной эволюции и принадлежит к большому классу эволюционных алгоритмов. Генетические алгоритмы используют для решения сложных задач глобальной оптимизации (например, NP-полных задач), в которых один вызов целевой функции может занимать значительное время. В настоящее время генетические алгоритмы являются перспективным направлением в области искусственного интеллекта.

В этой работе я попытаюсь увеличить эффективность работы генетического алгоритма, путем внедрения распределенных вычислений.

# Цель и задачи работы

Целью работы является реализация и исследование эффективности модификации генетического алгоритма, для работы в распределенной сети.

Были поставлены следующие задачи работы:

1. Реализовать стандартный генетический алгоритм на языке Python.
2. Составить тестовые задачи для имитации сложной целевой функции.
3. Получить результаты работы стандартного алгоритма на тестовых функциях.
4. Предложить несколько вариантов модификаций генетического алгоритма, для работы в распределенной сети.
5. Получить результаты работы нескольких модификаций алгоритма на тестовых функциях и провести их анализ.
6. Сравнить полученные данные, и найти самую эффективную модификацию.

# Методика выполнения работы

Для реализации генетического алгоритма был выбран язык Python.

Был сделал выбор в пользу Python по следующим причинам:

1. В современном мире Python для алгоритмов искусственного интеллекта является де факто стандартом.
2. Данный алгоритм планируется использовать для целевых функций реализованных на Python.
3. На Python написаны необходимые для работы библиотеки.
4. Большинство компаний на западе используют именно Python для таких задач.

Учитывая особенности алгоритма, многие минусы Python реализации нивелируются свойствами генетического алгоритма. В нашем случае вычисление целевой функции занимает гораздо больше процессорного времени, нежели сам алгоритм.

Стоит так же отметить, что автор работы, уже три года изучает генетические алгоритмы, и это не первая его работа по данной теме. Автор понимает, что выбор языка Python спорный, но стоит отметить, что изначально для всех реализаций своего генетического алгоритма автор использовал C++. Но ввиду вышеперечисленных причин автор, основываясь на своих старых наработках на С++ переписал алгоритм на Python. Проще говоря, текущая реализация полностью основана на C++ реализации автора (которая тоже доступна на репозитории автора в ветви RELOADED).

Был реализован простейший генетический алгоритм, с модульной структурой, легкой заменой тестовой функции. Суть алгоритма заключается в том, что, генерируется случайный набор бинарных индивидов (массивов из нулей и единиц), каждый из которых является одним из возможных решений для задачи, с этим набором индивидов (далее поколением) проводятся различные операции на многочисленных итерациях. На каждой итерации качество поколения улучшается (значения целевых функций приближаются к оптимальным) за счет различных действий, основанных на природных явлениях и эволюции.

Работа генетического алгоритма описывается в 7 этапах (рис. 1).

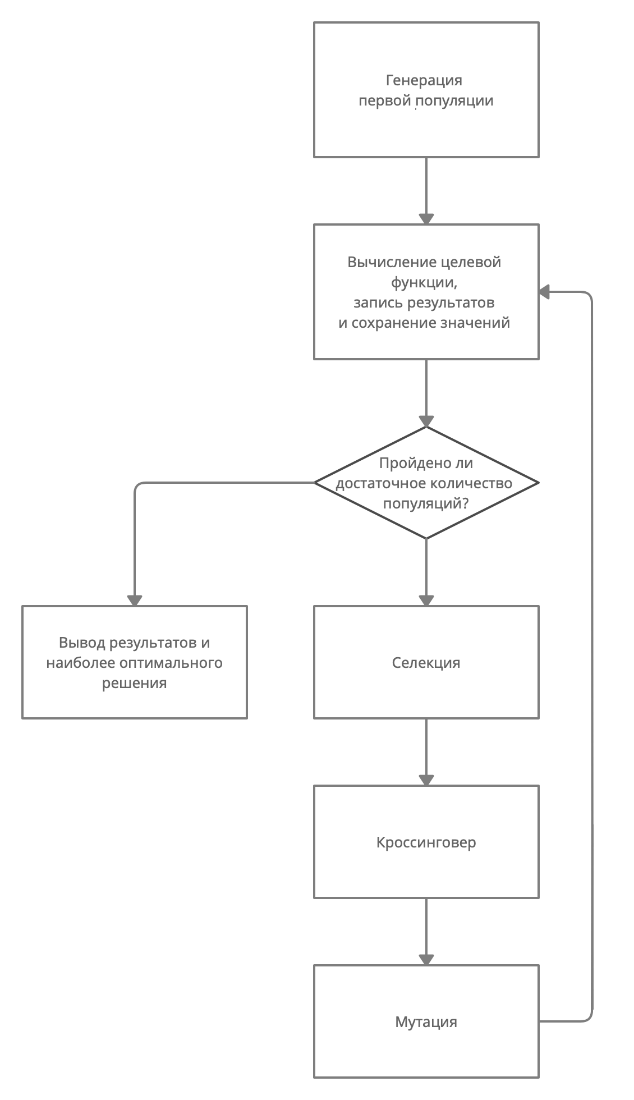


Рисунок 1 – Схема работы генетического алгоритма

Первый этап.

Задаются значения длины каждого индивида , размера популяции , режима работы мутации и количество итераций . Затем генерируется первая популяция (матрица ).

Второй этап.

Для каждого индивида в популяции вычисляется значение целевой функции, проще говоря каждый индивид подставляют в целевую функцию как аргумент, и записывают результат.

Третий этап.

Алгоритм сверяет значение текущей итерации и если оно равно максимальному, то алгоритм выводит наилучшее решение, и выводит данные с графиками.

Четвертый этап.

Из популяции случайно отбираются 4 индивида, из них выбираются два таких чье значение целевой функции наиболее близко к оптимальному, затем два полученных сохраняются в массив для следующей популяции. Эта операция проводится пока не наберется пар индивидов.

Пятый этап.

На этом этапе пары индивидов (родителей), выбранные в прошлом этапе, скрещиваются между собой, и таким образом получается новая популяция. В данном случае используется одноточечное скрещивание, это означает что каждый родитель разрывается в одном случайном месте. Затем берется первая часть первого родителя и вторая часть второго родителя и они соединяются. Так получается новый индивид (ребёнок), который унаследовал черты обоих родителей. Таким же образом получается второй ребёнок. В следующий этап проходит случайный из получившихся двух индивидов-детей.

Шестой этап.

На этом этапе алгоритм проходит по матрице и меняет значения на противоположные с заданной вероятностью. Проще говоря если, был ноль, то после мутации будет на его месте будет единица.

У мутации есть несколько режимов работы, которые отличаются вероятностью смены гена:

WEAK:

MEDIUM:

STRONG:

После этого алгоритм возвращается на второй этап, а счетчик повышается на единицу.

Седьмой этап.

Если алгоритм прошел все итерации t, то он сохраняет лучшее решение.

Были реализованы следующие модификации:

1. ТИП-0, Классический генетический алгоритм с распределением вычислений целевых функций.
2. ТИП-2, Модифицированный генетический алгоритм с распределением вычислений «кусками».
3. ТИП-3, Модификация алгоритма ТИП-2, в которой «куски» обмениваются индивидами.

Модификация ТИП-0 представляет собой инженерный подход к решению

задачи, модификация никак не влияет на ход работы алгоритма и влияет только на распределение самих вычислений между устройствами. Идея модификации заключается в том, что так-как в генетическом алгоритме часто вычисление функции занимает гораздо больше вычислительного времени нежели сама работа алгоритма, модификация пытается переложить на несколько устройств конкретно вычисления целевой функции, проще говоря, в этой модификации сам генетический алгоритма остается на центральной машине(сервере), но каждый вызов функции распределен между несколькими подключенными к распределенной сети компьютерами. Таким образом данная модификация должна показывать в сложных задачах уменьшение времени вычислений примерно в число равное количеству подключенных к распределенной сети компьютеров (без учета затрат на передачу данных).

Модификация ТИП-2 представляет собой научно-исследовательский подход к решению задачи. Данная модификация пытается увеличить эффективность самого алгоритма сохраняя константным количество вызов целевой функции , где - количество индивидов, а количество итераций (выращенных поколений). Суть данной модификации заключается в том, что сервер больше не занимается обсчетом алгоритма и лишь раздает параметры работы алгоритма всем подключенным к распределенной сети устройствам(клиентам), и, по сути, в данной модификации после начала работы, алгоритма запускается несколько «под-алгоритмов», количество которых равно количеству подключенных клиентов, проще говоря, каждый клиент представляет собой отдельный генетический алгоритм и параллельно остальным клиентам ищет свое решение. В конце работы алгоритма получается определенное количество решений целевой функции, которое равно количеству клиентов, из них сервер выбирает ближайшее к оптимуму и выдает как ответ. Чтобы сохранить константным количество вычислений целевых функций, которое в таком алгоритме равно , где - количество индивидов, количество итераций, а количество клиентов, нам приходится делить количество индивидов на количество клиентов или же количество популяций на количество клиентов, таким образом у нас получается 2 варианта использования данной модификации.

Модификация ТИП-3 является модификацией модификации ТИП-2, в которой параллельно работающие клиенты получают возможность наилучшими индивидами между собой, имитируя ситуации из реальной природы, когда есть несколько поселений каких-либо животных одного вида, которые разделяет условный каньон, который могут пересечь только самые сильные индивиды. В данной модификации на каждой популяции у всех клиентов есть 50% шанс отправки лучшего индивида серверу, где он попадет в банк индивидов, после чего сервер их случайно распределит между отправившими индивидов клиентами. Таким образом разные клиенты могут перенять к себе в популяции новые наборы генов, которые могут быть более эффективны чем те, что сформировались у них.

О реализации алгоритма и модификаций:

Была реализована библиотека с классами GeneticAlgorithm и GeneticIndividual, в дальнейшем используется наследование этих классов для модификаций. В качестве массивов использовались numpy.array из библиотеки numpy. Также в коде реализована модульность подключения тестовых функций. Для передачи данных по сети, используется встроенная библиотека socket, и протокол транспортного уровня TCP.

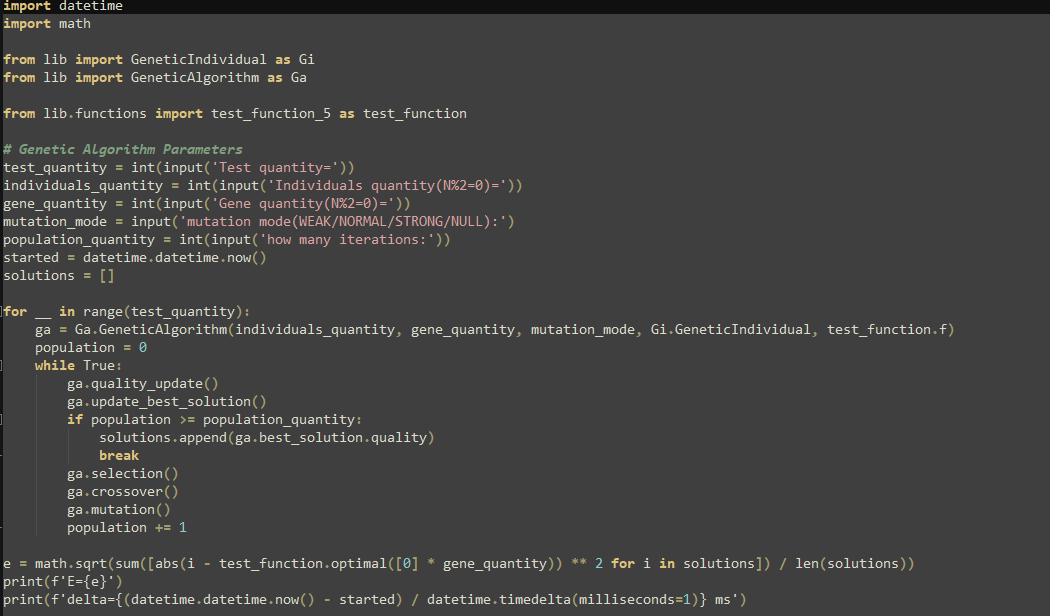


Рисунок 2 - Пример реализации классического генетического алгоритма, для работы с функцией с имитацией сложных вычислений.

Для тестирования и сравнения эффективности модификаций были выбраны следующие функции:

1. Двухмерный параболоид xi ∈ [-10;10] (рис. 3)

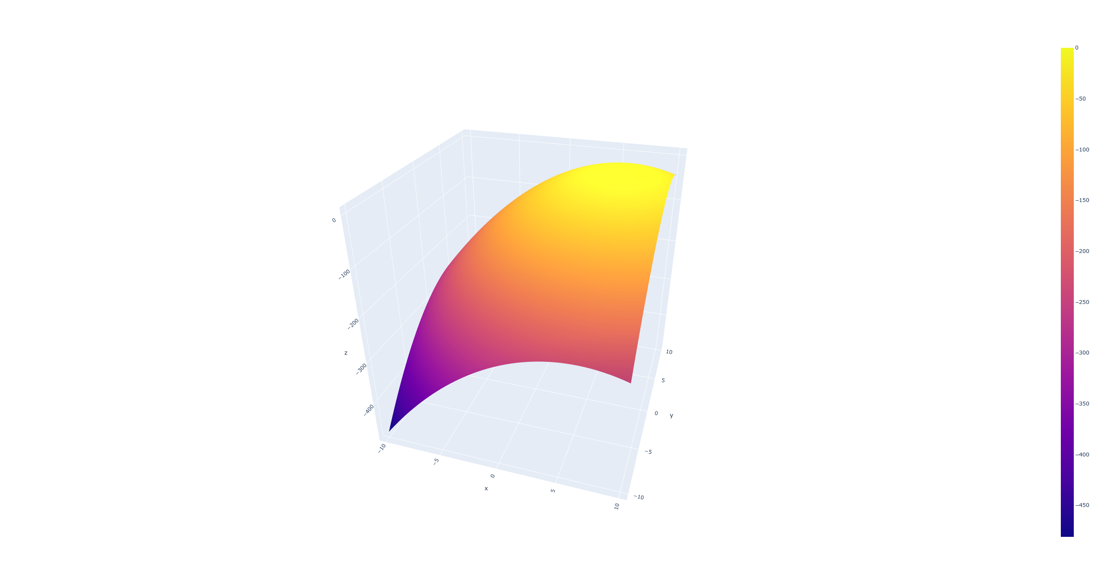


Рисунок 3 - График функции .

1. 4 экстремальная функция Химмельблау xi ∈ [-5;5] (рис. 4)

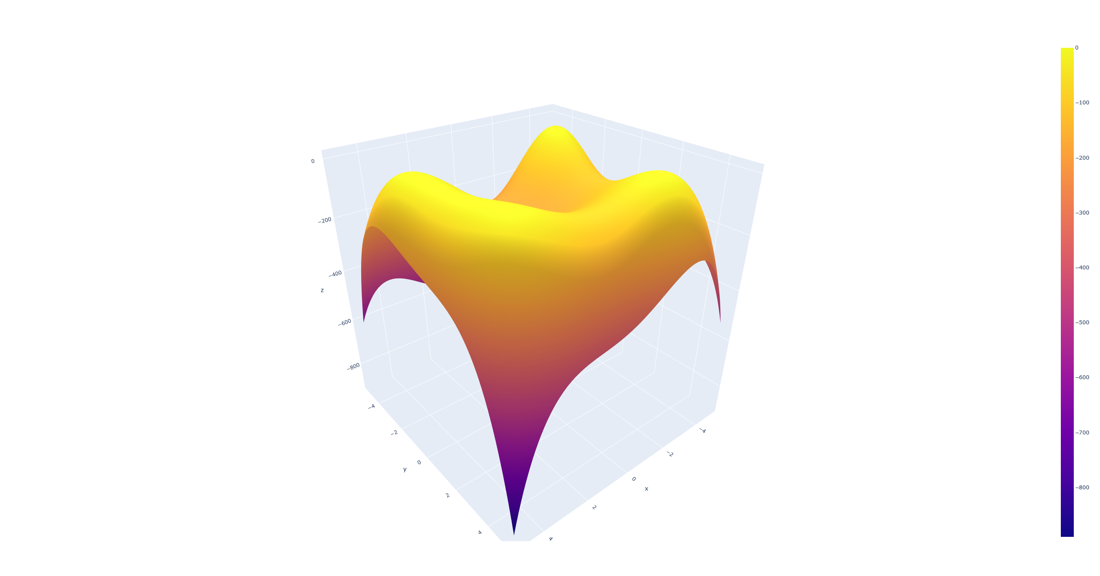


Рисунок 4 - График функции Химмельблау.

1. Тестовая функция бинарного максимума (максимальное количество единиц в индивиде), c имитацией сложных вычислений. (после вызова функции, алгоритм ждет 0,1 секунду).

В качестве задачи был выбран поиск максимума вышеперечисленных функций.

# Результаты и обсуждение

Перед началом исследования статистики стоит учесть следующую конфигурацию тестирования, тестирование проводилось через loopback ip адрес 127.0.0.1, для минимизации погрешности на передачу данных. Также в качестве процессора использовался 16-поточный Intel Core i7 10700k, с заблокированной частотой на 3,8Ghz, и 32Gb Дуалранговой ОЗУ, работающей в двухканальном режиме на частоте 3200Mhz c первичными таймингами 16-16-16-34-CR-2, G.Skill Trident Z 3200c16 RGB 32GB. Тестирование проводилось на чистой, отдельно установленной Windows 10 20H2, с отключенным интернетом.

Статистика для функции Химмельблау, чем краснее, тем больше, чем синее, тем меньше, время - дельта в миллисекундах от начала работы алгоритма до конца 12 теста, E - среднеквадратичное отклонение решения от оптимума функции (табл. 2).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Тип Алгоритма: | Классический генетический алгоритм | Модификация ТИП-0 | Модификация ТИП-2 | Модификация ТИП-2 | Модификация ТИП-3 | Модификация ТИП-3 | Случайный подбор (Метод Монте-Карло) |
| Параметры задаются: | Для Сервера | Для  Сервера | Для каждого  клиента | Для каждого клиента | Для каждого клиента | Для каждого клиента | - |
| Количество индивидов: | 60 | 60 | 60 | 5 | 60 | 5 | - |
| Количество генов: | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| Режим мутации: | WEAK | WEAK | WEAK | WEAK | WEAK | WEAK | - |
| Количество поколений: | 60 | 60 | 5 | 60 | 5 | 60 | - |
| Количество машин в сети: | - | 12 | 12 | 12 | 12 | 12 | - |
| Количество тестов для E: | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| Количество вычислений целевой функции: | 3600 | 3600 | 3600 | 3600 | 3600 | 3600 | 3600 |
| Первый запуск E: | 0,019685713 | 0,011243683 | 0,05261976 | 0,039330408 | 0,034386533 | 0,017010522 | 0,077405456 |
| Второй запуск E: | 0,061199051 | 0,060644476 | 0,053220199 | 0,081519489 | 0,05061008 | 0,011297449 | 0,073483737 |
| Третий запуск E: | 0,061317691 | 0,04193333 | 0,041744632 | 0,02500686 | 0,045772733 | 0,00343361 | 0,092388698 |
|  |  |  |  |  |  |  |  |
| Первый запуск Время(мс): | 9769 | 9750 | 1765 | 1484 | 1984 | 1921 | 6687 |
| Второй запуск Время(мс): | 9843 | 9812 | 1812 | 1500 | 2000 | 1921 | 6921 |
| Третий запуск Время(мс): | 9773 | 9734 | 1812 | 1531 | 2000 | 1906 | 6734 |

Таблица 2 - Статистика для функции Химмельблау.

На данной таблице мы можем увидеть, что самым быстрым оказался ТИП-2 5x60, а самым точным оказался ТИП-3 5x60. В данном примере ТИП-0 не показывает эффективности ввиду того, что функция выполняется мгновенно за O(1).

Статистика для двухмерного параболоида. Чем краснее, тем больше значение, чем синее, тем меньше, время - дельта в миллисекундах от начала работы алгоритма до конца 12 теста, E - среднеквадратичное отклонение решения от оптимума функции (табл. 3).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Тип Алгоритма: | Классический генетический алгоритм | Модификация ТИП-0 | Модификация ТИП-2 | Модификация ТИП-2 | Модификация ТИП-3 | Модификация ТИП-3 | Случайный подбор (Метод Монте-Карло) |
| Параметры задаются: | Для Сервера | Для Сервера | Для каждого клиента | Для каждого клиента | Для каждого клиента | Для каждого клиента | - |
| Количество индивидов: | 60 | 60 | 60 | 5 | 60 | 5 | - |
| Количество генов: | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| Режим мутации: | WEAK | WEAK | WEAK | WEAK | WEAK | WEAK | - |
| Количество поколений: | 60 | 60 | 5 | 60 | 5 | 60 | - |
| Количество машин в сети: | - | 12 | 12 | 12 | 12 | 12 | - |
| Количество тестов для E: | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| Количество вычислений целевой функции: | 3600 | 3600 | 3600 | 3600 | 3600 | 3600 | 3600 |
| Первый запуск E: | 0,018152795 | 0,02554495 | 0,020501846 | 0,002381636 | 0,010302005 | 0,001139132 | 0,037637033 |
| Второй запуск E: | 0,001957503 | 0,001970359 | 0,032372994 | 0,002892419 | 0,016675142 | 0,001185426 | 0,047323848 |
| Третий запуск E: | 0,013809952 | 0,001130313 | 0,026190101 | 0,00396748 | 0,011612958 | 0,001145028 | 0,039558625 |
|  |  |  |  |  |  |  |  |
| Первый запуск Время(мс): | 9811 | 9879 | 1832 | 1542 | 1994 | 1938 | 6843 |
| Второй запуск Время(мс): | 9683 | 9829 | 1823 | 1517 | 1989 | 1958 | 6800 |
| Третий запуск Время(мс): | 9645 | 9811 | 1846 | 1536 | 2003 | 1946 | 6796 |

Таблица 3 - Статистика для функции .

На данной таблице мы можем увидеть, что самым быстрым оказался ТИП-2 5x60, а самым точным оказался ТИП-3 5x60, вторым по точности идет ТИП-2 5x60. В данном примере ТИП-0 не показывает эффективности ввиду того, что функция выполняется мгновенно за O(1).

Статистика для функции с имитацией сложных вычислений. Чем краснее, тем больше, чем синее, тем меньше, время - дельта в миллисекундах от начала работы алгоритма до конца 12 теста, E - среднеквадратичное отклонение решения от оптимума функции (табл. 4).

|  |  |  |  |
| --- | --- | --- | --- |
| Тип Алгоритма: | Классический генетический алгоритм | Модификация ТИП-0 | Модификация ТИП-3 |
| Параметры задаются: | Для Сервера | Для Сервера | Для каждого клиента |
| Количество индивидов: | 60 | 60 | 5 |
| Количество генов: | 200 | 200 | 200 |
| Режим мутации: | WEAK | WEAK | WEAK |
| Количество поколений: | 60 | 60 | 60 |
| Количество машин в сети: | - | 12 | 12 |
| Количество тестов для E: | 12 | 12 | 12 |
| Количество вычислений целевой функции: | 3600 | 3600 | 3600 |
| Первый запуск E: | 27,72333554 | 27,95830229 | 43,00096898 |
|  |  |  |  |
| Первый запуск Время(мс): | 4805812 | 400312 | 400312 |

Таблица 4 - Статистика для функции тестовой функции с имитацией сложных вычислений.

Данная таблица приведена лишь для демонстрации разницы в работе с быстрой и медленной функцией. В ней мы видим, что классический алгоритм работает в 12 раз медленнее чем ТИП-0 и ТИП-3.

# Выводы

В тестах наиболее эффективным по значению ошибки себя показала Модификация ТИП-3 с 5 индивидами на 60 поколений. Также по таблицам видно, что в Модификациях ТИП-2 и ТИП-3 гораздо эффективнее уменьшать популяцию, нежели количество итераций(поколений). С точки зрения скорости работы, самой быстрой оказалась модификация ТИП-2, но разница между ТИП-3 и ТИП-2 в плане времени минимальна, а точность у ТИП-3 заметно выше. Подводя итог, можно сделать вывод о том, что самой удачной из модификаций оказалась модификация ТИП-3 с делением популяции.

Ссылка на репозиторий автора с кодом всех программ: <https://github.com/Formak21/GeneticAlgorithm> (Ветвь данного проекта называется RE\_Python)

# 

# Список используемой литературы

1. Джонс, М. Т. Программирование искусственного интеллекта в приложениях [Текст] / М. Тим Джонс. – М.: ДМК Пресс, 2006. – 312с
2. Cooperative and Adaptive Algorithms [Internet] <https://uwaterloo.ca/scholar/sites/ca.scholar/files/ahilal/files/lecture-6-1.pdf>
3. Данжу Джульен Путь Python. Черный пояс по разработке, масштабированию, тестированию и развертыванию.[Текст] -СПб.: Питер, 2020. -256 с.: ил. — (Серия «Библиотека программиста»).
4. Мэтиз Эрик Изучаем Python. Программирование игр, визуализация данных, веб-приложения. — СПб.: Питер, 2017. — 496 с.: ил. — (Серия «Библиотека программиста»).
5. Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 [3] с.