



МИНОБРНАУКИ РОССИИ
федеральное государственное образовательное учреждение
высшего образования
**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ
УНИВЕРСИТЕТ»**
(СПбГЭУ)

Факультет информатики и прикладной математики
 Кафедра прикладной математики и экономико-математических методов

КУРСОВАЯ РАБОТА
по дисциплине:
«Системы компьютерной математики»

Тема: «Реализация алгоритма для распознавания цифр на изображении»

Направление 01.03.02 «Прикладная математика и информатика»

Направленность «Прикладная математика и информатика в экономике и управлении»

Обучающийся Москвитин Марк Александрович

Группа ПМ-2401

Подпись _____

Проверил: Фридман Григорий Морицович

Должность: профессор, д.т.н

Оценка _____

Дата: «__» ____ 202_ г.

Подпись: _____

Санкт-Петербург
2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1. ЛИНЕЙНОЕ ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ	4
1.1. Генетические алгоритмы	4
1.2. Структура алгоритмов	4
1.3. Линейное генетическое программирование	5
1.4. Входные и получаемы данные	5
1.5. Функция приспособления	6
1.6. Метод отбора	6
1.7. Отбор лучших	7
1.8. Репродуктивная функция	7
1.9. Функция смешивания	7
1.10. Макромутация	7
1.11. Микромутация	7
1.12. Проблема результатов	7
1.13. Программа	8
1.14. Окончание работы алгоритма	9
2. РЕАЛИЗАЦИЯ АЛГОРИТМА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ WOLFRAM MATHEMATICA	10
2.1. Набор входных данных	10
2.2. Хранение информации	11
2.3. Функция-менеджер	12
2.4. Реализация функции приспособления и других операций	13
2.5. Настройка параметров	14
2.6. Анализ результатов функции приспособления	15
2.7. Проверка инструкции	18
ЗАКЛЮЧЕНИЕ	20
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	21
ПРИЛОЖЕНИЕ А	22

ВВЕДЕНИЕ

Сегодня нельзя представить свою жизнь без использования телефона для того, чтобы отсканировать текст или номер для отправки перевода денег. Однако этой инновации предшествовали довольно долгие поиски оптимального способа и бесчисленные попытки обучить машину узнавать любые данные.

Во второй половине XX века активно развивался технология оптического распознавания. Скоро на смену ей пришли машинное обучение и искусственный интеллект.

Одним из примеров алгоритмов является линейное генетическое программирование, построенное на идеи эволюционного моделирования. Эта модель активно развивалась в 1990-х и 2000-х годах.

Программа получает на вход данные об изображении и случайный набор программ, который необходимо адаптировать под решение конкретной задачи, отсеивая худшие варианты и изменяя лучшие для достижения наибольшей эффективности. В качестве тестируемых объектов исследования будут использоваться изображения наборы цифр размером 7x7. Соответственно, сами предметы исследования – это цифры.

Целью данной курсовой работы является реализация алгоритма для распознавания цифр на изображении.

Задачи курсовой работы:

- 1) знакомство с линейным генетическим программированием;
- 2) реализация алгоритма для распознавания цифр;
- 3) анализ влияния поколений на результат функции приспособления.

1. ЛИНЕЙНОЕ ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

1.1. Генетические алгоритмы

Как уже было сказано выше, генетические алгоритмы – это стандартный пример эволюционного моделирования. Впервые идею эволюции предложил И. Рехенберг в 1960-е годах, после чего ее подхватили и другие исследователи. Автором из основных трудов именно в генетическом программировании считается Джон Коза – он сформулировал сам термин и развил его реализацию для выполнения различных задач в 1990-е годы [2]. Впоследствии появилось множество различных способов выполнения алгоритмов: например, то же линейное генетическое программирование или программа на древовидной структуре. Хоть второй вариант встречается чаще первого, мы увидим, что он, напротив, эффективнее конкретно в нашей задаче.

1.2. Структура алгоритмов

Такое программирование строится на довольно простой идеи эволюции. Изначально случайным образом создается некая популяция. У каждого ее члена есть свои характеристики. Таким образом, сейчас речь идет о неком первом поколении.

Чтобы определить его соответствие стремлению получить итоговый результат, необходимо придумать функцию приспособления каждого из экземпляров. Одним из наиболее частых вариантов реализации является проверка на количество ошибок. С помощью такого подхода можно определить, какой из вариантов на данном этапе подходит лучше всего, а также ранжировать список для дальнейших действий.

Следующий шаг – применение к представителям популяции одной из операции для получения нового поколения. Выбор происходит либо случайным образом, либо на основе результатов функции приспособления.

Во-первых, чаще всего, самые лучшие варианты проходят в следующее поколение без изменений. Это помогает сохранить наилучшие способы, даже если результат от других действий не принесет никакой пользы.

Во-вторых, пары случайных объектов участвуют в обмене данными. В этом случае отлично подходит аналогия с ДНК и хромосомами, а именно генетической рекомбинации. Данные смешиваются, составляя нового представителя поколения, а предыдущие варианты больше не учитывают.

В-третьих, неудачные экземпляры подвергаются мутации: заменяя некоторые их части на другие.

Таким образом, всевозможные комбинации действий приведут к созданию нового поколения. Остается заново применить функцию приспособления и повторять всю процедуру до тех пор, пока не найдется решение поставленной задачи.

1.3. Линейный генетический алгоритм

В чем же отличие именно этой реализации эволюционной модели? Линейное генетическое программирование представляет каждое поколение как программу, составленную из набора различных инструкций.

В константе хранятся значение двух типов: входящие и полученные данные. Во входящие данные, помимо нужных для вычисления показателей, могут входить и случайные константы. Второй тип нужен для нахождения ответа и вывода результата.

Еще одним отличием является наличие двух вариантов мутации: макромутация и микромутации. В то время как микромутации связаны со стандартными изменениями, макромутации могут полностью удалять все инструкции из одной программы и составлять новые.

1.4. Входные и получаемые данные

Итак, за основу будут взяты данные статьи Genetic Programming for Image Recognition: An LGP Approach [1].

Как уже было отмечено раньше, на вход программа получает определенный объем информации. Для изучения цифр вполне могут сгодиться значения пикселей в бинарном изображении.

Так как потребуется определять, какой именно объект получила на вход программа, то в получаемых данных будет храниться сразу несколько значений. После подсчета инструкций нужно лишь будет посмотреть на то, какой из элементов хранит наибольшее значение. Таким и будет предполагаемый присвоенный тип от этой программы.

1.5. Функция приспособления

Довольно часто в такого рода задачах используют коэффициент ошибок, обычно показывающий объем неправильного результата по отношению к общему количеству. Однако дело в том, что разные ситуации могут показывать одинаковое значение, что не позволяет сделать выводы.

Представим ситуацию: программа классифицировала 13 из 15 объектов. В первом случае двое объектов из третьего класса попали во второй. Во втором – двое объектов из третьего класса попали по одному в первый и второй.

В данном случае можно воспользоваться другим вариантом функции приспособления: подсчет штрафа для каждой из программ на основе количества неправильных ответов.

$$f = \sum_c \sum_{i=0}^{M_c} \alpha^{\beta i}, \quad (1)$$

где $\sum_{i=0}^{M_c} \alpha^{\beta i}$ – проходит по каждому результату выполнения определенной

программы, а α и β – это константы, причем $\alpha > 1$, а $\beta > 0$, чтобы штраф за ошибку возрастил с каждым разом.

1.6. Метод отбора

Чтобы случайно выбирать программы для действий, все равно требуется какой-нибудь метод. Тут будет использоваться турнирный отбор: доставая несколько программ, выбирается та, у которой лучше значение функции приспособления.

1.7. Отбор лучших

Несколько лучших экземпляров без изменений проходят в следующее поколение для достижения наибольшей эффективности.

1.8. Репродуктивная функция

Здесь ничего сложного не будет: просто перенос образцов в следующее поколение без каких-либо изменений.

1.9. Функция смещивания

Аналогично устроена достаточно просто: берется две программы, в первой из них выбирается случайный набор инструкций. Затем случайно определяется аналогичный отрезок любой длины во второй программе. После чего два промежутка просто переставляют местами.

1.10. Макромутация

Случайно выбирается инструкция программы, после чего ее содержимое полностью стирается. В новом поколении для программы генерируется другое указание.

1.11. Микромутация

В этом случае точно также выбирается случайная инструкция программы, однако теперь заменяется лишь одна ее часть: либо функция, либо индекс получаемых или входных данных.

1.12. Проблема результатов

Дело в том, что несколько выходных данных могут иметь одинаковое значение. Конкретно здесь будем проверять, принадлежит ли нужный номер класса списку выбранных наибольших значений. Однако можно подобрать и другой способ.

1.13. Программа

Итак, каждая программа состоит из инструкций. Инструкция хранит в себе три типа данных: функция, выходной индекс и используемые индексы. Таким образом, выходные и входные данные хранятся в качестве единого списка, к которому каждый раз обращается алгоритм.

В этой реализации будут использоваться следующие функции: сложение, умножение, деление, вычитание и условный оператор if. При делении на ноль программа должна возвращать 0. Если первый аргумент при функции if меньше второго, то следующая инструкция в программе выполняется, в противном случае она пропускается.

На рисунке 1 представлен пример программы. Пусть r - это список, где хранятся данные. Индексам от 0 до 4 принадлежат входные данные, от 5 до 8 – результаты для определения типа.

```
r[5]= r[1] + r[2]
r[7] = r[1] / r[4]
if r[3] < r[4]:
    r[6] = r[0]+r[2]
    r[8] = r[1]+r[3]
```

Рисунок 1 – Пример инструкций программы

Если представить всю программу в виде графа, то получится не дерево, а ориентированный граф, что и показано на рисунке 2.

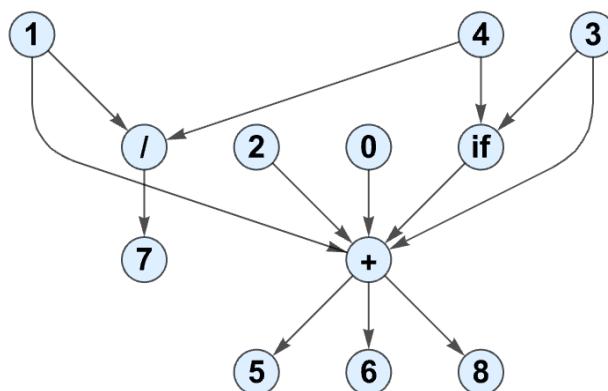


Рисунок 2 - Интерпретация программы

Обычно количество инструкций в программе и размеры списка задаются отдельно.

1.14. Окончание работы алгоритма

Важно отдельно обозначить момент, когда программа должна остановить свою работу. Очевидно, это либо ситуация, когда функция приспособления назначила штраф, равный нулю. Другой вариант – изначально установить возможность остановки программы после достижения определенного номера поколения.

2. РЕАЛИЗАЦИЯ АЛГОРИТМА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ WOLFRAM MATHEMATICA [3]

2.1. Набор входных данных

В качестве обучающего набора для алгоритма можно взять пиксельные изображения цифр в формате 7x7.

Кроме того, для практики будет использоваться полноценная картинка, прикрепленная в качестве рисунка 3. Оно подвергнется модификациям: сначала его преобразуют в режим Grayscale, затем определят пиксели, отличные от фона, отделят их и разбоят в разные элементы, анализируя пустое место между. Все пиксели приравняются к 0 и 1, а затем уже их можно будет использовать для обучения модели.



Рисунок 3 - Обучающее изображение

Для увеличения эффективности обучения на наборе из 1000 цифр, большая часть из них проходит случайное добавление шума, где часть пикселей каждой картинки меняют цвет с 1 на 0 или наоборот.

Реализация этого замещения представлена на рисунке 4. Установив в процентном соотношении, какое количество пикселей будет изменено, необходимо передать их список в функцию. Там будет выбрано случайное

значение от 1 до 49, а затем соответствующая позиция в матрице будет инвертирована.

```
createNumbers[digits_List, percent_Real] :=
Module [{digit = digits},
Table[
(digit[[l, Quotient[# - 1, 7] + 1, Mod[# - 1, 7] + 1]] =
1 - digit[[l, Quotient[# - 1, 7] + 1, Mod[# - 1, 7] + 1]]) & /@
RandomSample[Range[1, 49], Round[percent * 49]] ,
{l, 1, Length[digit]}];
digit
]
```

Рисунок 4 - Добавление случайного шума

В свою очередь, оставшиеся экземпляры размываются при помощи встроенной функции GausianFilter, в основе которой лежит работа с матрицей Гаусса. Визуальный пример входных данных после обработки пикселей – на рисунке 5.

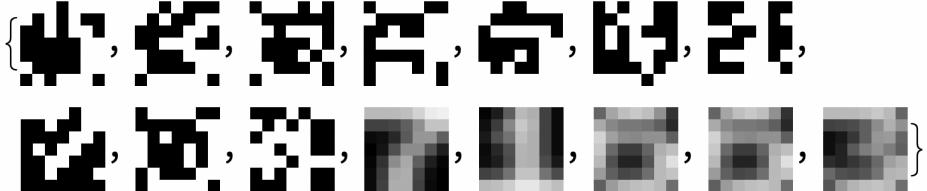
```
answers = {5, 3, 6, 9, 5, 1, 1, 7, 0, 4, 7, 1, 6, 6, 3}
Out[1501]=
{
```

Рисунок 5 - Примеры модификации

Как можно заметить, некоторые из цифр трудно различимы даже для человеческого глаза. Даже несмотря на такой малый процент инвертации.

2.2. Хранение информации

Основные объекты хранения - это наборы входных и выходных данных, а также популяция. Все объекты можно хранить в виде списков информации.

Во-первых, наборы входных и выходных данных будут храниться в одном списке. В начале программы создается список, куда помещаем 49 входных пикселей, константу и 10 выходных регистров (10 классов от 0 до 9). Получается объект из 60 элементов. Так как изначально на вход передается список цифр, то их преобразованные наборы также находятся в одном списке.

Что касается инструкций, их примеры хранения отображены на рисунке 6. Каждая инструкция представляет собой список из 4 элементов, а именно:

- номер выходного регистра в наборе данных цифры (от 51 до 60);
- название применяемой функции (сложение, вычитание, умножение, деление, оператор IF);
- номер используемого регистра (от 1 до 60);
- номер используемого регистра (от 1 до 60).

Соответственно, при подсчете инструкции, эти элементы применяются в нужном порядке, чтобы сформировать какое-либо действие.

{56, Plus, 22, 16}
{55, Subtract, 18, 5}
{55, Times, 21, 10}
{55, Divide, 33, 53}
{55, If, 11, 29}

Рисунок 6 - Примеры инструкций

Набор инструкций образует программу. В каждом поколении требуется определенное количество программы. Таким образом, все программы мы храним в одном большом многоуровневом списке.

2.3. Функция-менеджер

Для стабильной работы требуется стабильная функция-менеджер (см. Приложение А), которая будет вызывать остальные функции, в рамках цикла хранить поколение и обрабатывать его, а затем выводить результат после

достижения определенного числа поколений или нахождений идеального результата.

На вход передается большое количество параметров: terminal (список 49-пиксельных представлений цифр, answers (список ответов), destinationamount (общее количество входных, выходных регистров и констант), uptoconstant (количество входных регистров и констант), generetaionamount (количество поколений), populationsize (размер одного поколения), maxprogramlength (длина поколения), reproductionrate (шанс применения функции репродукции), crossoverrate (шанс применения функции смещения), macromutationrate (шанс применения макромутации), micromutationrate (шанс применения микромутации), alpha (константа для функции приспособления), beta (константа для функции приспособления), getbest (количество лучших программ), tournamentamount (количество программ, участвующих в турнире).

Затем набор входных и выходных данных преобразуется в единый список, генерируется первое поколение, после чего запускается цикл. Сохраняя результаты функции приспособления в специальный словарь, проходит проверка на достижение последнего номера поколения или на отсутствие ошибок. При необходимости создается другой словарь, с присвоением порядковому номеру в списке программ самой программы. Далее один из словарей записывается в файл для логирования.

На основе получившихся результатов, лучшие программы сразу проходят в следующие поколения, а для остальных с определенным шансом выбора назначается какое-то действие, покуда общее количество программ в новом поколении не достигнет нужного размера популяции. Это остановит цикл, запишет новое поколение и заново запустит основную часть – вычисление результатов, применение функции приспособления и так далее.

2.4. Реализация функции приспособления и других операций

В данной программе все функции были реализованы стандартными методами.

В функцию приспособления с рисунка 7 на вход подается список ответ, подсчитанные результаты и необходимые константы. Образуется список, где любой элемент – перечисление итогов программы для каждой инструкции. Затем индивидуально подсчитывается количество ошибок, после чего применяется формула функции приспособления и определяются ее показатели.

```
(*фитнес-функция*)
fitnessFunction[answers_List, results_List, alpha_Real, beta_Real,
  destinationamount_Integer, uptoconstant_Integer] :=
Module[{error, count},
  count = (countErrors[answers, #, destinationamount, uptoconstant]) & /@ results;
  Table[i → Total[(alpha * If[# > 0, Sum[beta^i, {i, 0, # - 1}], 0]) & /@ count[[i]], {i, 1, Length[count]}]]]

(*фитнес-функция, помощник*)
countErrors[answers_List, result_List, destinationamount_Integer,
  uptoconstant_Integer] := Module[{error}, error = Table[0, {i, 1, Length[answers]}];
Table[
  If[
    MemberQ[PositionLargest[
      result[[i]][uptoconstant + 1 ;; destinationamount + uptoconstant]],
      answers[[i]]] == False, error[[i]] = error[[i]] + 1], {i, 1, Length[result]}]; error];
```

Рисунок 7 - Функция приспособления

Функция смешивания, микромутации, макромутации, репродукции не менее просты. На вход они получают необходимую программу и константы. После чего либо выбирают еще одну программу для скрещивания частей, либо запускают свою операцию (полное или частичное изменение одной инструкции).

2.5. Настройка параметров

Следующий шаг перед запуском программы – установление оптимальных значений. Несмотря на то, что, как может показаться, здесь присутствует достаточная вариативность, на самом деле это не так.

Поведение программы достаточно предсказуемо, и устанавливать слишком большие значения не нужно. Это лишь увеличит общую продолжительность и нагрузит большее количество ресурсов компьютера.

Таблица 1 - Параметры запуска алгоритма

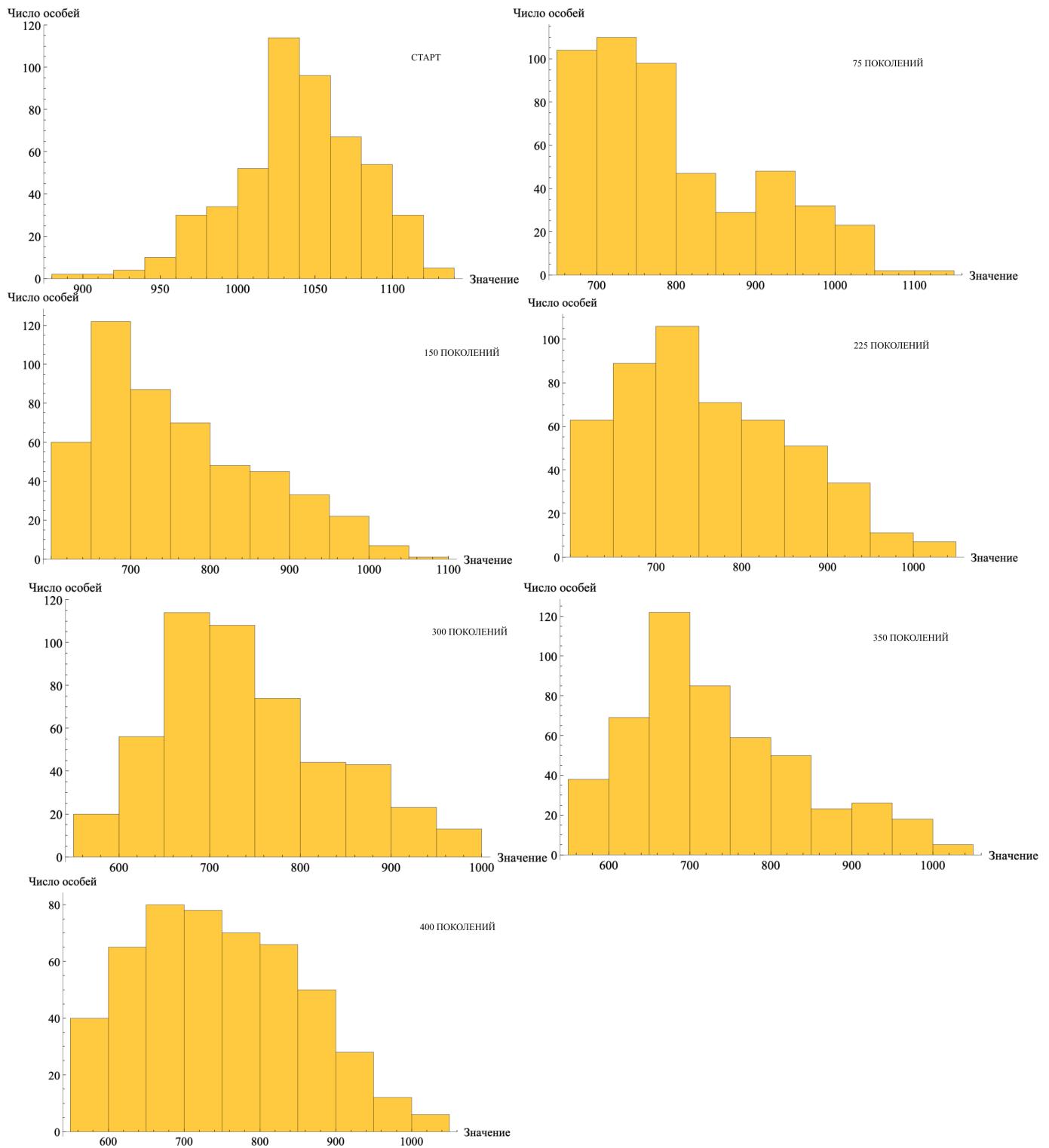
Общее количество регистров	60
Количество регистров входных и константных	50
Количество поколений	400
Размер поколения	500
Максимальная длина программы	50
Шанс репродукции	10 %
Шанс смешения	30 %
Шанс макромутации	30 %
Шанс микромутации	30 %
α	1.15
β	0.08
Количество лучших программ, переходящих в новое поколение	5
Количество участников турнирного отбора	3

Каждый шаг будут фиксироваться показатели функции приспособления, чтобы впоследствии проанализировать их.

2.6. Анализ результатов функции приспособления

По итогам работы алгоритма мы собрали его показатели. На рисунках 8 и 9 представлены изменения показателей функции приспособления. Можно заметить, что с каждым разом все больше и больше программ постепенно стремятся к 0 – это правильная тенденция. Чем выше значение, тем чаще ошибается алгоритм.

Кроме того, также видно, что максимальное значение постепенно убывает, а с ним уменьшается и количество программ, получающих максимальное значение. Хотя, конечно, скорость этого изменения далеко не высока, сама по себе тенденция положительна и тоже подчеркивает точность выполнения поставленных задач.



Рисунки 8, 9 - Гистограммы распределения значения

Не менее важно проанализировать и сами значения на рисунках 10, 11 и 12. Это поможет понять, что действительно большие значения параметров не приводят к сильным изменениям и не всегда достигают необходимой точности ответа.

Начнем с наименьших в каждом поколении.

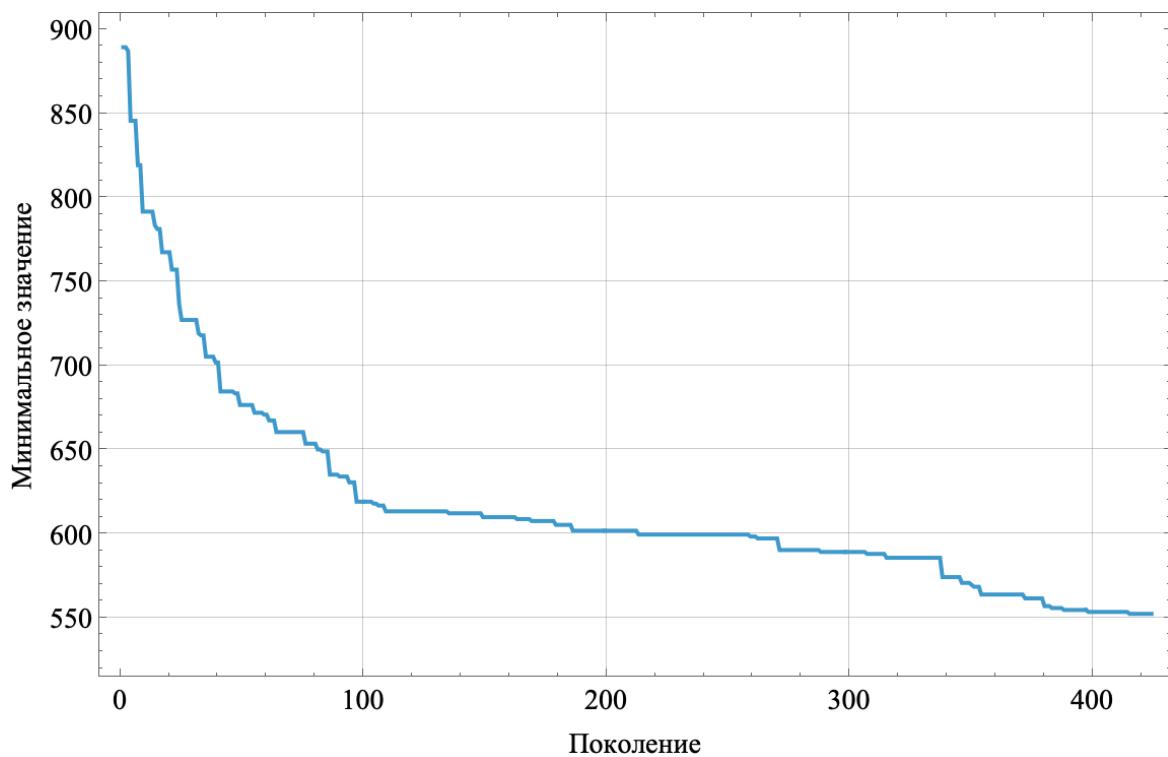


Рисунок 10 - Минимальное значение

Среднее значение изменяется в таком же темпе.

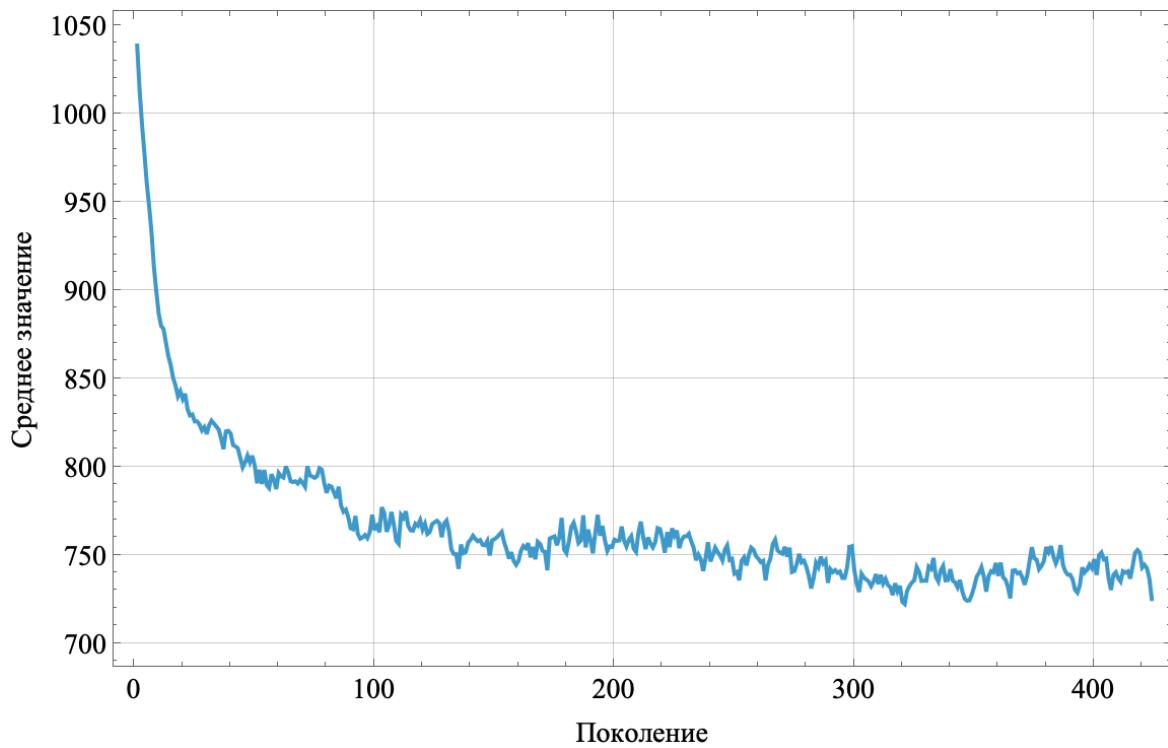


Рисунок 11 - Среднее значение

А максимальное значение скачет, так как все операции приводят к изменению некоторых инструкций, и далеко не всегда это означает положительный эффект.

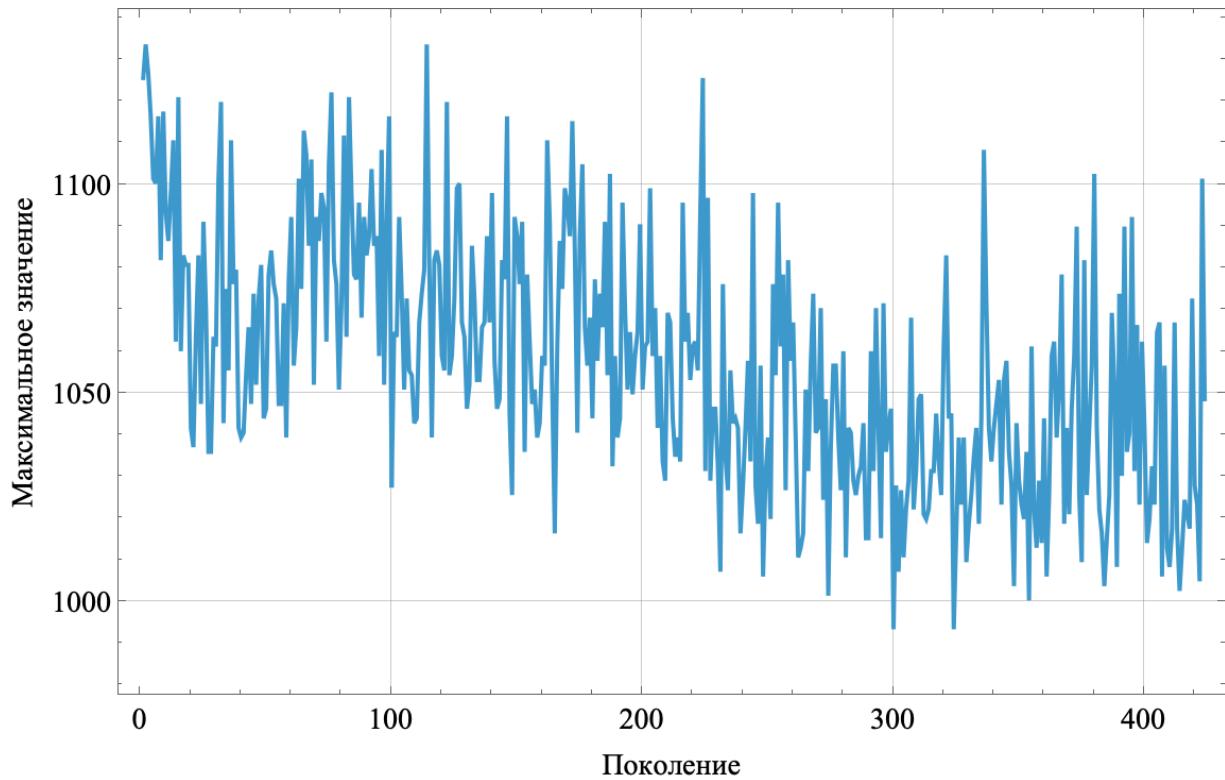


Рисунок 12 - Максимальное значение

Теперь осталось проверить лучшую инструкцию через случайную выборку.

2.7. Проверка инструкции

Возьмем и создадим входные данные таким же способом, только в меньших значениях. Получим примерно 40-50% точности: наиболее проблемными местами стали 0, 2, 4, 6.

На рисунке 13 сверху представлены примеры ответов, под ними – решения алгоритма.

```

Out[1325]=
{5, 4, 1, 7, 5, 6, 3, 7, 7, 8, 3, 3, 2, 4, 0}

Out[1326]=
{{{5}, {7, 8, 10}, {1, 2, 6, 7, 9, 10}, {7}, {5}, {7},
{3, 5, 8}, {7}, {7}, {3, 6, 8}, {3}, {3}, {1}, {3}, {3}}}

Out[1390]=
{5, 8, 7, 6, 5, 4, 3, 5, 8, 4, 6, 0, 3, 0, 2}

Out[1391]=
{{{5}, {6, 8}, {7}, {6}, {2, 3, 5, 7, 8}, {7},
{2, 3, 8}, {5}, {8}, {2, 4, 5}, {3}, {3}, {3}, {1}}}

```

Рисунок 13 - Результаты алгоритма

Таким образом, хоть и с большой погрешностью и перспективой на улучшение, была выработана программа, способная определить примерный результат.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы были изучены линейные генетические алгоритмы, нюансы их работы и влияние параметров на эффективность результатов. Чем выше номер поколения, тем медленнее будет улучшаться значения функции приспособления.

Кроме этого, этот способ действительно подходит для создания алгоритма распознавания цифр. Несмотря на погрешность, модель вычисляла и верные значения. Наилучший результат зависит от размера обучающей выборки.

Алгоритму не страшны шумы или размытие: цифры распознаются в любых условиях.

Подводя итог проделанной работе, можно отметить следующие выполненные задачи:

- 1) знакомство с линейным генетическим программированием;
- 2) реализация алгоритма для распознания цифр;
- 3) анализ влияния количества поколений на результат функции приспособления.

Цель была достигнута с использованием технологий Wolfram Mathematica [3].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Fogelberg C.G., Zhang M. Genetic programming for image recognition: An LGP approach // Applications of Evolutionary Computing. EvoWorkshops 2007: Proceedings. – Berlin: Springer, 2007. – С. 340–350.
2. Koza, J. R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. – Cambridge, MA: MIT Press, 1992. – 840 с.
3. Wolfram Mathematica: Version 14.1 [Электронный ресурс]. – Champaign, IL: Wolfram Research, Inc., 2025. – Режим доступа: <https://www.wolfram.com/mathematica/>, свободный. – Загл. с экрана. – Яз. англ.

ПРИЛОЖЕНИЕ А

ФУНКЦИЯ-МЕНЕДЖЕР ГЕНЕТИЧЕСКОГО АЛГОРИТМА

```
recognitionLGP[terminals_List, answers_List, destinationamount_Integer, uptoconstant_Integer, generetaionamount_Integer,
  populationsize_Integer, maxprogramlength_Integer, reproductionrate_Real, crossoverrate_Real, macromutationrate_Real,
  micromutationrate_Real, alpha_Real, beta_Real, getbest_Integer, tournamentamount_Integer] :=
Module[{terminationset, instruction = {}, result, newinstruction = List[], listend, resprint, nums},
  terminationset = setTerminal[#, destinationamount, uptoconstant] & /@ terminals;
  instruction = programCreate[maxprogramlength, populationsize, destinationamount, uptoconstant];
  nums = 1;
  result = Association[{0 \[Rule] 0}];
  Do[
    Print[{nums, Min[Values[result]]}];
    result = fitnessFunction[answers, countFunction[instruction, terminationset], alpha, beta, destinationamount, uptoconstant];
    result = Association[result];
    If[
      MemberQ[Values[result], 0.] || nums == generetaionamount, listend = {};
      If[result[[#]] == 0. || result[[#]] == Min[Values[result]], listend = Join[listend, {result[[#]], # \[Rule] instruction[[#]]}] & /@ Keys[result];
      Return[listend];
      newinstruction = reproductionBest[result, instruction, getbest];
      While[Length[newinstruction] < populationsize - getbest,
        newinstruction =
          Join[newinstruction,
            RandomChoice[{reproductionrate, crossoverrate, macromutationrate, micromutationrate} \[Rule]
              {reproductionOperation, crossoverOperation, macromutationOperation, micromutationOperation}][
              instruction[[tournamentChoose[result, tournamentamount]], result, instruction, destinationamount, uptoconstant,
              maxprogramlength, tournamentamount]]];
      If[Length[newinstruction] == populationsize, instruction = newinstruction, instruction = newinstruction[[1 ;; populationsize - getbest]]];
      {nums, generetaionamount}
    ]; listend
  ]
]
```