**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра технологий программирования**

**ПРОЕКТИРОВАНИЕ БИБЛИОТЕКИ ДЛЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ**

Отчет по преддипломной практике

|  |
| --- |
| Доскоча Романа Дмитриевича |
| студента 4 курса, специальность  «прикладная информатика» |
| Научный руководитель: |
| кандидат технических наук, |
| доцент И.С. Войтешенко |

|  |  |
| --- | --- |
|  |  |

Минск, 2023 РЕФЕРАТ

Курсовой проект, 42 c., 33 рис.

**Ключевые слова:** Генетический алгоритм, Unity, Генотип, RL

**Объект исследования –** В качестве предмета исследования выбрано исследование применения генетического алгоритма на практике и зависимость его от разных модификаций подходов в выборе операторов алгоритма.

**Цели работы –** изучение теории генетического алгоритма и исследованиевозможностей его применения, а также проектирование библиотеки.

**Методы исследования –** изучение литературы по теории, так же изучение общеизвестных библиотек генетического алгоритма.

**Результатами являются –** структурная схема библиотека с удобным API для использования возможностей генетического алгоритма.

**Область применения –** Задачи оптимизации, Искусственный интеллект.

РЭФЕРАТ

Курсавы праект, 42 с., 33 рыс.

**Ключавыя словы**: Генетычны алгарытм, Unity, Генатып, RL

**Аб'ект даследавання** **–** У якасці прадмета даследавання абраны даследаванне прымянення генетычнага алгарытму на практыцы і залежнасць яго ад розных мадыфікацый падыходаў у выбары аператараў алгарытму.

**Мэты працы** **–** вывучэнне тэорыі генетычнага алгарытму і даследаванне магчымасцей яго прымянення, а таксама праектаванне бібліятэкі.

**Метады даследавання** **–** вывучэнне літаратуры па тэорыі, гэтак жа вывучэнне агульнавядомых бібліятэк генетычнага алгарытму.

**Вынікамі з'яўляюцца** **–** структурная схема бібліятэкі з зручным API для выкарыстання магчымасцяў генетычнага алгарытму.

**Вобласць ужывання** **–** Задачы аптымізацыі, Штучны інтэлект.

ESSAY

Course project, 42 p., 33 illustrations.

**Keywords**: Genetic Algorithm, Unity, Genotype, RL

**Object of research –** findthe optimal application of the genetic algorithm in practice and its dependence on various modifications of approaches in the choice of algorithm operators.

**Purpose –** study the theory of the genetic algorithm and explore the possibilities of its application, as well as designing the class library.

**Methods of research** **–** study of literature on theory, as well as the study of well-known libraries of the genetic algorithm.

**The results are** **–** structured diagram of library with a convenient API for using the capabilities of the genetic algorithm.

**Scope** **–** Optimization problems, Artificial intelligence.

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

|  |  |
| --- | --- |
| ГА | Генетический алгоритм – эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора |
| TDD | Разработка через тестирование (Test Driven Development) – это методология разработки программного обеспечения, которая основана на подходе: изначально пишутся тесты затем программный код для реализации нужного поведения. |
| RL | Reinforcement learning – обучение с подкреплением. Один из способов машинного обучения, в ходе которого испытуемая система обучается, взаимодействуя с некоторой окружающей средой. |
| Unity 3D | Межплатформенная среда разработки компьютерных игр. позволяет создавать приложения, работающие на более чем 25 различных платформах |
| Python DEAP | Distributed Evolutionary Algorithms in Python – это эволюционная вычислительная среда для быстрого прототипирования, разработана специально для создания реализаций ГА на Python. |
|  |  |

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc129959140)

[**Глава 1 ОСНОВЫ ГЕНЕТИЧЕСКОГО АЛГОРИТМА** 5](#_Toc129959141)

[1.1 История возникновения генетического алгоритма 5](#_Toc129959142)

[1.2 Что такое генетические алгоритмы 6](#_Toc129959143)

[1.3 Основные понятия и определения 7](#_Toc129959144)

[1.4 Примеры использования генетических алгоритмов. 9](#_Toc129959145)

[1.5 Преимущества и недостатки генетических алгоритмов 11](#_Toc129959146)

[1.6 Вывод 13](#_Toc129959147)

[**Глава 2 ИССЛЕДОВАНИЕ МНОГООБРАЗИЯ ПОДХОДОВ ГЕНЕТИЧЕСКОГО АЛГОРИТМА** 14](#_Toc129959148)

[2.1 Структура генетического алгоритма 14](#_Toc129959149)

[2.2 Операторы отбора 15](#_Toc129959150)

[2.3 Операторы скрещивания 18](#_Toc129959151)

[2.4 Операторы Мутации 20](#_Toc129959152)

[2.5 “Hello World” в мире генетических алгоритмов 22](#_Toc129959153)

[2.6 Эксперименты с параметрами ГА 25](#_Toc129959154)

[2.7 Вывод 28](#_Toc129959155)

[**Глава 3 ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ И РАЗРАБОТКА БИБЛИОТЕКИ** 30](#_Toc129959156)

[3.1 Обучение с подкреплением 30](#_Toc129959157)

[3.2 Приложение Генетического алгоритма в RL 32](#_Toc129959158)

[3.3 Структурная схема библиотеки 35](#_Toc129959159)

[3.4 TDD & Unit Testing 38](#_Toc129959160)

[3.5 Вывод 38](#_Toc129959161)

[**ЗАКЛЮЧЕНИЕ** 40](#_Toc129959162)

[**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ** 41](#_Toc129959163)

[**ПРИЛОЖЕНИЯ** 42](#_Toc129959164)

[*Приложение A* 42](#_Toc129959165)

[*Приложение Б* 43](#_Toc129959166)

# **ВВЕДЕНИЕ**

В последние годы генетические алгоритмы (ГА) стали популярным методом оптимизации благодаря их способности эффективно решать сложные задачи оптимизации в различных областях. Их эвристическая природа, вдохновленная принципами естественного отбора и генетической наследственности, позволяет им исследовать обширные области решений и находить оптимальные или почти оптимальные решения.

В главе 1 «Основы генетического алгоритма» приводится краткое введение в теорию и принципы работы генетических алгоритмов. Рассматриваются также различия между генетическим алгоритмом и традиционными методами, и описываются сценарии, в которых имеет смысл его применения.

В главе 2 «Исследование многообразия подходов генетического алгоритма» рассматривается и описываются шаги алгоритма, так же рассматриваться основные разновидности операторов отбора скрещивания и мутации. В конце главы на примере задачи OneMax и с помощью пакета Python DEAP реализован стандартный подход алгоритма, так же применены разные методы при выборе операторов и гиперпараметров алгоритма. Все сравнения приведены в качестве графиков.

В главе 3 «Практическое применение и разработка библиотеки» описано применение ГА в нейронных сетях в частности на подходе обучения с подкреплением на примере самоуправляемой машины в Unity 3D. Так же описана структура библиотеки генетического алгоритма и ее использование в задаче OneMax. Написаны юнит-тесты для подхода проектирования TDD.

# **Глава 1 ОСНОВЫ ГЕНЕТИЧЕСКОГО АЛГОРИТМА**

Применение генетических алгоритмов в оптимизации и моделировании представляет собой перспективное и актуальное направление. Эти алгоритмы могут решать задачи на графах, компоновки и составления расписаний, а также настройки и обучения искусственных нейронных сетей, и многие другие задачи.

## История возникновения генетического алгоритма

История генетических алгоритмов (ГА) восходит к началу 1960-х годов, когда математик и ученый-компьютерщик Джон Холланд начал исследовать идею использования естественного отбора и генетического наследования в качестве вычислительной парадигмы. Кульминацией работы Холланда над ГА стала публикация его основополагающей книги «Адаптация в естественных и искусственных системах» в 1975 году, в которой изложены основы генетического алгоритма и его потенциального приложения в науке.

В последующие годы исследователи из различных областей начали принимать и расширять концепции ГА, что привело к разработке новых алгоритмов и методов. Одним из примечательных примеров является введение Джоном Коза в 1980-х годах парадигмы генетического программирования, которая применяет генетические алгоритмы к проблеме автоматического синтеза компьютерных программ.

В 1990-е годы наблюдался быстрый рост исследований ГА, что вызвало всплеск интереса со стороны сообществ искусственного интеллекта и машинного обучения. В 1995 году Дэвид Голдберг опубликовал «Генетические алгоритмы в поиске, оптимизации и машинном обучении», в котором был представлен всесторонний обзор области и укреплены позиции ГА как основного метода оптимизации.

С тех пор генетические алгоритмы нашли применение в самых разных областях, включая инженерию, финансы, биоинформатику и робототехнику. Достижения в вычислительной мощности и параллельной обработке также позволили использовать ГА для решения более крупных и сложных задач оптимизации.

В целом, история генетических алгоритмов отражает продолжающийся поиск эффективных и действенных методов оптимизации, которые могут использовать принципы естественного отбора и генетической наследственности.

## Что такое генетические алгоритмы

Прежде чем знакомиться с понятием самого алгоритма, рассмотрим подводящие понятия. Как известно, оптимизационные задачи заключаются в нахождении минимума (максимума) заданной функции. Такую функцию называют целевой. Как правило целевая функция — сложная функция, зависящая от некоторых входных параметров. В оптимизационной задаче требуется найти значения входных параметров, при которых целевая функция достигает минимального (максимального) значения.

Существует целый класс оптимизационных методов. С их помощью можно найти экстремальное значение целевой функции, но не всегда можно быть уверенным, что получено значение принадлежит глобальному экстремуму. Нахождение локального экстремума вместо глобального называется **преждевременной сходимостью**. Помимо проблемы преждевременной сходимости существует другая проблема — время процесса вычислений. Зачастую более точные оптимизационные методы работают очень долго.

Для решения поставленных проблем и проводится поиск новых оптимизационных алгоритмов. Одним из таких алгоритмов можно считать генетический алгоритм.

**Генетические алгоритмы** – это семейство поисковых алгоритмов, идеи которых подсказаны принципами эволюции в природе. Имитируя процессы естественного отбора и воспроизводства, генетические алгоритмы могут находить высококачественные решения задач, включающих поиск, оптимизацию и обучение.

В то же время аналогия с естественным отбором позволяет этим алгоритмам преодолевать некоторые препятствия, встающие на пути традиционных алгоритмов поиска и оптимизации, особенно в задачах с большим числом параметров и сложными математическими представлениями.

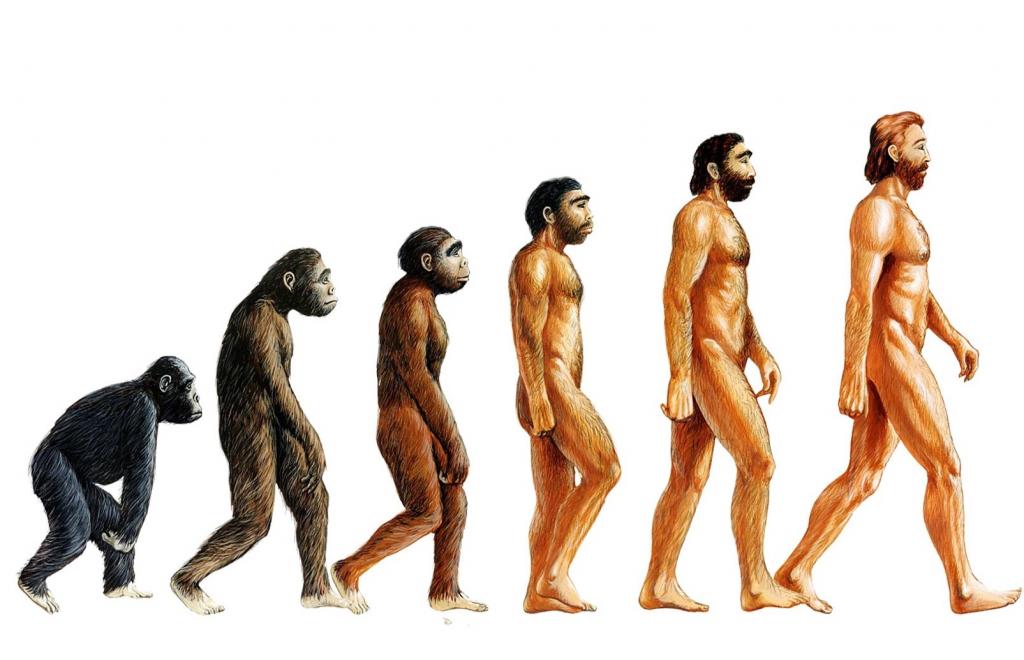


Рисунок 1.1 Эволюция по Дарвину

Генетический алгоритм представляет собой адаптивный поисковый метод, основанный на селекции лучших элементов популяции. Область поиска для генетического алгоритма называется популяцией, элементами которой являются хромосомы. Начальная выборка состоит из случайных хромосом, каждая из которых является потенциальным решением. Качество решения определяется степенью приспособленности каждой хромосомы.

Генетический алгоритм использует адаптивный эвристический поиск, выбирая лучшие решения из популяции с помощью операций селекции, скрещивания и мутации. Создаются новые особи - потомки, и более приспособленные хромосомы переходят в следующее поколение. Процесс повторяется до тех пор, пока не будет получено наилучшее решение. С каждой итерацией средняя приспособленность популяции увеличивается, поэтому большее число итераций приводит к лучшему результату.

## Основные понятия и определения

Опишем основные понятия теории генетического алгоритма.

***Генотип.*** В контексте генетических алгоритмов генотип относится к генетической информации индивидуума, которая представлена в виде последовательности символов или двоичных цифр (битов). Он содержит инструкции, которые определяют характеристики или черты организма или решения в конкретной предметной области. Генотип особи представляет ее потенциал адаптации и выживания в данной среде. В генетическом алгоритме генотип подвергается таким операциям, как скрещивание и мутация, для создания новых возможных решений, которые затем оцениваются на основе их соответствия рассматриваемой проблеме. Например, хромосому можно представить двоичной строкой, в которой каждый бит соответствует одному гену:

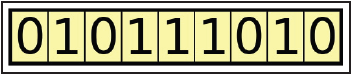


Рисунок 1.2 Простое двоичное кодирование хромосомы

На рисунке выше показан пример такой двоично-кодированной хромосомы, представляющей одного индивидуума.

***Популяция.*** В любой момент времени генетический алгоритм хранит популяцию **индивидуумов** – набор потенциальных решений поставленной задачи. В генетическом алгоритме популяция относится к группе возможных решений, представленных в виде хромосом, которые используются для поиска в пространстве решений оптимального или близкого к оптимальному решения данной проблемы. Начальная популяция генерируется случайным образом и представляет собой отправную точку генетического алгоритма.

Размер популяции является важным параметром, который следует учитывать при разработке генетического алгоритма. Больший размер популяции увеличивает разнообразие решений-кандидатов, но также увеличивает вычислительные затраты на оценку функции пригодности и выполнение операций выбора, скрещивания и мутации. Меньший размер популяции может сходиться быстрее, но может привести к преждевременной сходимости.

***Функция приспособленности (Fitness function).*** На каждой итерации алгоритма индивидуумы оцениваются с помощью **функции приспособленности** (или **целевой функции**). Это функция, которую мы стремимся оптимизировать, или задача, которую пытаемся решить.

Фитнес-функция зависит от проблемы поставленной задачи и может принимать различные формы. Это может быть простая функция, вычисляющая расстояние между двумя точками, или более сложная функция, включающая множество критериев и ограничений.

Индивидуумы, для которых функция приспособленности дает наилучшую оценку, представляют лучшие решения и с большей вероятностью будут отобраны для воспроизводства и представлены в следующем поколении. Со временем качество решений повышается, значения функции приспособленности растут, а когда будет найдено удовлетворительное значение, процесс можно остановить.

Если фитнес-функция хорошо спроектирована и точно отражает решаемую задачу, генетический алгоритм может сходиться к оптимальному или близкому к оптимальному решению.

***Отбор (Selection).*** После того как вычислены приспособленности всех индивидуумов в популяции, начинается процесс отбора, который определяет, какие индивидуумы будут оставлены для воспроизводства, т. е. создания потомков, образующих следующее поколение. Цель отбора состоит в том, чтобы склонить популяцию к лучшим решениям путем отбора наиболее приспособленных особей для размножения. Этого можно добиться с помощью различных методов выбора, таких как выбор колеса рулетки, выбор турнира или выбор на основе рейтинга. Плохо приспособленные индивидуумы все равно могут быть отобраны, но с меньшей вероятностью. Таким образом, их генетический материал не полностью исключен. Выбирая наиболее подходящих людей, алгоритм склоняется к лучшим решениям и может сходиться к оптимальному или близкому к оптимальному решению.

***Скрещивание (Crossover).*** Целью скрещивания является объединение генетического материала родителей для создания нового решения, которое наследует желательные черты от обоих родителей.Для создания пары новых индивидуумов родители обычно выбираются из текущего поколения, а части их хромосом меняются местами (скрещиваются), в результате чего создаются две новые хромосомы, представляющие потомков.

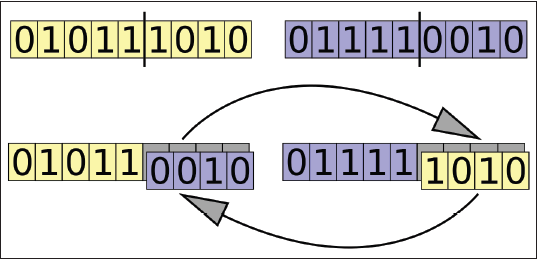


Рисунок 1.3 Операция скрещивания двух двоично-кодированных хромосом

***Мутация (Mutation).*** Цель оператора мутации – периодически случайным образом **обновлять** популяцию, т. е. вносить новые сочетания генов в хромосомы, стимулируя тем самым поиск в неисследованных областях пространства решений. Мутация может проявляться как случайное изменение гена. Мутации реализуются с помощью внесения случайных изменений в значения хромосом, например инвертирования одного бита в двоичной строке.

Вероятность мутации обычно устанавливается на низкое значение, например 1% или меньше, чтобы предотвратить нарушение генетического материала популяции слишком большим количеством случайных изменений. Скорость мутаций можно регулировать в зависимости от желаемого уровня разнообразия в популяции и сложности решаемой задачи.

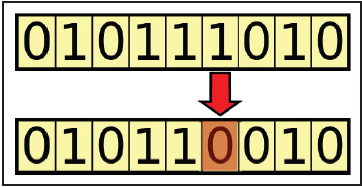


Рисунок 1.4 Применение оператора мутации к двоично-кодированной хромосоме

## Примеры использования генетических алгоритмов.

Генетические алгоритмы часто и эффективно применяются в многочисленных пространствах задач. Их часто используют для сложных задач, которые не требуют абсолютно оптимальных решений, таких как задачи с **ограничениями**, если они слишком велики, чтобы их можно было решить с помощью традиционных методов. Одним из примеров таких задач являются сложные **проблемы планирования**.

Генетические алгоритмы (ГА) применялись для решения широкого круга задач оптимизации и поиска в различных областях, в том числе:

* **Оптимизация проектирования и проектирования**: ГА использовались для оптимизации конструкции конструкций, схем и механических систем. Также их можно использовать для оптимизации параметров сложных инженерных систем, таких как системы отопления, вентиляции и кондиционирования, ветроустановки.
* **Финансовая и экономическая оптимизация**: ГА применялись для решения различных финансовых и экономических проблем, включая оптимизацию портфеля, управление рисками и оценку опционов.
* **Обработка изображений и сигналов**: GA можно использовать для задач обработки изображений, таких как выбор признаков, сегментация и классификация. Их также можно использовать в приложениях обработки сигналов, таких как проектирование фильтров и сжатие сигналов.
* **Машинное обучение**: ГА использовались для настройки параметров и выбора функций в алгоритмах машинного обучения, таких как искусственные нейронные сети, деревья решений и машины опорных векторов.
* **Робототехника и системы управления**: ГА можно использовать для оптимизации параметров управления робототехническими системами и для планирования пути робота.
* **Составление расписания и планирование**: ГА использовались для решения задач составления расписания и планирования, таких как составление расписания производства, планирование проектов и планирование мастерских.
* **Биоинформатика**: ГА использовались для решения задач биоинформатики, таких как анализ экспрессии генов, предсказание структуры белка и выравнивание последовательностей.
* **В компьютерном искусстве:** ГА алгоритмы иногда применяются для имитации фотографий с помощью стохастических методов. Представьте себе 50 многоугольников, случайным образом размещенных на экране и постепенно скручиваемых, поворачиваемых, перемещаемых, изменяющих размеры и цвет, пока они не будут как можно точнее соответствовать фотографии. Результат выглядит как работа художника-абстракциониста или, если использовать более угловатые формы, как витраж.

В целом, генетические алгоритмы полезны в любой ситуации, когда имеется большое пространство для поиска и оптимальное решение не сразу очевидно.

## Преимущества и недостатки генетических алгоритмов

Основные преимущества генетических алгоритмов:

* Способность выполнять глобальную оптимизацию;
* Применимость к задачам сложной математической структуры;
* Применимость к задачам, не имеющим математического образа;
* Поддержка распараллеливания и распределенной обработки;
* Пригодность к непрерывному обучению.

***Глобальная оптимизация.***  Большинство традиционных алгоритмов поиска и оптимизации основаны на вычислении градиента и могут застревать в локальном максимуме, что препятствует нахождению глобального оптимума. Причина заключается в том, что небольшое изменение решения в окрестности локального максимума может привести к ухудшению оценки. В отличие от них, генетические алгоритмы более устойчивы к этому и имеют больше шансов найти глобальный максимум.

Объясняется это тем, что используется популяция потенциальных решений, а не единственное решение, а операции скрещивания и мутации зачастую порождают решения, далеко отстоящие от ранее рассмотренных. Это остается справедливым при условии, что мы поддерживаем разнообразие популяции и избегаем **преждевременной сходимости**, о чем поговорим в следующем разделе.

***Применимость к сложным задачам.*** Генетические алгоритмы могут быть применены к задачам со сложным математическим представлением, так как они нуждаются только в значении функции приспособленности каждого индивидуума, не требуя знания производных и других свойств функции. Кроме того, генетические алгоритмы могут быть применены к задачам, которые не имеют математического представления, например, в случаях, когда оценка приспособленности основана на мнении человека. В таких случаях, генетические алгоритмы могут использоваться для поиска наилучшего решения, используя функцию приспособленности, основанную на оценках пользователей. Допустим, вы хотите найти самую привлекательную цветовую палитру для веб-сайта. Вы можете попробовать разные цветовые сочетания и попросить пользователей оценить привлекательность сайта. Затем вы можете применить генетический алгоритм для поиска наилучшей комбинации, используя фитнес-функцию, основанную на оценках пользователей. Алгоритм будет работать, даже если нет математического представления и невозможно напрямую рассчитать счет данной комбинации.

**Параллелизм.** Генетические алгоритмы хорошо поддаются распараллеливанию и распределенной обработке. Функция приспособленности независимо вычисляется для каждого индивидуума, а это значит, что все индивидуумы в популяции могут обрабатываться одновременно. Кроме того, операции отбора, скрещивания и мутации могут одновременно выполняться для индивидуумов и пар индивидуумов. Поэтому подход, основанный на генетических алгоритмах, естественно адаптируется к распределенным и облачным реализациям.

***Непрерывное обучение.*** В природе эволюция никогда не прекращается. Если окружающие условия изменяются, популяция приспосабливается к ним. Так и генетические алгоритмы могут непрерывно работать в постоянно изменяющихся условиях, и мы всегда можем получить и использовать лучшее на данный момент решение. Но это возможно, только если окружающая среда изменяется медленно по сравнению со скоростью смены поколений в генетическом алгоритме.

Итак, преимущества мы обсудили, теперь перейдем к **недостаткам** или же ограничениям генетических алгоритмов. К ним можно отнести следующие:

* Необходимы специальные определения;
* Необходима настройка гиперпараметров;
* Опасность преждевременной сходимости;
* Отсутствие гарантированного решения.

Рассмотрим их поочередно.

***Специальные определения и настройка гиперпараметров.*** Пытаясь применить генетические алгоритмы к некоторой задаче, мы должны создать подходящее представление – определить функцию приспособленности и структуру хромосом, а также операторы отбора, скрещивания и мутации. Зачастую это совсем не просто и занимает много времени. Поведение генетических алгоритмов контролируется набором гиперпараметров, например размером популяции и скоростью мутации. Точных правил для выбора значений гиперпараметров не существует. Однако так обстоит дело практически со всеми алгоритмами поиска и оптимизации.

***Преждевременная сходимость.*** Если приспособленность какого-то индивидуума гораздо больше, чем у всей остальной популяции, то не исключено, что он продублируется так много раз, что в конечном счете, кроме него, в популяции ничего не останется. В результате генетический алгоритм может застрять в локальном максимуме и не найдет глобального. Чтобы предотвратить такое развитие событий, важно поддерживать разнообразие популяции. В следующей главе мы рассмотрим различные способы достижения этой цели.

***Отсутствие гарантированного решения.*** Использование генетических алгоритмов не гарантирует нахождения глобального максимума. Однако это типично для всех алгоритмов поиска и оптимизации, если только у задачи не существует аналитического решения. Но, вообще говоря, при правильном применении генетические алгоритмы находят хорошие решения на разумное время.

## Вывод

Резюмируя изложенное в предыдущих разделах, можно сказать, что генетические алгоритмы лучше применять для решения следующих задач.

* **Задачи со сложным математическим представлением.** Поскольку генетическим алгоритмам нужно знать только значение функции приспособленности, их можно использовать для решения задач, в которых целевую функцию трудно или невозможно продифференцировать, задач с большим количеством параметров и задач с параметрами разных типов.
* **Задачи, не имеющие математического представления.** Генетические алгоритмы не требуют математического представления задачи, коль скоро можно получить значение оценки или существует метод сравнения двух решений.
* **Задачи с зашумленной окружающей средой.** Генетические алгоритмы устойчивы к зашумленным данным, например прочитанным с датчика или основанным на оценках, сделанных человеком.
* **Задачи, в которых окружающая среда изменяется во времени.** Генетические алгоритмы могут адаптироваться к медленным изменениям окружающей среды, поскольку постоянно создают новые поколения, приспосабливающиеся к изменениям.

С другой стороны, если для задачи известен специализированный способ решения традиционным или аналитическим методом, то вполне вероятно, что он окажется эффективнее.

# **Глава 2 ИССЛЕДОВАНИЕ МНОГООБРАЗИЯ ПОДХОДОВ ГЕНЕТИЧЕСКОГО АЛГОРИТМА**

В этой главе будут рассмотрены все шаги алгоритма, особое внимание будет уделено операторам наиболее изменяющих популяцию. Так же на примере простой задачи будет разобран пример – “Hello World” в мире ГА, посмотрим на разные подходы и насколько сильно влияет выбор правильной конфигурации алгоритма.

1. Структура генетического алгоритма

Перечислим каждый этап базового алгоритма и далее опишем каждый из них более подробно.

1. Начало
2. Создать начальную популяцию
3. Вычислить приспособленность каждого индивидуума
4. Selection (Отбор)
5. Crossover (Скрещивание)
6. Mutation (Мутация)
7. Отбор наилучших индивидов
8. Условия остановки выполнены? Если нет вернуться к пункту 3
9. Выбрать индивидуума с максимальной приспособленностью
10. Конец

***Создание начальной популяции.*** Первым шагом в ГА является создание начальной популяции потенциальных решений, где каждое такое решение-кандидат представляет отдельного индивидуума в популяции. Начальная популяция – это, по сути дела, набор хромосом. Формат хромосом должен соответствовать принятым для решаемой задачи правилам, например это могут быть двоичные строки определенной длины или векторы действительных чисел.

***Вычисление приспособленности.*** Для каждого индивидуума вычисляется фитнес-функция. Это делается для каждого нового поколения после применения операторов отбора, скрещивания и мутации. Поскольку приспособленность любого индивидуума не зависит от всех остальных, эти вычисления можно производить параллельно. Функция пригодности определяет, насколько хорошо человек решает поставленную задачу. Результатом оценки является оценка пригодности для каждого человека.

***Применение операторов отбора, скрещивания и мутации.*** Применение генетических операторов к популяции приводит к созданию новой популяции, основанной на лучших индивидуумах из текущей.

* Оператор **отбора** отвечает за отбор индивидуумов из текущей популяции таким образом, что предпочтение отдается лучшим. Особи с более высокими показателями фитнес-функции с большей вероятностью будут отобраны для разведения.
* Оператор **скрещивания**, при нем происходит процесс объединения генетической информации от двух родителей для создания нового потомства. Обычно для этого берутся два случайных индивидуума, и части их хромосом меняются местами, в результате чего создаются две новые хромосомы, представляющие двух потомков. Этот процесс повторяется до тех пор, пока не будет создана новая популяция потомков.
* Оператор **мутации** вносит случайные изменения в один или несколько генов хромосомы. Мутация вносит разнообразие в популяцию и помогает предотвратить преждевременную сходимость. Частота мутаций обычно устанавливается для контроля частоты мутаций.

***Проверка условий остановки.*** Это заключительный этап генетического алгоритма, на котором алгоритм решает прекратить поиск решения. Существует несколько причин, по которым может произойти этап завершения, например:

* **Алгоритм нашел удовлетворительное решение**. ГА завершится, если найдет решение, отвечающее желаемым критериям, таким как определенный уровень пригодности.
* **Достигнуто максимальное количество поколений.** Это условие заодно позволяет ограничить время работы алгоритма и потребление им ресурсов системы;
  + **За последнее время не наблюдается заметных улучшений**. Это можно реализовать путем запоминания наилучшей приспособленности, достигнутой в каждом поколении, и сравнения наилучшего текущего значения со значениями в нескольких предыдущих поколениях. Если разница меньше заранее заданного порога, то алгоритм можно останавливать.

1. Операторы отбора

Отбор выполняется в начале каждой итерации цикла генетического алгоритма, чтобы выбрать из текущей популяции тех индивидуумов, которые станут родителями индивидуумов в следующем поколении. Рассмотрим самые популярные вариации этого оператора:

* **FPS** – Правило рулетки;
* **SUS** – Стохастическая универсальная выборка;
* **Rank Selection** – Ранжированный отбор;
* **Fitness scaling** Масштабирование приспособленности;
* **Tournament Selection** – Турнирный отбор;
* **Elitism –** Элитизм;

Поговорим про некоторые из них более подробно.

***Правило рулетки.***  В этом методе особи отбираются для размножения на основе их значения фитнес-функции, при этом более приспособленные имеют более высокую вероятность быть выбранными, другими словами, величина приспособленности прямо пропорциональна вероятности отбора. Приведем аналогию с рулеткой, это делается путем назначения каждому индивидууму части сектора колеса рулетки, пропорционального его значению физической подготовки, а затем выбора путем вращения колеса.

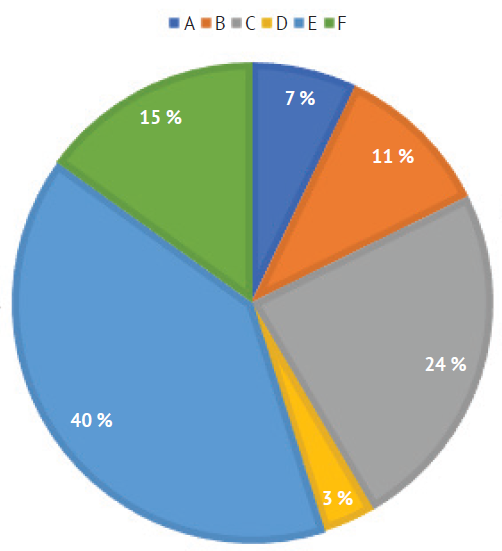


Рисунок 2.1 Пример отбора по правилу рулетки

Рулетка запускается до тех пор, пока не наберется достаточно индивидуумов для образования следующего поколения, недостатком может считаться, то, что один и тот же индивид может выпасть несколько раз.

***Стохастическая универсальная выборка.***  Это измененный вариант правила рулетки. Используется та же рулетка, но вместо одной точки отбора мы используем несколько равномерно расположенных точек и крутим один раз. Тем самым все индивидуумы выбираются одновременно, как показано на рисунке далее.

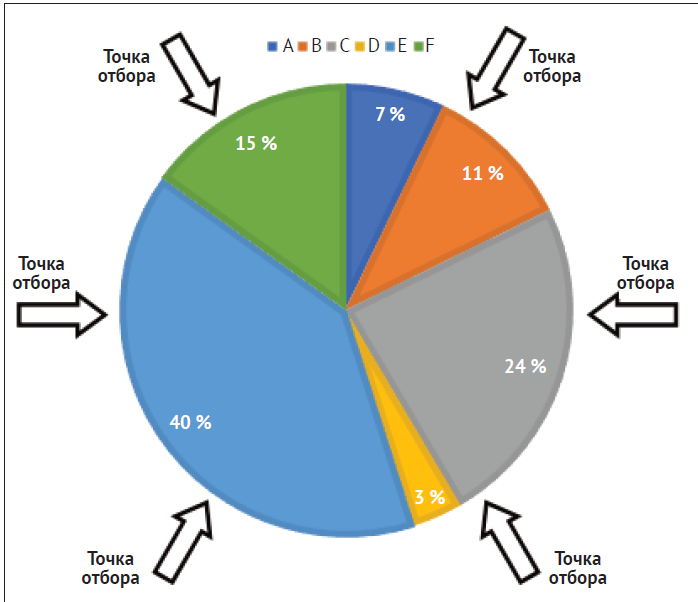


Рисунок 2.2 Пример Стохастической универсальной выборки

Такой подход не дает особям с высокой приспособленностью заполнить все следующее поколение в результате повторного выбора. Поэтому более слабым индивидуумам предоставляется шанс.

**Турнирный отбор.** Это самый популярный метод отбора. Этот подход разделяются на раунды в каждом таком раунде из популяции выбираются два или более индивидуумов, отбирается тот у кого приспособленность больше, выигрывает и отбирается в следующее поколение. Пример подобного поведения можно увидеть на рисунке ниже.

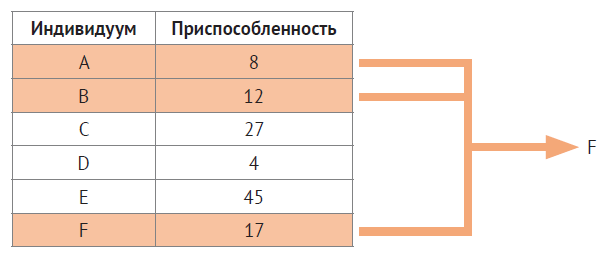


Рисунок 2.3 Пример турнирного отбора на турнире с тремя участниками

Преимущество турнирного отбора в том, что он позволяет сбалансировать разнообразие. Количество индивидуумов, участвующих в каждом раунде турнирного отбора, называется размером турнира. Выбор турнира также можно изменить, добавив параметр, называемый «давление отбора». Этот параметр определяет вероятность того, что будет выбран человек с наибольшей приспособленностью. Высокое давление отбора означает, что с большей вероятностью будет выбран наиболее приспособленный индивидуум, а низкое давление отбора означает, что отбор более случайный. Давление отбора можно регулировать, изменяя размер турнира или вероятность выбора победителя турнира.

***Элитизм.*** Элитарный отбор — это вариант отбора, при котором наиболее приспособленные особи из текущего поколения автоматически переносятся в следующее поколение без каких-либо изменений или модификаций. Это гарантирует, что так называемая «Элита» не будет потеряна в процессе эволюции и сможет продолжить вносить свой вклад в поиск оптимального решения.

Преимущество элитарного отбора состоит в том, что он гарантирует сохранение лучших индивидуумов и их дальнейший вклад в поиск оптимального решения. Без элитарности лучшие представители одного поколения могут быть потеряны в следующем поколении из-за случайного отбора, который может предотвратить сходимость к оптимальному решению.

Однако недостатком элитарного выбора является то, что он может привести к преждевременной сходимости. Всегда выбирая лучших особей, генетический алгоритм может застрять в локальном оптимуме и не сможет исследовать другие области пространства поиска. Чтобы решить эту проблему, можно использовать другие методы отбора в сочетании с элитарным отбором.

1. Операторы скрещивания

Оператор скрещивания служит для совмещения генетической информации двух родительских индивидуумов при создании потомства. Обычно этот оператор не всегда используется, а применяется с определенной вероятностью. Если скрещивание не выполняется, то оба родительских индивидуума копируются в следующее поколение без изменений.

Разновидности операторов скрещивания:

* Одноточечное скрещивание
* k-точечное скрещивание
* Равномерное скрещивание
* Упорядоченное скрещивание
* BLX – Скрещивание Смешением
* SBX – Имитация двоичного скрещивания

***Одноточечное скрещивание.*** Одноточечное скрещивание выбирает одну точку кроссинговера. Генетическая информация от одного родителя копируется в потомство до точки кроссинговера, а генетическая информация от другого родителя копируется в потомство от этой точки и до конца. Пример можно увидеть ниже на рисунке.

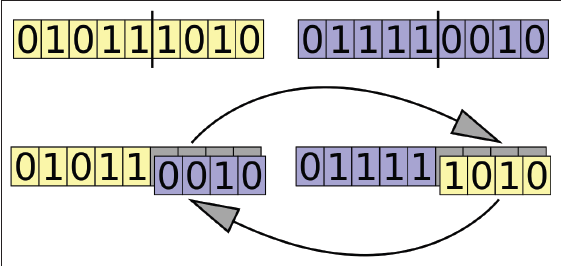


Рисунок 2.4 Пример одноточечного скрещивания

Проблема такого подхода заключается в потере разнообразия популяции, если одна и та же точка кроссинговера используется повторно.

***Двухточечное и k-точечное скрещивание.*** При двухточечном скрещивании в каждой хромосоме случайным образом выбираются по две точки кроссинговера. Гены одной хромосомы, расположенные между этими точками, обмениваются с точно так же расположенными генами другой хромосомы.

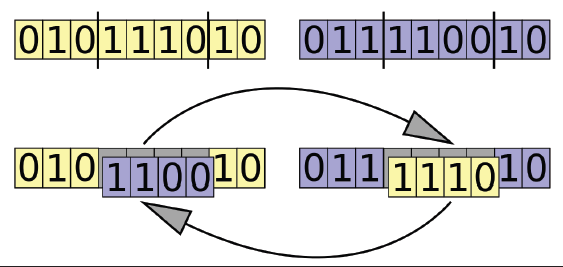


Рисунок 2.5 Пример двухточечного скрещивания

По аналогии работает обобщенный метод k-точечного скрещивания. Где k – целое число. Такое скрещивание в разы лучше одноточечного аналога, из-за разнообразия полученного потомства.

***Равномерное скрещивание.*** При равномерном скрещивании каждый ген потомства выбирается с равной вероятностью от одного из его родителей. Это означает, что существует 50% вероятность того, что каждый ген потомства будет получен от любого из родителей.

Процесс равномерного скрещивания выполняется путем создания бинарной маски, указывающей, какие гены происходят от какого родителя. Например, если длина родительских хромосом равна n, генерируется двоичная маска длины n со значениями 0 и 1, указывающими, какой родительский ген будет выбран. Затем эта бинарная маска применяется к обоим родителям для создания потомства.

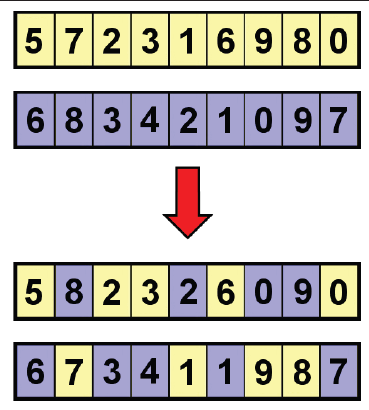


Рисунок 2.6 Пример равномерного скрещивания

Такой метод может существенно повысить появление разнообразных потомком, так как в этом методе не производится обмен целых участков хромосом.

1. Операторы Мутации

Мутация – это последний оператор, который применяется для создания новых потомков. Она происходит случайно и имеет низкую вероятность, поскольку может привести к ухудшению качества потомства. Если мутации происходят слишком часто, то генетический алгоритм может превратиться в случайный поиск, что нежелательно. Это важный шаг для поддержания генетического разнообразия и предотвращения преждевременной сходимости.

Список популярных вариаций операторов мутаций:

* Инвертирование бита
* Мутация обменом
* Мутация обращением
* Мутация перестановкой
* Мутация Гаусса

***Инвертирование бита.*** Для двоичной хромосомы случайным образом выбирается ген который в последствии инвертируя его.



Рисунок 2.7 Пример мутации инвертированием бита

Плюсом такого подхода является простота в реализации, так же такая мутация может быть полезна при исследовании пространства решений когда алгоритм сошелся к неоптимальному решению.

***Мутация обменом.*** Этот метод применим как к двоичным, так и к целочисленным хромосомам: случайно выбираются два гена, и их значения меняются местами.



Рисунок 2.8 Пример мутации обменом

Эта операция может быть полезна для сохранения хороших решений, найденных в предыдущих поколениях.

***Мутация обращением.*** При применении этого метода к двоичной или целочисленной хромосоме выбирается случайная последовательность генов, и порядок генов в ней меняется на противоположный.



Рисунок 2.9 Пример мутации обращением

Преимущество инверсионной мутации заключается в том, что она может сохранять относительный порядок генов в подмножестве, что может быть полезно в некоторых проблемных областях.

**Мутация Гаусса.** В отличие от мутаций с переворачиванием битов, в генетическом материале, мутация по Гауссу добавляет случайное значение, полученное из распределения Гаусса, к каждому элементу генетического материала индивидуума. Количество мутаций, внесенных в каждый элемент генетического потомка, контролируется параметром, называемым частотой мутаций, который определяет стандартное отклонение распределения Гаусса.

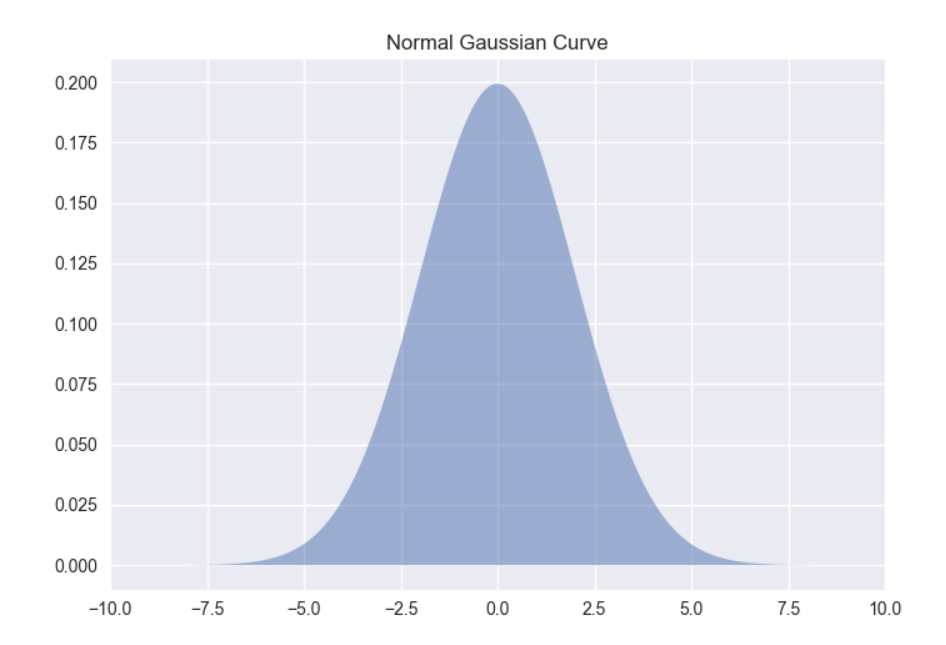


Рисунок 2.10 График нормального распределения

Преимущество гауссовской мутации заключается в том, что она позволяет более точно контролировать количество мутаций, внесенных в генетический материал. Это может быть полезно в тех случаях, когда область поиска велика и сложна, а небольшие изменения в генетическом материале могут оказать существенное влияние на приспособленность человека.

Один потенциальный недостаток гауссовой мутации заключается в том, что она может внести слишком много случайности в процесс поиска, что приведет к потере направленности и потенциально замедлит сходимость алгоритма. Как и в случае с другими генетическими операторами, важно тщательно настроить частоту мутаций, чтобы сбалансировать исследование и использование пространства поиска.

1. “Hello World” в мире генетических алгоритмов

При знакомстве с миром генетических алгоритмов в первую очередь решают простую задачу оптимизации OneMax. Ее задача в том, чтобы найти двоичную строку заданной длины с максимальной суммой в ней. Очевидно, что решением считается строка, состоящая из всех единиц, тем самым максимизируя ее сумму. Хоть задача очень тривиальна, но зато хорошо показывает этапы и суть алгоритма. Итак приступим к рассмотрению.

Наша цель – найти решение, которое бы давало максимальную сумму цифр этого списка. Логично предположить что фитнес-функцией будет являться просто сумма в каждого гена:

Тут N – длина списка. Лучший индивид будет тот у которого все числа единицы, это и есть решение данной задачи.

В рамках задачи OneMax мы воспользуемся идеей турнирного отбора с выборкой из трех претендентов на каждой итерации. Будем использовать одноточечное скрещивание (одноточечный кроссинговер). А при мутации будем выполнять инвертирование бита с некоторой очень небольшой вероятностью.

***Реализация на Python.*** Для работы с генетическими алгоритмами создан целый ряд каркасов на Python, например GAFT, Pyevolve и PyGMO. Остановимся на каркасе DEAP, поскольку он прост в использовании и предлагает широкий набор функций, поддерживает расширяемость и может похвастаться подробной документацией.

DEAP (сокращение от Distributed Evolutionary Algorithms in Python – распределенные эволюционные алгоритмы на Python) поддерживает быструю разработку решений с применением генетических алгоритмов и других методов эволюционных вычислений. DEAP предлагает различные структуры данных и инструменты, необходимые для реализации самых разных решений на основе генетических алгоритмов [2].

Определим фитнесс функцию, которая является суммой всех элементов в индивидууме.

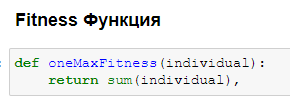


Рисунок 2.11 Фитнес функция для задачи OneMax

Для генерации начальной популяции можем воспользоваться ToolBox, возможностью пакета DEAP,

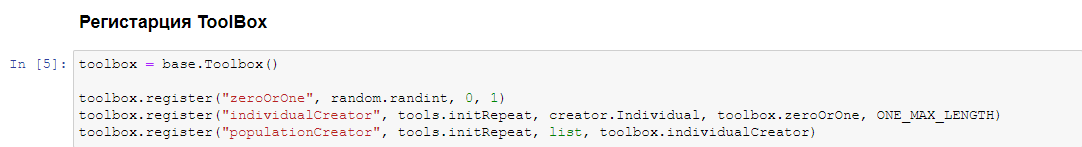


Рисунок 2.12 настройка начальной популяции

Здесь мы используем существующую функцию initRepeat из модуля tools, которая как раз и разработана для формирования списков. В данном случае, мы ей указываем:

tools.initRepeat(<контейнер для хранения генов>, <функция генерации значения гена>, <число генов в хромосоме>)

То есть, на выходе функция individualCreator будет выдавать экземпляр класса creator.Individual – список заполненный случайными величинами 0 или 1 с длиной хромосомы.Таким образом мы сможем сформировать начальную популяцию. Далее воспользуемся встроенными функциями отбора мутации и скрещивания, соответственно selTournament, mutFlipBit и cxOnePoint.

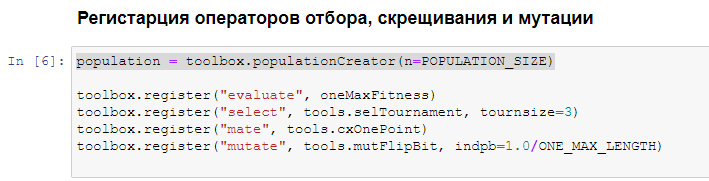


Рисунок 2.13 Настройка операторов отбора, скрещивания и мутации

Все готово для запуска ГА. Для этого воспользуемся готовой функцией eaSimple() модуля algorithms.

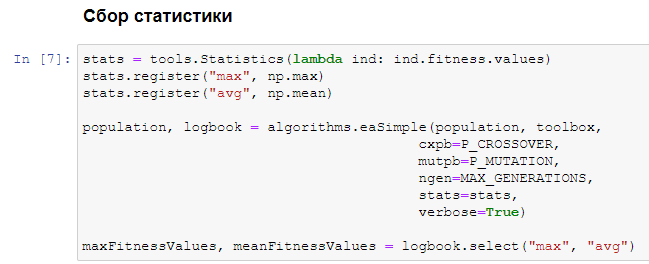


Рисунок 2.14 Сбор статистики

Тут мы будем сравнивать среднюю и максимальную приспособленность от поколения и отобразим это на графике.

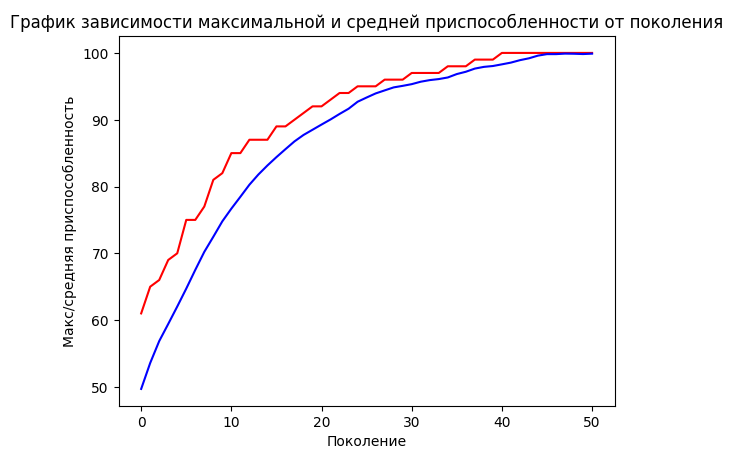


Рисунок 2.15 График зависимости максимальной и   
средней приспособленности от поколения

Начиная с 40-го поколения максимальное значение приспособленности перестает изменяться, а среднее продолжает расти, пока в конце концов не станет почти равным максимальному. Это означает, что в конце прогона почти все индивидуумы стали равны лучшему.

1. Эксперименты с параметрами ГА

Проведем несколько экспериментов с целью узнать как различные подходы в выборе стратегий алгоритма способны повлиять на итоговый результат. Изменения мы будем проводить по следующим показателям алгоритма:

* размер популяции
* оператор скрещивания
* оператор мутации
* оператор отбора

***Размер популяции.*** Изменим размер популяции и количество поколений. Размер популяции определяется константой POPULATION\_SIZE. Увеличим ее с 200 до 400, результаты можно видеть на рисунке ниже:

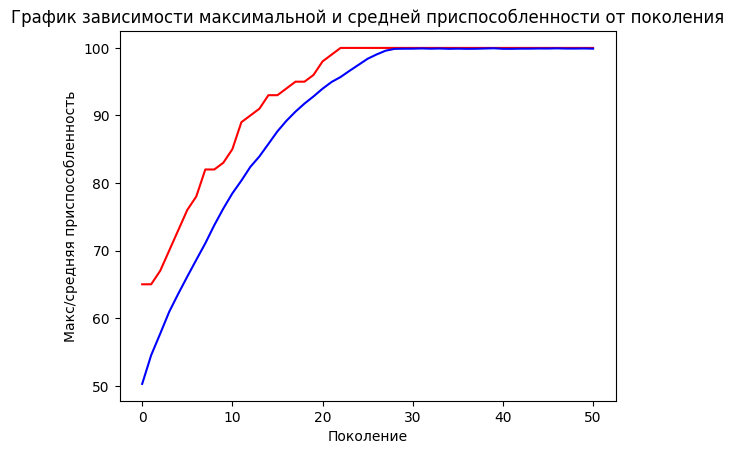


Рисунок 2.16 Результат увеличения размера популяции

Как видно из графика лучшее решение достигается уже на 20 поколении. Такое поведение ожидаемо так как при увеличении популяции для нахождения решения требуется меньше поколений. Однако с ростом размера популяции повышаются и требования к вычислительной мощности и ресурсам, поэтому обычно стремятся найти компромисс в данных параметрах.

Для чистоты эксперимента все последующие изменения будем проводить с параметрами начальной системы, что бы не мешать одни изменения с другими.

***Оператор скрещивания.***  Заменим одноточечное скрещивание на двуточечное и сравним полученные результаты. Что бы сделать это нужно заменить функцию скрещивания на двуточечное:

toolbox.register("mate", tools.cxTwoPoint)

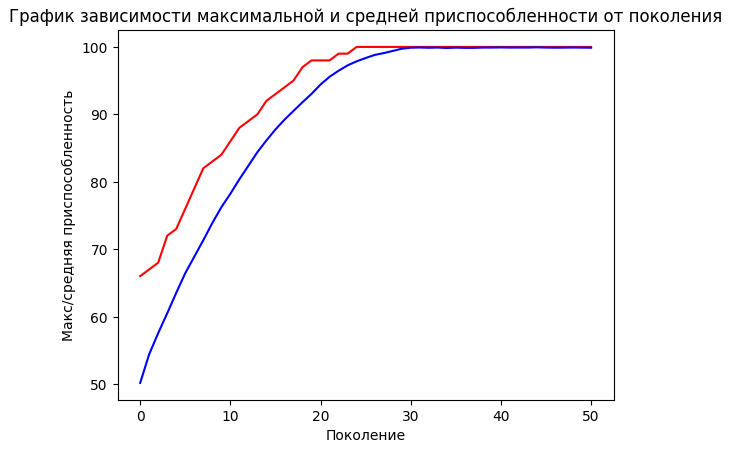


Рисунок 2.17 Результат изменения оператора скрещивания на двуточечное

Теперь алгоритм имеет решение уже на 24 поколении. Это объясняется тем, что двухточечное скрещивание – более гибкий способ смешивания генов родителей по сравнению с одноточечным.

***Оператор мутации.***  Изменим вероятность мутации у каждого конкретного индивидуума. Увеличим константу P\_MUTATION до 0.9 и увеличим значение indpb в десять раз. В результате поведение алгоритма становится неустойчивым.

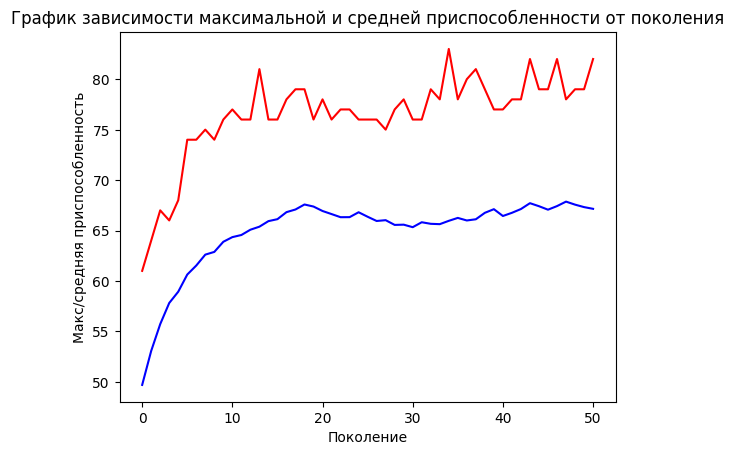


Рисунок 2.18 Увеличение вероятности мутации каждого индивидуума

Такое поведение объясняется тем, что при увеличении вероятности мутации полученные гены можно будет сравнить с случайными, тем самым превратив ГА в случайный поиск.

***Оператор отбора.*** Заменим турнирный отбор отбором по правилу рулетки. Для этого заменим на эту строчку: toolbox.register("select", tools.selRoulette). Результаты, следующие:



Рисунок 2.19 Изменение оператора отбора на отбор по правилам рулетки

На графике наблюдается, что несколько точек с наилучшими решениями не получают достаточного внимания, и максимальная приспособленность колеблется, в то время как средняя приспособленность монотонно возрастает.

Это объясняется тем, что в правиле рулетки вероятность выбора индивидуума зависит от его приспособленности. Если разница в приспособленности между индивидуумами невелика, то более слабые могут быть выбраны чаще, чем при использовании турнирного отбора.

1. Вывод

В этой главе мы познакомились с общей структурой генетического алгоритма. Затем мы рассмотрели детали: создание популяции, вычисление функции приспособленности, применение генетических операторов и проверку условий остановки. Более подробно разобрались с понятиями и видами самой главной части алгоритма, операторами отбора скрещивания и мутации. На практических примерах продемонстрировали эффективность ГА в решении задач оптимизации, включая задачу Onemax.

Важно отметить, что хотя генетический алгоритм может быть мощным инструментом, его производительность сильно зависит от выбора параметров и конкретной решаемой задачи. Поэтому очень важно тщательно настроить эти параметры и понять проблему для достижения оптимальных результатов.

Из-за множества вариаций генетических алгоритмов возникает вопрос о правильной настройке параметров и выборе методов для достижения максимальной эффективности в рамках конкретной задачи. Для начала нужно определить фитнес-функцию, которая оценивает индивидуумов. Затем необходимо выбрать подходящий способ кодирования хромосом, основанный на параметрах, передаваемых фитнес-функции. Однако если фитнес-функция недоступна, можно воспользоваться турнирным методом отбора. При выборе методов оператора скрещивания и мутации нужно опираться на специфику конкретной задачи, иногда используются смешанные и собственные методы. Наконец, следует учитывать гиперпараметры алгоритма. Наиболее распространенные:

* Размер популяции;
* Частота скрещивания;
* Частота мутаций;
* Максимальное количество поколений;
* Другие условия остановки;
* Элитизм.

Как можно было видеть даже на простой задаче OneMax, правильный подбор гипперпараметров и выборов операторов отбора скрещивания и мутации довольно сильно влияет на итоговый результат.

# **Глава 3 ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ И РАЗРАБОТКА БИБЛИОТЕКИ**

Рассматривая практическое применение генетического алгоритма, можно указать на разнообразные виды задач. Однако, если учитывать направления развития мировой практики, то в настоящее время наиболее распространенным и полезным применением является искусственный интеллект. Генетические алгоритмы находят применение во многих областях искусственного интеллекта, включая машинное обучение, обработку естественного языка, компьютерное зрение, управление роботами и многое другое.

Одно из наиболее распространенных применений ГА - это обучение нейронных сетей, которое позволяет улучшить точность модели и повысить скорость обучения. Другим примером является решение задач оптимизации, которые возникают в машинном обучении, например, поиск оптимальных гиперпараметров модели или подбор наилучшего набора признаков.

ГА также применяются в обработке естественного языка, например, для генерации текстов, классификации текстовых документов и машинного перевода. В области компьютерного зрения, ГА используются для решения задач распознавания образов, поиска объектов на изображении и других задач, связанных с обработкой изображений и видео. Кроме того, ГА могут использоваться для управления роботами, например, для определения оптимальной траектории движения робота или для оптимизации параметров управляющего алгоритма.

В целом, применение генетических алгоритмов в искусственном интеллекте огромно и продолжает развиваться с появлением новых задач и технологий. Одним из перспективных направлений можно считать метод машинного обучения с подкреплением, его мы и рассмотрим подробнее.

1. Обучение с подкреплением

Обучение с подкреплением RL (reinforcement learning) - это один из видов машинного обучения, который используется для обучения агентов принимать оптимальные решения в средах, где есть возможность выполнения действий и получения обратной связи за эти действия.

В отличие от обучения с учителем, в котором модель обучается на основе правильных ответов, и в отличие от обучения без учителя, в котором модель пытается найти скрытые закономерности в данных, RL работает с помощью системы наград и штрафов. Агент получает награду за выполнение правильных действий и штраф за неправильные действия. Таким образом, агенту нужно определить оптимальную стратегию действий, чтобы максимизировать полученную награду.

Основная идея обучения с подкреплением заключается в том, чтобы агент мог самостоятельно исследовать среду и определить оптимальную стратегию действий. Агент начинает со случайной стратегии, затем постепенно улучшает ее, используя полученный опыт и обратную связь. Обучение с подкреплением является одним из самых эффективных методов обучения в задачах, связанных с принятием решений в динамических средах, таких как игры, управление роботами, финансовые торги и другие.

Существует множество случаев использования обучения с подкреплением (RL). Некоторые из них включают:

* Игры: RL может использоваться для обучения агентов играть в компьютерные игры, такие как шахматы, го, покер и видеоигры. Например, в марте 2016 года программа AlphaGo, основанная на обучении с подкреплением и предназначенная для игры в игру го, сумела победить сильнейшего в мире игрока в матче, который получил широкое освещение в СМИ.
* Робототехника: RL может использоваться для обучения роботов выполнять задачи, такие как управление роботом, балансирование, манипулирование объектами и пр.
* Автоматическая торговля: RL может использоваться для обучения агентов, которые принимают решения о покупке или продаже финансовых инструментов на основе текущих рыночных условий.
* Управление ресурсами: RL может использоваться для оптимизации использования ресурсов, таких как электроэнергия, вода или топливо.
* Медицина: RL может использоваться для обучения агентов, которые принимают решения о лечении пациентов на основе симптомов, лабораторных данных и истории болезни.
* Рекомендательные системы: RL может использоваться для обучения агентов, которые предлагают пользователю рекомендации на основе его действий и предпочтений.

Это лишь несколько примеров использования RL, и его применение продолжает расширяться в различных областях.

Основным преимуществом обучения с подкреплением можно считать то, что ему не требуется наличие размеченных обучающих данных, т. е. пар, состоящих из входа и соответствующего ему выхода. Вместо этих данных их место занимает окружающая среда, от которой требуется получить максимальное вознаграждение за длительный период. Это означает, что иногда алгоритму приходится отступать назад, чтобы в итоге достичь долгосрочной цели.

***Концептуальная модель.*** Представляет собой абстрактный каркас, описывающий основные компоненты и процессы в RL. Она включает в себя три основных элемента: агента, среду и награду. Агент - это сущность, которая принимает решения в среде на основе опыта, полученного из награды. Среда - это внешний мир, в котором действует агент. Среда может быть физическим миром, игрой или любой другой формой окружающей среды. Награда - это сигнал, который агент получает от среды, когда он выполняет действия.

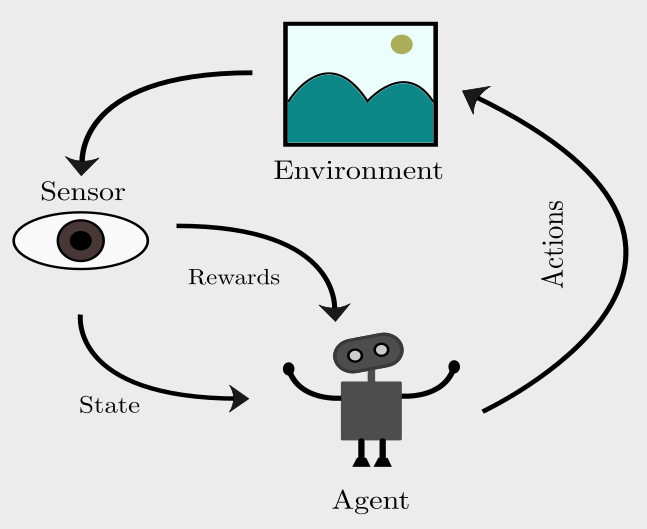


Рисунок 3.1 Концептуальная схема RL

Основная цель агента в RL состоит в том, чтобы максимизировать награду, которую он получает от среды. Для этого агент должен научиться выбирать правильные действия на основе опыта, полученного взаимодействуя со средой. Агент получает опыт, выполняя действия и наблюдая за результатами взаимодействия. Он использует этот опыт, чтобы корректировать свои действия и улучшать свои стратегии.

Наконец, концептуальная модель RL описывает процессы обучения агента. Обучение в RL происходит на основе обратной связи, которую агент получает от среды в виде награды. Агент использует эту обратную связь, чтобы корректировать свои действия и улучшать свои стратегии. Процесс обучения в RL может быть представлен как итерационный процесс, включающий в себя выполнение действий, получение обратной связи и корректировку стратегии агента.

1. Приложение Генетического алгоритма в RL

Рассмотрим практическое применение генетического алгоритма в RL на примере задачи о самоуправляемой машине (self-driving car). Добавим ограничения: машина может передвигаться только в пределах виртуальной трассы. Тогда задачу можно будет сформулировать следующим образом: обучить модель нейронной сети автомобиля так что бы он смог проехать один круг по виртуальной трассе при этом не врезавшись. Выглядит это следующим образом:

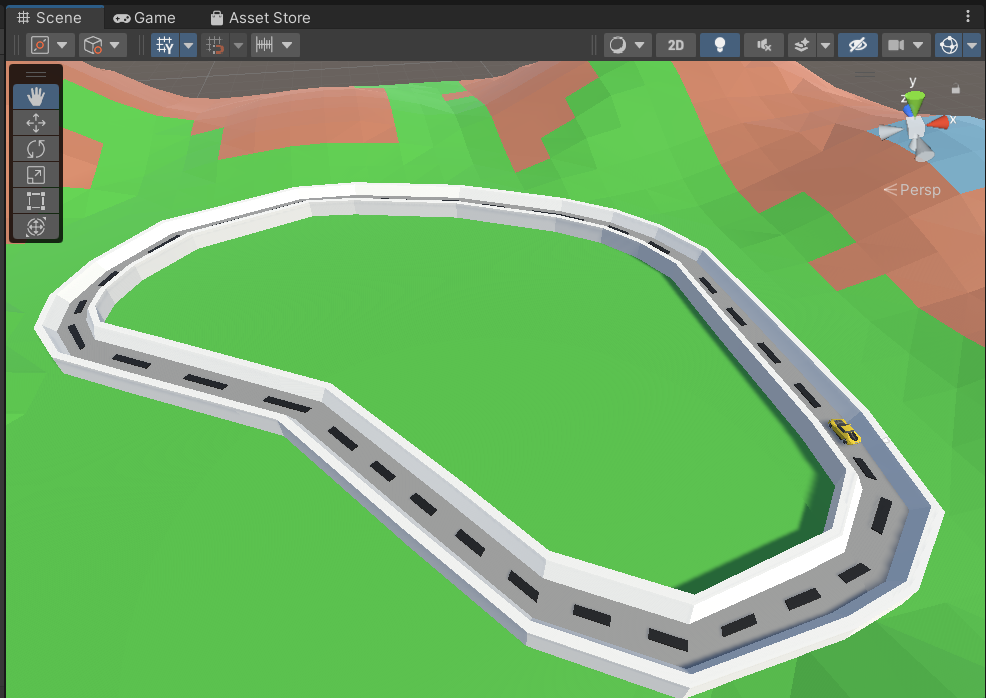


Рисунок 3.2 Окружающая среда виртуальной трассы

Для эффективного применения ГА в любой задаче нужно правильно настроить параметры, после некоторых экспериментов можно выделить следующие:

* Вероятность мутации: 5% для каждого индивидуума
* Процент скрещивания от поколения: 50%
* Размер начальной популяции: 80 особей

Уже на 23 поколении машина смогла преодолеть 3 круга ни разу не врезавшись.

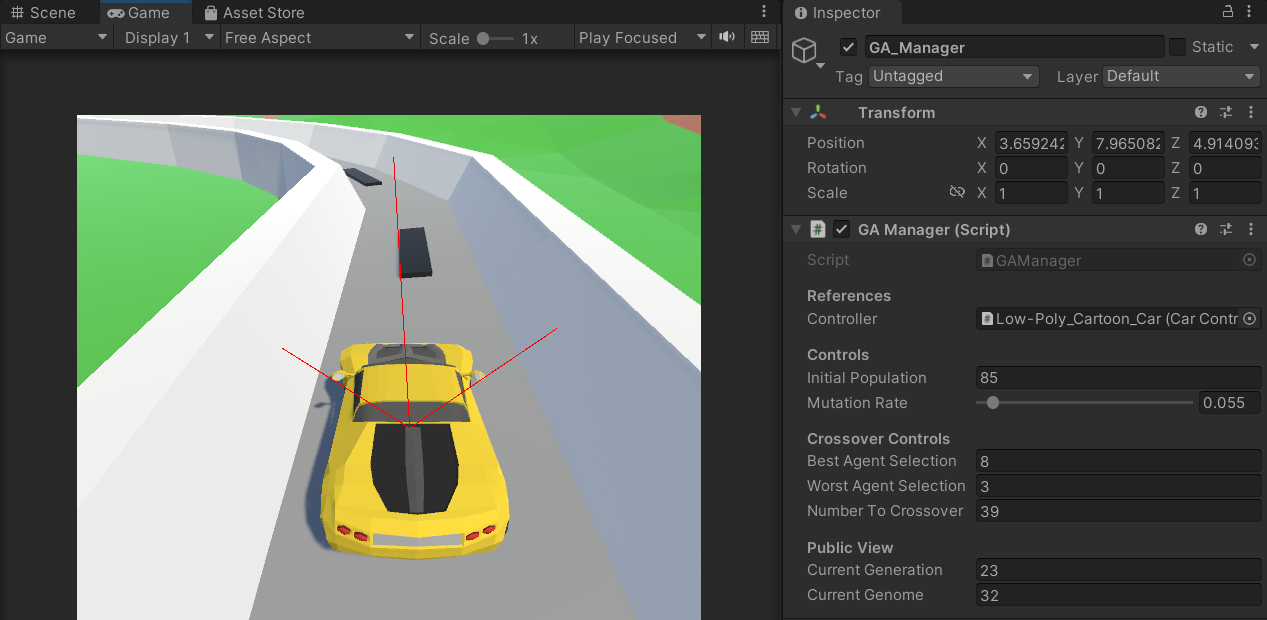


Рисунок 3.3 Трассировочные лучи

Принцип работы в следующем: у виртуальной машины есть 3 луча, которые в любой момент времени содержат расстояние до конкретной стенки. Эти данные служат описанием окружающей среды в текущий момент времени, что позволяет настроенной нейронной сети понимать, как нужно изменить параметры ускорения А (acceleration) и поворота Т (turning).

Каждый раз, когда машина врезается, она записывает пройденное расстояние для сравнения их с фитнес функцией, и в зависимости от результата попытки, принимается решение о рейтинге текущего индивида, затем идет переход хода следующему индивидууму. Следующим этапом после прогона всего поколения следуют операторы отбора скрещивания и мутации, в результате которых образуется новое поколение. На определенном поколении, пройдя все этапы эволюции, задача достигает поставленной цели.

В качестве результатов были проведены тесты с разными подходами выбора операторов. В первом случае (синий график) я выбрал размер популяции 40, вероятность мутации 30 % и количество нейронов 5 для сравнения во втором случае (красный график) – размер популяции: 80, вероятность мутации: 5%, количество нейронов 10. Результаты на графике ниже.

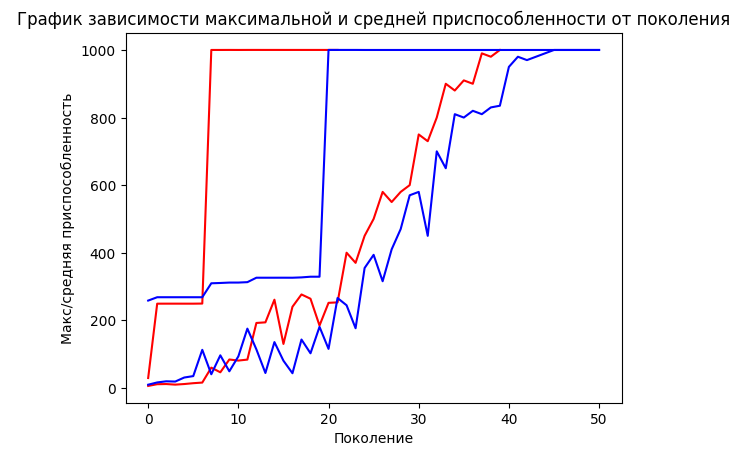


Рисунок 3.4 Сравнение результатов разных параметров

По графикам видно, что выбор гиперпараметров алгоритма очень важный этап, который может существенно ускорить процесс нахождения решения.

1. Структурная схема библиотеки

Для проектирования структурной схемы библиотеки сначала мы должны определиться какие основные классы будут фигурировать в основном Domain слое. Сам алгоритм похож на конвейер и поэтому нужно создать отдельные группы сущностей на каждый структурный элемент.

Определим базовые сущности: Gene, Chromosome, Generation, Population, Fitness. Так же нужно определить базовые этапы в виде групп классов: Selections, Mutations, Crossovers. У каждого такого оператора есть несколько видов и логично определить базовый интерфейс для каждого из них: ISelection, IMutation, ICrossover соответственно.

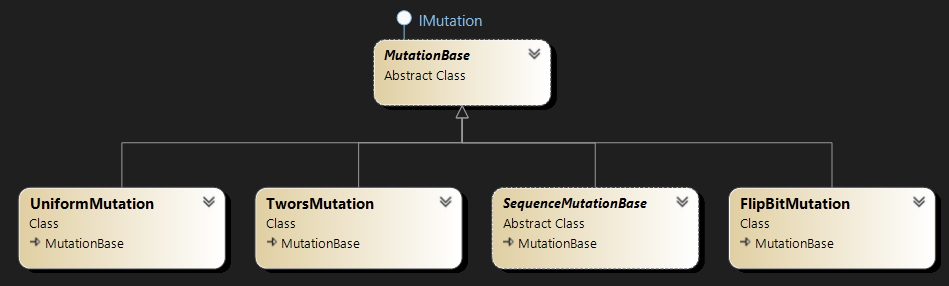


Рисунок 3.5 Структура классов оператора мутации

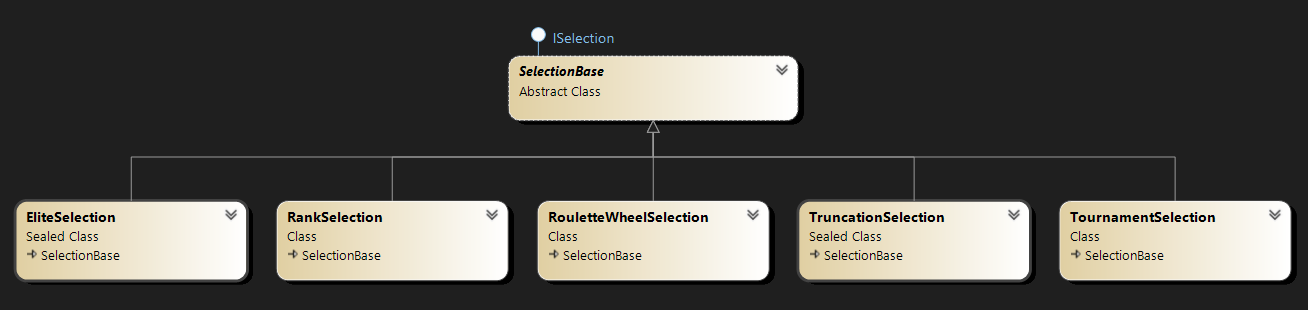


Рисунок 3.6 Структура классов оператора отбора

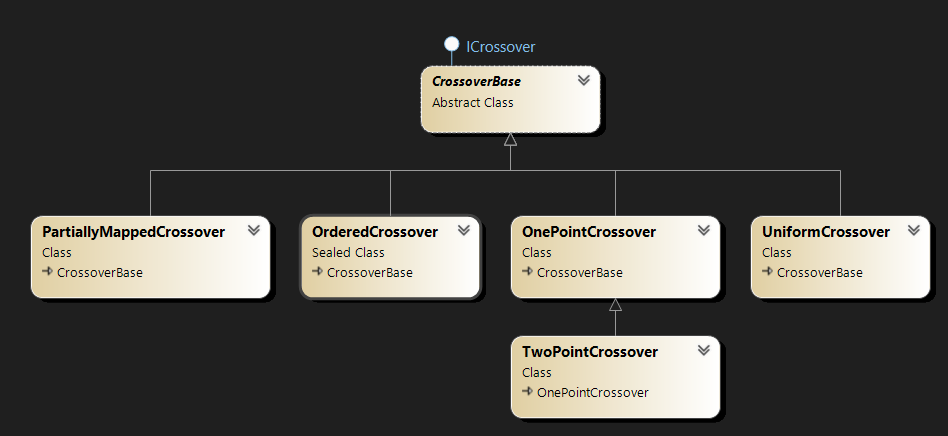


Рисунок 3.7 Структура классов оператора скрещивания

Кроме основных операторов определим и другие этапы алгоритма: Chromosome, Population, Termination.

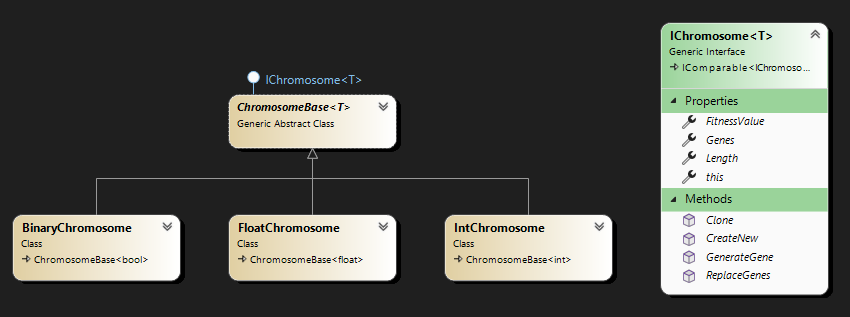


Рисунок 3.8 Структура классов Хромосомы

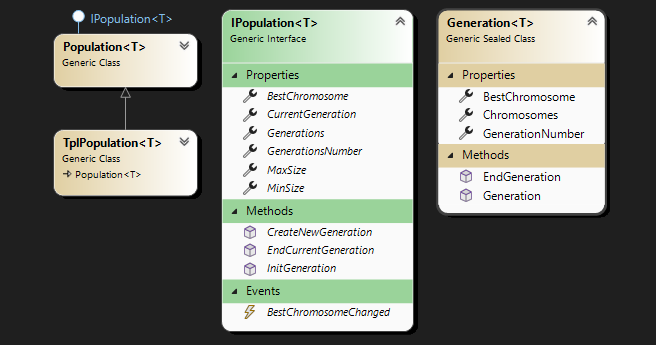


Рисунок 3.9 Структура классов популяции и поколения

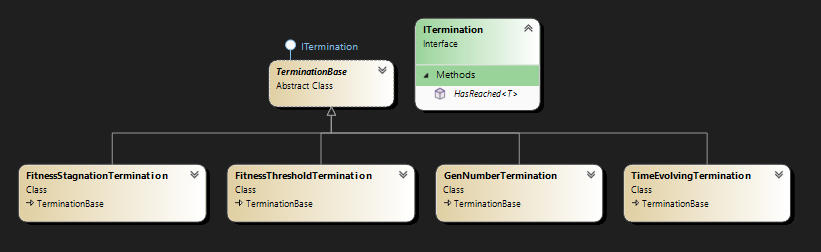


Рисунок 3.10 Структура классов Условия остановки

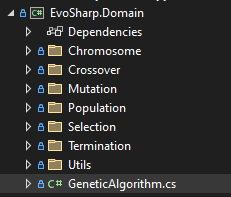


Рисунок 3.8 Файловая структура проекта

Были реализованы все этапы алгоритма. Каждый этап включает в себя несколько вариаций, пользователь библиотеки сможет настроить под свои задачи конкретный набор действий и операторов. В центре всего проекта стоит класс *GeneticAlgorithm*, он принимает начальную популяцию и ссылки на классы операторов алгоритма, так же в ответственность этого класса входит запуск алгоритма посредствам метода *Start.* Пример использования библиотеки на примере задачи OneMax, которая описывалась в 1 главе можно увидеть в приложении Б.

1. TDD & Unit Testing

При модульном тестировании отдельные модули или компоненты кода тестируются изолированно от остальной системы. Это позволяет разработчикам протестировать поведение каждого модуля и убедиться, что оно правильное. При разработке библиотеки я решил прибегнуть к подходу известному как **TDD.**

Разработка через тестирование (**TDD**) — это подход к разработке программного обеспечения, при котором тесты пишутся для фрагмента кода до того, как будет написан сам код. Тесты используются для управления разработкой кода и обеспечения его правильности и соответствия требованиям.

Лучшее качество кода: TDD помогает обеспечить тщательное тестирование кода и обнаружение ошибок на ранних этапах процесса разработки, что приводит к более высокому качеству кода. Более быстрое время разработки: выявляя ошибки на ранних этапах процесса разработки, TDD может фактически сэкономить время в долгосрочной перспективе, сократив количество времени, затрачиваемого на отладку.

Хоть и TDD имеет столько плюсов это не панацея для разработки ПО и подходит не для каждого проекта. Однако во многих случаях он может быть полезным инструментом для улучшения качества и эффективности кода. TDD может быть сложно реализовать для определенных типов кода, таких как код, который сильно зависит от внешних зависимостей, или код, взаимодействующий с пользовательским интерфейсом.

Будем тестировать основные этапы алгоритма и их вариации: операторы выборки, скрещивания имутации, условие остановки, и класс популяции. В приложении A находиться некоторые юнит-тесты написанные для них. Используя при этом, возможно, фреймворка NUnit. Каждый юнит-тест будет структурирован по шаблону ААА ­(Arange Act Assert) предоставляет простую единообразную структуру для всех тестов это единообразие дает большое преимущество: привыкнув к нему, можно легко прочитать и понять любой тест.

Все базовые классы наследуются от интересов, на первый взгляд это кажется избыточным, но для тестирования это играет особую роль. Это помогает в тестировании основных поведений объекта без внедрения жестких зависимостей и облегчает разработку.

1. Вывод

В данной главе была рассмотрена роль генетических алгоритмов в искусственном интеллекте. Было выяснено, что ГА широко используются в задачах оптимизации, машинном обучении, обработке изображений и других областях искусственного интеллекта.

Была проведена разработка библиотеки для реализации ГА на основе TDD тестирования. Это позволяет разработчикам уверенно изменять код и проверять его на корректность без риска нарушения функциональности.

Кроме того, была представлена задача обучения самоуправляемой машины с помощью генетического алгоритма в Unity. Первоначальные результаты практического применения показали, что ГА хорошо подходят для оптимизации параметров ИИ, и наши результаты показывают, что ГА может находить значения параметров, которые приводят к более быстрому обучению и лучшей производительности в выбранных нами задачах. Таким образом, мы предоставляем дополнительные доказательства того, что эвристический поиск, выполняемый эволюционными (генетическими) вычислительными алгоритмами, является жизнеспособным инструментом для оптимизации производительности обучения с подкреплением в нескольких областях.

В целом, генетические алгоритмы являются мощным инструментом в искусственном интеллекте и могут применяться для решения различных задач. Разработка библиотеки позволяет лучше понять принципы работы и использования генетических алгоритмов в практических задачах.

# **ЗАКЛЮЧЕНИЕ**

Мы живем в век искусственного интеллекта и генетические алгоритмы хорошо приспособлены к этому и могут стать мощным оружием в арсенале, когда придется решать очередную задачу. Как было показано, генетические алгоритмы и родственные им методы применимы к самым разным задачам практически в любых областях вычислений и техники. Для применения генетического алгоритма нужен лишь способ представить решение и получить его численную оценку – или хотя бы сравнить два решения.

Созданная библиотека хорошо адаптирована для реальных проектов в использовании, из основных плюсов можно выделить ее гибкость и простота в использовании.

В ходе выполнения работы были разработаны и решены следующие задачи:

1. Рассмотрены различия между генетическими алгоритмами и традиционными методами.
2. Исследовано влияние разных подходов в выборе операторов на конечный результат.
3. Реализовано использование ГА в нейронных сетях в частности обучения с подкреплением в задаче self-driving car.
4. Написаны Unit тесты для TDD подхода.
5. Реализована библиотека генетического алгоритма с гибкой настройкой всех параметров.

В планах на дальнейшее развитие этой темы – написание API для запросов алгоритма и непосредственно проверка действенности библиотеки в различных задачах оптимизации.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

* 1. Панченко Т. В. Генетические алгоритмы: учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань: Издательский дом «Астраханский университет», 2007. — 87с.
  2. Вирсански Э. Генетические алгоритмы на Python пер. с англ. А. А. Слинкина. ДМК пресс, 2020. – 286 с.: ил.
  3. Копец Дэвид Классические задачи Computer Science на языке Python. — СПб.: Питер, 2020. — 256 с.: ил. — (Серия «Библиотека программиста»).
  4. Кононюк А. Е. Дискретно-непрерывная математика. Часть 3. Генетические Алгоритмы. — В 12-и кн. Кн. 10 — К.: 2017. — 444 с.
  5. Генетические алгоритмы: цикл статей [Электронный ресурс]. – 2021. – Режим доступа: https://proproprogs.ru/ga – Дата доступа: 11.10.2022

# **ПРИЛОЖЕНИЯ**

# *Приложение A*

[Test]

public void Cross\_ParentsWithTwoGenes\_Cross() {

// Arrange

var target = new OnePointCrossover(0);

var chromosome1 = Substitute.For<Chromosome>(2);

chromosome1.ReplaceGenes(0, new []{ new Gene(1), new Gene(2) });

chromosome1.CreateNew().Returns(Substitute.For<Chromosome>(2));

var chromosome2 = Substitute.For<Chromosome>(2);

chromosome2.ReplaceGenes(0, new[]{ new Gene(3), new Gene(4) });

chromosome2.CreateNew().Returns(Substitute.For<Chromosome>(2));

// Act

var actual = target.Cross(new List<IChromosome>{ chromosome1, chromosome2 });

// Assert

Assert.AreEqual(1, actual[0].GetGene(0).Value);

Assert.AreEqual(4, actual[0].GetGene(1).Value);

Assert.AreEqual(3, actual[1].GetGene(0).Value);

Assert.AreEqual(2, actual[1].GetGene(1).Value);

}

[Test]

public void Mutate\_NoProbality\_NoInsertion() {

// Arrange

var target = new InsertionMutation();

var chromosome = Substitute.For<Chromosome>(4);

chromosome.ReplaceGenes(0, new[] {

new Gene(1), new Gene(2), new Gene(3), new Gene(4),

});

// Act

target.Mutate(chromosome);

// Assert

Assert.AreEqual(4, chromosome.Length);

Assert.AreEqual(1, chromosome.GetGene(0).Value);

Assert.AreEqual(2, chromosome.GetGene(1).Value);

Assert.AreEqual(3, chromosome.GetGene(2).Value);

Assert.AreEqual(4, chromosome.GetGene(3).Value);

}

[Test]

public void SelectChromosomes\_InvalidNumber\_Exception() {

// Arrange

var target = new TruncationSelection();

// Assert

Assert.Catch<ArgumentOutOfRangeException>(() =>

{

// Act

target.SelectChromosomes(-1, null);

},

"The number of selected chromosomes should be at least 2.");

}

# *Приложение Б*

var chromosome = new IntChromosome(32, 0, 1);

var population = new Population<int>(50, 100, chromosome);

var fitness = new Func<IChromosome<int>, double>(c => c.Genes.Sum());

var selection = new RankSelection<int>();

var crossover = new UniformCrossover(0.5f);

var mutation = new TworsMutation();

var termination = new GenNumberTermination { MaxGenCount = 50 };

var ga = new GeneticAlgorithm<int>(population, fitness, selection, crossover, mutation)

{

    Termination = termination

};

var latestFitness = 0.0;

ga.GenerationRan += (\_, \_) =>

{

    var bestChromosome = ga.Population.BestChromosome as IntChromosome;

    var bestFitness = bestChromosome.FitnessValue.Value;

    if (bestFitness != latestFitness)

    {

        latestFitness = bestFitness;

        var phenotype = bestChromosome.ToString();

        Console.WriteLine($"Generation {ga.Population.GenerationsNumber}: Fitnes = {bestFitness}, best gene: {ga.Population.BestChromosome}");

    }

};

ga.Start();