**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра технологий программирования**

ДОСКОЧ

Роман Дмитриевич

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БИБЛИОТЕКИ ДЛЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ**

Дипломная работа

|  |  |
| --- | --- |
| Научный руководитель: | |
| кандидат технических наук, | |
| доцент И.С. Войтешенко | |
| Допущена к защите  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2023 г  Зав. кафедрой технологий программирования  доктор технических наук, профессор, Заслуженный деятель науки РБ А.Н. Курбацкий | |

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

Минск, 2023

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 7](#_Toc135574872)

[**Глава 1 ОСНОВЫ ГЕНЕТИЧЕСКОГО АЛГОРИТМА** 8](#_Toc135574873)

[1.1 История возникновения генетического алгоритма 8](#_Toc135574874)

[1.2 Что такое генетический алгоритм 9](#_Toc135574875)

[1.3 Основные понятия и определения 10](#_Toc135574876)

[1.4 Примеры использования генетических алгоритмов. 12](#_Toc135574877)

[1.5 Преимущества и недостатки генетического алгоритма 14](#_Toc135574878)

[1.6 Выводы 16](#_Toc135574879)

[**Глава 2 ИССЛЕДОВАНИЕ РАЗЛИЧНЫХ ПОДХОДОВ ГЕНЕТИЧЕСКОГО АЛГОРИТМА** 18](#_Toc135574880)

[2.1 Структура генетического алгоритма 18](#_Toc135574881)

[2.2 Операторы отбора 20](#_Toc135574882)

[2.3 Операторы скрещивания 22](#_Toc135574883)

[2.4 Операторы Мутации 25](#_Toc135574884)

[2.5 “Hello World” в мире генетического алгоритма 27](#_Toc135574885)

[2.6 Эксперименты с параметрами ГА 29](#_Toc135574886)

[2.7 Выводы 32](#_Toc135574887)

[**Глава 3 ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ И РАЗРАБОТКА БИБЛИОТЕКИ** 34](#_Toc135574888)

[3.1 Обучение с подкреплением 34](#_Toc135574889)

[3.2 Приложение Генетического алгоритма в RL 37](#_Toc135574890)

[3.3 Структурная схема библиотеки 40](#_Toc135574891)

[3.4 TDD & Unit Testing 43](#_Toc135574892)

[3.5 Задача коммивояжера 45](#_Toc135574893)

[3.6 Публикация NuGet 49](#_Toc135574894)

[3.7 Выводы 51](#_Toc135574895)

[**ЗАКЛЮЧЕНИЕ** 52](#_Toc135574896)

[**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ** 53](#_Toc135574897)

[**ПРИЛОЖЕНИЯ** 54](#_Toc135574898)

[*Приложение А* 54](#_Toc135574899)

[*Приложение Б* 55](#_Toc135574900)

[*Приложение В* 56](#_Toc135574901)

[*Приложение Г* 58](#_Toc135574902)

[*Приложение Д* 59](#_Toc135574903)

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

|  |  |
| --- | --- |
| ГА | Генетический алгоритм – эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного отбора. |
| 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. |
| NuGet | NuGet — это менеджер пакетов для платформы разработки Microsoft, который позволяет разработчикам легко управлять сторонними библиотеками и инструментами для своих приложений. |
| TSP | TSP (задача коммивояжёра) — известная задача в области информатики и математики, которая включает в себя поиск кратчайшего возможного маршрута, по которому продавец может посетить заданный набор городов и вернуться в исходную точку, не посещая ни один город более одного раза. |
|  |  |

**РЕФЕРАТ**

Дипломная работа, 52 c., 38 рис., 5 приложений.

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

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

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

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

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

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

**РЭФЕРАТ**

Дыпломная работа, 52 ст., 38 мал., 5 дадаткаў.

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

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

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

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

**Вынікамі з'яўляюцца** **–** бібліятэка класаў з зручным API для выкарыстання магчымасцяў генетычнага алгарытму.

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

**ABSTRACT**

Graduate Work, 52 p., 38 illustrations, 5 appendixes.

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

**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 develop 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** **–** class library with a convenient API for using the capabilities of the genetic algorithm.

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

# **ВВЕДЕНИЕ**

В последние годы генетический алгоритм (ГА) стал популярным методом оптимизации благодаря его способности эффективно решать сложные задачи оптимизации в различных областях. Эвристическая природа ГА позволяет исследовать обширные области решений и находить решения близкие к оптимальным.

В главе 1 «Основы генетического алгоритма» приводится краткое введение в теорию и принципы работы генетического алгоритма. Рассматриваются также различия между генетическим алгоритмом и традиционными методами, и описаны 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 при подходе обучения с подкреплением на примере самоуправляемой машины в Unity 3D. Так же описана структура библиотеки классов генетического алгоритма и ее использование в задачах OneMax и TSP. Написаны юнит-тесты для подхода проектирования TDD.

# **Глава 1 ОСНОВЫ ГЕНЕТИЧЕСКОГО АЛГОРИТМА**

Применение генетического алгоритма в оптимизации и моделировании представляет собой перспективное и актуальное направление. В круг возможный применений можно выделить задачи на графах, компоновки и составления расписаний, а также настройки и обучения искусственных нейронных сетей, и многие другие задачи.

В этой главе приводится краткое введение в теорию генетического алгоритма, описаны сценарии, в которых имеет смысл его применения.

## 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 «Генетический алгоритм в поиске, оптимизации и машинном обучении», в котором был представлен всесторонний обзор области и укреплены позиции ГА как основного метода оптимизации.

В наши дни генетический алгоритм применяется 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 генетического алгоритма отражает продолжающийся поиск эффективных и действенных методов оптимизации, которые могут использовать принципы естественного отбора и генетической наследственности.

## Что такое генетический алгоритм

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 группа методов оптимизации, которые могут определить экстремальное значение целевой функции. Однако не всегда можно с уверенностью сказать, что найденное значение является глобальным экстремумом. Когда вместо глобального экстремума обнаруживается локальный, такой случай называется преждевременной сходимостью. Так же существует проблема длительного времени расчета у точных методов оптимизации. Для решения этих задач ведется поиск новых алгоритмов, и одним из них является генетический алгоритм.

**Генетический алгоритм** – это семейство поисковых алгоритмов, идеи которого подсказаны принципами эволюции в природе. Имитируя процессы естественного отбора и воспроизводства, генетический алгоритм может находить высококачественные решения задач, включающих поиск, оптимизацию и обучение [2].

В силу того что в основу идеи алгоритма входит аналогия с естественным отбором, следует обширная применимость к сложным задачам, таким задачам что традиционные алгоритмы могут не справиться.

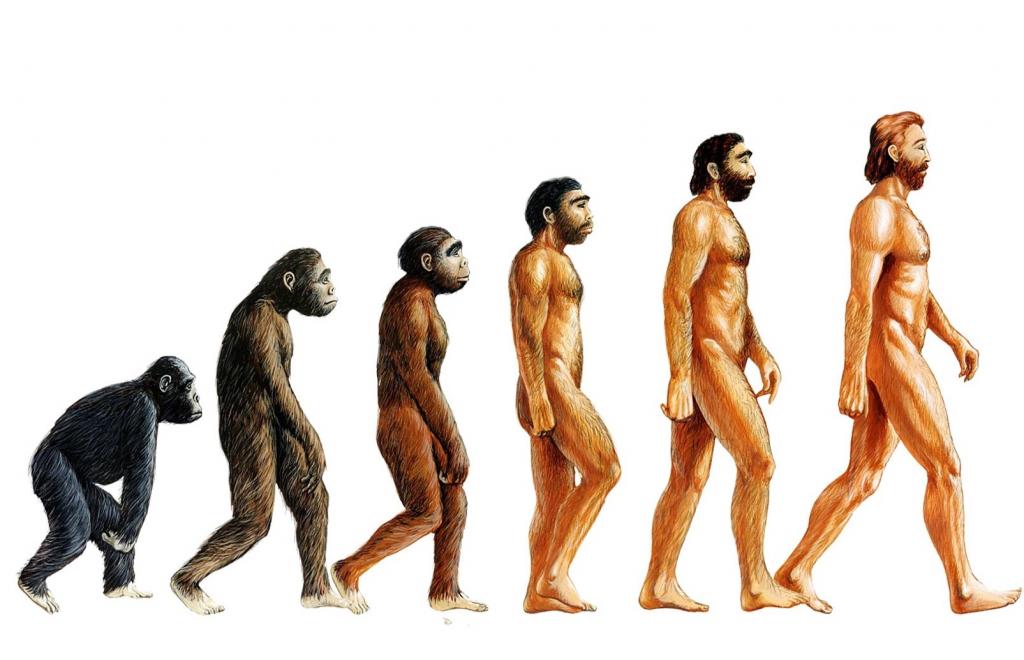


Рисунок 1.1 Эволюция по Дарвину

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 генетического алгоритма 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:

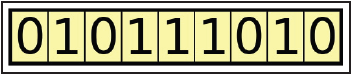


Рисунок 1.2 Простое двоичное кодирование хромосомы

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.

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.

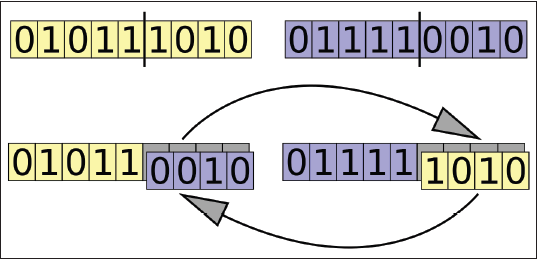


Рисунок 1.3 Операция скрещивания двух двоично-кодированных хромосом

***Мутация (Mutation).*** Основная цель мутации - привнести новую генетическую информацию в популяцию и предотвратить преждевременную сходимость. Мутация работает путем случайного изменения значения одного или нескольких генов в хромосоме. Одним из примеров мутации можно считать инвертирования одного гена 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.

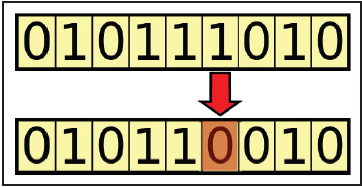


Рисунок 1.4 Применение оператора мутации к двоично-кодированной хромосоме

## Примеры использования генетических алгоритмов.

Генетический алгоритм часто и эффективно применяется в многочисленных пространствах задач. Его часто используют для сложных задач, которые не требуют точных 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**.

Генетический алгоритм (ГА) применяется для решения широкого круга задач оптимизации и поиска в различных областях, в том числе:

* **Оптимизация проектирования**: ГА используется для оптимизации 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 применяется для решения различных финансовых и экономических проблем, включая управление рисками и оценку опционов.
* **Обработка изображений и сигналов**: ГА 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 используется 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 используется для решения задач составления расписания и планирования, таких как составление расписания производства, планирование проектов и планирование мастерских.
* **Биоинформатика**: ГА используется для решения задач биоинформатики, таких как анализ экспрессии генов, предсказание структуры белка и выравнивание последовательностей.
* **В компьютерном искусстве:** ГА иногда применяется для имитации фотографий с помощью стохастических методов. В пример возьмем 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 генетического алгоритма

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.***  Задачи глобальной оптимизации — это трудоемкие задачи, которые включают в себя поиск наилучшего возможного решения из всех возможных решений в пределах заданного пространства поиска. Традиционные алгоритмы оптимизации обычно используют математические модели и аналитические методы для определения оптимального решения. Однако эти алгоритмы могут сталкиваться с проблемами, которые имеют большое количество переменных или нелинейные целевые функции, что может затруднить поиск глобального оптимального решения. Напротив, ГА — это тип алгоритма оптимизации, основанный на принципах естественного отбора и эволюции. Что позволяет им выполнять поиск по всему пространству поиска и избегать преждевременную сходимость.

***Применимость к сложным задачам.*** Генетический алгоритм может быть применен к задачам со сложным математическим представлением, так как для решения он нуждаются только в значении функции приспособленности каждого индивидуума, не требуя знания производных и других свойств функции. Кроме того, ГА может быть применен к задачам, которые не имеют математического представления, например, в случаях, когда оценка приспособленности основана на мнении человека. В таких случаях генетический алгоритм может 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 целевую функцию, основанную на оценках пользователей. Алгоритм будет работать, даже если нет математического представления и невозможно напрямую рассчитать счет данной комбинации.

**Параллелизм.** ГА по своей сути распараллеливаемы, что означает, что их можно легко запускать параллельно на нескольких процессорах или ядрах. Это связано с тем, что вычисления, в основном вычисление фитнес-фугкции, в генетическом алгоритме структурированы вокруг совокупности хромосом, каждая из которых может быть оценена независимо. В течение каждого поколения популяция модифицируется с помощью генетических операторов, таких как мутация, скрещивание и отбор. Эти операторы могут применяться независимо к каждому элементу совокупности, что упрощает распараллеливание вычислений на нескольких процессорах или ядрах. Распараллеливание может помочь сократить время, необходимое для решения крупных задач оптимизации, и может обеспечить другие преимущества, такие как отказоустойчивость и масштабируемость.

***Непрерывное обучение.*** Непрерывное обучение — это способность алгоритма адаптироваться и извлекать уроки из новых данных по мере их поступления. В контексте ГА непрерывное обучение может относиться к способности алгоритма корректировать свои параметры или обновлять свою популяцию на основе новой информации. ГА динамически корректирует свои параметры в зависимости от приспособленности текущей популяции. Например, частота мутаций или скрещиваний алгоритма может быть скорректирована на основе разнообразия или приспособленности популяции. Это может помочь алгоритму более эффективно исследовать пространство поиска и избежать преждевременной сходимости.

Итак, преимущества мы обсудили, теперь перейдем к **недостаткам** или же ограничениям генетического алгоритма. 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 ГА 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 ГА 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.*** Существует несколько причин, по которым в ГА может произойти преждевременная сходимость. Одна из причин заключается в том, что совокупность решений может быть недостаточно разнообразной. Это может произойти, когда вероятности мутаций или других операторов слишком низки или когда критерии отбора слишком строгие.

Другая причина, заключается в том, что ГА может сходиться слишком быстро из-за давления окружающей среды, такого как давление отбора. Давление отбора возникает, когда ГА отбирает наиболее приспособленных особей в популяции для размножения, что может привести к потере разнообразия, если наиболее приспособленные особи похожи друг на друга.

Для решения проблемы преждевременной конвергенции в ГА можно использовать несколько методов. Один из подходов заключается в увеличении частоты мутаций или других генетических воздействий для увеличения разнообразия популяции. Другой подход заключается в использовании многоцелевого ГА, который оптимизируется для нескольких целей одновременно, что может помочь алгоритму исследовать различные части пространства поиска.

***Отсутствие гарантированного решения.*** Использование ГА 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 ГА находит хорошие решения за разумное время.

## Выводы

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

* **Задачи со сложным математическим представлением.** Поскольку ГА 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.** ГА не требует математического представления задачи, так как достаточно только получить значение оценки или существует метод сравнения двух решений.
* **Задачи с зашумленной окружающей средой.** ГА устойчив к зашумленным данным, например прочитанным с датчика или основанным на оценках, сделанных человеком.
* **Задачи, с динамически изменяющейся средой.** ГА может 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 РАЗЛИЧНЫХ ПОДХОДОВ ГЕНЕТИЧЕСКОГО АЛГОРИТМА**

В этой главе будут рассмотрены все шаги алгоритма, особое внимание будет уделено операторам наиболее изменяющих популяцию. Так же на примере решения задачи OneMax, называемой “Hello World” в мире ГА, будет разобран базовый вариант использования, так же будут рассмотрены разные подходы и степень влияния выбора 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 является создание начальной популяции потенциальных решений, где каждое такое решение-кандидат представляет отдельного индивидуума в популяции. Говоря простыми словами, начальная популяция – это 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** изменяет случайные гены 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**. Алгоритм завершится, если найдет решение, отвечающее желаемым критериям, таким как определенный уровень пригодности.
* **Достигнуто максимальное количество поколений.** Алгоритм завершиться по достижению определенного количества поколений, это полезно если нужно сравнить с предыдущей попыткой, задав точное число поколений.
  + **За последнее время не наблюдается заметных улучшений**. ГА может быть прекращен, когда приспособленность популяции приблизится к стабильному значению. Такой подход полезен, когда пространство поиска относительно невелико, и ожидается, что оптимальное решение будет найдено быстро. Это можно реализовать путем запоминания наилучшей приспособленности в каждом поколении, и сравнением его с со значениями в предыдущих поколениях.

1. Операторы отбора

Отбор первый из операторов, выполняется для того чтобы выбрать из текущего поколения те хромосомы, которые будут использованы в качестве родителей следующего поколения. 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.

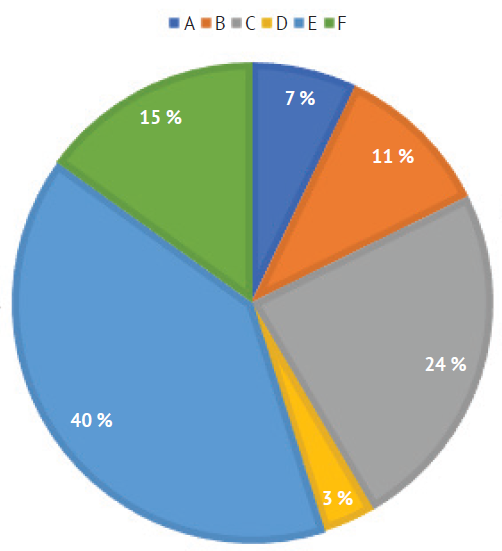


Рисунок 2.1 Пример отбора по правилу рулетки

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.

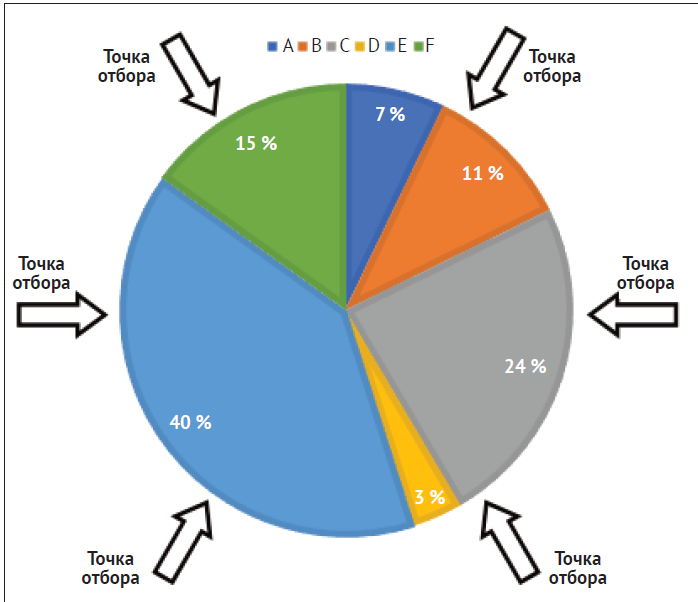


Рисунок 2.2 Пример Стохастической универсальной выборки

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. Принцип такого отбора можно описать следующим образом:

* 1. Из популяции случайным образом выбирается подмножество хромосом.
  2. Вычисляется значение фитнес-функции для каждого из них.
  3. Затем выбирается победитель, победителем турнира считается хромосома с самой высокой пригодностью
  4. Турнир повторяется до тех пор, пока не будет выбрано желаемое количество родителей.

Пример подобного поведения можно увидеть на рисунке ниже.

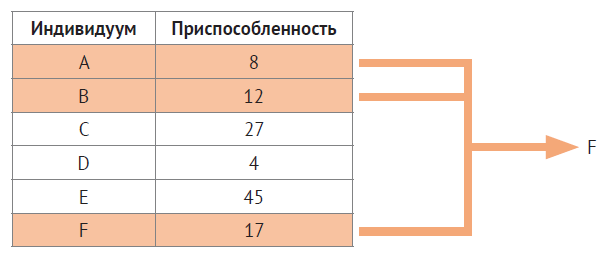


Рисунок 2.3 Пример турнирного отбора на примере с тремя участниками

Преимущество турнирного отбора в том, что он позволяет сбалансировать разнообразие. Размером турнира можно определить количество хромосом, участвующих в каждом раунде отбора. Выбор турнира также можно изменить, добавив параметр, называемый «давление отбора». Высокое давление отбора означает, что с большей вероятностью будет выбран наиболее приспособленный индивидуум, а низкое, следовательно, указывает на более случайный отбор. 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 отбора 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 и другие методы отбора в совокупности с элитарным.

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.

Некоторые из 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 – индекс в массиве хромосомы для последующего скрещивания. 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.

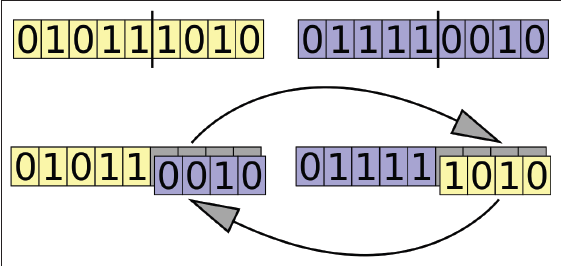


Рисунок 2.4 Пример одноточечного скрещивания

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.

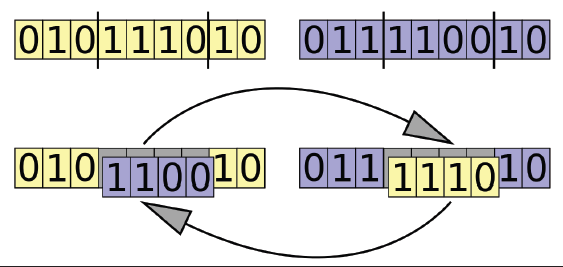


Рисунок 2.5 Пример двухточечного скрещивания

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.

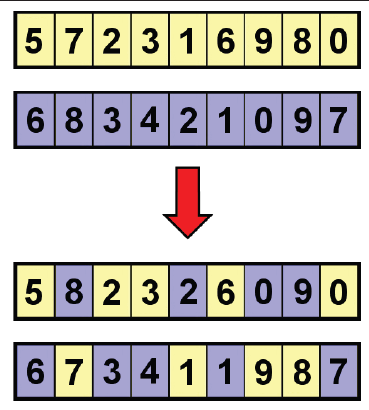


Рисунок 2.6 Пример равномерного скрещивания

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 оператора мутации:

* 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.*** Принцип следующий: 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.*** В хромосоме выбирается случайная последовательность генов, и порядок 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.

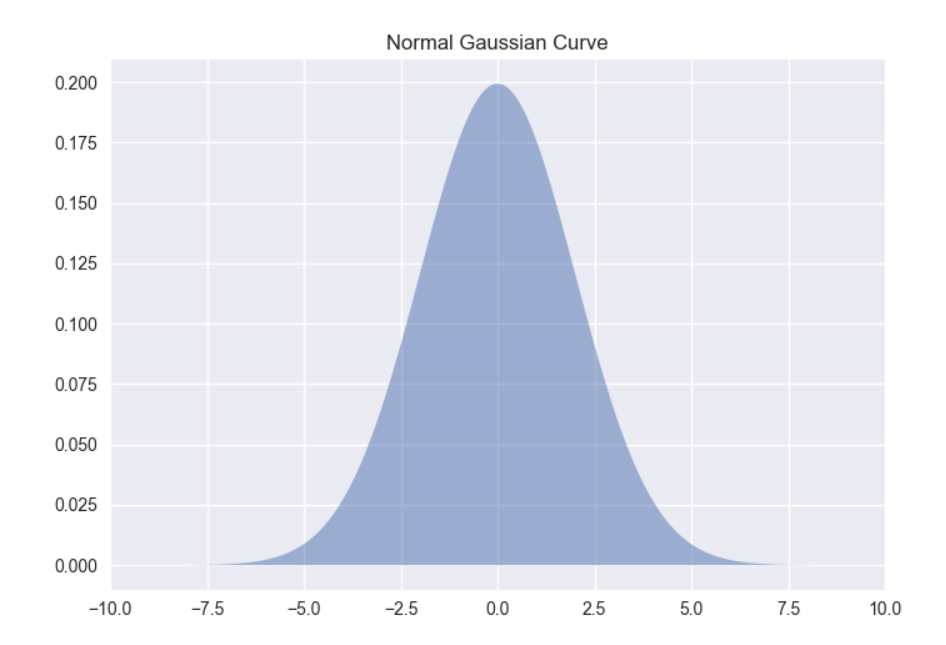


Рисунок 2.10 График нормального распределения

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 генетического алгоритма

При знакомстве с миром генетического алгоритма в первую очередь решают простую задачу оптимизации OneMax. Ее задача в том, чтобы найти двоичную строку заданной длины с максимальной суммой. Очевидно, что решением считается строка, состоящая из всех единиц, тем самым максимизируя ее сумму. Хоть задача является тривиальной, но на ее примере хорошо показаны этапы и суть алгоритма.

Наша цель – найти решение, которое бы давало максимальную сумму цифр этого списка. Логично предположить, что фитнес-функцией будет являться просто сумма 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 ГА создан целый ряд каркасов на Python, например DEAP, GAFT, Pyevolve и PyGMO. Остановимся на каркасе DEAP, основные преимущества: простота в использовании, широкий набор функций, подробная документация.

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 фитнес-функцию, которая является суммой всех элементов в индивидууме.

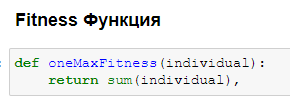


Рисунок 2.11 Фитнес-функция для задачи OneMax

Для генерации начальной популяции можем воспользоваться ToolBox.

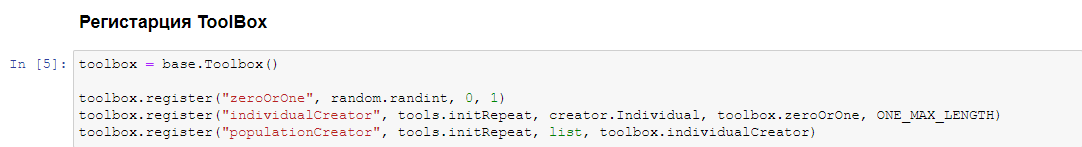


Рисунок 2.12 Настройка начальной популяции

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.

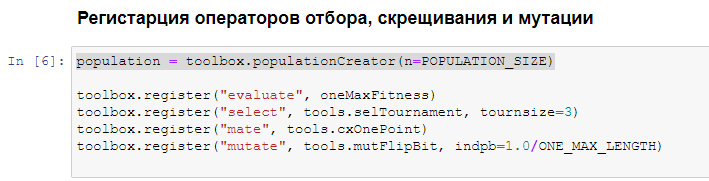


Рисунок 2.13 Настройка операторов отбора, скрещивания и мутации

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

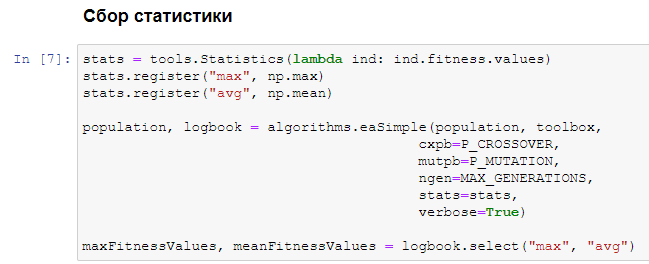


Рисунок 2.14 Сбор статистики

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

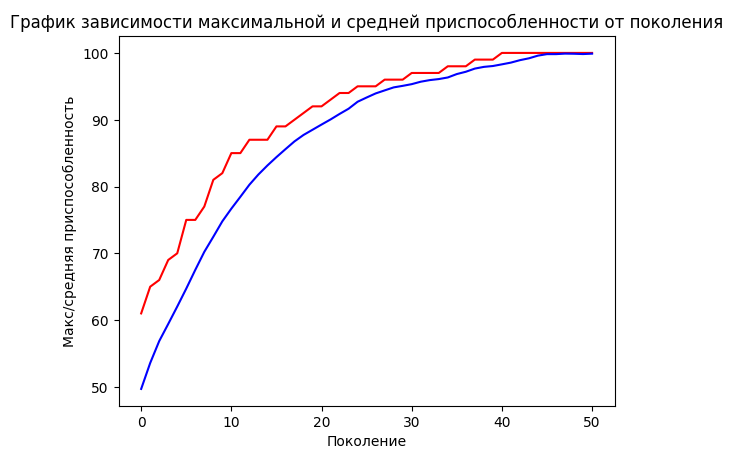


Рисунок 2.15 График зависимости максимальной и   
средней приспособленности от поколения

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:

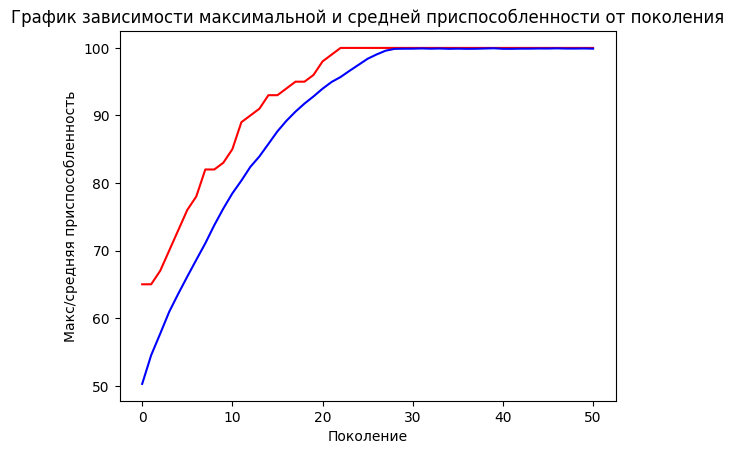


Рисунок 2.16 Результат увеличения размера популяции

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, чтобы 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)

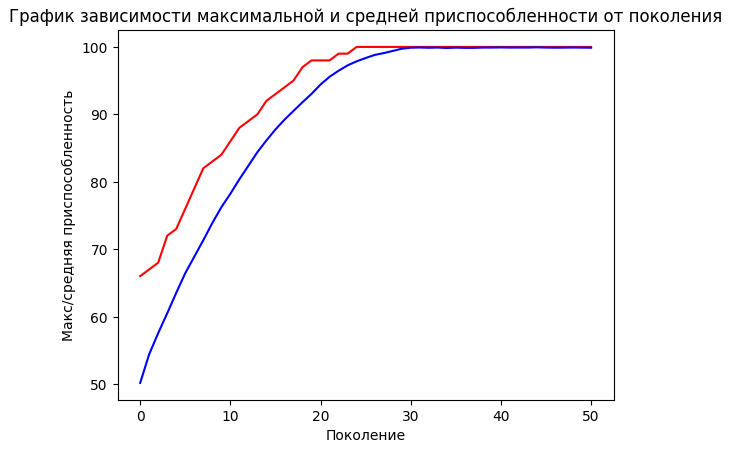


Рисунок 2.17 Результат изменения оператора скрещивания на двуточечное

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 (независимая вероятность обмена каждым атрибутом) в десять раз. В результате поведение алгоритма становится неустойчивым.

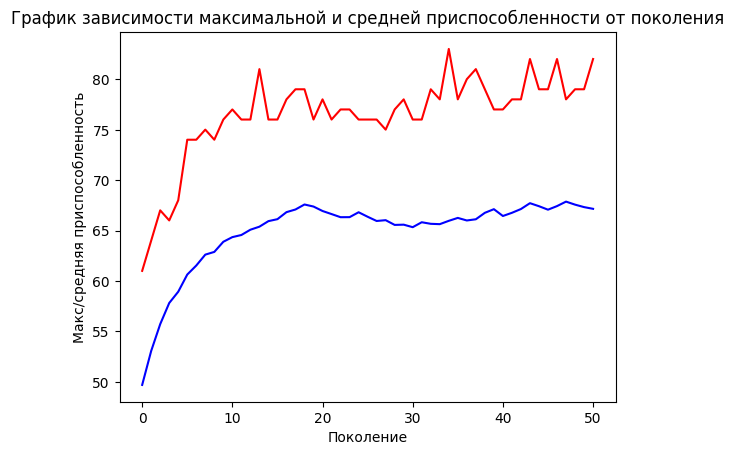


Рисунок 2.18 Увеличение вероятности мутации каждого индивидуума

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. Выводы

В этой главе мы познакомились с общей структурой генетического алгоритма. Затем рассмотрели детали: создание популяции, вычисление фитнес-функции, 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. 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 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. ГА находит 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. Наиболее приемлемым способом настройки весовых коэффициентов искусственных нейронных сетей можно считать генетические алгоритмы [7].

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 ГА 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 сущностей (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 [6]. 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 RL можно считать то, что ему не требуется наличие размеченных обучающих данных, т. е. пар, состоящих из входа и 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.*** Концептуальная модельпредставляет 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.

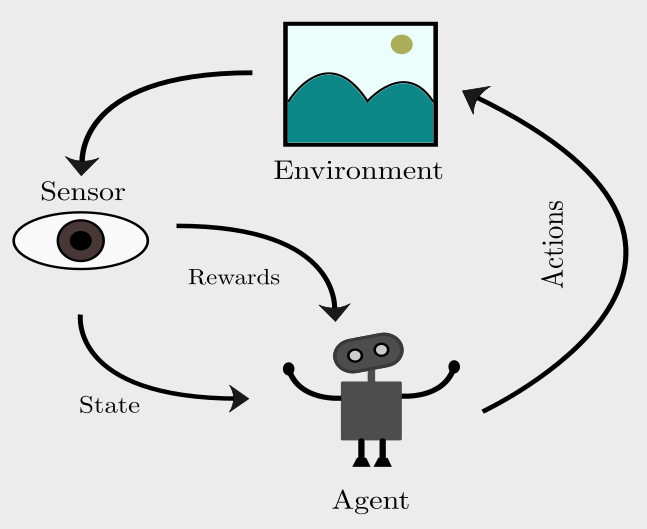


Рисунок 3.1 Концептуальная схема RL

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.

Существует бесчисленное множество статей о нейронных сетях и их обучении с использованием ГА. Существует также множество подобных работ, демонстрирующих способность обучения нейронной сети играть в некоторые компьютерные игры. В качестве примера можно привести проект OpenAI, представляющий собой среду разработки для создания алгоритмов обучения нейронных сетей. Одна из его частей реализует Arcade Learning Environment (ALE) для игры Atari. Эта среда де-факто стала эталонной платформой для тестирования производительности различных алгоритмов, в основном RL.

В основном для тренировки RL используют, такие методы как Deep-Q Networks (DQN), Actor Advantage Critic (A2C) и эволюционные стратегии, но оказывается, что подход с ГА конкурирует с вышеупомянутыми алгоритмами и даже превосходит их в некоторых задачах. В статье Deep Neuroevolution [8], были представлены результаты сравнения популярных алгоритмов обучения и подхода ГА при решении задачи игр Atari. Результаты исследования показали, что ГА работает быстро, позволяя обучать Atari за ∼4 часа на одной рабочей машине или ∼1 час, распределенный на 720 процессорах. Эта работа показала, что ГА не только имеет более низкие требования к времени выполнения, но и достигает лучших результатов, чем более трудоемкие алгоритмы RL.

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 как можно больше кругов по виртуальной трассе при этом не врезавшись. Выглядит это следующим образом:

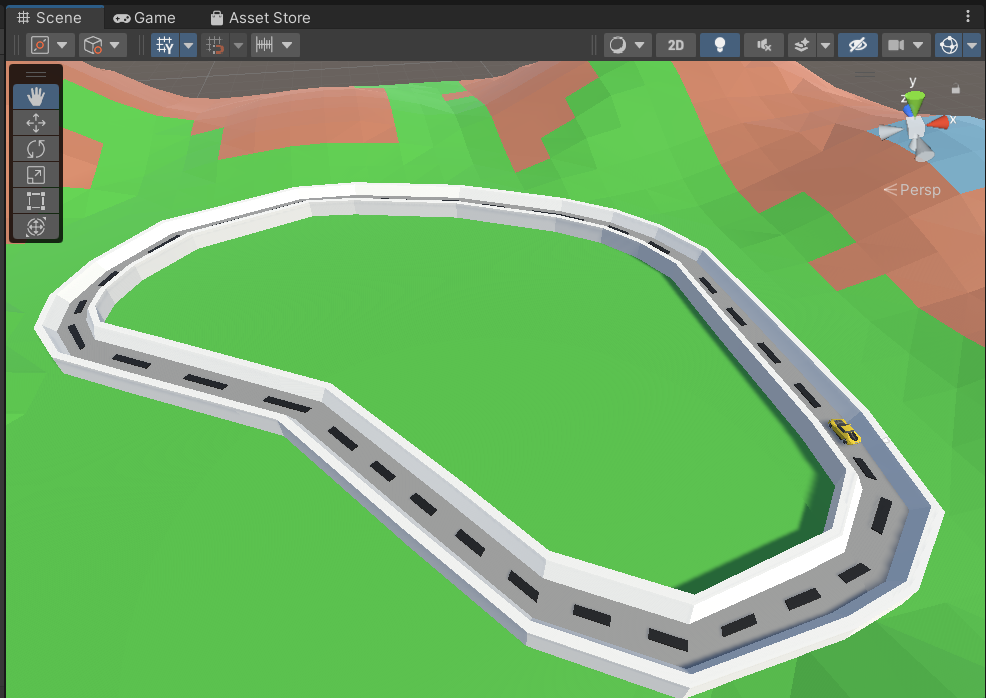


Рисунок 3.2 Окружающая среда виртуальной трассы

Для эффективного применения ГА в любой задаче нужно правильно настроить параметры, после нескольких 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.

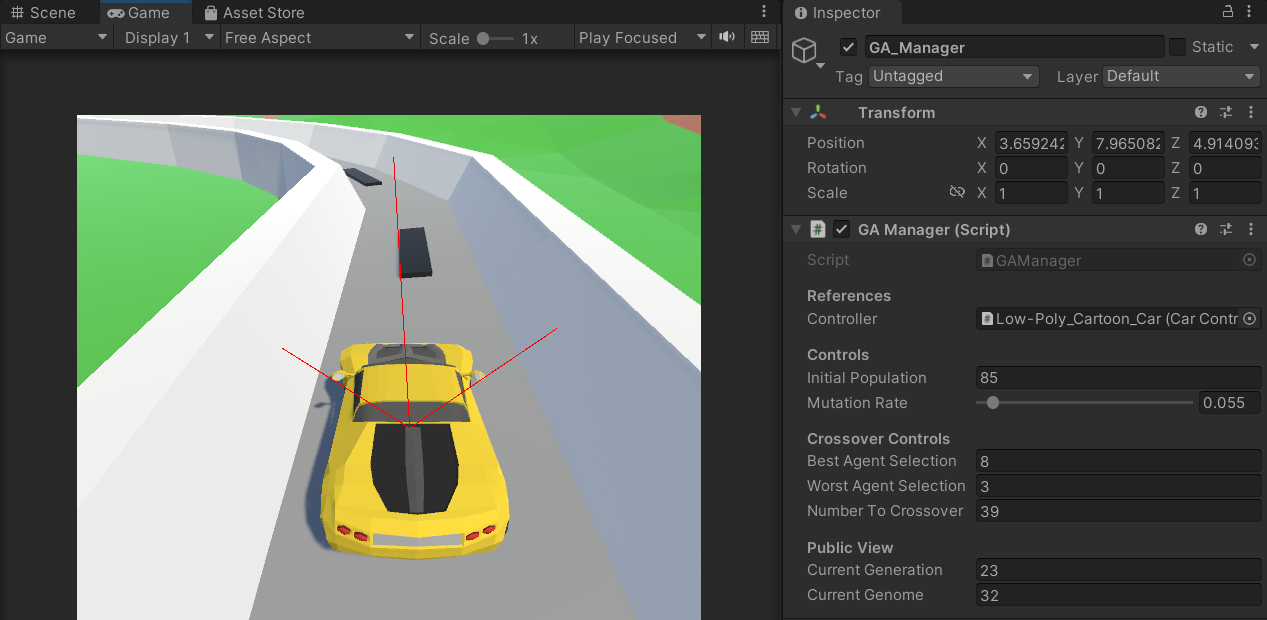


Рисунок 3.3 Трассировочные лучи

Принцип работы в следующем: у виртуальной машины есть 3 трассировочных 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).

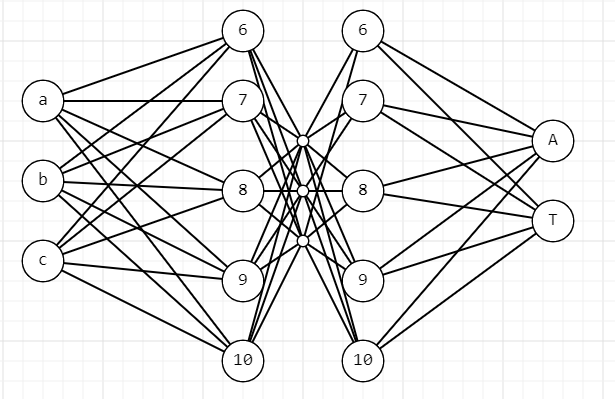


Рисунок 3.4 Схема нейронной сети

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.

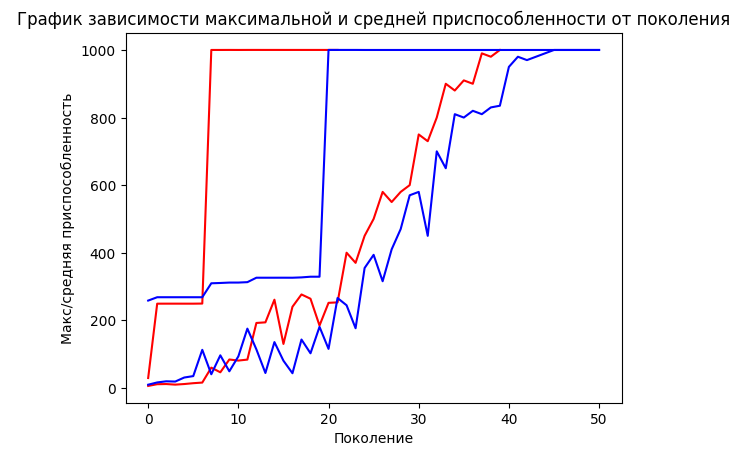


Рисунок 3.5 Сравнение результатов разных параметров

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

1. Структурная схема библиотеки

Изучив базовую теорию ГА и рассмотрев применение на практике, можем перейдем к основной задаче – реализации библиотеки классов ГА.

Для начала спроектируем будущую структурную схему библиотеки. Для проектирования схемы сначала мы должны определиться какие основные классы будут фигурировать в основном слове бизнес логики, так называемым – Domain слое. Сам алгоритм похож на конвейер и поэтому логично создать отдельные группы сущностей на каждый структурный элемент.

Определим базовые сущности: Gene, Chromosome, Generation, Population, Fitness. Так же нужно определить этапы операторов 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.

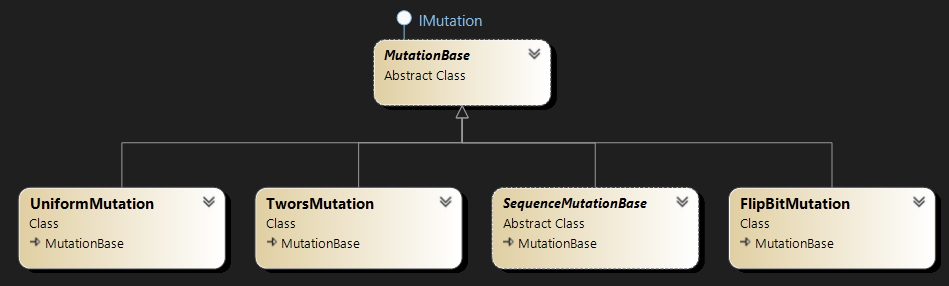


Рисунок 3.6 Структура классов оператора мутации

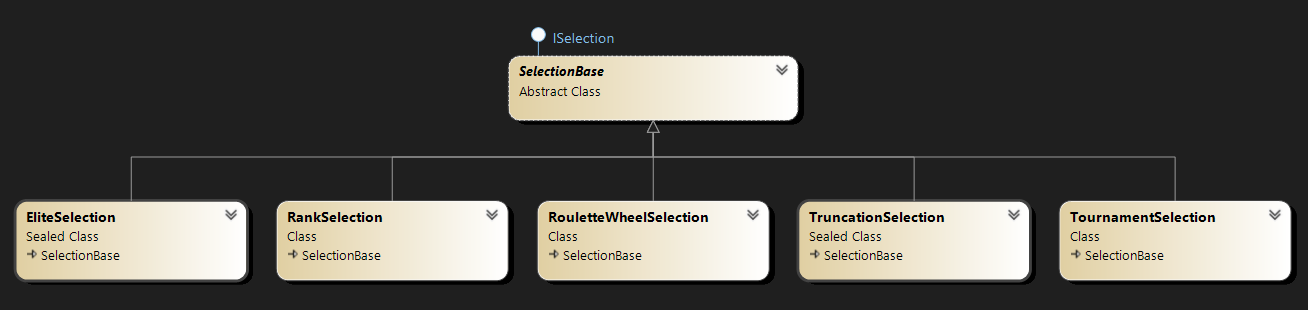


Рисунок 3.7 Структура классов оператора отбора

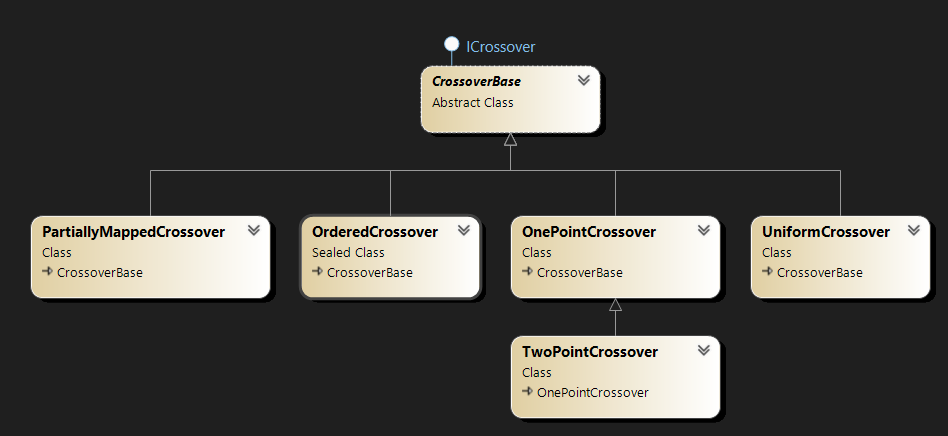


Рисунок 3.8 Структура классов оператора скрещивания

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

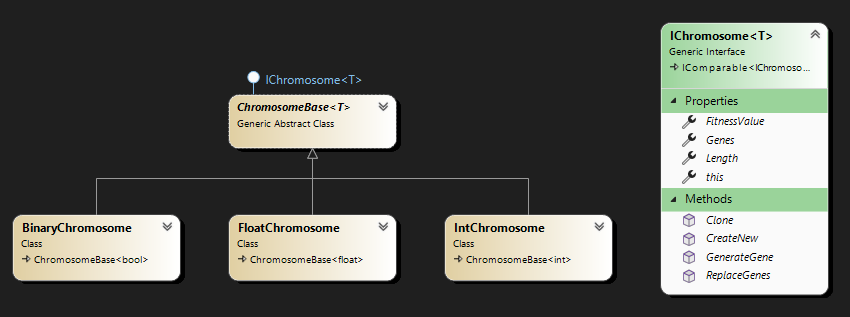


Рисунок 3.9 Структура классов Хромосомы

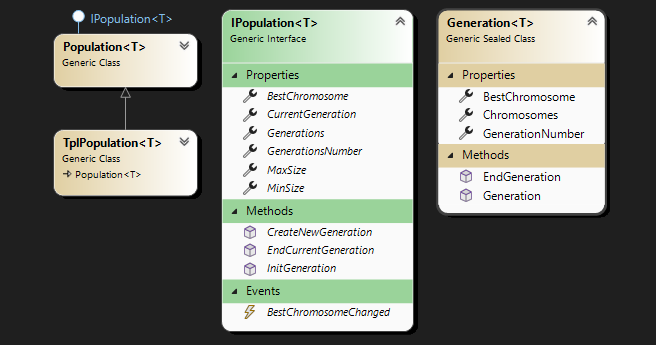


Рисунок 3.10 Структура классов популяции и поколения

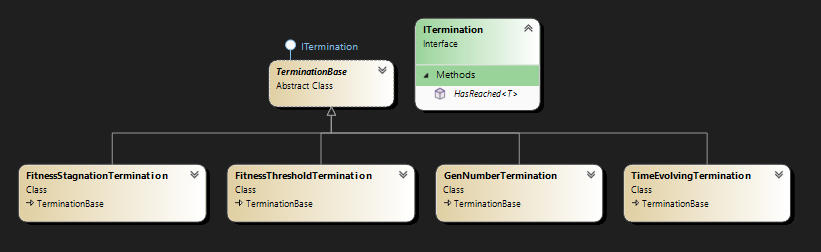


Рисунок 3.11 Структура классов Условия остановки

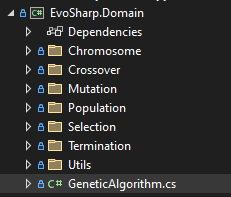


Рисунок 3.12 Файловая структура проекта

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.*

1. TDD & Unit Testing

Далеко не секрет что написание модульных тестов уже стала мировой практикой хорошего тона, и не с проста. 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.

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 проектов, таких как, проект, который сильно зависит от внешних зависимостей, или, взаимодействующий с пользовательским интерфейсом.

Будем тестировать основные этапы алгоритма и их вариации, но для начала нужно реализовать классы, отвечающие за состояние алгоритма: Chromosome, Generation, Population, программную реализацию которых можно найти в приложениях А, Б, В, соответственно. Каждый из этих классов хранит определенные состояния этапа алгоритма. Класс Chromosome содержит в себе такие поля как: значение фитнесс-функции, длину хромосомы, непосредственно массив генов, а также вспомогательные методы для создания или получения внутреннего состояния. Класс Generation содержит в себе массив хромосом текущей генерации так же хранит в состоянии лучшую на момент хромосому. Класс Population представляет собой хранилище списка всех генераций, созданных алгоритмом, так же содержит текущую генерацию.

Как можно заметить на все эти классы повешено ограничение обобщение класса типа – ”<T>”, оно характеризует тип значения которым будет представлена хромосома, из возможных вариантов может быть целочисленные значения, битовые значения (1 или 0), вещественные значения либо пользовательские типы. Такой подход позволяет быстро и легко настроить будущую структуру алгоритма.

Для операторов выборки, скрещивания и мутации напишем тесты. В приложении Г находиться некоторые юнит-тесты, написанные для них. В основном эти тесты на проверку валидации входных параметров и основной функциональности. Используя при этом, возможности, 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 интерфейсов, 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.

После написания тестов можно приступить к реализации основных операторов – выборки, скрещивания и мутации. К каждому из них один подход: написание интерфейса с одним методом, который представляет основу оператора, затем последующая реализация каждого вида оператора. После написания очередной разновидности оператора проверяем тесты, это обезопасит нас от последующих ошибок, и поможет определить на ранних этапах разработки потенциальные угрозы.

После реализации всех компонентов, и проверки тестов, остаётся вопрос в способе применения. Пример использования библиотеки на примере задачи OneMax, которая описывалась в 1 главе можно увидеть в приложении Д. Удобство написанной библиотеки в гибкости ее настройки: так как все операторы наследуют интерфейсы, пользователь что будет применять библиотеку у себя в проектах, сможет легко кастомизировать любой оператор под свои нужды, или использовать уже написанные.

1. Задача коммивояжера

Для полной демонстрации работы написанной библиотеки, решим небезызвестную задачу коммивояжера.

TSP расшифровывается как "Задача коммивояжера", которая является хорошо известной задачей в информатике и математике. Проблема заключается в нахождении кратчайшего возможного маршрута, по которому коммивояжер может посетить несколько городов и вернуться в исходную точку, посетив каждый город ровно один раз.

**Математическая модель.** Математически TSP можно сформулировать следующим образом:

Пусть G = (V, E) - полный неориентированный граф с n вершинами, где V = {1, 2, ..., n} - множество узлов (городов), а E - множество ребер (дорог), соединяющих их. Каждое ребро e = (i, j) имеет неотрицательный вес w(e), представляющий расстояние между городами i и j.

Цель TSP состоит в том, чтобы найти гамильтонов цикл (цикл, который посещает каждый узел ровно один раз) с минимальным общим весом, т.е. найти перестановку вершин, которая минимизирует сумму весов ребер в соответствующем цикле.

Формально пусть P = (v1, v2, ..., vn, v1) - гамильтонов цикл в G, где vi ∈ V для всех i. Тогда TSP можно сформулировать как следующую оптимизационную задачу:

при соблюдении следующих ограничений:

* vi ∈ V для всех i,
* vi ≠ vj для всех i ≠ j
* v1 = vn+1 = 1

Первое ограничение гарантирует, что каждая вершина посещается ровно один раз, второе ограничение гарантирует, что ни одна вершина не посещается более одного раза, а третье ограничение гарантирует, что цикл начинается и заканчивается в вершине 1.

Обратите внимание, что TSP, как известно, является NP-трудной задачей, что означает, что не известен эффективный алгоритм, который мог бы оптимально решить все экземпляры задачи. Поэтому были разработаны различные эвристики и алгоритмы аппроксимации для поиска хороших решений на практике.

TSP – это задача оптимизации, которая имеет широкий спектр практических применений, включая логистику, планирование транспортировки. Несмотря на свою простоту, поиск оптимального решения для TSP является NP-трудной задачей, что означает, что ее вычислительно невозможно решить при больших размерах задачи.

Было разработано несколько эвристических алгоритмов для нахождения приближенных решений для TSP, включая алгоритм ближайшего соседа, алгоритм 2-opt и так же **генетический алгоритм**. Эти алгоритмы могут найти хорошие решения для многих практических случаев TSP за разумный промежуток времени. Однако поиск точного оптимального решения остается сложной открытой проблемой в информатике.

В генетическом алгоритме проблема TSP представлена в нахождении такой хромосомы, которая будет минимизировать суммарное расстояние между всеми пунктами маршрута. Представим нашу хромосому целочисленным массивом индексов городов. Каждая хромосома оценивается на основе фитнес-функции, к данной задаче подойдет функция, которая на основе общего расстояния, необходимого для достижения всех городов на маршруте, вычисляет пригодность. Чем короче расстояние, тем лучше хромосома. Хромосомы с более высокой приспособленностью с большей вероятностью будут отобраны для размножения, в то время как хромосомы с более низкой приспособленностью с большей вероятностью будут отброшены. В качестве критерия остановки будем использовать временной промежуток, возьмем класс TimeEvolvingTermination, и зададим параметр MaxTime на минуту. Минимальный размер популяции поставим 50, а максимальный 100. Для оператора скрещивания выберем равномерное скрещивание, для оператора выборки выберем выбор на основе рулетки и наконец для оператора мутации выберем мутацию обращением, которая сохранит общий вид хромосом, что в данной задаче желательно.

Реализовывать данную задачу я буду в Unity 3D, что уже ранее использовалась для демонстрации работы ГА в RL на примере задачи self-driving car. Для имитации городов и маршрута, буду использовать иконки домов и LineRenderer для прорисовки маршрутов. При каждом новом запуске на карту случайным образом будут помещены N городов, и сразу же запущен алгоритм, с подробной статистикой. Что бы мы могли видеть, как меняется текущее поколение и лучшее текущее решение, прибегнем к дополнительному потоку, в одном из которых будет запускаться сам алгоритм, а в другом просчет фитнесс-функции. Для просчета расстояния между городами будем вычислять длину вектора между точками координат 2 городов на сетке. Так же для большей наглядности добавим возможность перетягивать города во временя работы алгоритма, навесим на иконку города контроллер CityController, в котором и определены событие OnMouseDrag.

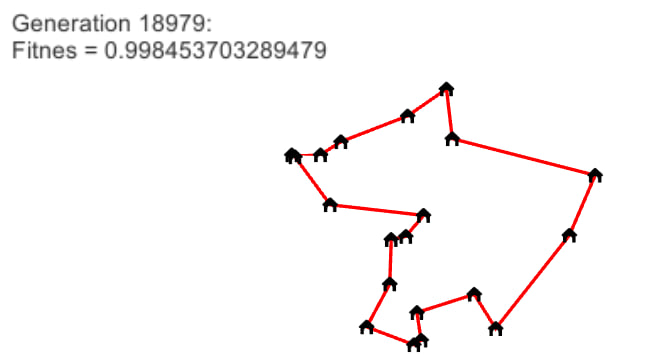


Рисунок 3.13 Итоговое решение

Хочу отметить, что при количестве городов равное 20 решение находиться за достаточно приемлемое время. Этим примером показан один из всевозможных вариантов использования пользователями данной библиотеки. Далее остается последний этап ­ ­– публикация библиотеки в репозитории NuGet.

Задача коммивояжера является NP-сложной, что означает, что нет известного алгоритма, который бы решал ее за полиномиальное время для всех возможных входных данных. В связи с этим, большинство алгоритмов для решения задачи коммивояжера являются приближенными, т.е. они дают решение, которое может быть неоптимальным, но все еще достаточно близким к оптимальному.

Среди приближенных алгоритмов для решения задачи коммивояжера наиболее эффективными являются: локальный поиск (Local Search), муравьиные алгоритмы (Ant Colony Optimization), симуляционное отжигание (Simulated Annealing). Каждый из этих алгоритмов имеет свои преимущества и недостатки, и лучший алгоритм для решения задачи коммивояжера зависит от особенностей решаемой проблемы и требуемой точности.

Рассмотрим некоторые из преимуществ и недостатков ГА по сравнению с другими алгоритмами для решения TSP:

Преимущества:

1. ГА может найти несколько решений за один прогон, в силу их подхода с хромосомами, что полезно для создания разнообразного набора решений.
2. ГА может быть распараллелен, что позволяет ускорить вычисления на современных вычислительных архитектурах.
3. ГА может быть легко расширен для решения многоцелевых задач TSP, где целью является оптимизация нескольких задач одновременно.

Недостатки:

1. ГА может быть медленным по сравнению с некоторыми другими алгоритмами TSP, особенно для небольших экземпляров TSP.
2. ГА требует правильной настройки параметров, таких как размер популяции, частота мутаций и скрещиваний, которые может быть трудно настроить для достижения оптимальной производительности.
3. ГА может плохо работать на экземплярах TSP с сильно симметричными или кластеризованными городскими распределениями.

В целом, генетические алгоритмы представляют собой мощный и гибкий класс алгоритмов для решения TSP. Хотя они не всегда могут быть наиболее эффективным или точным алгоритмом для всех экземпляров TSP, они часто позволяют находить высококачественные решения и являются полезным инструментом для оптимизации TSP.

1. Публикация NuGet

Для того что бы любые пользователи могли иметь доступ к нашей библиотеке, ее нужно опубликовать на общедоступном ресурсе, где любой сможет установить и использовать в своих нуждах, к счастью, Майкрософт предоставляет такое решение в виде менеджера пакетов – NuGet.

NuGet – это менеджер пакетов для платформы .NET, который упрощает процесс поиска, установки и управления сторонними библиотеками и инструментами. Это позволяет разработчикам легко искать и загружать пакеты, содержащие библиотеки кода, фреймворки и другие зависимости, которые могут быть использованы в их проектах.

NuGet – это проект с открытым исходным кодом, разработанный Microsoft и ставший важным инструментом для разработчиков для управления зависимостями своих проектов. NuGet предоставляет централизованное хранилище пакетов, к которым можно получить доступ и установить из Visual Studio или из командной строки. Это также позволяет разработчикам создавать и публиковать свои собственные пакеты в репозитории, упрощая обмен библиотеками кода с сообществом. С помощью NuGet разработчики могут избежать ручной работы по загрузке зависимостей и управлению ими, что может сэкономить им значительное количество времени и усилий. В целом, NuGet упрощает процесс управления зависимостями в проектах .NET, что делает его незаменимым инструментом для любого разработчика .NET.

Для публикации своего пакета, сначала нужно зарегистрироваться на портале [nuget.org](https://www.nuget.org/), после создать проект Class Library, но так как у нас уже он есть, перейдем к следующему этапу – настройке метаданных будущего пакета. В настройках нашего проекта откроем панель Package, в нем пропишем имя проекта, версию, автора, описание, и т.д.

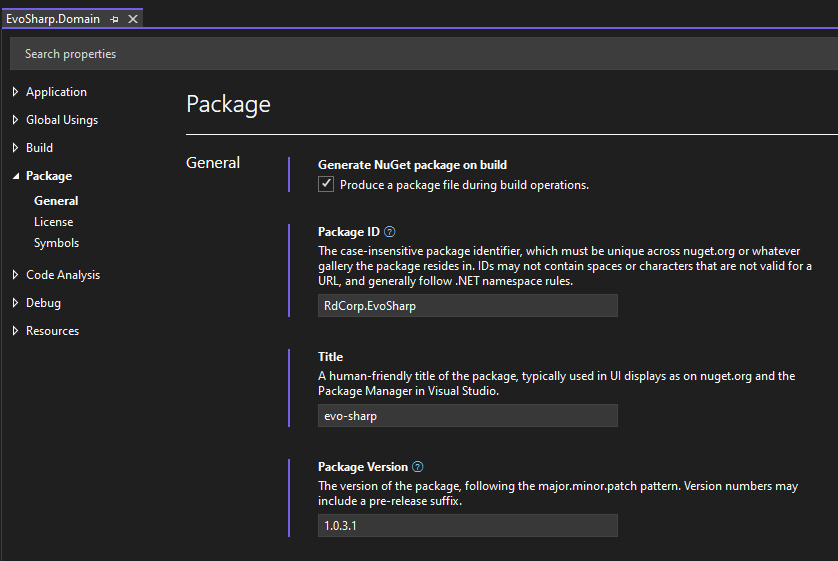


Рисунок 3.14 Настройка метаданных

После того как мы соберем проект создаться пакет для публикации, далее на самом портале, выбираем опцию загрузки пакетов и публикуем наш пакет. После загрузки наша библиотека будет доступна всем желающим.

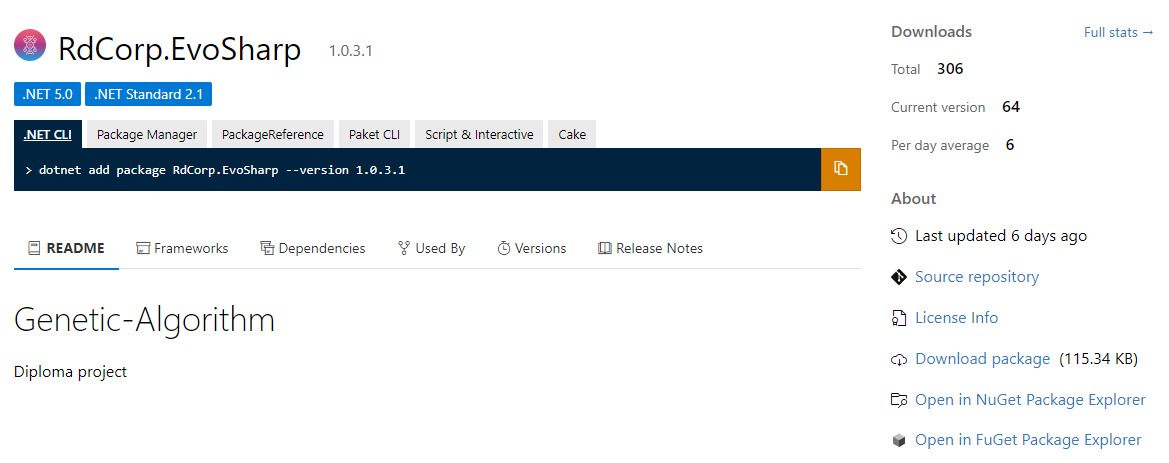


Рисунок 3.15 Страница NuGet пакета

1. Выводы

В данной главе была рассмотрена роль генетического алгоритма в искусственном интеллекте. Было выяснено, что ГА широко используется в задачах оптимизации, машинном обучении, обработке изображений и других смежных областях.

Была проведена разработка библиотеки для реализации ГА на основе подхода TDD тестирования. Это позволяет разработчикам уверенно изменять код и проверять его на корректность без риска нарушения функциональности. На примере решения задачи TSP было показано практическое применение библиотеки. Так же была выполнена публикация библиотеки на общедоступный ресурс ­– NuGet платформу управления пакетов, что позволит любому пользователю использовать ее в своих целях, и ускорит разработку проектов, благодаря гибкости структуры библиотеки.

Если сравнивать мою реализацию библиотеки с популярными решениями, то можно выделить следующие плюсы: поддержка нескольких версий (.Net Standart 2.1, .Net 5, .Net 6, .Net 7), легковесность, простота в использовании, открытый исходный код, наличие юнит тестов, поддержка многопоточности для ускорения работы алгоритма.

Кроме того, была представлена задача обучения самоуправляемой машины с помощью генетического алгоритма в Unity. Первоначальные результаты практического применения показали, что ГА хорошо подходит 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, генетический алгоритм является мощным инструментом в искусственном интеллекте и может 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 генетический алгоритм хорошо приспособлен к этому и может стать мощным оружием в арсенале, когда придется решать очередную сложную задачу. Как было показано, генетический алгоритм и родственные им методы применимы к самым разным задачам – практически в любых областях вычислений и техники. Прелесть данного подхода в том, что для применения генетического алгоритма нужно всего лишь знать способ представить решения и получить его численную оценку.

Созданная библиотека классов 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.
6. Разобрано практическое применение библиотеки в решении проблемы коммивояжёра.
7. Произведена загрузка на общедоступную платформу управления пакетов ­– NuGet.

Для проектирования библиотеки использовалась среда Visual Studio 2022.

Для реализации задачи self-driving car и решение задачи коммивояжёра использовались среда Unity 2021.3.13f1.

# **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
  6. Reinforcement Learning. Successes of RL [Electronic resource] – Mode of access: [http://umichrl.pbworks.com/w/page/7597597/Successes  
     %20of%20Reinforcement%20Learning](http://umichrl.pbworks.com/w/page/7597597/Successes%20of%20Reinforcement%20Learning).– Date of access: 14.02.2023
  7. Мищенко В.А., Коробкин А.А. Использование генетических алгоритмов в обучении нейронных сетей [Электронный ресурс]. – 2011. – Режим доступа: https://s.science-education.ru/pdf/2011/6/mischenko.pdf. – Дата доступа: 29.04.2023
  8. Madhavan V., Conti E. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning [Electronic resource] – Mode of access: <https://arxiv.org/pdf/1712.06567.pdf>. – Date of access: 01.05.2023

# **ПРИЛОЖЕНИЯ**

# *Приложение А*

public abstract class ChromosomeBase<T> : IChromosome<T>

{

private T[] \_genes;

protected int \_length;

protected readonly T \_minValue = default;

protected readonly T \_maxValue = default;

protected readonly Random \_random = new Random();

     public ChromosomeBase(T[] genes)

     {

         \_genes = genes ?? throw new ArgumentNullException();

         \_length = \_genes.Length;

     }

     public T this[int i]

     {

         get => \_genes[i];

         set

         {

             if (i < 0 || i >= \_length)

                 throw new ArgumentOutOfRangeException(nameof(i));

             \_genes[i] = value;

             FitnessValue = null;

         }

     }

     public double? FitnessValue { get; set; }

     public int Length => \_length;

     public T[] Genes => \_genes;

     public abstract T GenerateGene();

     public abstract IChromosome<T> CreateNew();

     public void ReplaceGenes(int startIndex, T[] genes)

     {

         if (genes == null)

             throw new ArgumentNullException(nameof(genes));

         if (genes.Length == 0) return;

         Array.Copy(genes, 0, \_genes, startIndex, Math.Min(genes.Length, \_length - startIndex));

            FitnessValue = null;

    }

      public int CompareTo(IChromosome<T> other)

      {

          if (other == null)

              return -1;

          var otherFitness = other.FitnessValue;

          if (FitnessValue == otherFitness)

              return 0;

          return FitnessValue > otherFitness ? 1 : -1;

      }

      public override bool Equals(object obj) =>

obj is IChromosome<T> other && CompareTo(other) == 0;

      public override int GetHashCode() =>

FitnessValue.GetHashCode();

      private T[] CreateGenes(int length)

      {

          var result = new T[length];

          for (int i = 0; i < length; i++)

              result[i] = GenerateGene();

          return result;

      }

}

# *Приложение Б*

public sealed class Generation<T>

{

public Generation(IList<IChromosome<T>> chromosomes)

      {

if (chromosomes is null || chromosomes.Count < 2)

            throw new ArgumentOutOfRangeException()

          GenerationNumber = generationNumber;

          Chromosomes = chromosomes;

      }

      public IList<IChromosome<T>> Chromosomes { get; internal set; }

      public IChromosome<T> BestChromosome { get; internal set; }

      public void EndGeneration()

      {

          Chromosomes = Chromosomes

              .Where(c => c.FitnessValue.HasValue)

              .OrderByDescending(c => c.FitnessValue.Value)

              .ToList();

          BestChromosome = Chromosomes.First();

      }

    }

# *Приложение В*

public class Population<T> : IPopulation<T>

{

public event EventHandler BestChromosomeChanged;

public Population(int minSize, int maxSize,

IChromosome<T> firstChromosome)

{

if (minSize < 2)

throw new ArgumentOutOfRangeException(nameof(minSize);

           if (maxSize < minSize)

                throw new ArgumentOutOfRangeException();

          MinSize = minSize;

          MaxSize = maxSize;

          FirstChromosome = firstChromosome;

          Generations = new List<Generation<T>>();

     }

     public IList<Generation<T>> Generations { get; protected set; }

     public Generation<T> CurrentGeneration { get; protected set; }

     public int GenerationsNumber { get; protected set; }

     public int MinSize { get; set; }

     public int MaxSize { get; set; }

     public IChromosome<T> BestChromosome { get; protected set; }

     protected IChromosome<T> FirstChromosome { get; set; }

     public virtual void InitGeneration()

     {

         Generations = new List<Generation<T>>();

         GenerationsNumber = 0;

         var chromosomes = Enumerable

.Range(0, MinSize)

.Select(x => FirstChromosome.CreateNew())

.ToList();

         CreateNewGeneration(chromosomes);

     }

     public virtual void CreateNewGeneration(IList<IChromosome<T>> chromosomes)

     {

if (chromosomes == null)

            throw new ArgumentNullException(nameof(chromosomes));

           CurrentGeneration = new Generation<T>(++GenerationsNumber, chromosomes);

           Generations.Add(CurrentGeneration);

    }

     public virtual void EndCurrentGeneration()

     {

      CurrentGeneration.EndGeneration();

           if (BestChromosome is null || BestChromosome.CompareTo(CurrentGeneration.BestChromosome) != 0)

           {

            BestChromosome = CurrentGeneration.BestChromosome;

                OnBestChromosomeChanged(EventArgs.Empty);

           }

     }

     protected virtual void OnBestChromosomeChanged(EventArgs args) =>

            BestChromosomeChanged?.Invoke(this, args);

    }

# *Приложение Г*

[Test]

public void Mutate\_SwapsTwoRandomGenesWithProbabilityLessThanOne()

{

// Arrange

var chromosome = new IntChromosome(5, 1, 10);

var mutation = new TworsMutation();

var originalGenes = chromosome.Genes.ToArray();

     // Act

     mutation.Mutate(chromosome, 0.5f);

     // Assert

     var mutatedGenes = chromosome.Genes.ToArray();

     Assert.That(mutatedGenes,Has.Length.EqualTo(5);

}

[Test]

public void PerformMutate\_MutatesGenesWithProbabilityLessThanOne()

{

// Arrange

var mutableGenesIndexes = new int[] { 1, 3, 5 };

var chromosome = new IntChromosome(0, 10);

var mutation = new UniformMutation(mutableGenesIndexes);

var originalGenes = chromosome.Genes.ToArray();

// Act

mutation.Mutate(chromosome, 0.5f);

var mutatedGenes = chromosome.Genes.ToArray();

// Assert

for (int i = 0; i < originalGenes.Length; i++)

if (!mutableGenesIndexes.Contains(i))

Assert.That(

mutatedGenes[i],

Is.EqualTo(originalGenes[i])

);

}

[Test]

public void Cross\_SwapIndexLarge\_ThrowsArgumentOutOfRangeException()

{

// Arrange

var crossover = new OnePointCrossover(10);

var parents = new List<IChromosome<int>>

{

new IntChromosome(new[] {1, 2, 3, 4, 5}),

new IntChromosome(new[] {6, 7, 8, 9, 10}),

};

// Act & Assert

Assert.Throws<ArgumentOutOfRangeException>(

() => crossover.Cross(parents)

);

}

# *Приложение Д*

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 Number: {{ga.Population.GenerationsNumber}}:

Fitnes: {{bestFitness}},

Best chromosome: {{ga.Population.BestChromosome}}

"""

);

    }

};

ga.Start();