МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра технологий программирования

ПРОЕКТИРОВАНИЕ БИБЛИОТЕКИ ДЛЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

Курсовой проект

Доскоча Романа Дмитриевича

студента 4 курса, специальность

«прикладная информатика»

Научный руководитель:

кандидат технических наук,

доцент И.С. Войтешенко

Минск, 2022

РЕФЕРАТ

Курсовой проект, 36 с., 26 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 исследование возможностей 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, 36 с., 26 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, 36 p., 26 illustrations.

Keywords: Genetic Algorithm, Unity, Genotype, RL

Object of research – find the 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

Берущие начало в эволюционной теории Дарвина, генетические алгоритмы являются одним из самых удивительных методов решения задач поиска, оптимизации и обучения. Они могут привести к успеху там, где традиционные алгоритмы не способны дать адекватные результаты за приемлемое время.

В главе 1 «Основы генетического алгоритма» приводится краткое введение в теорию и принципы работы генетических алгоритмов. Рассматриваются также различия между генетическими алгоритмами и традиционными методами, и описываются сценарии, в которых имеет смысл применять их.

В главе 2 «Исследование многообразия подходов генетического алгоритма» рассматривается и описываются шаги алгоритма, так же рассмотрены 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 проектирование библиотеки» реализовано применение ГА в нейронных сетях в частности на подходе обучения с подкреплением на примере самоуправляемой машины в Unity 3D. Так же спроектирована библиотека генетического алгоритма и написаны юнит-тесты для подхода проектирования TDD.

Глава 1
ОСНОВЫ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

Генетические алгоритмы является весьма перспективным и актуальным направлением в области оптимизации и моделирования. С помощью генетических алгоритмов решаются множество задач на графах, задачи компоновки и составления расписаний, производится настройка и обучение искусственных нейронных сетей, и многое другое.

1.1 История

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 глобального экстремума. 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 (ГА) основаны на принципах естественного отбора Ч. Дарвина. ГА относятся к стохастическим методам. Эти алгоритмы успешно применяются в различных областях деятельности (экономика, физика, и т.п.). Созданы различные модификации ГА и разработан ряд тестовых функций.

Генетические алгоритмы относят к области мягких вычислений. Термин «мягкие вычисления» введен Лофти Заде в 1994 году. Это понятие объединяет такие области, как нечеткая логика, нейронные сети, вероятностные рассуждения, сети доверия и эволюционные алгоритмы, которые дополняют друг друга и используются в различных комбинациях или самостоятельно для создания гибридных интеллектуальных систем. Первая схема генетического алгоритма была предложена в 1975 году в Мичиганском университете Джоном Холландом (John Holland), а предпосылками этому послужили исследования Л.Дж.Фогеля, А.Дж. Оуэнса, М.Дж.Волша по эволюции простых автоматов, предсказывающих символы в цифровых последовательностях.

Новый алгоритм получил название «репродуктивный план Холланда» и в дальнейшем активно использовался в качестве базового алгоритма в эволюционных вычислениях. Идеи Холланда развили его ученики Кеннет Де Йонг из университета Джорджа Мейсона и Дэвид Голдберг из лаборатории ГА. Благодаря им, был создан классический ГА, описаны все операторы и исследовано поведение группы тестовых функций (именно алгоритм Голдберга и получил название «генетический алгоритм»).

1.2 Что такое генетические алгоритмы

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.

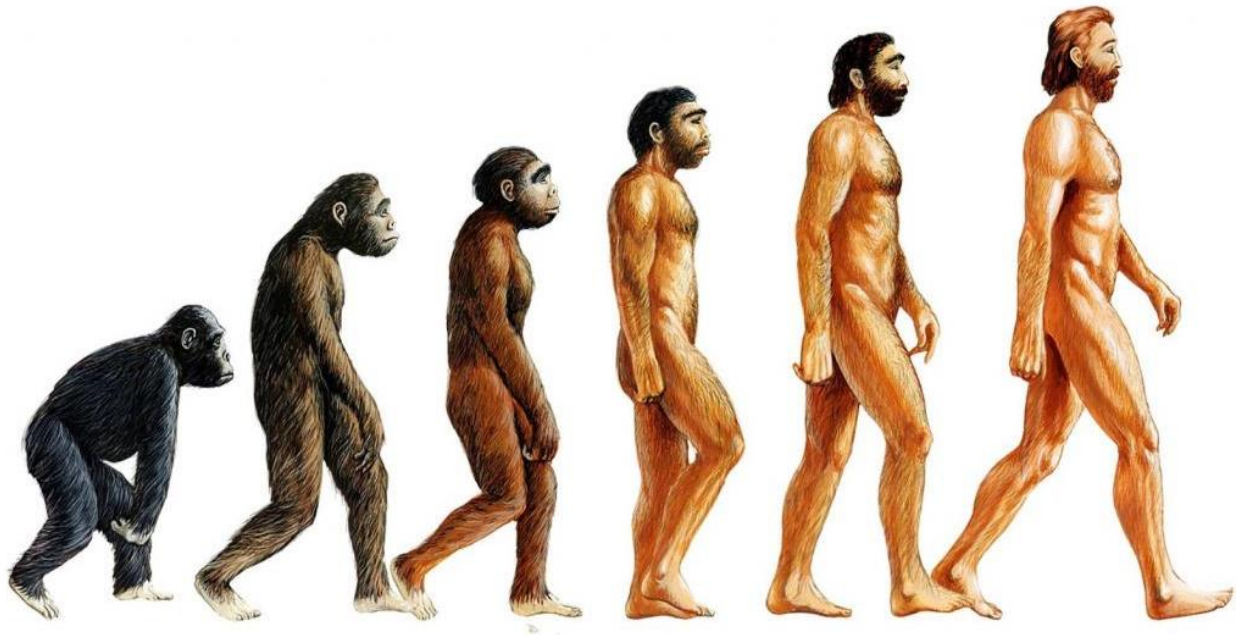

Рисунок 1.1 Эволюция по Дарвину

Генетический алгоритм представляет собой адаптивный поисковый метод, основанный на селекции лучших элементов популяции. Область поиска для генетического алгоритма называется популяцией, элементами которой являются хромосомы. Алгоритм начинается со случайной выборки совокупности допустимых решений из всей популяции. Каждая хромосома при этом уже является сама по себе решением. Качество решения определяется степенью приспособленности каждой хромосомы.

Генетический алгоритм использует методику адаптивного эвристического поиска, которая выбирает совокупность наилучших решений среди всей популяции. Операции селекции, скрещивания и мутаций позволяют получить новые особи – потомков. Более приспособленные хромосомы переходят в следующее поколение. Менее сильные особи имеют меньшие шансы на это перемещение. Процесс повторяется до момента получения наиболее приспособленного решения задачи. Средняя приспособленность популяции возрастает с каждой итерацией, таким образом, большее число итераций дает лучший результат.

1.3 Основные понятия и определения

Опишем основные понятия теории генетического алгоритма.

Генотип. В природе скрещивание, воспроизводство и мутация реализуются посредством генотипа – набора генов, сгруппированных в хромосомы. Когда две особи скрещиваются и производят потомство, каждая хромосома потомка несет комбинацию генов родителей. В случае генетических алгоритмов каждому индивидууму соответствует хромосома, представляющая набор генов. Например, хромосому можно представить двоичной строкой, в которой каждый бит соответствует одному гену:

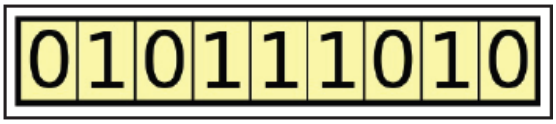

Рисунок 1.2 Простое двоичное кодирование хромосомы

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. Поскольку каждый индивидуум представлен некоторой хромосомой, эту популяцию можно рассматривать как коллекцию хромосом. Популяция всегда представляет текущее поколение и эволюционирует со временем, когда текущее поколение заменяется новым.

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.

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. Процесс отбора основан на оценке приспособленности индивидуумов. Те, чья оценка выше, имеют больше шансов передать свой генетический материал следующему поколению. Плохо приспособленные индивидуумы все равно могут быть отобраны, но с меньшей вероятностью. Таким образом, их генетический материал не полностью исключен.

Скрещивание (Crossover). 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. Эта операция называется скрещиванием, или рекомбинацией.

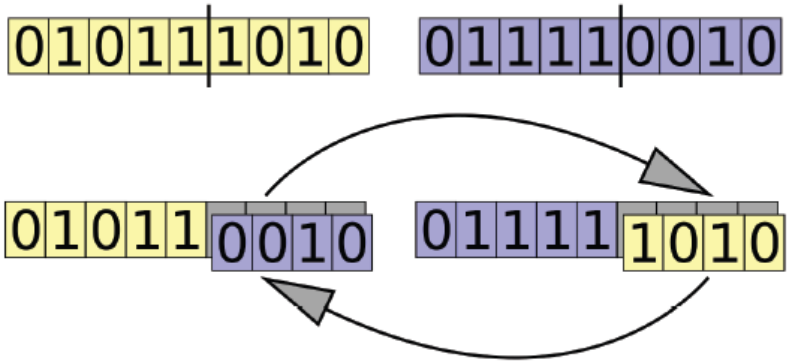

Рисунок 1.3 Операция скрещивания двух двоично-кодированных хромосом

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.

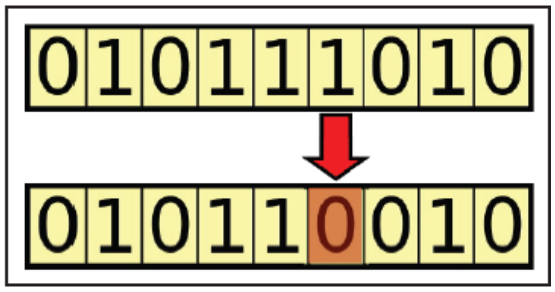

Рисунок 1.4 Применение оператора мутации к двоично-кодированной хромосоме

1.4 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 нашли широкое применение в вычислительной биологии. Они были успешно использованы для соединения белка-лиганда, при котором требовался поиск конфигурации маленькой молекулы, связанной с реципиентом. Эти алгоритмы используются в фармацевтических исследованиях и для того, чтобы лучше понять механизмы природных явлений.

Еще одним их примеров использования генетического алгоритма можно считать одну из самых известных задач в области дискретной оптимизации – задача коммивояжера. Бродячий торговец желает найти по карте кратчайший маршрут, чтобы посетить каждый город ровно один раз и вернуться в исходную точку. Решение задачи коммивояжера представляет собой гигантский цикл, который сводит к минимуму затраты на его прохождение.

Как было показано, генетические алгоритмы находят неоптимальные, но довольно хорошие решения за короткий промежуток времени. Эта задача широко применяется для эффективного распределения товаров. Например, диспетчеры грузовых автомобилей служб FedEx и UPS используют программное обеспечение для решения задачи коммивояжера в повседневной деятельности. Алгоритмы, помогающие решить эту задачу, позволяют сократить расходы в самых разных отраслях.

В компьютерном искусстве генетические 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.

1.5 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, а особенно те, что основаны на вычислении градиента, могут застревать в локальном максимуме, вместо того чтобы найти глобальный. Это связано с тем, что в окрестности локального максимума всякое небольшое изменение решения ухудшает оценку. Генетические алгоритмы менее подвержены этой напасти и имеют больше шансов отыскать 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. Поскольку генетическим алгоритмам нужно знать только значение функции приспособленности каждого индивидуума, а все остальные ее свойства, в частности производные, несущественны, их можно применять к задачам со сложным математическим представлением, включающим функции, которые трудно или невозможно продифференцировать.

Генетические алгоритмы применимы и к задачам, вообще не имеющим математического представления. Один из таких случаев, представляющий особый интерес, – когда оценка приспособленности основана на мнении человека. Пусть, например, требуется найти наиболее привлекательную цветовую палитру для веб-сайта. Мы можем попробовать разные комбинации цветов и попросить пользователей оценить привлекательность сайта. А затем применить генетический алгоритм, чтобы найти лучшую комбинацию, используя функцию приспособленности, основанную на оценках пользователей. Алгоритм будет работать, несмотря на то что никакого математического представления нет и невозможно вычислить оценку заданной комбинации непосредственно.

Распараллеливание. 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.

1.6 Выводы

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 будет подробнее рассмотрены основные компоненты и детали реализации генетического алгоритма.

2.1 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. Начальная популяция состоит из случайным образом выбранных потенциальных решений (индивидуумов). 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.

Вычисление приспособленности. Для каждого индивидуума вычисляется функция приспособленности. 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. Если в какой-то задаче нужен минимум, то при вычислении приспособленности нужно инвертировать каждое найденное значение, умножив его на -1.

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.
- Оператор **скрещивания** создает потомка выбранных индивидуумов. Обычно для этого берутся два индивидуума, и части их хромосом

меняются местами, в результате чего создаются две новые хромосомы, представляющие двух потомков.

- Оператор **мутации** вносит случайные изменения в один или несколько генов хромосомы вновь созданного индивидуума.

Проверка условий остановки. Может существовать несколько условий, при выполнении которых процесс останавливается. Сначала отметим два самых распространенных:

- Достигнуто максимальное количество поколений. Это условие заодно позволяет ограничить время работы алгоритма и потребление им ресурсов системы;
- На протяжении нескольких последних поколений 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.

Перечислим также другие возможные условия:

- С момента начала прошло заранее определенное время;
- Превышен некоторый лимит затрат, например процессорного времени или памяти;
- Наилучшее решение заняло часть популяции, большую заранее заданного порога.

Подытожим. Генетический алгоритм начинается с популяции случайно выбранных потенциальных решений (индивидуумов), для которых вычисляется функция приспособленности. Алгоритм выполняет цикл, в котором последовательно применяются операторы отбора, скрещивания и мутации, после чего приспособленность индивидуумов пересчитывается. Цикл продолжается, пока не выполнено условие остановки, после чего лучший индивидуум в текущей популяции считается решением.

2.2 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. Отбор носит вероятностный характер, причем вероятность выбора индивидуума зависит от его приспособленности, так что у более приспособленных индивидуумов шансы отобраться выше.

Существуют разные методы отбора вот самые популярные из них:

- **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. Метод отбора по правилу рулетки, устроен так, что вероятность отбора индивидуума прямо пропорциональна его приспособленности. Тут можно провести аналогию с вращением колеса рулетки, где каждому индивидууму соответствует сектор, стоимость которого равна приспособленности индивидуума. Шансы, что шарик остановится в секторе индивидуума, пропорциональны размеру этого сектора.

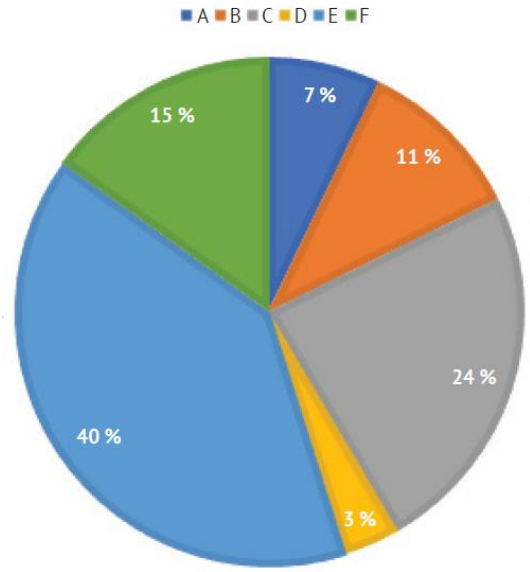

Рисунок 2.1 Пример отбора по правилу рулетки

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 немного модифицированный вариант правила рулетки. Используется та же рулетка с такими же секторами, но вместо одной точки отбора и многократного запуска рулетки мы вращаем колесо только один раз, а отбор индивидуумов производим в нескольких точках, равномерно расставленных по окружности. Тем самым все индивидуумы выбираются одновременно, как показано на рисунке ниже.

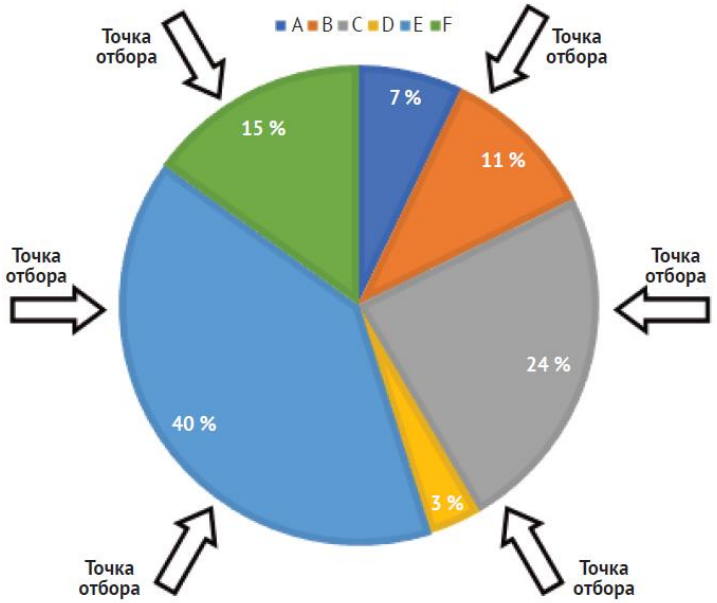

Рисунок 2.2 Пример Стохастической универсальной выборки

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

Турнирный отбор. Самым популярным методом отбора является Турнирный метод. В каждом раунде турнирного отбора из популяции выбираются два или более индивидуумов, и тот, у кого приспособленность больше, выигрывает и отбирается в следующее поколение.

Индивидуум	Приспособленность
A	8
B	12
C	27
D	4
E	45
F	17

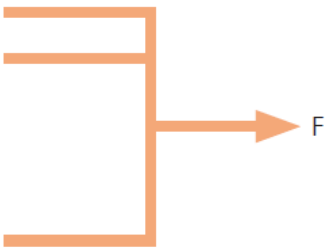

Рисунок 2.3 Пример турнирного отбора на турнире с тремя участниками

Количество индивидуумов, участвующих в каждом раунде турнирного отбора, называется размером турнира. Чем больше размер турнира, тем выше шансы, что в раундах будут участвовать лучшие индивидуумы, и тем меньше шансов у слабых участников победить в турнире и отобраться.

У этого метода отбора есть интересная особенность: если мы умеем сравнивать любых двух индивидуумов и определять, какой из них лучше, то сами значения функции приспособленности и не нужны.

Элитизм. Еще один механизм, применяемый совместно с оператором отбора. В следующее поколение мы гарантированно отбираем небольшое число наиболее приспособленных (элитных) особей. Причем, эти элитные индивидуумы также участвуют в скрещивании на правах родителей. Такой подход позволяет сохранять в популяции лучшие решения (не теряя их и не переоткрывая заново). В ряде задач это существенно увеличивает скорость сходимости ГА к некоторому приемлемому или оптимальному решению.

2.3 Операторы Скрещивания

Оператор скрещивания используется для комбинирования генетической информации двух индивидуумов, выступающих в роли родителей, в процессе порождения потомков. Как правило, оператор скрещивания применяется не всегда, а с некоторой вероятностью. Если скрещивание не применяется, то копии обоих родителей переходят в следующее поколение без изменения.

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 скрещивания. Где точкой скрещивания считается позиция разреза, таким образом что правая часть первого родителя получает левую часть второго родителя и так же для второго родителя. В результате мы получим двух потомком которые несут информацию обоих родителей.

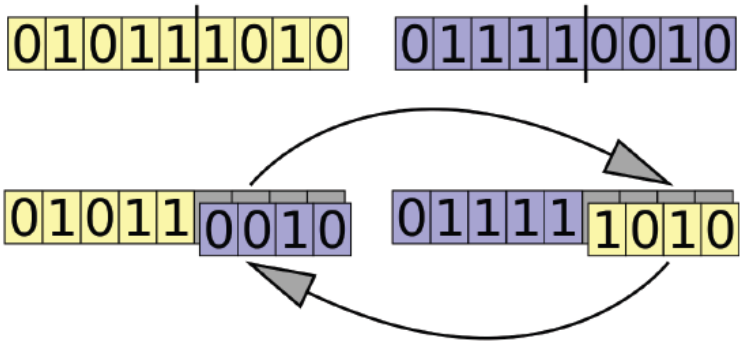

Рисунок 2.4 Пример одноточечного скрещивания

Двухточечное и k-точечное скрещивание. При двухточечном скрещивании случайным образом выбираются по две точки скрещивания в каждой хромосоме. Гены одной хромосомы, расположенные между этими

точками, обмениваются с точно так же расположенными генами другой хромосомы.

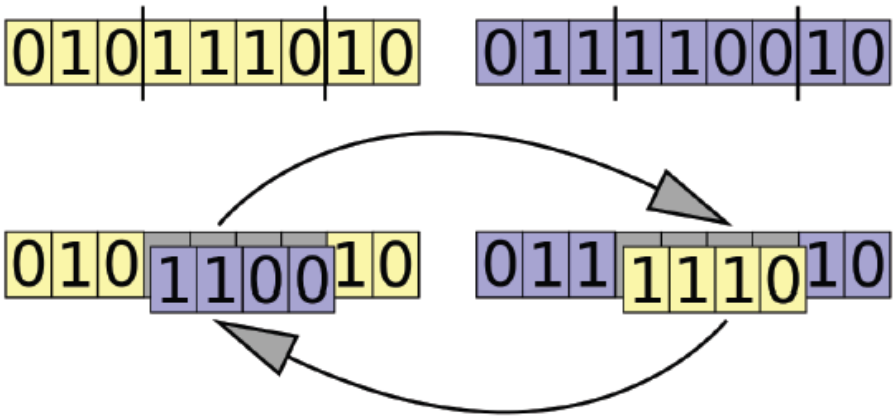

Рисунок 2.5 Пример двухточечного скрещивания

По аналогии работает обобщенный метод k-точечного скрещивания. Где k – целое число.

Равномерное скрещивание. При равномерном скрещивании каждый ген обоих родителей определяется независимо путем случайного выбора с равномерным распределением. Когда выбирается 50 % генов, оба родителя имеют одинаковые шансы повлиять на потомков.

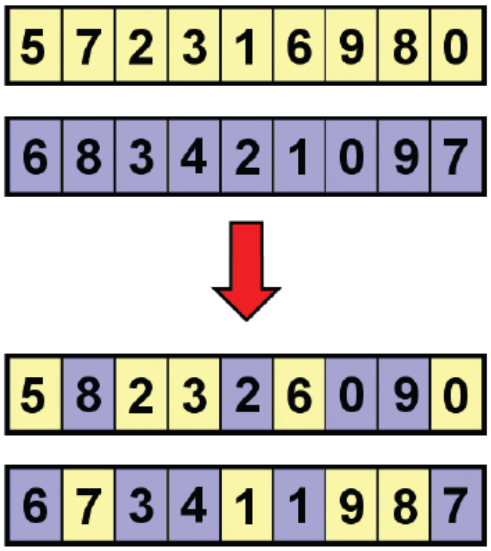

Рисунок 2.6 Пример равномерного скрещивания

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

2.4 Операторы Мутации

Мутация – последний генетический оператор, применяемый при создании нового поколения. Операция мутации вероятностная, обычно с очень

низкой вероятностью, поскольку может ухудшить качество индивидуума. С другой стороны, если частота мутации слишком велика, то генетический алгоритм выродится в случайный поиск.

Список популярных вариаций операторов мутаций:

- Инвертирование бита
- Мутация обменом
- Мутация обращением
- Мутация перестановкой

Инвертирование бита. Для двоичной хромосомы мы можем случайным образом выбрать ген и инвертировать его.

Рисунок 2.7 Пример мутации инвертированием бита

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

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

2.5 “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. Очевидно, что решением считается строка, состоящая из всех единиц тем самым максимизируя ее. Хоть задачи очень тривиальна, но зато очень хорошо показывает этапы и суть генетического алгоритма.

Наша цель – найти решение, которое бы давало максимальную сумму цифр этого списка:

$$fitness = \sum_{i=0}^{N-1} individ[i]$$

Здесь 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. Остановиться 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.

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

Fitness Функция

```
def oneMaxFitness(individual):
    return sum(individual),
```

Рисунок 2.10 Фитнес функции для задачи OneMax

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

```
toolbox = base.Toolbox()

toolbox.register("zeroOrOne", random.randint, 0, 1)
toolbox.register("individualCreator", tools.initRepeat, creator.Individual, toolbox.zeroOrOne, ONE_MAX_LENGTH)
toolbox.register("populationCreator", tools.initRepeat, list, toolbox.individualCreator)
```

Рисунок 2.11 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 (фактически, 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.

```
population = toolbox.populationCreator(n=POPULATION_SIZE)

toolbox.register("evaluate", oneMaxFitness)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("mate", tools.cxOnePoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=1.0/ONE_MAX_LENGTH)
```

Рисунок 2.12 Настройка операторов отбора, скрещивания и мутации

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

```
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("max", np.max)
stats.register("avg", np.mean)

population, logbook = algorithms.eaSimple(population, toolbox,
                                          cxpb=P_CROSSOVER,
                                          mutpb=P_MUTATION,
                                          ngen=MAX_GENERATIONS,
                                          stats=stats,
                                          verbose=True)

maxFitnessValues, meanFitnessValues = logbook.select("max", "avg")
```

Рисунок 2.13 Сбор статистики

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

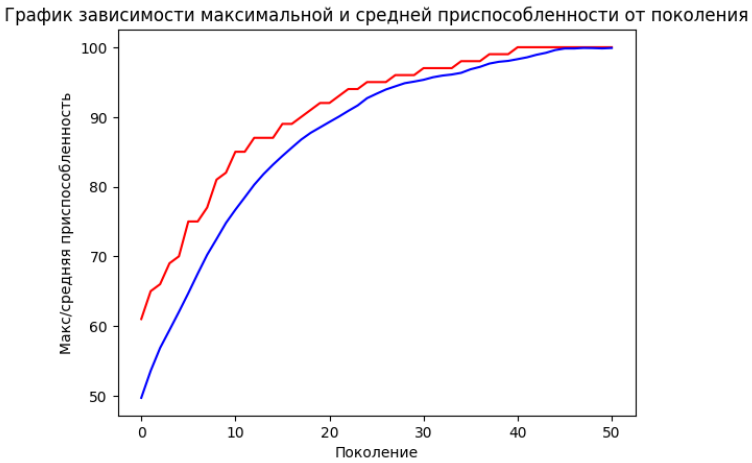

Рисунок 2.14 График зависимости максимальной и средней приспособленности от поколения

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.

Проверка эффективности

2.6 Эксперименты с параметрами ГА

Проведем несколько экспериментов с целью узнать как различные подходы в выборе стратегий алгоритма способны повлиять на итоговый результат.

Размер популяции. Изменим размер популяции и количество поколений. Размер популяции определяется константой POPULATION_SIZE. Увеличим ее с 200 до 400:

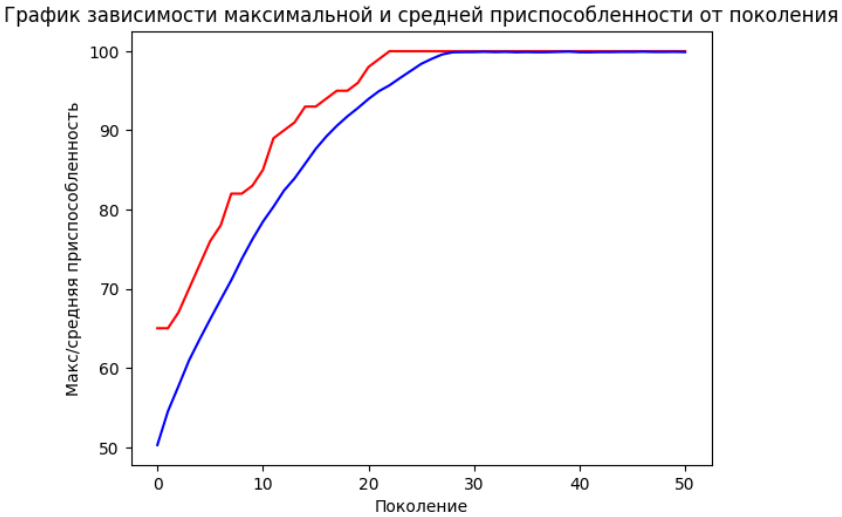

Рисунок 2.15 Результат увеличения размера популяции

Как видно из графика лучшее решение достигается уже на 20 поколении. Такое поведение неудивительно – при увеличении популяции для нахождения решения требуется меньше поколений. Однако с ростом размера популяции повышаются требования к вычислительной мощности, поэтому обычно стремимся найти умеренный размер популяции.

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)

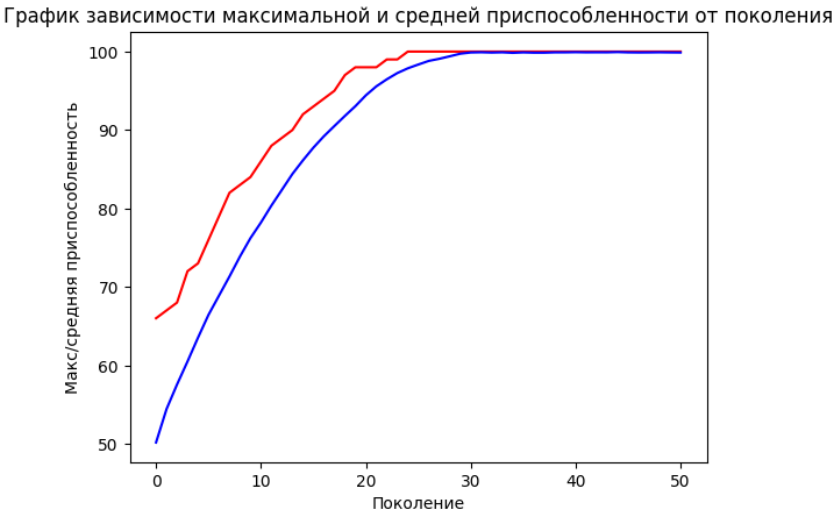

Рисунок 2.16 Результат изменения оператора скрещивания на двуточечное

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 2.17 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 2.18 Изменение оператора отбора на отбор по правилам рулетки

На графике видно, что есть много точек, в которых наилучшее решение оказывается забыто и максимальная приспособленность ведет себя скачками, хоть средняя приспособленность монотонно возрастает. Это объясняется тем, что правило рулетки отбирает индивидуумов с вероятностью,

пропорциональной их приспособленности. Если различия между индивидуумами невелики, то у более слабых появляется больше шансов отобраться по сравнению с турнирным отбором.

2.7 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.

Так как ГА имеет в себе достаточно вариаций собственной реализации, то возникает вопрос, как правильно настроить все параметры и какие методы лучше выбрать для достижения максимальной эффективности в рамках текущей задачи.

Для этого есть несколько подходов. Прежде всего необходимо определить функцию приспособленности. С ее помощью оцениваются все индивидуумы: чем больше приспособленность, тем лучше индивидуум. Функция необязательно должна быть выражена в виде математической формулы. Она может быть представлена алгоритмом, или являться результатом игры. Затем нужно выбрать подходящий способ кодирования хромосом. Он основан на параметрах, передаваемых функции приспособленности. Бывают задачи со смешанными типами параметров, а иногда даже приходится создавать собственные кодировки. Далее нужно определиться с методом отбора. Если функция приспособленности как таковая недоступна, но мы все-таки можем сказать, какой из двух кандидатов лучше, то можно воспользоваться турнирным методом отбора. При выборе методов оператора скрещивания и мутации нужно опираться от специфики конкретной задачи, иногда используются смешанные и собственные методы. Наконец, следует помнить о гиперпараметрах 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 ПРОЕКТИРОВАНИЕ БИБЛИОТЕКИ

Думая о практическом применении генетического алгоритма, можно перечислить множество типов задач, но если смотреть на тенденции мировой практики, то в настоящее время наибольшую популярность и пользу приносит искусственный интеллект.

3.1 Приложения ГА в искусственном интеллекте

Одним из самых перспективных машинных обучений можно считать основанное на обучении с подкреплением – RL (reinforcement learning). Причины – в разнообразии сложных повседневных задач, которые потенциально можно разрешить. 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.

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. Два основных компонента задачи обучения с подкреплением – окружающая среда и агент. Агент представляет алгоритм, который взаимодействует с окружающей средой и стремится решить поставленную задачу путем максимизации полного вознаграждения.

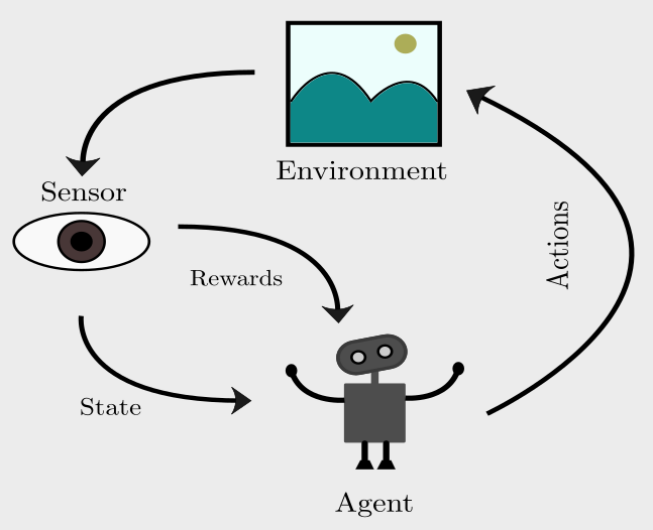

Рисунок 3.1 Концептуальная схема RL

Взаимодействие между агентом и окружающей средой можно выразить в виде последовательности шагов. На каждом шаге среда сообщает агенту некоторое состояние (state). В свою очередь, агент предпринимает некоторое действие (action). Среда реагирует переходом в новое состояние, а также промежуточным вознаграждением (reward). Это взаимодействие повторяется, пока не будет выполнено условие остановки.

Для задач обучения с подкреплением разработано много специализированных алгоритмов: Q-обучение, SARSA, DQN и др. Но поскольку в этих задачах фигурирует максимизация долгосрочного вознаграждения, мы можем рассматривать их как задачи оптимизации. Поэтому в обучении с подкреплением есть место для генетических алгоритмов.

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:

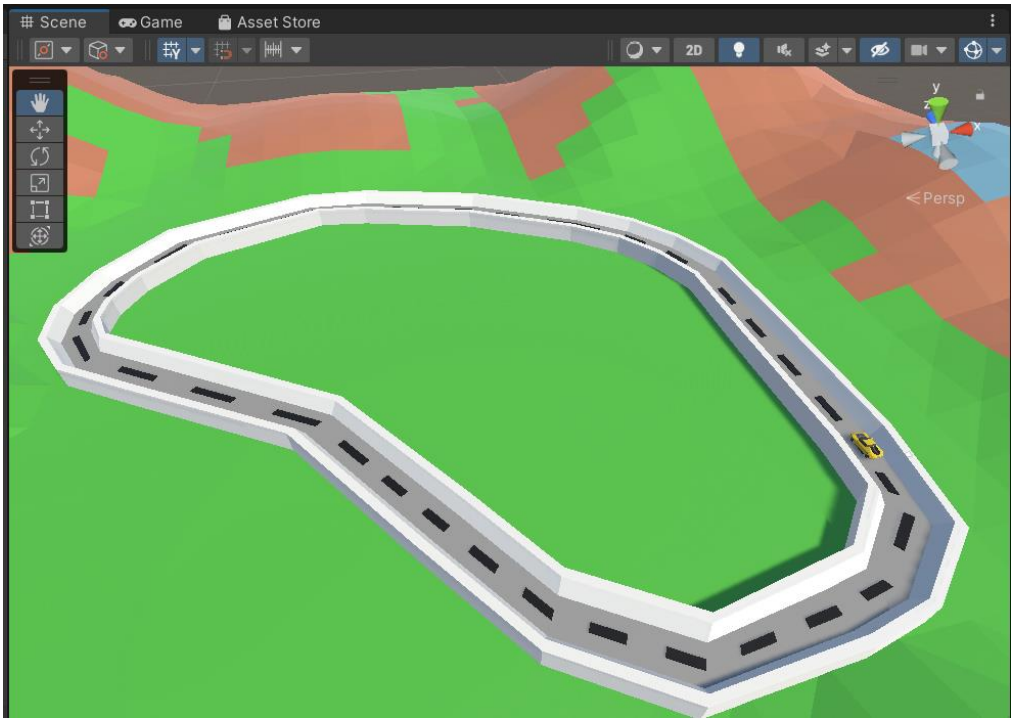

Рисунок 3.2 Окружающая среда виртуальной трассы

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.

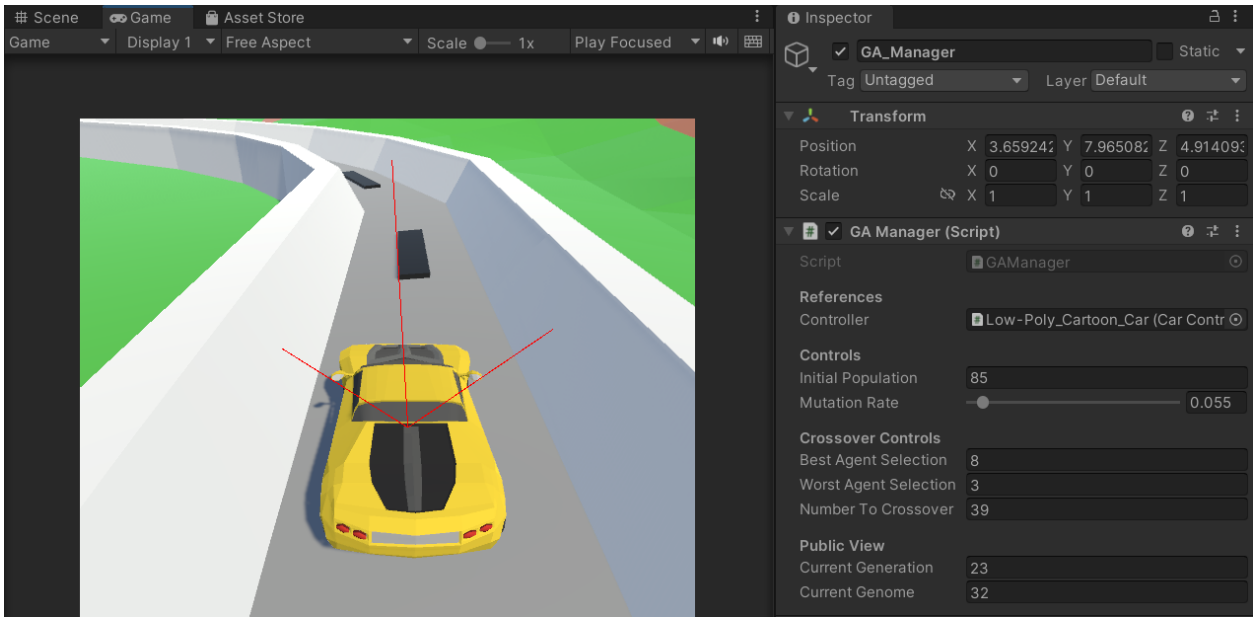

Рисунок 3.3 Трассировочные лучи

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 A (acceleration) и поворота T (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.

3.2 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 этапа есть несколько видов и логично определить базовый интерфейс для каждого из них: ISelection, IMutation, ICrossover соответственно.

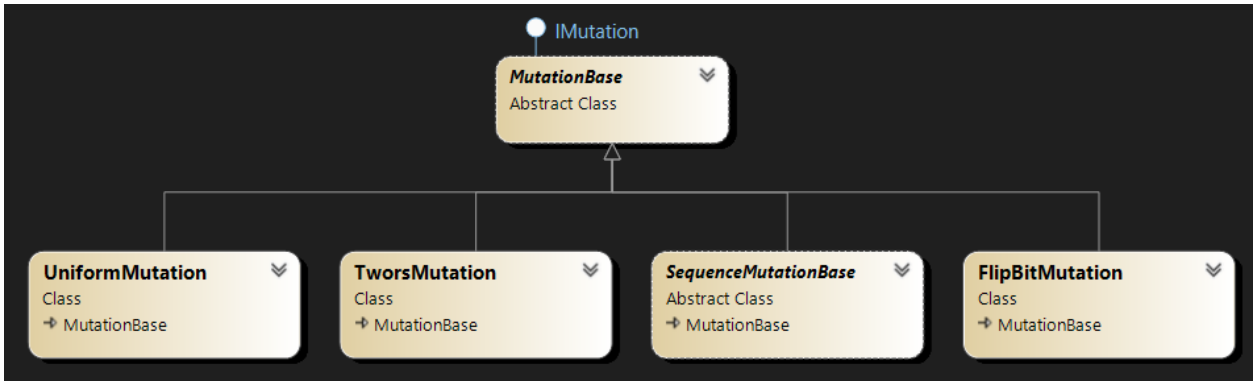

Рисунок 3.5 Структура классов оператора мутации

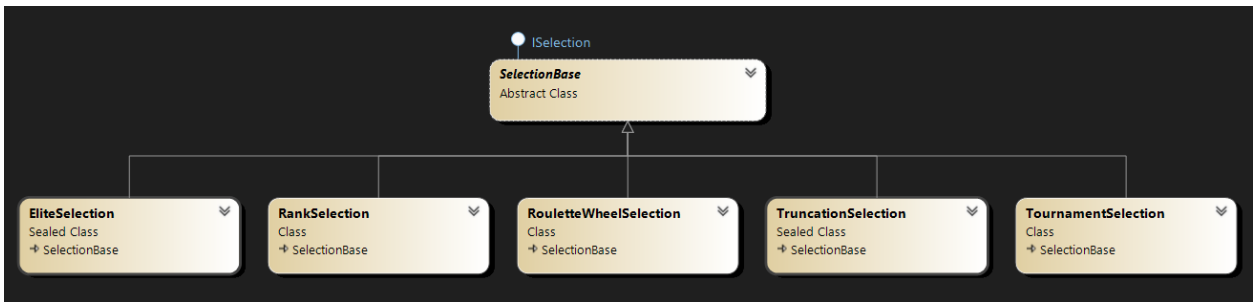

Рисунок 3.6 Структура классов оператора отбора

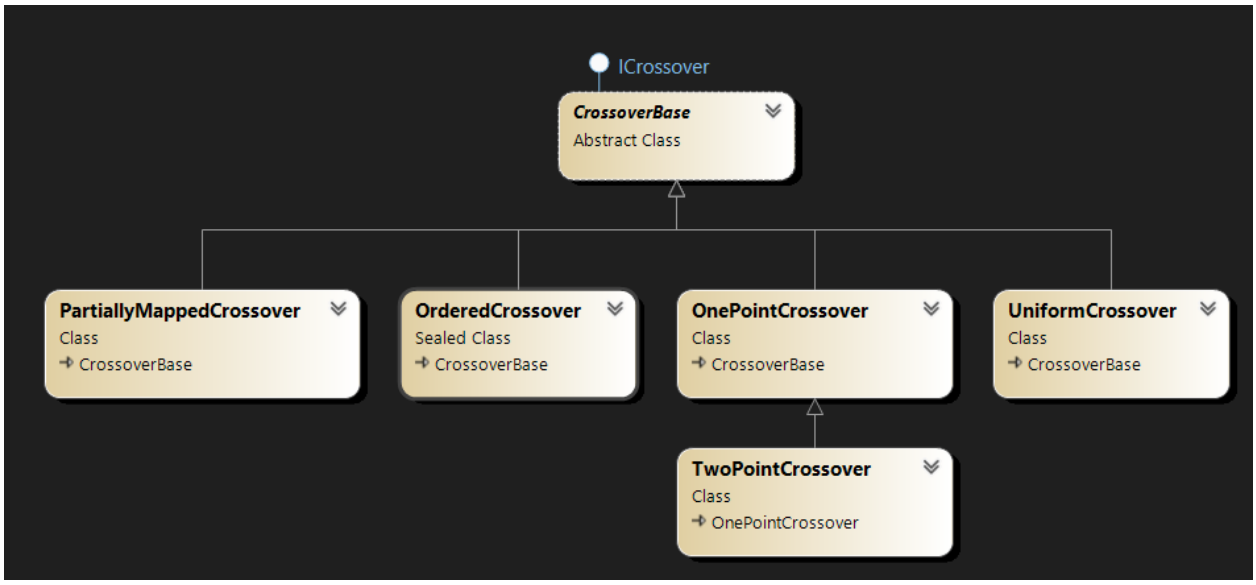

Рисунок 3.7 Структура классов оператора скрещивания

Так же немаловажным будет создать класс, который будет отвечать за сбор необходимый статистики, на примере номера популяции, лучшего индивидуума, времени затраченного на алгоритм

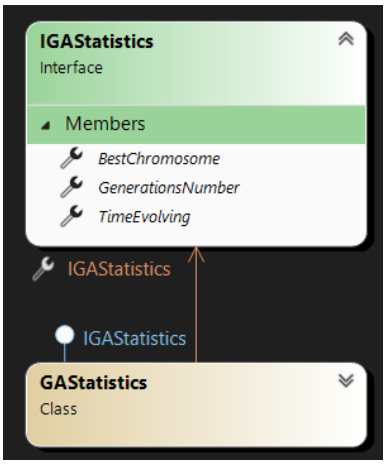

Рисунок 3.8 Структура классов предоставления статистики

3.3 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.

Одно из основных преимуществ TDD заключается в том, что он помогает обнаруживать ошибки на ранних этапах процесса разработки, что может сэкономить время и силы. Это также может помочь улучшить дизайн кода и сделать код более модульным и простым в обслуживании. TDD часто используется в сочетании с другими методологиями разработки программного обеспечения, такими как Agile, чтобы обеспечить быструю и эффективную доставку высококачественного программного обеспечения.

Будем тестировать основные этапы алгоритма, в приложении А находиться некоторые юнит-тесты написанные для них. Используя при этом, возможно, фреймворка NUnit. Каждый юнит-тест будет структурирован по шаблону AAA (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.

3.4 Вывод

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.

Так же стоит отметить TDD и модульное тестирование, они часто используются вместе, при этом тесты, написанные на TDD, служат основой для модульных тестов. Это уже на ранних этапах разработки может помочь убедиться, что код правильный и соответствует требованиям, а также может помочь улучшить дизайн кода.

ЗАКЛЮЧЕНИЕ

Как было показано, генетические алгоритмы и родственные им методы применимы к самым разным задачам практически в любых областях вычислений и техники, в т. ч., весьма возможно, и в тех, в которых подвизаетесь вы сами. Напомним, что для применения генетического алгоритма нужен лишь способ представить решение и получить его численную оценку – или хотя бы сравнить два решения. Мы живем в век искусственного интеллекта и облачных вычислений, а генетические алгоритмы хорошо приспособлены к тому и другому и могут стать мощным оружием в вашем арсенале, когда придется решать очередную задачу.

Созданная схема архитектуры хорошо адаптирована для реальных проектов. 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 конечных результат.
3. Реализовано использование ГА в нейронных сетях на примере обучения с подкреплением.
4. Разработана схема библиотеки генетического алгоритма.
5. Написаны Unit тесты для TDD подхода.

В планах на дальнейшее развитие этой темы – реализация библиотеки генетического алгоритма с учетом разных подходов при выборе стратегии решения под конкретную задачу.

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 А

```
[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.");
}
```