Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Ульяновский государственный Технический университет

Кафедра «Вычислительная техника»

Дисциплина «Системы искусственного интеллекта»

**Лабораторная работа №1**

**«Генетические алгоритмы»**

**Выполнил**:

студент гр. ИВТАСбд-41

Бородин К. Ю.

**Проверил работу:**

Хайруллин И. Д.

Ульяновск,

2025

**Общее задание**

1. Необходимо разработать программу на языке python, реализующую генетический алгоритм по предложенному вариантом заданию.
2. Провести эксперименты по разным способам скрещивания (не менее 3-х), разным способам мутирования (не менее трех). Результат отобразить в виде графиков.
3. Моделирование данных производить на основе максимально правдоподобных данных. Т.е. если рассматривается задача, в которой есть калорийность продуктов, то должны использоваться данные о реальных продуктах с реальной калорийностью.
4. Предоставить отчет о проделанной работе.

**Задача. Вариант №8:**

На языке Python разработайте скрипт, который с помощью генетического алгоритма и полного перебора решает следующую задачу. Дано N наименований продуктов, для каждого из которых известно m характеристик. Необходимо получить лучший по характеристикам рацион из k наименований, удовлетворяющий заданным ценовым рамкам. Лучшим считается рацион с минимальным отклонением от нормы.

**Теоретическая подготовка**

**Что такое генетический алгоритм?**  
Генетический алгоритм (ГА) — это метод поиска оптимальных решений, основанный на принципах естественного отбора и эволюции. Вместо того чтобы перебирать все возможные комбинации продуктов, алгоритм работает с набором кандидатов (популяцией рационов) и постепенно улучшает их, отбирая и комбинируя наиболее удачные.

**В моём коде это выглядит так:**

1. **Создание популяции.**  
   Алгоритм начинает с набора случайных рационов. Каждый рацион кодируется как бинарный вектор длиной *N*, где 1 означает, что продукт выбран, а 0 — нет. В каждом векторе ровно *k* единиц, то есть в рационе всегда выбрано строго *k* продуктов.
2. **Оценка приспособленности.**  
   Для каждого рациона вычисляется функция *fitness*, которая отражает отклонение суммарных характеристик (калорий, белков, жиров, углеводов, клетчатки, натрия) от заданных норм. Чем меньше отклонение от нормы и чем больше рацион удовлетворяет ценовым ограничениям, тем лучше приспособленность. В коде это реализовано через функцию fitness(), где штрафы добавляются за выход за пределы цены и за превышение натрия.
3. **Эволюция поколений.**  
   Алгоритм повторяет несколько шагов, формируя всё более качественные поколения рационов:
   * **Отбор.** Из текущей популяции выбираются лучшие решения. В коде используется турнирный отбор: случайно берутся несколько рационов, и выигрывает тот, у которого fitness меньше.
   * **Скрещивание.** Два «родительских» рациона комбинируются, чтобы создать потомков. Используются три метода: одноточечный кроссовер, двуточечный и равномерный. Каждый из них по-своему «перемешивает» выбранные продукты у родителей.
   * **Мутация.** В потомков вносятся небольшие случайные изменения. Это позволяет исследовать новые комбинации продуктов и не застревать в локальных решениях. У меня реализовано три типа мутаций: swap (обмен выбранного и невыбранного продукта), bitflip+repair (инверсия битов с восстановлением ровно *k* единиц) и random-reset (полная случайная замена на новый рацион).
4. **Конечный результат.**  
   После заданного числа поколений (generations) алгоритм возвращает найденный лучший рацион. Для проверки корректности решения дополнительно используется полный перебор всех комбинаций из *N* по *k*, который гарантированно находит глобальный минимум, но занимает значительно больше времени.

**Почему именно ГА?**  
Задача поиска оптимального рациона относится к классу NP-трудных: число комбинаций быстро растёт с увеличением *N* и *k*. Полный перебор возможен только для небольших наборов данных. Генетический алгоритм позволяет за разумное время найти решение, очень близкое к оптимальному, что и является целью данной лабораторной работы.

**Скрещивание (Crossover)**

Скрещивание — это процесс, при котором два родительских решения (хромосомы, представляющие наборы продуктов) обмениваются своими частями для создания новых рационов (потомков). Его цель — объединить сильные стороны обоих родителей. Например, если один рацион хорош по калориям, а другой — по содержанию белка и углеводов, скрещивание может дать потомка, который сочетает эти преимущества.

В моём коде реализованы три типа скрещивания:

* **crossover\_one\_point**: хромосома разрезается в одной случайной точке, и хвостовые части родителей обмениваются.
* **crossover\_two\_point**: хромосома разрезается в двух случайных точках, и средние сегменты родителей меняются местами.
* **crossover\_uniform**: для каждого гена (продукта) случайно выбирается, от какого родителя его взять.

**Мутация (Mutation)**

Мутация — это небольшое случайное изменение в хромосоме (выборе продуктов). Её цель — поддерживать разнообразие в популяции, чтобы алгоритм не застрял в локальном оптимуме и мог исследовать новые комбинации продуктов.

В моей реализации использовались три типа мутации:

* **mutation\_swap**: один выбранный продукт заменяется на один невыбранный (сохраняется ровно *k* продуктов).
* **mutation\_bitflip**: некоторые биты (0/1) инвертируются случайно, а затем выполняется «ремонт» хромосомы, чтобы снова было ровно *k* продуктов.
* **mutation\_random\_reset**: вся хромосома случайным образом заменяется на новый набор из *k* продуктов.

**Типы скрещивания**

Скрещивание, или кроссовер, объединяет «генетический материал» от двух рационов для создания нового потомка. Это один из главных операторов эволюции в генетическом алгоритме.

1. **Одноточечное скрещивание (one-point crossover)**
   * **Как работает:** бинарная хромосома «разрезается» в одной случайной точке. Первая часть одного родителя объединяется с хвостом другого.
   * **Преимущества:** очень простое и быстрое. Хорошо сохраняет крупные блоки генов (например, целые наборы продуктов).
   * **Недостатки:** если полезные продукты распределены по всей хромосоме, а не сосредоточены в одном блоке, эффективность может снижаться.
2. **Двухточечное скрещивание (two-point crossover)**
   * **Как работает:** выбираются две точки разреза, и средние сегменты родителей меняются местами.
   * **Преимущества:** обеспечивает обмен сразу несколькими блоками продуктов. Это даёт больше разнообразия и комбинаций.
   * **Недостатки:** всё ещё «жёстко блочное», и полезные гены, которые находятся вне этих сегментов, не всегда сохраняются.
3. **Равномерное скрещивание (uniform crossover)**
   * **Как работает:** для каждого гена (продукта) бросается «монетка», и он копируется либо от первого, либо от второго родителя.
   * **Преимущества:** даёт максимальное разнообразие, позволяет детально перемешивать продукты из разных рационов. Это помогает избежать застревания в локальных решениях.
   * **Недостатки:** может разрушать удачные комбинации продуктов, которые лучше работали вместе.

Мутация вносит случайные изменения в хромосому (набор выбранных продуктов). Это поддерживает генетическое разнообразие и помогает находить решения, которые не удаётся получить только с помощью скрещивания.

1. **Swap-мутация (mutation\_swap)**
   * **Как работает:** выбирается один продукт из уже выбранных (1) и один из невыбранных (0), после чего они меняются местами. Таким образом сохраняется условие «ровно *k* продуктов в рационе».
   * **Преимущества:** простая и «мягкая» мутация — изменяется только один элемент рациона. Это позволяет точечно корректировать состав без резкого ухудшения решения.
   * **Недостатки:** небольшие изменения могут быть слишком слабыми, если алгоритм застрял в локальном оптимуме.
2. **Битфлип с восстановлением (mutation\_bitflip)**
   * **Как работает:** каждый бит хромосомы (продукт) инвертируется с небольшой вероятностью. Если после этого количество выбранных продуктов перестаёт быть равно *k*, применяется «ремонт» — случайно удаляются или добавляются продукты, чтобы вернуть ровно *k* единиц.
   * **Преимущества:** может одновременно поменять сразу несколько продуктов, что позволяет делать более крупные шаги в пространстве решений.
   * **Недостатки:** без ремонта хромосома может стать некорректной; а после ремонта часть случайных изменений теряется.
3. **Мутация случайной замены (mutation\_random\_reset)**
   * **Как работает:** с определённой вероятностью вся хромосома заменяется на полностью новый случайный набор из *k* продуктов.
   * **Преимущества:** сильная мутация, которая может вывести алгоритм из тупика и исследовать совершенно новые комбинации продуктов.
   * **Недостатки:** слишком резкое изменение может уничтожить уже найденные полезные комбинации, и придётся снова «учиться» их находить.

**Что такое отбор**

Отбор — это процесс выбора «родителей» из текущей популяции рационов для создания потомков. Чем меньше значение функции приспособленности (то есть чем ближе рацион к нормам и ценовым ограничениям), тем выше вероятность, что он будет выбран.

В моём коде используется **турнирный отбор (tournament selection)**:

* **Как работает:** случайным образом выбирается небольшая группа особей (например, 3 рациона). Из них побеждает тот, у которого значение функции приспособленности наименьшее. Этот процесс повторяется, пока не будет набрано нужное количество родителей.
* **Почему это эффективно:** турнирный отбор прост в реализации и хорошо балансирует: он чаще выбирает сильных кандидатов, но при этом не исключает полностью шанс слабых решений. Благодаря этому в популяции сохраняется генетическое разнообразие, что важно для поиска глобально лучшего рациона.

**Описание набора данных**

В основе набора данных лежат два ключевых элемента: **продукты и нормы потребления**.  
Продуктов 20 (*N=20*), и для каждого заданы реальные характеристики на 100 г: калорийность (kcal), количество белков, жиров, углеводов, клетчатки, натрия, а также цена в евро. Эти параметры определяют питательную и экономическую ценность каждого продукта.

Нормы потребления выступают в роли целевых ориентиров, к которым должен стремиться алгоритм. Они включают рекомендуемое суточное количество калорий, белков, жиров, углеводов, клетчатки и максимально допустимое количество натрия.

Помимо основных характеристик, набор данных включает **ценовые ограничения**: суммарная стоимость выбранного рациона должна находиться в пределах от 2.5 до 5.0 евро. Эти рамки помогают алгоритму находить практичные решения, соответствующие реальным условиям.

Таким образом, набор данных сочетает в себе реальные пищевые характеристики продуктов и ограничения по цене и нормам. Эти параметры задают условия, в которых генетический алгоритм ищет наилучший рацион с минимальным отклонением от целевых значений.

**Описание реализации**

Мой код представляет собой комплексное решение для задачи оптимизации рациона питания, использующее два подхода: генетический алгоритм (ГА) и полный перебор. Главная цель — найти такой набор из *k* продуктов, который одновременно удовлетворяет ценовым ограничениям и как можно точнее соответствует нормам по калориям, белкам, жирам, углеводам, клетчатке и натрию.

Программа разделена на несколько логических блоков:

1. **Инициализация и данные**  
   В начале определяются все исходные данные: список продуктов (*N* наименований) и их характеристики (калорийность, содержание белков, жиров, углеводов, клетчатки, натрия и цена за 100 г). Также задаются параметры задачи: количество продуктов в рационе (*k*), ценовые рамки (минимальная и максимальная стоимость), а также нормы по каждому из нутриентов. Эти данные составляют основу, с которой работает алгоритм.
2. **Функция приспособленности**  
   Функция fitness() является «сердцем» генетического алгоритма. Она берёт любое решение (хромосому — бинарный вектор длины *N* с ровно *k* единицами) и оценивает его качество. Расчёт включает в себя:
   * **Суммарные нутриенты**: вычисляется общая калорийность, количество белков, жиров, углеводов, клетчатки и натрия для выбранных продуктов.
   * **Отклонение от нормы**: рассчитываются относительные ошибки по каждому параметру; для натрия учитывается только превышение нормы.
   * **Штрафы**: добавляются большие штрафы, если рацион выходит за рамки цены или если число выбранных продуктов отличается от *k*.

В результате эта функция возвращает значение приспособленности: чем оно меньше, тем рацион ближе к нормам и ограничениям, и тем более «приспособленным» считается решение.

**3. Операторы генетического алгоритма**

Этот блок содержит функции, которые имитируют эволюционные процессы:

* **Создание начальной популяции** (random\_mask\_k): формируются первые случайные решения — бинарные векторы длиной *N* с ровно *k* единицами (каждая особь — рацион из *k* продуктов).
* **Отбор** (tournament\_select): выбирает родителей для следующего поколения с помощью турнирного отбора.
* **Скрещивание (crossover):** функции crossover\_one\_point, crossover\_two\_point и crossover\_uniform реализуют разные варианты обмена генами (продуктами) между родителями и создают новых потомков.
* **Мутация (mutation):** функции mutation\_swap, mutation\_bitflip и mutation\_random\_reset вносят случайные изменения в хромосомы, что поддерживает генетическое разнообразие в популяции и предотвращает преждевременную сходимость.

**4. Основной цикл ГА**

Функция run\_ga объединяет все операторы и управляет процессом эволюции. Она выполняет цикл заданного числа поколений (generations), в каждом из которых:

1. Оценивается приспособленность всех рационов в популяции с помощью функции fitness().
2. Отбираются родители для скрещивания (турнирный отбор).
3. Создаются новые потомки с помощью выбранного кроссовера и мутации.
4. Обновляется популяция: сохраняется несколько лучших решений (элитизм), остальные заменяются новыми потомками.

В конце цикла возвращается лучшее найденное решение и история изменения значения функции приспособленности.

**5. Полный перебор**

Функция brute\_force решает задачу абсолютно точно, перебирая все возможные комбинации из *N* по *k* продуктов и выбирая среди них оптимальный рацион, удовлетворяющий ценовым ограничениям. Такой подход гарантирует нахождение глобального оптимума, но становится крайне затратным по времени при увеличении количества продуктов. Это наглядно показывает, почему для практических задач используется генетический алгоритм.

**6. Запуск и анализ результатов**

В финальном блоке программы запускаются эксперименты:

* сравниваются разные виды скрещивания (3 варианта) и мутации (3 варианта),
* строятся графики сходимости по поколениям,
* выполняется проверка найденных решений с помощью полного перебора.

В результате можно увидеть, как приспособленность популяции меняется от поколения к поколению, какие операторы дают наилучший результат, и что генетический алгоритм позволяет находить почти оптимальные решения гораздо быстрее, чем полный перебор.

**Общее задание**

1. **Какова структура хромосомы для вашей задачи?**

Хромосома в моей задаче представляет собой **бинарный вектор длиной *N***, где *N* — количество доступных продуктов.

* Каждая позиция хромосомы соответствует одному продукту.
* Если значение равно 1, продукт включён в рацион; если 0 — продукт не выбран.
* При этом в каждом решении должно быть выбрано ровно *k* продуктов (количество единиц = *k*).

Таким образом, хромосома полностью описывает один рацион питания. Например, вектор [0, 1, 0, 1, 1, …] означает, что в рацион вошли продукты под индексами 2, 4 и 5.

1. **Какую функцию приспособленности вы выбрали и почему?**

Функция приспособленности (fitness) вычисляет, насколько близко рацион соответствует заданным нормам по калориям, белкам, жирам, углеводам, клетчатке и натрию, а также удовлетворяет ценовым ограничениям.

Расчёт выполняется следующим образом:

1. Суммируются все характеристики выбранных продуктов.
2. Для каждого параметра вычисляется отклонение от нормы (в относительных величинах).
   * Для натрия учитывается только превышение нормы (если меньше нормы — штрафа нет).
3. Все квадраты отклонений суммируются.
4. Если нарушены ценовые рамки или количество выбранных продуктов не равно *k*, добавляется большой штраф.

**Почему именно такая функция:**

* Она напрямую минимизирует **отклонение от нормы**: чем рацион ближе к целевым значениям, тем меньше значение fitness.
* Введённые штрафы исключают непригодные решения (например, слишком дешёвые или слишком дорогие рационы).
* Такая формулировка создаёт чёткую иерархию решений: рацион с меньшими отклонениями всегда считается лучше.

1. **Что такое кроссовер и мутация?**

**Кроссовер** — это процесс, при котором два родительских рациона обмениваются частями своих «генов» (выбранных продуктов), чтобы создать новых потомков. Его цель — объединять сильные стороны разных решений.  
В моей реализации предусмотрены три вида скрещивания:

* **Одноточечное (one-point):** хромосома разрезается в одной случайной точке, и хвостовые части родителей обмениваются.
* **Двухточечное (two-point):** выбираются две точки разреза, и средние сегменты родителей меняются местами.
* **Равномерное (uniform):** для каждого продукта случайно определяется, от какого родителя он будет взят.

**Мутация** — это небольшое случайное изменение в хромосоме. Она вносит разнообразие и помогает алгоритму исследовать новые комбинации продуктов.  
В моей задаче реализованы три типа мутации:

* **Swap:** один выбранный продукт заменяется на один невыбранный.
* **Bitflip с восстановлением:** некоторые биты случайно инвертируются, после чего хромосома «чинится», чтобы снова содержать ровно *k* продуктов.
* **Random reset:** вся хромосома случайно сбрасывается и заменяется на новый набор из *k* продуктов.