МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Применение метода имитации отжига для решения комбинаторных задач

Обучающийся Д.А. Осягин
(Инициалы Фамилия) (личная подпись)

Руководитель д.т.н., доцент, С.В. Мкртычев
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.п.н, доцент, А.В. Егорова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

## Аннотация

Тема выпускной квалификационной работы (далее – ВКР): «Применение метода имитации отжига для решения комбинаторных задач».

Исследование и особенности практического применения метода имитации отжига для решения комбинаторных задач представляет актуальность и научно-практический интерес, поскольку данный метод используется для нахождения оптимальных решений в задачах, где пространство возможных решений очень велико, и нет гарантии нахождения идеального решения.

Объектом исследования бакалаврской работы является метод имитации отжига.

Предметом исследования бакалаврской работы являются решения комбинаторных задач с использованием метода имитации отжига.

Цель работы заключается в исследовании и реализации метода имитации отжига для решения комбинаторных задач.

Практическая значимость работы заключается в разработке и тестировании программы, которая может эффективно реализовывать алгоритм имитации отжига.

Результаты бакалаврской работы представляют научно-практический интерес и могут быть рекомендованы для анализа и программной реализации метода имитации отжига для решения комбинаторных задач.

Бакалаврская работа состоит из 49 страниц текста, 18 рисунков, 5 таблиц и 24 источников.

## Abstract

The title of the graduation project is Application of the Simulated Annealing Method for Solving Counting Problems.

This research addresses the study and practical application of the simulated annealing method for solving counting problems, which is of current scientific and practical interest. This method is employed to find optimal solutions in scenarios where the solution space is vast, and there is no guarantee of finding a perfect solution.

The object of the study is the simulated annealing method.

The subject of the study is the solutions to counting problems using the simulated annealing method.

The aim of the work is to investigate and implement the simulated annealing method for solving counting problems.

The practical significance of this work lies in the development and testing of a program capable of effectively implementing the simulated annealing algorithm.

The results of the thesis have both scientific and practical relevance and can be recommended for the analysis and software implementation of the simulated annealing method to solve counting problems.

The graduation work consists of an explanatory note on 49 pages, introduction, including 18 figures, 5 tables, and the list of 24 references.

## Оглавление

# Введение

Решение комбинаторных задач является важной и актуальной проблемой в различных областях, таких как логистика, планирование и оптимизация. Для достижения эффективных результатов в этих областях необходимы алгоритмы, способные находить качественные решения в условиях большого пространства возможных вариантов.

Метод имитации отжига (англ. Simulated Annealing, SA) – это один из таких алгоритмов, который имитирует физический процесс отжига металлов. «В процессе отжига металл нагревается до высокой температуры, а затем медленно охлаждается, что приводит к улучшению его структуры» [4]. В контексте алгоритмов оптимизации метод имитации отжига применяется для поиска оптимальных решений в сложных задачах. Алгоритм стартует с произвольного начального решения и улучшает его путем внесения случайных изменений. Если новое решение оказывается лучше текущего, оно принимается. Если новое решение хуже, его также могут принять, но с определенной вероятностью, которая зависит от разницы в качестве решений и текущей температуры. «В процессе выполнения алгоритма температура постепенно снижается, что уменьшает вероятность принятия менее эффективных решений. Процесс продолжается до достижения заданного критерия остановки или исчерпания лимита итераций» [1].

Современные исследования показывают, что метод имитации отжига эффективен для решения задач, в которых пространство возможных решений чрезвычайно велико и отсутствует гарантия нахождения идеального решения. Этот метод может находить приближенные решения для таких задач, как задача коммивояжера, раскраска графа, расстановка ферзей и многие другие.

Однако, несмотря на значительные достижения, остаются нерешенные вопросы, связанные с оптимизацией параметров метода и улучшением его производительности для различных типов задач. Необходимы дальнейшие

исследования и разработка новых подходов, которые могли бы повысить эффективность и точность данного алгоритма.

Объектом исследования ВКР является метод имитации отжига.

Предметом исследования ВКР являются решения комбинаторных задач с использованием метода имитации отжига.

Цель работы заключается в исследовании и реализации метода имитации отжига для решения комбинаторных задач.

Для достижения этой цели необходимо выполнить следующие задачи:

- выполнить постановку задачи исследования и проанализировать существующие методы решения комбинаторных задач;
- проанализировать алгоритм имитации отжига;
- разработать и протестировать программу, реализующую алгоритм имитации отжига для решения комбинаторных задач.

Практическая значимость работы заключается в разработке и тестировании программы, которая может эффективно реализовывать алгоритм имитации отжига.

Данная работа включает в себя введение, три главы, заключение и список литературы.

Первая глава посвящена постановке задачи исследования и анализу методов решения комбинаторных задач.

Вторая глава посвящена анализу алгоритма имитации отжига для решения комбинаторных задач.

В третьей главе рассматривается программная реализация и тестирование алгоритма метода имитации отжига для решения комбинаторных задач.

В заключении описываются результаты выполнения бакалаврской работы.

# Глава 1 Постановка задачи исследования и анализ методов решения комбинаторных задач

Постановка задачи.

- описание комбинаторных задач, выбор и анализ примера;
- изучение существующих методов решения комбинаторных задач;
- анализ теоретических основ и схем алгоритма имитации отжига;
- реализация различных вариантов алгоритма имитации отжига для решения комбинаторных задач;
- оценка эффективности алгоритмов на примере задачи о ранце;
- тестирование программной реализации с использованием разных схем отжига и параметров температуры;
- сравнительный анализ результатов работы программных реализаций алгоритмов.

Рассмотрим определение комбинаторных задач более подробно и проведем анализ одной из них.

## 1.1 Определение комбинаторных задач

Комбинаторная оптимизация – это область математической оптимизации, которая занимается нахождением наилучшего решения из конечного множества возможных решений. В таких задачах возможные решения либо дискретны, либо могут быть сведены к дискретному набору.

Задачи можно разделить на два класса:

- задачи, решаемые за полиномиальное время: эти задачи могут быть сформулированы, как задачи линейного программирования, где число допустимых значений, для которых находится оптимум (максимум или минимум) целевой функции, бесконечно. Однако, стандартные методы решения сводят выбор к конечному множеству вариантов;

- задачи с конечным множеством допустимых вариантов: в таких задачах можно использовать полный перебор всех возможных вариантов, хотя их количество может значительно превышать длину описания задачи (например, задача коммивояжера). Обычно такие задачи сводятся к задаче целочисленного программирования. Решение задач второго класса за полиномиальное время является открытой проблемой. Известно, что наличие полиномиального алгоритма для любой из задач второго класса подразумевает существование такого алгоритма для всех задач данного класса.

Примеры типичных комбинаторных задач:

- задача коммивояжера (англ. Traveling Salesman Problem): задача заключается в нахождении кратчайшего пути, проходящего через заданные города ровно один раз и возвращающегося в исходный город;
- задача о минимальном остовном дереве (англ. Minimum Spanning Tree): задача состоит в нахождении подмножества ребер графа, соединяющего все вершины с минимальной суммой весов ребер;
- задача о расстановке N-ферзей (англ. N-Queen Problem): задача размещения N ферзей на шахматной доске NxN так, чтобы ни один ферзь не угрожал другому;
- задача о ранце (англ. Knapsack Problem): задача максимизации стоимости предметов, помещаемых в рюкзак с ограниченной вместимостью, при условии, что суммарный вес предметов не превышает эту вместимость.

Рассмотрим задачу о ранце.

Задача о ранце является типичным примером комбинаторной оптимизации. Она относится к классу NP-полных задач, для которых не существует полиномиального алгоритма, способного решить её за приемлемое время. Сложность заключается именно в этом.

Основная цель задачи – максимизировать суммарную стоимость предметов в рюкзаке, при этом соблюдая ограничение на вес, чтобы общая масса не превышала вместимость рюкзака. В данной задаче имеется только один экземпляр каждого предмета, и для каждого предмета существуют только два возможных состояния: либо он включен в рюкзак, либо нет. Каждый предмет нельзя класть в рюкзак более одного раза или частично класть в рюкзак.

«KP ограничивает количество копий каждого вида товара нулем или единицей. Задан набор из n предметов, пронумерованных от 1 до n, каждый из которых имеет вес $w_i$ и стоимость $v_i$, а также максимальную грузоподъемность W. Значение KP может быть сформулировано следующим образом» [18] (1,2):

$$\sum_i^n v_i x_i \ , \ x_i = \begin{cases} 1, \text{если предмет в рюкзаке} \\ 0, \text{в противном случае} \end{cases}, \quad (1)$$

$$\sum_i^n v_i x_i \leq W \ \text{и} \ x_i \in \{0, 1\}, \quad (2)$$

где $x_i$ – количество экземпляров элемента i, которые должны быть включены в рюкзак.

Предположим, у нас есть рюкзак вместимостью W = 15 и несколько предметов разного веса и с разной стоимостью. Мы хотим, чтобы в рюкзак были включены только те предметы, которые принесут наибольшую пользу в рамках вместимости рюкзака. Есть три возможных предмета: A, B и C (их вес и стоимость приведены в таблице 1).

Таблица 1 – Вес и стоимость предметов

| Предмет | Вес | Стоимость |
|---|---|---|
| A | 9 | 2 |
| B | 6 | 5 |
| C | 7 | 4 |

Мы хотим максимизировать общую выгоду (3):

$$\sum_{i=1}^{3} v_i x_i = 2x_1 + 5x_2 + 4x_3 \quad (3)$$

При соблюдении следующих ограничений (4):

$$\sum_{i=1}^{3} v_i x_i = 9x_1 + 6x_2 + 7x_3 \leq 15, x_i \in \{0,1\}, i = 1,2..n \quad (4)$$

Чтобы найти наилучшее решение, мы должны определить подмножество, которое удовлетворяет ограничению и имеет максимальную общую выгоду. Для этой задачи существует 8 возможных подмножества элементов, как показано в таблице 2.

Таблица 2 – Возможные подмножества решений для задачи о ранце

| A | B | C | Стоимость | Вес |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 4 | 7 |
| 0 | 1 | 0 | 5 | 6 |
| 1 | 0 | 0 | 2 | 9 |
| 0 | 1 | 1 | 9 | 13 |
| 1 | 0 | 1 | - | 16 |
| 1 | 1 | 0 | 7 | 15 |
| 1 | 1 | 1 | - | 22 |

Оптимальная выгода для данного ограничения (W = 15) может быть получена только при взятии предметов B и C, и результирующая выгода равна 9.

### 1.2 Традиционные методы решения комбинаторных задач

Прежде чем обсудить каждый из методов, разделим их на две группы:

а) точные методы:

1) метод полного перебора,

2) метод ветвей и границ,

3) динамическое программирование;

б) приближенные и эвристические методы:

1) жадный алгоритм,

2) имитация отжига.

Рассмотрим каждый из методов более подробно, чтобы понять их особенности и области применения.

Точные методы.

Метод полного перебора.

«Метод полного перебора принадлежит к категории методов, которые ищут решение путем исчерпывающего перебора всех возможных вариантов» [13].

Метод ветвей и границ.

«Метод ветвей и границ (ВВ) впервые был предложен Лендом и Дойгом в 1960 году для задач целочисленного программирования» [11]. Позже этот алгоритм стал известен как «алгоритм Литтла». Фактически, этот метод является улучшенной версией полного перебора, в рамках которого последовательно отбрасываются решения, кажущиеся невыгодными. «ВВ используется для нахождения оптимального решения, сохраняя лучшее найденное на данный момент решение.

Алгоритм ветвей и границ (ВВ) строит кандидатные решения по одному компоненту за раз и оценивает частично построенные решения. Если частичное решение не может улучшить текущее оптимальное решение, оно отбрасывается. Этот подход позволяет решать некоторые крупные задачи сложной комбинаторной оптимизации. Алгоритм основан на построении дерева состояний. Дерево состояний – это корневое дерево, где каждый уровень представляет выбор в пространстве решений, зависящий от уровня выше, и любое возможное решение представлено некоторым путем,

начинающимся с корня и заканчивающимся на листе. Значение границы узла сравнивается со значением текущего лучшего решения. Если значение границы не лучше текущего лучшего решения, то есть не меньше для минимизации и не больше для максимизации, узел считается бесперспективным и может быть завершен, поскольку ни одно решение, полученное из него, не может дать лучшее решение, чем уже имеющееся. Это основная идея алгоритма ветвей и границ» [21,22]. Далее мы обсудим стратегию ветвления.

Существует два основных способа реализации ветвления:

- «ветвление на узле с наименьшей границей: поиск всех узлов и нахождение узла с наименьшей границей и установка его в качестве следующего ветвящегося узла» [13]. Преимущество: проверяет меньше подзадач, что экономит вычислительное время. Недостаток: требует больше памяти;
- «ветвление на вновь созданном узле с наименьшей границей: поиск вновь созданных узлов и нахождение узла с наименьшей границей и установка его в качестве следующего ветвящегося узла» [20]. Преимущество: экономит память. Недостаток: требует больше усилий для ветвления и поэтому не является вычислительно эффективным.

В контексте алгоритма ветвей и границ для решения задачи о ранце (KP), «если имеется N возможных предметов для выбора, тогда k-й уровень представляет состояние, в котором было решено, какие из первых k предметов включены или не включены в рюкзак. В этом случае на k-м уровне существует$2^k$ узлов, и листья дерева состояний находятся на уровне N» [16].

Динамическое программирование.

«Еще одним широко используемым точным подходом является динамическое программирование (DP). Этот метод был разработан Ричардом Беллманом в 1950-х годах и нашел применение во многих областях, от аэрокосмической техники до экономики» [23,24].

«В обоих контекстах это означает упрощение сложной задачи путем разбиения ее на более простые подзадачи рекурсивным способом. Хотя некоторые задачи, связанные с принятием решений, невозможно разделить таким образом, решения, которые охватывают несколько моментов времени, часто разделяются рекурсивно» [19]. «Аналогично, в информатике, если проблему можно решить оптимальным образом, разбив ее на подзадачи и затем рекурсивно найдя оптимальные решения подзадач, то говорят, что она имеет оптимальную подструктуру» [3].

Если подзадачи могут быть рекурсивно вложены в более крупные задачи, так что методы динамического программирования применимы, то существует связь между значением более крупной задачи и значениями подзадач.

С точки зрения математической оптимизации, динамическое программирование обычно означает упрощение решения путем его разбиения на последовательность шагов принятия решения во времени.

«Это делается путем определения последовательности функций стоимости $V_1$, $V_2$, ..., $V_n$, принимающих y в качестве аргумента, представляющего состояние системы в моменты времени i от 1 до n.

Определение Vn(y) – это значение, полученное в состоянии y в последний момент времени n.

Значения $V_i$ в более ранние моменты времени i = n - 1, n - 2, ..., 2, 1 могут быть найдены путем работы в обратном направлении, используя рекурсивное соотношение, называемое уравнением Беллмана.

Для i = 2, ..., n, $V_{i-1}$ в любом состоянии y вычисляется из $V_i$ путем максимизации простой функции (обычно суммы) выигрыша от решения в момент времени i - 1 и функции $V_i$ в новом состоянии системы, если это решение будет принято.

Поскольку $V_i$ уже вычислена для нужных состояний, вышеописанная операция дает $V_{i-1}$ для этих состояний.

Наконец, $V_1$ в начальном состоянии системы является значением оптимального решения» [19]. Оптимальные значения переменных решения могут быть восстановлены по одному, прослеживая назад уже выполненные вычисления.

Приближенные и эвристические методы.

Эвристические методы для приближенного решения задач показывают высокую эффективность, так как позволяют избежать полного перебора всех возможных вариантов. Вместо поиска оптимального решения они вычисляют приблизительное решение, которое затем можно улучшить. «Жадный алгоритм, в частности, основывается на выборе локально оптимальных решений на каждом этапе, с предположением, что конечное решение будет близким к оптимальному» [2].

Рассмотрим жадный алгоритм.

Жадный алгоритм (GA) использует эвристическую технику для принятия локально оптимальных решений на каждом шаге, стремясь таким образом найти глобальный оптимум. Однако, из-за своей особенности, жадный алгоритм часто не может достичь глобально оптимального решения, поскольку не рассматривает все возможные варианты. «Он может делать преждевременные выборы, которые препятствуют нахождению наилучшего общего решения» [6].

Допустим, что есть целевая функция, которую необходимо оптимизировать (максимизировать или минимизировать). Жадный алгоритм на каждом этапе делает выбор, направленный на оптимизацию целевой функции, но у него есть только одна попытка вычислить оптимальное решение, и он никогда не возвращается к уже принятым решениям для их изменения.

Некоторые ограничения жадного алгоритма:

- жадный алгоритм не всегда находит глобально оптимальное решение;
- он требует дополнительных усилий для правильного понимания задач. Даже если алгоритм правильный, доказать это бывает сложно;
- в некоторых случаях жадный алгоритм может привести к наихудшему возможному решению.

Рассмотрим метод имитации отжига (SA).

Метод имитации отжига (Simulated Annealing, SA) вдохновлен аналогией с процессом отжига металлов, который способствует улучшению структуры материала и уменьшению его дефектов. В 1983 году С. Киркпатрик, С. Джелатто и М. Веччи предложили SA как метод оптимизации для решения задач, связанных с распределением энергии. «Основываясь на сходстве с процессом отжига в материаловедении, где металл постепенно остывает для достижения состояния с наименьшей энергией, метод имитации отжига использует этот принцип для поиска глобального минимума целевой функции в задачах оптимизации» [16].

«На каждой итерации алгоритма SA, применяемого к дискретной оптимизационной задаче, сравниваются значения объективной функции для двух решений (текущего и вновь сгенерированного соседнего). Лучшие решения принимаются всегда, а часть худших решений принимается в надежде избежать локальных оптимумов в поисках глобального оптимума. Ключевая особенность SA заключается в том, что он позволяет избежать локальных оптимумов, допуская худшие ходы (т. е. ходы к решению, которое соответствует худшему значению объективной функции)» [8].

«В рамках задачи KP, основные компоненты процесса SA включают пространство решений, структуру окрестностей, функцию стоимости и график отжига» [15]. Обычной практикой является игнорирование невыполнимых решений (т. е. решений, в которых вес превышает максимальную вместимость ранца) и «выкидывание» элементов из ранца до тех пор, пока они не станут

выполнимыми. Однако существует четкий компромисс между качеством решения и временем, необходимым для его получения: точность используемых чисел может существенно повлиять на качество результата.

Представление: используется двоичное представление, где значение 1 означает, что предмет помещен в ранец, в то время как значение 0 означает, что предмет оставлен.

Генерируется цепь Маркова конечной длины при определенной температуре $T_k$ и параметре управления температурой α, который медленно охлаждает металл до определенной температуры, при которой металл замораживается.

Генерация состояния: Генерация состояния выполняется путем случайного изменения текущего решения.

Критерии завершения: Алгоритм завершается, когда температура достигает минимально допустимой температуры, что указывает на то, что металл заморожен.

Преимущества алгоритма имитации отжига:

- способность избегать застревания в локальных минимумах за счет приема худших решений;
- простота реализации и широкая адаптируемость к различным видам оптимизационных задач;
- эффективность в поиске глобального оптимума для широкого круга проблем, включая сложные комбинаторные задачи.

Ограничения и недостатки:

- зависимость от правильной настройки параметров, включая начальную температуру и график охлаждения, что может потребовать значительного количества времени и экспериментов;
- потенциально высокая вычислительная сложность и время выполнения, особенно для задач большой размерности;

− не гарантирует нахождение абсолютно оптимального решения, особенно при недостаточном количестве итераций или слишком быстром охлаждении.

Представим сводную таблицу по основным методам, преимущества и недостатки (таблица 3).

Таблица 3 – Преимущества и недостатки основных методов решения комбинаторных задач

| Методы решения комбинаторных задач | Преимущества | Недостатки |
|---|---|---|
| Динамическое программирование | Может найти оптимальное решение. | Требует много вычислительных ресурсов и памяти. |
| Метод ветвей и границ | Может найти оптимальное решение при условии, что проблема ограниченного размера и задача может быть решена за разумное время. | Могут быть проблемы с эффективностью при больших размерах задач. |
| Жадный алгоритм | Довольно просто написать жадный алгоритм для решения задачи.<br>Требует меньше вычислительных ресурсов.<br>Быстро выполняется. | Часто не находит оптимального решения. |
| Алгоритм Метрополиса (имитация отжига) | Он может работать с сильно нелинейными моделями, хаотичными данными.<br>Он гибок и способен избегать локального оптимума. | Не гарантирует нахождение оптимального решения. |

Выводы по главе 1

В первой главе ВКР были рассмотрены основные методы решения комбинаторных задач, такие как точные и эвристические методы.

Основное внимание уделено задаче о ранце, которая является одной из классических задач комбинаторной оптимизации. Метод имитации отжига, благодаря своей способности избегать локальных минимумов и гибкости настройки, выделяется как перспективный инструмент для решения таких задач.

# Глава 2 Анализ алгоритма имитации отжига для решения комбинаторных задач

## 2.1 Теоретические основы алгоритма имитации отжига

Алгоритм SA основан на итеративном процессе, где каждая итерация представляет собой попытку улучшить текущее решение за счет случайного изменения его параметров и оценки полученного нового решения с использованием целевой функции. Важной особенностью SA является его способность принимать худшие решения с некоторой вероятностью, которая зависит от параметра, называемого температурой. Эта особенность позволяет алгоритму избежать застревания в локальных минимумах целевой функции.

«Посредством моделирования данного процесса определяется точка или множество точек, при которых достигается минимум определённой числовой функции $F(x)$, где $x=(x_1,..,x_m) \in X$. Задается последовательность точек $x_0, x_1, \ldots, x_n$ пространства $X$. Алгоритм последовательно определяет следующую точку, начиная с начальной точки $x_0$, которая является начальным приближением. Алгоритм останавливается по достижении точки $x_n$, которая оказывается при температуре, равной нулю» [12].

Переход от точки $x_i$ к точке $x_{i+1}$ осуществляется следующим образом: к точке $x_i$ применяется оператор $A$, который случайным образом модифицирует соответствующую точку, в результате чего получается новая точка $x^*$.

«Точка $x^*$становится точкой $x_{i+1}$ с вероятностью $P$ $(x^* \rightarrow x_{i+1})$, которая вычисляется в соответствии с распределением Гиббса (5):

$$P(x^* \rightarrow x_i + 1|x_i) = exp\left(-\frac{\frac{1}{F(x)}-F(x)}{T_i}\right), \quad (5)$$

где $T_i > 0$ – элементы произвольной убывающей, сходящейся к нулю положительной последовательности, аналогичной понижающейся температуры в металле»[13].

Если переход не произошёл, состояние системы остаётся прежним (6):

$$x_{i+1} = x_i. \tag{6}$$

На рисунке 1 представлена схема работы алгоритма имитации отжига

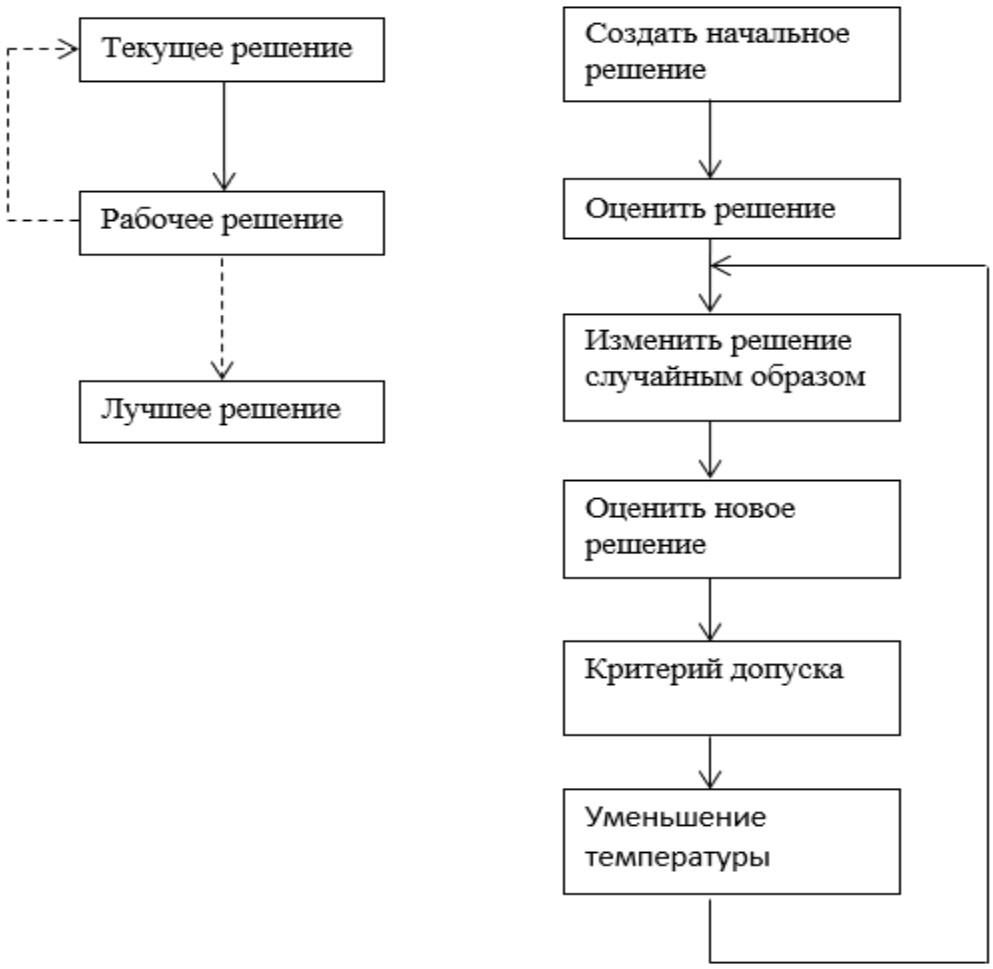


Рисунок 1 – Схема алгоритма имитации отжига

Процесс работы алгоритма включает в себя следующие шаги:

Шаг 1: задается начальное решение и начальная температура;

Шаг 2: генерируется новое решение путем случайного изменения текущего решения;

Шаг 3: рассчитывается разница в качестве между новым и текущим решениями с помощью целевой функции;

Шаг 4: у нас есть два решения - оригинальное (текущее) и найденное (рабочее). Каждому из них присвоена определенная энергия, отражающая их эффективность. «Чем ниже энергия, тем эффективнее решение. Рабочее решение сравнивается с текущим. Если его энергия меньше, чем у текущего решения, мы заменяем текущее решение на рабочее. Если же энергия рабочего решения выше, то мы применяем критерий допуска для определения дальнейших действий с текущим рабочим решением. Вероятность перехода в новое состояние рассчитывается по формуле, причем по мере снижения температуры эта вероятность также уменьшается. При высоких температурах алгоритм позволяет принимать менее эффективные решения для более широкого поиска. С понижением температуры уменьшается и диапазон поиска до тех пор, пока при температуре 0° не наступит равенство;

Шаг 5: Температура уменьшается по заранее заданному графику охлаждения, что снижает вероятность принятия менее эффективных решений на поздних этапах. В данной реализации используется простая геометрическая функция (7):

$$T_{i+1} = aT_i, \quad (7)$$

где a < 1.

Также могут использоваться другие стратегии охлаждения, такие как линейные или нелинейные функции» [10];

Шаг 6: шаги 2-5 повторяются до тех пор, пока не будет выполнено условие остановки, например, достижение заданного числа итераций или стабилизация значения целевой функции. В конечном итоге температура снижается до нуля.

**2.2 Схемы алгоритма имитации отжига**

Рассмотрим больцмановский отжиг.

Первой схемой метода отжига является схема отжига Больцмана. При отжиге Больцмана изменение температуры определяется формулой (8):

$$T(k) = \frac{T_0}{ln(1+k)}, k > 0. \quad (8)$$

Формула для вычисления новой начальной точки (9):

$$x(k) = x(k-1) + T(k) \cdot N(0,1), \quad (9)$$

где N (0,1) – стандартное нормальное распределение.

«Основным недостатком отжига по Больцману является очень медленное снижение температуры. Например, чтобы снизить начальную температуру в 40 раз, требуется $e^{40} \approx 2{,}35\ 10^{17}$ итераций, что уже вряд ли приемлемо при решении каких-либо задач» [7].

Рассмотрим отжиг Коши (быстрый отжиг).

«Цу и Хартли предложили алгоритм, который позволяет использовать следующую схему для изменения температуры без потери гарантии нахождения глобального минимума» [14] (10):

$$T(k) = \frac{T_0}{k}. \quad (10)$$

Формула для вычисления новой начальной точки (11):

$$x(k) = x(k-1) + T(k) \cdot \mathrm{C}(0,1), \quad (11)$$

где C (0,1) – распределение Коши.

Рассмотрим сверхбыстрый отжиг.

«Недостатки двух предыдущих методов привели к разработке сверхбыстрого метода отжига в 1989 году американским исследователем Л. Ингбером» [9]. Температура по каждой из координат может различаться.

Доказано, что закон изменения температуры (12):

$$T(k) = T_0 \cdot \ exp\left(-c_k \ \cdot \ k^{\frac{1}{n}}\right), \tag{12}$$

где $c_i > 0$ и вычисляется по формуле (13):

$$c_k = m(k) \cdot exp\left(-\frac{p(k)}{n}\right), \tag{13}$$

где m(k), p(k) – дополнительные параметры алгоритма, которые, для простоты работы алгоритма, не изменяются.

Формула для вычисления новой начальной точки (14):

$$x(k) = x(k-1) + (P_{max} - P_{min}) \cdot z(k), \tag{14}$$

где $P_{max}$, $P_{min}$ – минимальное и максимальное значение оптимизируемого параметра.

Величина z(k) вычисляется по формуле (15):

$$z(k) = sign\left(a - \frac{1}{2}\right) \cdot T(k) \cdot \left(\left(1 + \frac{1}{T(k)}\right)^{|2a-1|} - 1\right), \tag{15}$$

где a – случайная величина, равномерно распределенная на интервале [0,1);

$sign\left(a - \frac{1}{2}\right)$ определяется по формуле (16):

$$sign\left(a - \frac{1}{2}\right) = \begin{cases} 1, \text{при } a - \frac{1}{2} > 0 \\ -1, \text{при } a - \frac{1}{2} < 0. \\ 0, \text{при } a - \frac{1}{2} = 0 \end{cases} \tag{16}$$

Рассмотрим метод «тушения».

Метод «тушения» быстро находит близкое решение, а на практике часто и сам оптимум. «Зачастую он основан либо на нормальном распределении, либо на распределении для сверхбыстрого отжига, с более быстрым законом понижения температуры» [5].

Например, «можно проанализировать нормальное распределение по Больцману, но при этом снизить температуру по закону (17):

$$T_{k+1} = cT_k, \tag{17}$$

где $c$ выбирается между 0.7 и 0.99.

Этот метод работает очень быстро и для конкретных задач может дать очень хорошее решение, близкое к оптимальному, в режиме реального времени» [17].

Выводы по главе 2

Во второй главе ВКР была проведена всесторонняя оценка алгоритма имитации отжига, применяемого для решения комбинаторных задач. Рассмотрены основные теоретические аспекты алгоритма, его математическая модель и процесс работы, включая этапы инициализации, выбора нового решения, его оценки, принятия решения и охлаждения, его схемы. Анализ показал, что метод имитации отжига является сильным инструментом для решения комбинаторных задач благодаря своей гибкости и возможности адаптации к различным типам задач. Это делает его особенно полезным в ситуациях, где требуется найти приближенные решения в большом пространстве возможных решений.

# Глава 3 Программная реализация и тестирование алгоритма имитации отжига для решения комбинаторных задач

## 3.1 Программная реализация алгоритма имитации отжига

Алгоритм был реализован в PyCharm.

Реализация была разработана на версии Python 3.12.3. Были использованы следующие библиотеки:

- matplotlib для визуализации данных в виде графиков;
- numpy для работы с многомерными массивами;
- tkinter для создания графического пользовательского интерфейса;
- time для замера времени работы программы;
- math для использования математических формул.

Приведем таблицу технических данных персонального компьютера (далее – ПК) (Таблица 4).

Таблица 4 – Технические данные ПК

| Компоненты ПК | Характеристика |
|---|---|
| Процессор | Intel core i5 4460 3.2GHz |
| Оперативная память | 16 Gb DDR3 1600MHz |
| Видеокарта | AMD Radeon RX 580 8Gb |
| Операционная система | Windows 10 |

Для вычисления оптимальной стоимости в задаче о ранце в программе реализован метод имитации отжига, схемы метода имитации отжига.

Для тестирования алгоритма были созданы два файла:

- items.txt, который содержит вес и стоимость предметов (рисунок 2);

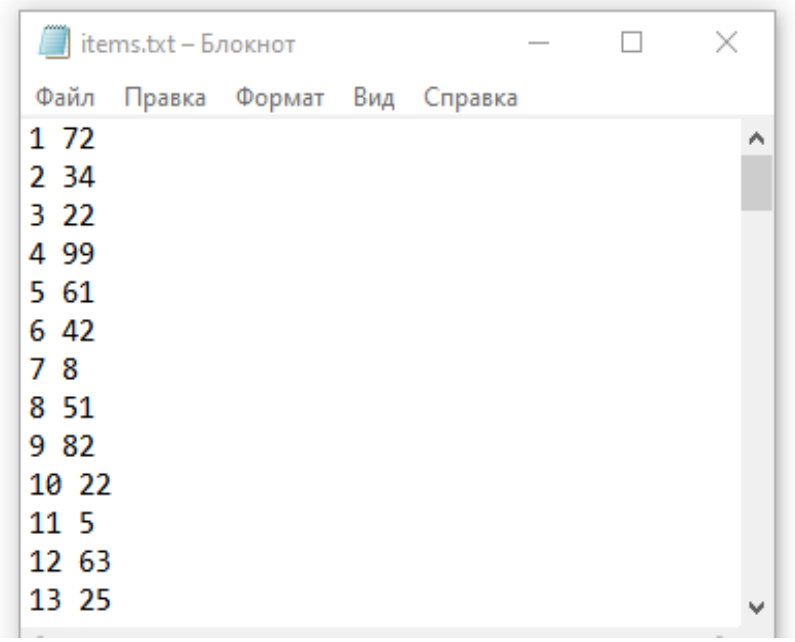


Рисунок 2 – Файл «items.txt»

– settings.txt, который содержит максимальный вес рюкзака, начальную температуру, конечную температуру и изменение температуры (рисунок 3).

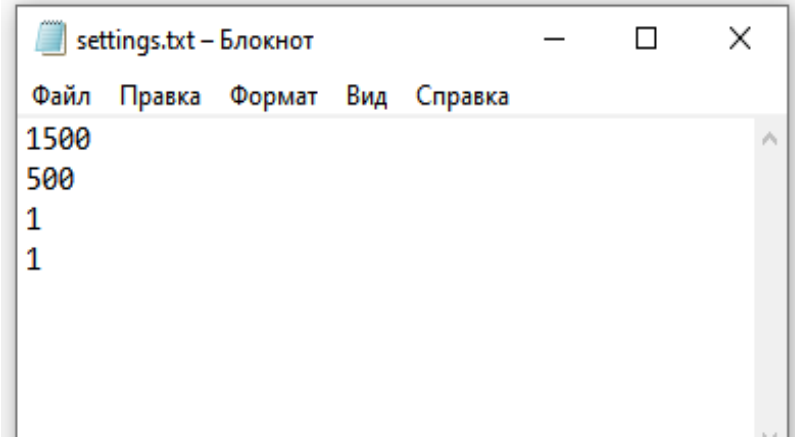


Рисунок 3 – Файл «settings.txt»

Параметры для каждого из способов:

– начальная температура – 2000;

– конечная температура – 1;

– предел количества итерации – 5000.

Для удобства выбора схемы отжига, был разработан интерфейс (рисунок 4).

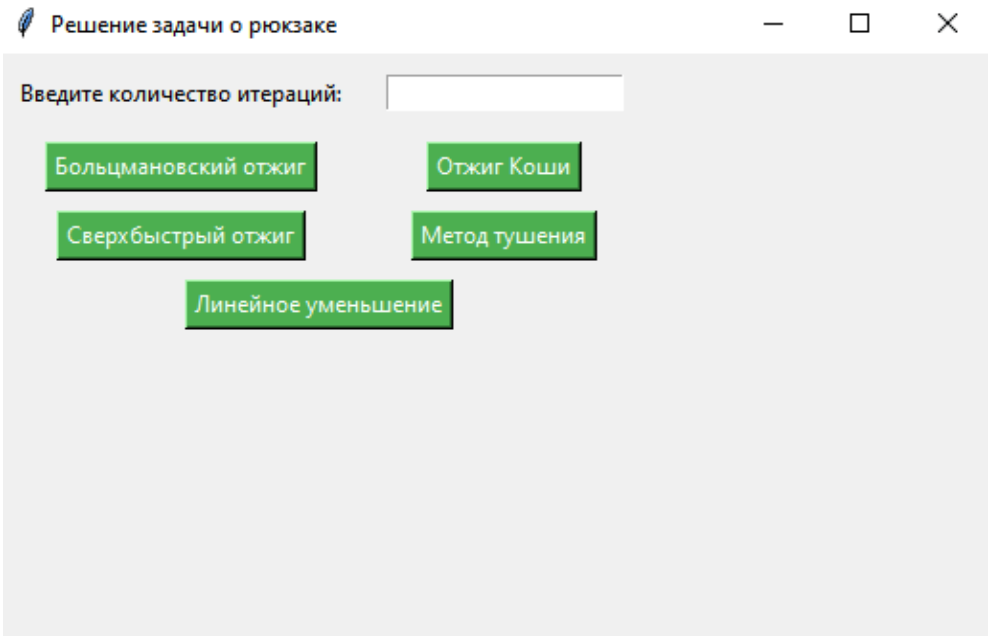


Рисунок 4 – Интерфейс выбора метода отжига

Весь описанный алгоритм программы представлен в Приложении А.

### 3.2 Тестирование алгоритма имитации отжига

Линейное уменьшение.

При изменении температуры = 1 программа завершилась с результатом стоимость: 2695, вес: 1496 за 6 секунд и потребовала 2000 итераций. График показан на рисунке 5.

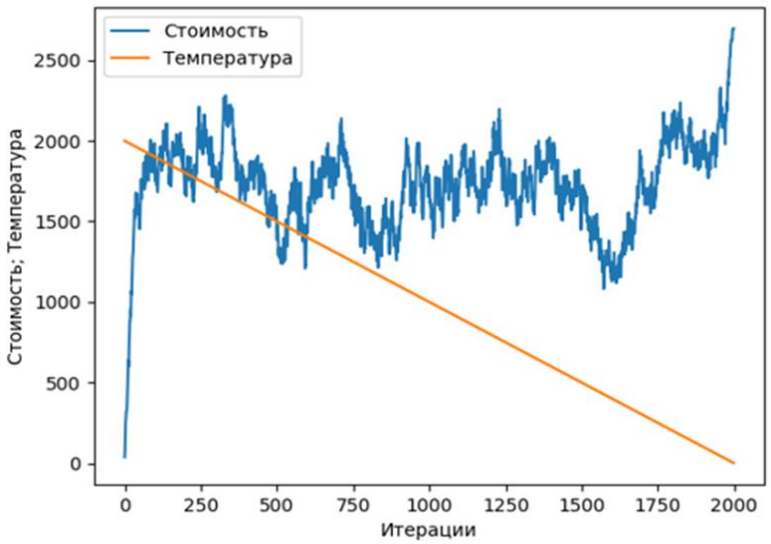


Рисунок 5 – Линейное уменьшение при изменении температуры, равному 1

Можно заметить, что значение на протяжении всего времени сильно скачет, что помогает избежать «ловушек» в локальных минимумах, но из-за этого не доводит каждое из вариантов до логического конца.

При изменении температуры = 0.5 программа завершилась с результатом стоимость: 2974, вес: 1498 за 12 секунд и потребовала 4000 итераций (рисунок 6).

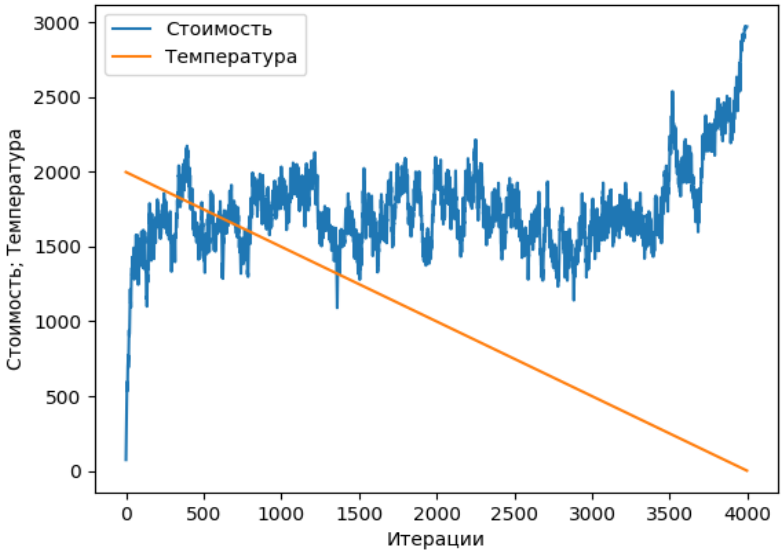


Рисунок 6 – Линейное уменьшение при изменении температуры, равному 0.5

Запустим программу 5 раз для поиска средних значений (изменение температуры = 0.5), визуализация показана на рисунке 7.

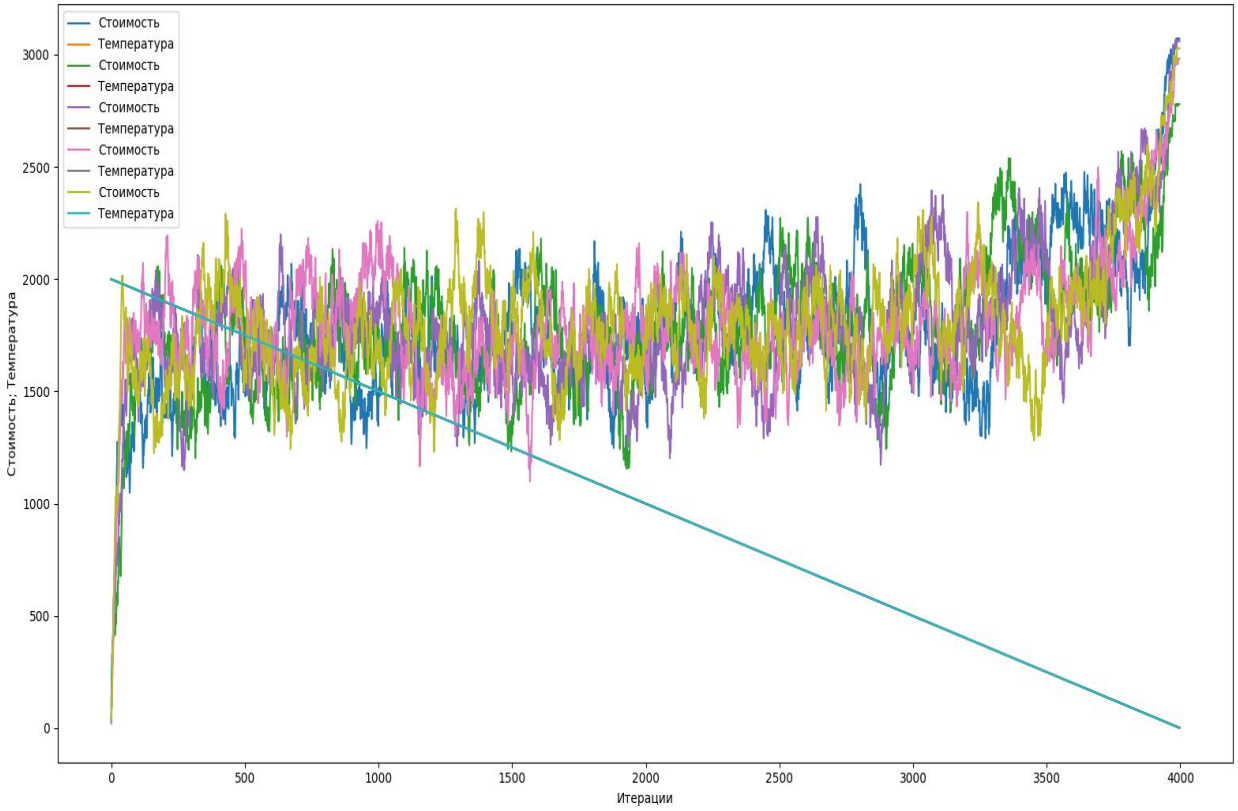


Рисунок 7 – Динамика значений при схеме линейного уменьшения

Результат программы показал, что средняя стоимость – 2984, среднее время – 12.2 секунд.

Метод «тушения».

При изменении температуры = 0.995 программа завершилась с результатом стоимость: 3039, вес: 1500 за 5 секунд и потребовала 1536 итераций (рисунок 8).

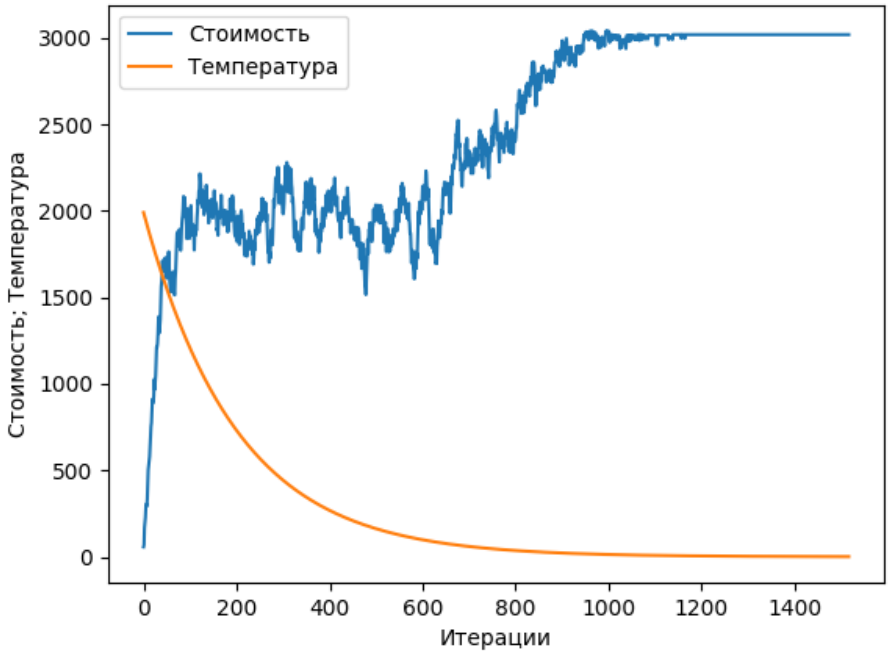


Рисунок 8 – Тушение при изменении температуры, равному 0.995

При изменении температуры = 0.9975 программа завершилась с результатом: стоимость: 3160, вес: 1498за 9,1 секунды и потребовала 3048 итераций (рисунок 9).

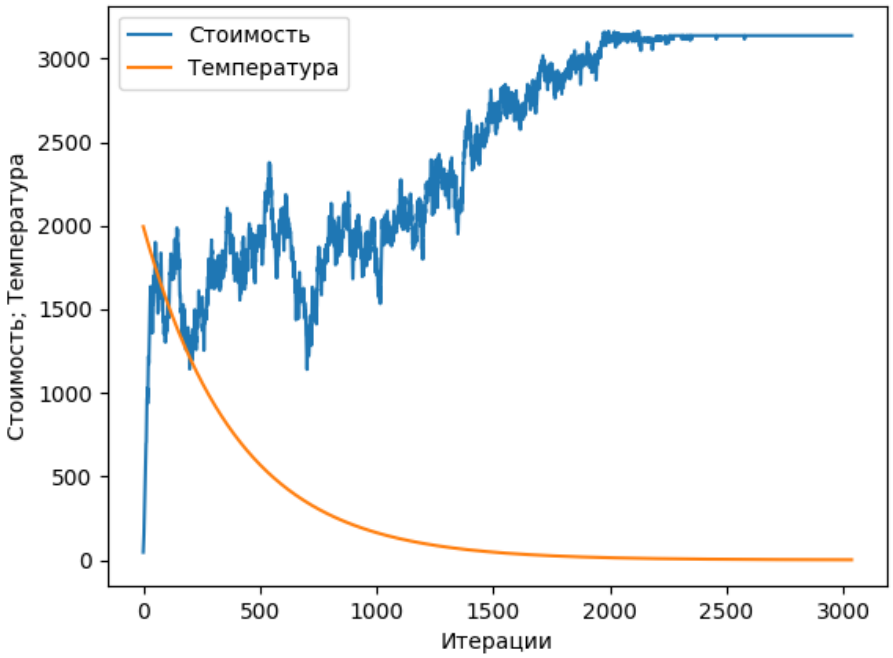


Рисунок 9 – Тушение при изменении температуры, равному 0.9975

Запустим программу 5 раз для поиска средних значений (изменение температуры = 0.9975).

Визуализацию результата можно увидеть на рисунке 10.

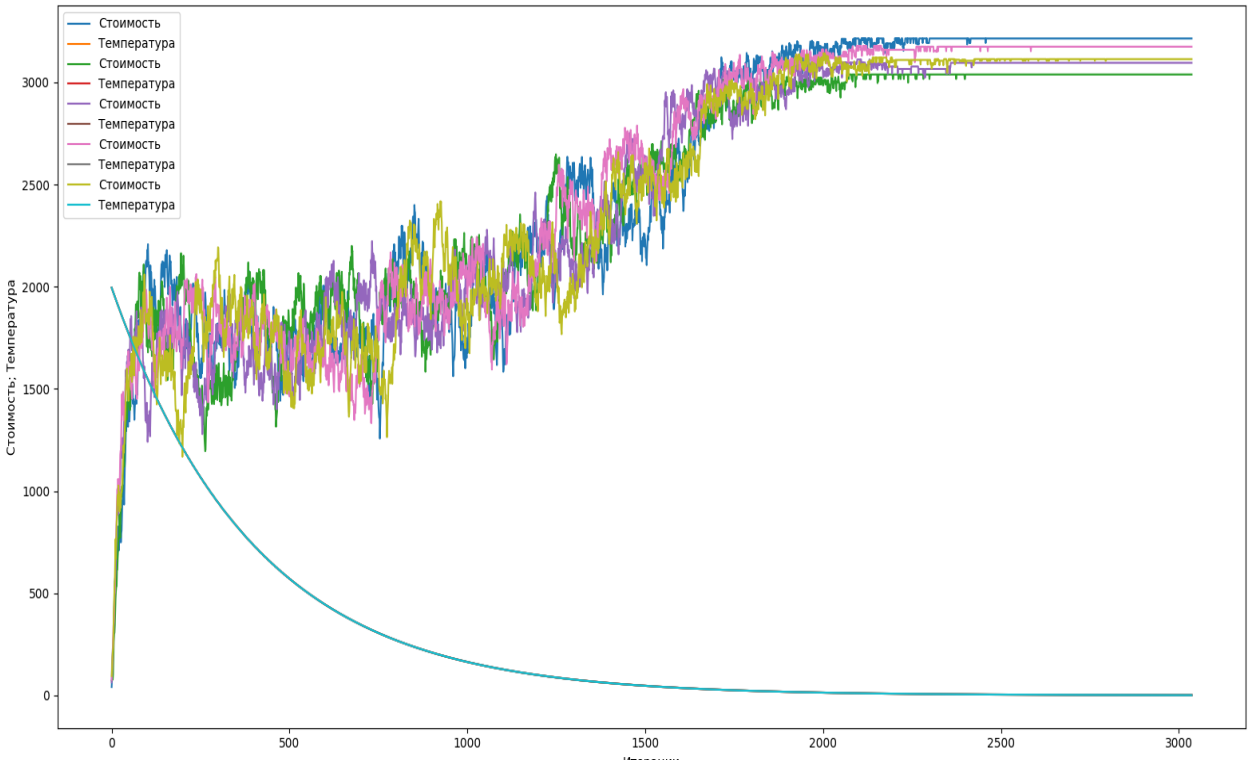


Рисунок 10 – Динамика значений при схеме тушения

Результат программы показал, что средняя стоимость – 3138, среднее время – 9.1 секунд.

Больцмановский отжиг.

При начальной температуре, равной 2000 и изменению температуры, равному 1 программа завершилась с результатом: стоимость: 2479, вес: 1478 за 13,52 секундыи потребовала 5000 итераций, однако температура не достигла финальной (рисунок 11).

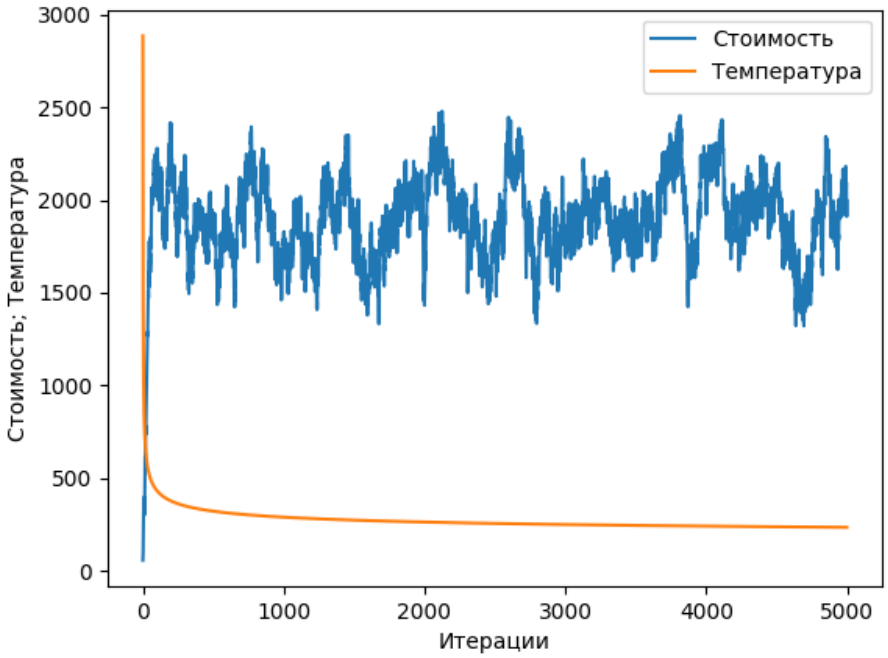


Рисунок 11 – Больцмановский отжиг при начальной температуре = 2000

При начальной температуре, равной 500 программа завершилась с результатом: стоимость: 2941, вес: 1491 за 13,9 секунды и потребовала 5000 итераций (рисунок 12).

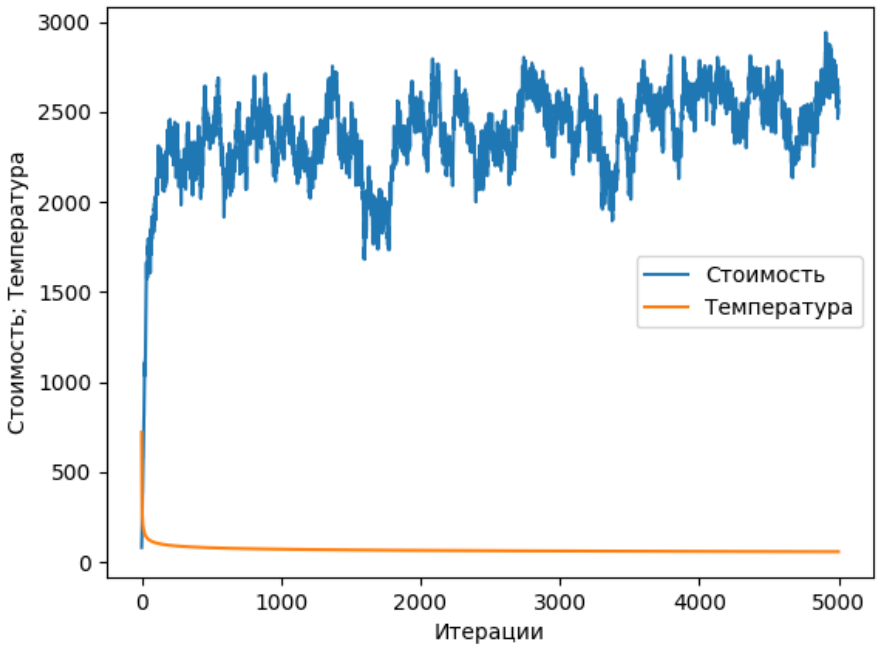


Рисунок 12 – Больцмановский отжиг при начальной температуре = 500

При начальной температуре, равной 100 программа завершилась с результатом: стоимость: 3226, вес: 1499 за 14,2 секунды и потребовала 5000 итераций (рисунок 13).

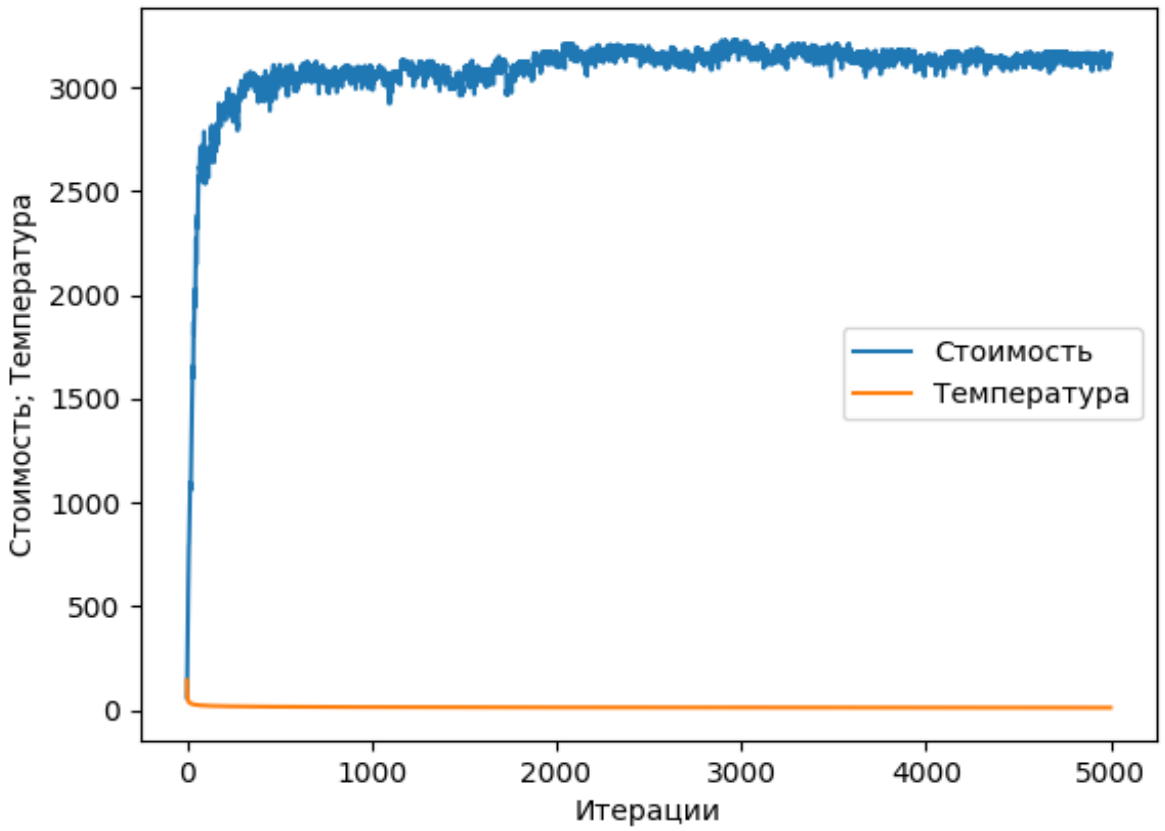


Рисунок 13 – Больцмановский отжиг при начальной температуре = 100

Запустим программу 5 раз для поиска средних значений (начальная температура = 100). На рисунке 14 представлена динамика значений.

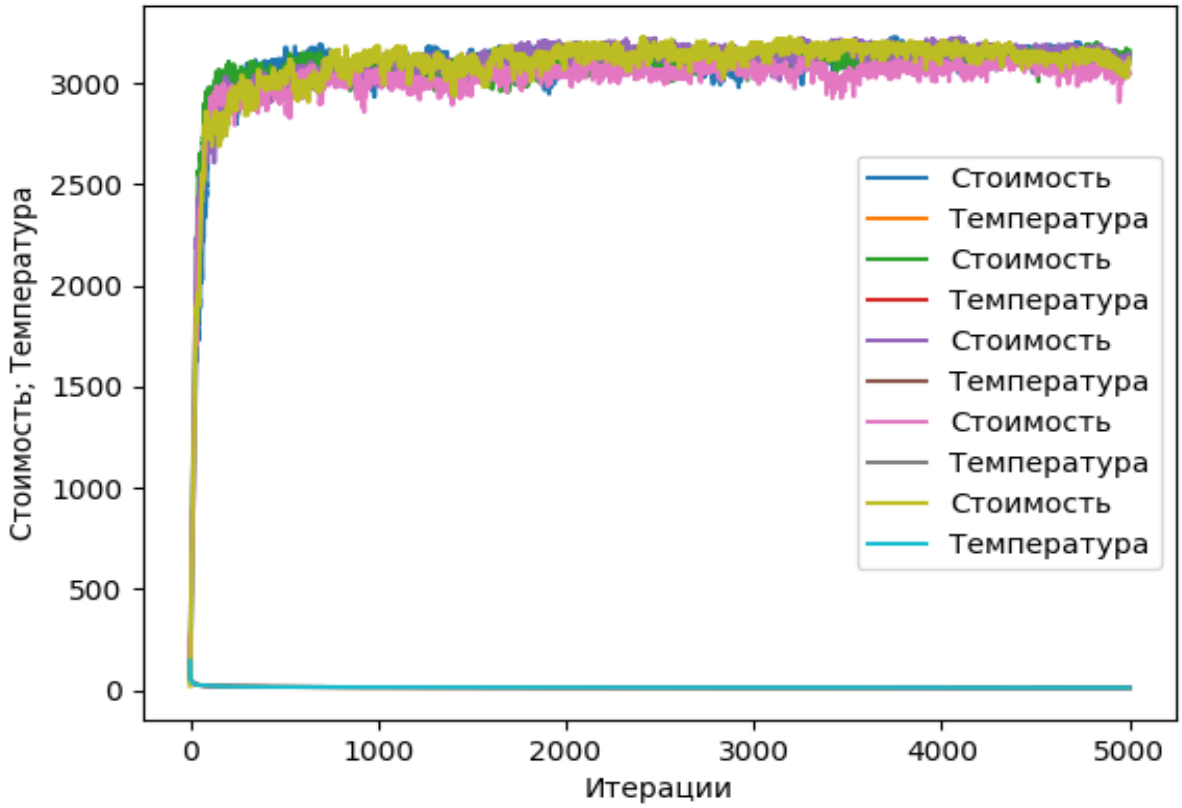


Рисунок 14 – Динамика значений при схеме больцмановского отжига

Результат программы показал, что средняя стоимость – 3201, среднее время – 14,9 секунды.

Рассмотрим отжиг Коши.

При начальной температуре, равной 2000 и изменении температуры, равной 1, программа завершилась с результатом: стоимость: 3187, вес: 1497 за 6,46 секунд, и потребовала 2000 итераций (рисунок 15).

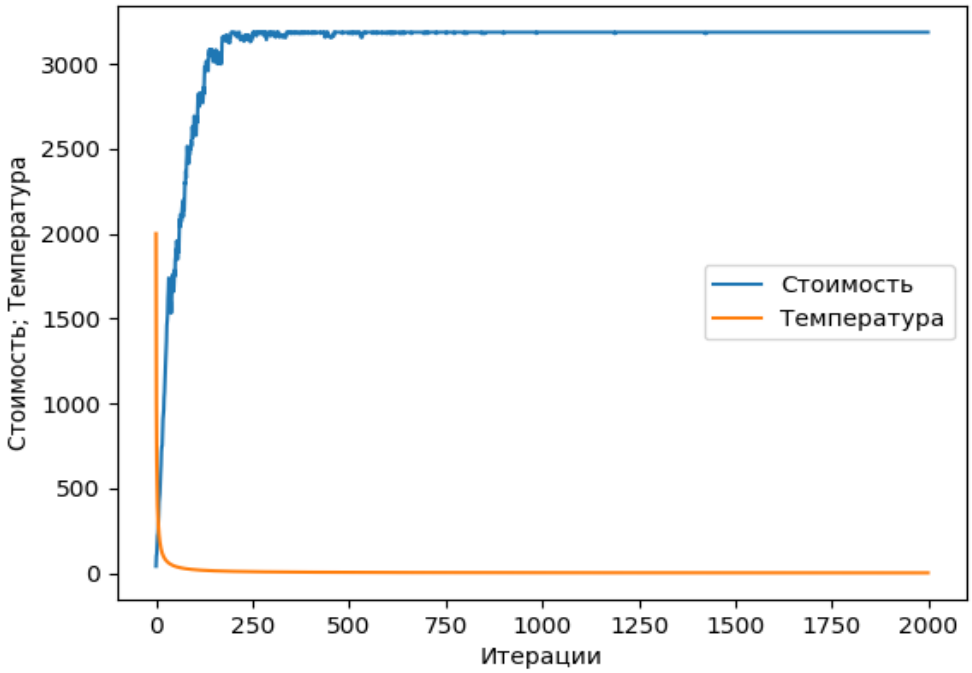


Рисунок 15 – Отжиг Коши при начальной температуре = 2000

Однако на графике видно, что программа практически не изменяла значение уже на 400-й итерации.

Запустим программу 5 раз для поиска средних значений (рисунок 16).

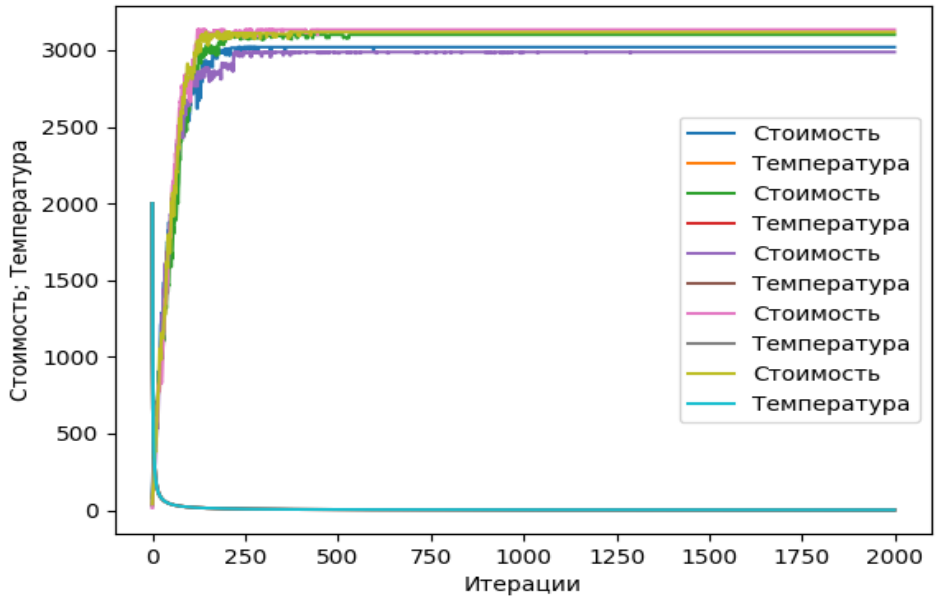


Рисунок 16 – Динамика значений при схеме отжига Коши

Результат программы показал, что средняя стоимость – 3073, среднее время – 6.44 секунды.

Сверхбыстрый отжиг.

Установим количество итераций: 2000. Программа завершилась с результатом Стоимость: 3149 Вес: 1499 за 6,09 секунд, и потребовала 2000 итераций (рисунок 17).

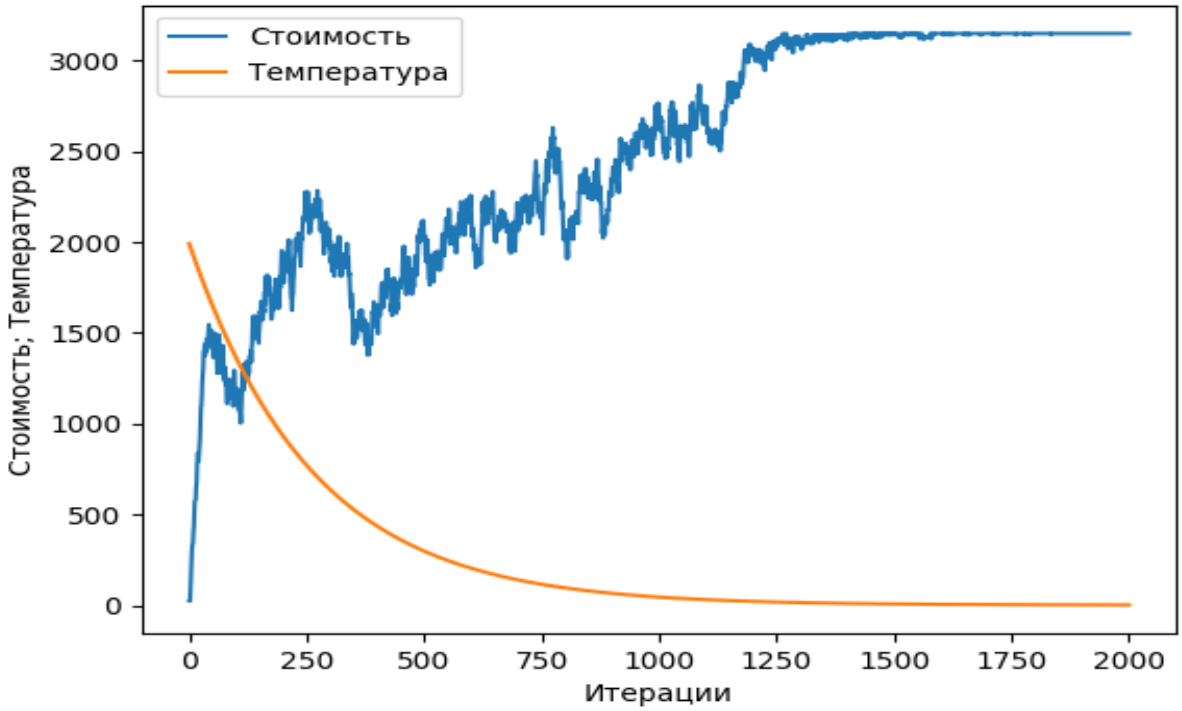


Рисунок 17 – Сверхбыстрый отжиг при количестве итераций: 2000

Запустим программу несколько раз для поиска средних значений (установлено 2000 итераций). График представлен на рисунке 18.

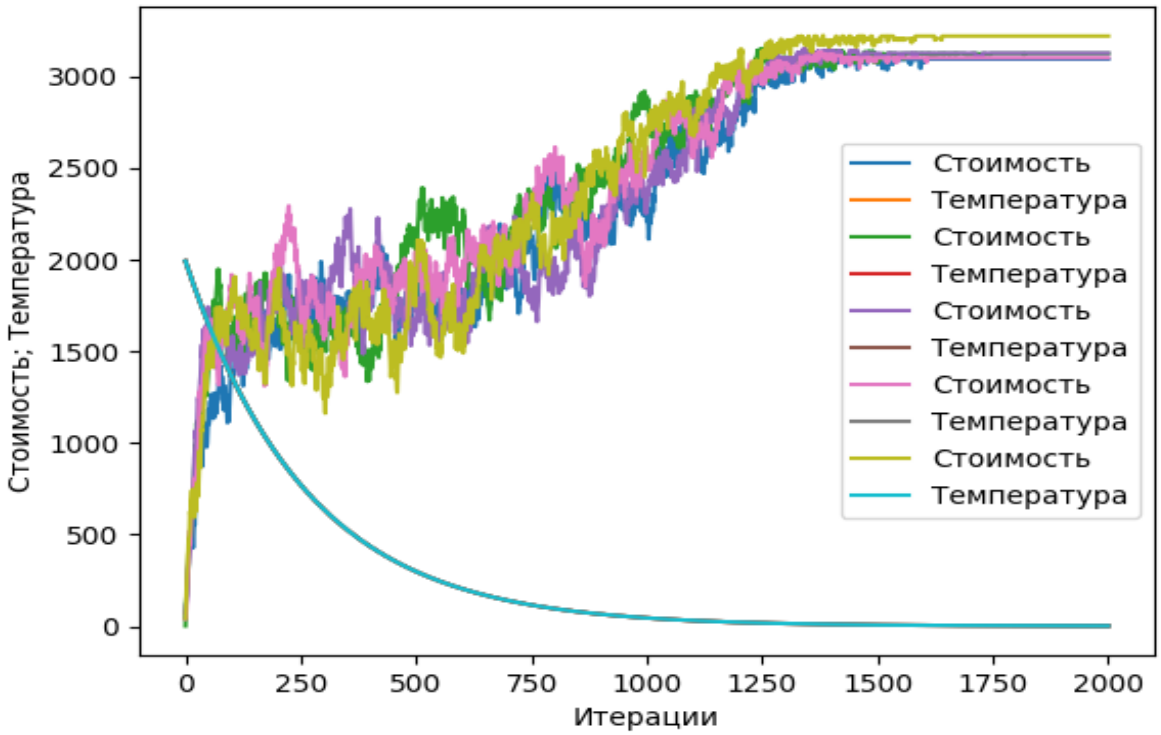


Рисунок 18 – Динамика значений при схеме сверхбыстрого отжига

Результат программы показал, что средняя стоимость – 3150, среднее время – 6.11 секунд.

Предоставим сравнительную таблицу различных схем имитации отжига (таблица 5).

Таблица 5 – Сравнение различных схем имитации отжига

| Схема | Среднее значение | Среднее время (с) |
|---|---|---|
| Линейное уменьшение | 2984 | 12.2 |
| Метод тушения | 3138 | 9.1 |
| Больцмановский отжиг | 3201 | 14.9 |
| Отжиг Коши | 3073 | 6.44 |
| Сверхбыстрый отжиг | 3150 | 6.11 |

Выводы по главе 3

Во третьей главе ВКР было проведено тестирование различных схем алгоритма имитации отжига для решения задачи о ранце.

Сравнительный анализ показывает, что методы, требующие меньше времени на выполнение (отжиг Коши, сверхбыстрый отжиг, метод тушения), демонстрируют хорошие результаты. Методы с более долгим временем выполнения не всегда показывают значительное улучшение результата по сравнению с быстрыми методами, что указывает на их возможную неэффективность.

Выбор схемы имитации отжига должен основываться на конкретных требованиях задачи:

- если ключевым фактором является время, предпочтение следует отдавать быстрым методам;
- если на первом месте стоит точность, стоит рассмотреть более ресурсоемкие методы.

Этот вывод подчеркивает важность тщательного анализа задачи и условий ее выполнения при выборе оптимального алгоритма.

# Заключение

В ходе выполнения ВКР на тему «Применение метода имитации отжига для решения комбинаторных задач» проведено исследование, объектом которого является метод имитации отжига, различные схемы и алгоритм метода имитации отжига.

Цель бакалаврской работы заключается в исследовании и реализации метода имитации отжига для решения комбинаторных задач.

В ходе данной работы поставлены и выполнены следующие задачи:

- выполнена постановка задачи исследования и проанализированы традиционные методы решения комбинаторных задач, а именно точные и приближенные и эвристические методы. Основное внимание было уделено задаче о ранце, которая является одной из классических задач комбинаторной оптимизации. В ходе её анализа было выявлено, что точные методы, такие как полный перебор и метод ветвей и границ, обладают высокой вычислительной сложностью и требуют значительных временных затрат. Эвристические методы предлагают более быстрые и приближенные решения, что делает их более применимыми для задач большой размерности.

  Для эффективного решения задачи о ранце необходимы методы, способные быстро находить хорошие приближенные решения. Метод имитации отжига, благодаря своей способности избегать локальных минимумов и гибкости настройки, выделяется как перспективный инструмент для решения таких задач;
- проанализирован алгоритм имитации отжига. Также были рассмотрены основные теоретические аспекты алгоритма, его математическая модель и процесс работы, включая этапы инициализации, выбора нового решения, его оценки, принятия решения и охлаждения, его схемы.;

- выполнена программная реализация и тестирование различных схем алгоритма имитации отжига для решения задачи о ранце. Сравнительный анализ показал, что методы, требующие меньше времени на выполнение (отжиг Коши, сверхбыстрый отжиг, метод тушения), демонстрируют хорошие результаты. Методы с более долгим временем выполнения не всегда показывают значительное улучшение результата по сравнению с быстрыми методами, что указывает на их возможную неэффективность.

Основные результаты исследования включают:

- подтверждение эффективности метода имитации отжига для решения задач с большим пространством возможных решений, где традиционные методы полного перебора неэффективны;
- реализация программного обеспечения, которое может быть использовано для решения различных комбинаторных задач с использованием метода имитации отжига;
- проведение сравнительного анализа разработанных алгоритмов, который показал высокую точность и эффективность каждой схемы метода имитации отжига.

Работа метода имитации отжига сильно зависит от корректной настройки начальной температуры, скорости охлаждения и критерия остановки под конкретную задачу. Правильная настройка важна для того, чтобы алгоритм мог исследовать большое пространство решений и избегать локальных минимумов на начальных этапах, когда температура высока.

Результаты бакалаврской работы подтверждают, что метод имитации отжига является мощным и универсальным инструментом для решения задачи о ранце, что открывает перспективы для его дальнейшего применения и совершенствования в различных прикладных задачах.

Цель работы была полностью достигнута. Задачи, поставленные для достижения цели работы, были выполнены в полном объеме.

## Список используемой литературы


1. Акулич И. Л. Математическое программирование в примерах и задачах. СПб: Лань, 2024. 348 с.
2. Громкович Ю., Мельников Б. Ф. Алгоритмизация труднорешаемых задач. Часть I. простые примеры и простые эвристики // Философские проблемы информационных технологий и киберпространства. 2013. №2. С. 17-30.
3. Габасов Р. П., Кириллова Ф. М. Основы динамического программирования. М.: Ленанд, 2021. 264 с.
4. Елкин Д. И. Метод отжига. М.: Мир, 2008, 46 с.
5. Зуенко А. А. Применение методов локального поиска при решении задач удовлетворения ограничений // Труды Кольского научного центра РАН. 2015. №3. С. 59-74.
6. Кормен Т, Лейзерсон Ч, Ривест Р, Штайн К. Алгоритмы. Построение и анализ. М.: Диалектика (Вильямс), 2020. 1328 с.
7. Лопатин А. С. Метод отжига // Стохастическая оптимизация в информатике. 2005. Вып.1. С.133-149.
8. Beisiegel B. Simulated annealing based algorithm for the bin packing problem with impurities. Proc., 2005, 113 c.
9. Dantas B. A., Cáceres E. N. An experimental evaluation of a parallel simulated annealing approach for the 0-1 multidimensional Knapsack problem, J. Parallel Distrib. Comput., vol. 120, pp. 211–221, Oct. 2018.
10. Goffe W. L., Ferrier G. D., Rogers J. Global optimization of statistical functions with simulated annealing, J. Econometrics, vol 60, pp. 65–99, Jan. 1994.
11. Hristakeva M., Shrestha D. Different approaches to solve the 0/1 Knapsack problem, in Proc. Midwest Instruct. Comput. Symp., Apr. 2005, pp. 1–15.
12. Kirkpatrick, S. Optimization by Simulated Annealing // Science, 1983. Vol. 220, No. 4598. P. 671-680.

13. Kellerer H., Pferschy U., Pisinger D. Knapsack problems. // Springer-Verlag, 2003. 548 c.

14. Kolesar P. J. A branch and bound algorithm for the Knapsack problem, Manage. Sci., vol. 13, no. 9, pp. 723–735, 1967.

15. Land A. H., Doig A. G. An automatic method of solving discrete programming problems // Econometrica. 1960 Vol. 28 P. 497-520.

16. Lawler E. L., Wood D. E. Branch-and-bound methods: A survey, Oper. Res., vol. 4, no. 4, pp. 669–719, 1966.

17. Liu A. Improved simulated annealing algorithm solving for 0/1 knapsack problem // Sixth International Conference on Intelligent Systems– 2006. – P. 1159–1164

18. Merkle R., Hellman M. Hiding information and signatures in trapdoor knapsacks, IEEE Trans. Inf. Theory, vol. 24, no. 5, pp. 525–530, Sep. 1978.

19. Martello S., Pisinger D., Toth P. Dynamic programming and strong bounds for the 0-1 Knapsack problem, Manage. Sci., vol. 45, no. 3, pp. 414–424, 1999.

20. Martello S. D. New trends in exact algorithms for the 0-1 Knapsack problem, Eur. J. Oper. Res., vol. 123, no. 2, pp. 325–332, 2000.

21. Pfeiffer J., Rothlauf F. Analysis of greedy heuristics and weightcoded eas for multidimensional Knapsack problems and multi-unit combinatorial auctions, in Proc. Conf. 9th Annu. Genetic Evol. Comput., Jul. 2007, p. 1529.

22. Ross G. T., Soland R. M. A branch and bound algorithm for the generalized assignment problem, Math. Program., vol. 8, no. 1, pp. 91–103, Dec. 1975.

23. Sonuc E., Sen B., Bayir S. Solving bin packing problem using simulated annealing, in Proc. 65th ISERD Int. Conf., Mecca, Saudi Arabia, Jan. 2017, p. 3.

24. Truong T. K., Li K., Xu Y. Chemical reaction optimization with greedy strategy for the 0-1 Knapsack problem, Appl. Soft Comput., vol. 13, no. 4, pp. 1774–1780, 2013.

Приложение А

**Листинг программы**

Листинг 1 – реализация программы

```
import numpy as np
import matplotlib.pyplot as plt
import time
import math
import tkinter as tk
from tkinter import messagebox
# Глобальные переменные для хранения температуры и значений на каждой итерации
temp_list = []
val_list = []
best_val_list = []
iteration_list = []
class Item:
    def __init__(self, item_weight, item_value):
        self.item_weight = item_weight
        self.item_value = item_value
    def getWeight(self):
        return self.item_weight
    def getValue(self):
        return self.item_value
class Knapsack:
    def __init__(self, capacity, items, initial_temp, final_temp, temp_update, method):
        self.current_solution = [False] * len(items)
        self.best_solution = [False] * len(items)
```

Продолжение Листинга 1

```
        self.neighbour_solutions = []
        self.current_temp = initial_temp
        self.final_temp = final_temp
        self.temp_update = temp_update
        self.capacity = capacity
        self.items = items
        self.iteration_count = 0
        self.method = method
        self.temp_factor = -math.log(float(initial_temp) / final_temp)
    def generate_neighbour_solutions(self):
        n = len(self.current_solution)
        self.neighbour_solutions = []
        for j in range(n):
            neighbour_solution = self.current_solution.copy()
            neighbour_solution[j] = not neighbour_solution[j]
            self.neighbour_solutions.append(neighbour_solution)
    def find_next_solution(self):
        n = len(self.neighbour_solutions)
        solution_weights = [0] * n
        solution_values = [0] * n
        for j in range(n):
            for i in range(n):
                if self.neighbour_solutions[j][i]:
                    solution_weights[j] += self.items[i].getWeight()
                    solution_values[j] += self.items[i].getValue()
        max_neighbour_value = self.get_value(self.current_solution)
        best_neighbour_index = -1
```

Продолжение Листинга 1

```
            for attempt in range(n):
                i = np.random.randint(n)
                current_solution_value = self.get_value(self.current_solution)
                probability = np.exp(-np.abs(current_solution_value -
solution_values[i]) / self.current_temp)
                if solution_weights[i] <= self.capacity and (solution_values[i] >
max_neighbour_value or probability > np.random.random()):
                    max_neighbour_value = solution_values[i]
                    best_neighbour_index = i
            if self.current_temp > self.final_temp:
                if self.method == "boltzmann":
                    self.current_temp = initial_temp / np.log(float(self.iteration_count
+ 2))
                elif self.method == "cauchy":
                    self.current_temp = initial_temp / float(self.iteration_count + 1)
                elif self.method == "superfast":
                    self.current_temp = initial_temp * math.exp(self.temp_factor *
(self.iteration_count + 1) / 2000)
                elif self.method == "quenching":
                    self.current_temp *= self.temp_update
                elif self.method == "linear":
                    self.current_temp -= self.temp_update
            else:
                self.current_temp = self.final_temp
            if best_neighbour_index != -1:
                self.current_solution =
self.neighbour_solutions[best_neighbour_index]
```

Продолжение Листинга 1

```
                neighbour_value =
self.get_value(self.neighbour_solutions[best_neighbour_index])
                best_solution_value = self.get_value(self.best_solution)
                if neighbour_value > best_solution_value:
                    self.best_solution =
self.neighbour_solutions[best_neighbour_index]
            temp_list.append(self.current_temp)
            iteration_list.append(self.iteration_count)
        def print_current_solution(self):
            self.iteration_count += 1
            v = self.get_value(self.current_solution)
            val_list.append(int(v))
        def print_best_solution(self):
            v = self.get_value(self.best_solution)
            print('\nЛучшее решение: ' + str(list(map(int, self.best_solution))) + '
Стоимость: ' + str(v) + ' Вес: ' + str(self.get_weight(self.best_solution)) + '\n')
            best_val_list.append(int(v))
        def get_weight(self, solution):
            return sum(item.getWeight() for item, included in zip(self.items,
solution) if included)
        def get_value(self, solution):
            return sum(item.getValue() for item, included in zip(self.items,
solution) if included)
        def get_temperature(self):
            return self.current_temp
    def run_knapsack(method):
        try:
```

Продолжение Листинга 1

```
            iterations = int(iteration_entry.get())
        except ValueError:
            messagebox.showerror("Ошибка", "Введите корректное число
итераций")
            return
        best_val_list.clear()
        iteration_indices = list(range(iterations))
        start_time = time.time()
        for i in range(iterations):
            val_list.clear()
            iteration_list.clear()
            temp_list.clear()
            knapsack = Knapsack(capacity, items, initial_temp, final_temp,
temp_update, method)
            while knapsack.get_temperature() > final_temp and len(iteration_list) <
5000:
                knapsack.generate_neighbour_solutions()
                knapsack.find_next_solution()
                knapsack.print_current_solution()
            plt.plot(iteration_list, val_list, label='Стоимость')
            plt.plot(iteration_list, temp_list, label='Температура')
            knapsack.print_best_solution()
        time_elapsed = time.time() - start_time
        print("--- Программа завершила работу за: %s секунд ---" %
(time_elapsed))
        plt.xlabel("Итерации")
        plt.ylabel("Стоимость; Температура")
```

Продолжение Листинга 1

```
    plt.legend()
    plt.show()
    plt.plot(iteration_indices, best_val_list)
    plt.show()
    print("Средняя стоимость:", np.mean(best_val_list))
    print("Среднее время на итерацию:", float(time_elapsed) / iterations)
# Загрузка данных
with open('items.txt', 'r') as f, open('settings.txt', 'r') as setting:
    weights = []
    values = []
    for line in f:
        temp = line.split()
        weights.append(int(temp[0]))
        values.append(int(temp[1]))
    settings = [float(line.split()[0]) for line in setting]
num_items = len(weights)
capacity = int(settings[0])
initial_temp = int(settings[1])
final_temp = int(settings[2])
temp_update = settings[3]
print('Вместимость:', capacity)
print('Количество предметов:', num_items)
print('Вес:', weights)
print('Стоимость:', values)
print()
print('Начальная Температура:', initial_temp)
print('Изменение температуры на каждом шаге', temp_update)
```

Продолжение Листинга 1

```
print()
items = [Item(weights[i], values[i]) for i in range(num_items)]
# Функции для запуска программы с выбранным методом отжига
def start_boltzmann():
    run_knapsack('boltzmann')
def start_cauchy():
    run_knapsack('cauchy')
def start_superfast():
    run_knapsack('superfast')
def start_quenching():
    run_knapsack('quenching')
def start_linear():
    run_knapsack('linear')
# Создание графического интерфейса
root = tk.Tk()
root.title("Решение задачи о рюкзаке")
root.geometry("520x420")
root.configure(bg="#f0f0f0")
# Поле для ввода количества итераций
tk.Label(root, text="Введите количество итераций:",
bg="#f0f0f0").grid(row=0, column=0, padx=10, pady=10)
iteration_entry = tk.Entry(root)
iteration_entry.grid(row=0, column=1, padx=10, pady=10)
# Кнопки для выбора метода отжига
tk.Button(root, text="Больцмановский отжиг", command=start_boltzmann,
bg="#4CAF50", fg="white").grid(row=1, column=0, padx=10, pady=5)
```

Продолжение Листинга 1

```
tk.Button(root, text="Отжиг Коши", command=start_cauchy,
bg="#4CAF50", fg="white").grid(row=1, column=1, padx=10, pady=5)
tk.Button(root, text="Сверхбыстрый отжиг", command=start_superfast,
bg="#4CAF50", fg="white").grid(row=2, column=0, padx=10, pady=5)
tk.Button(root, text="Метод тушения", command=start_quenching,
bg="#4CAF50", fg="white").grid(row=2, column=1, padx=10, pady=5)
tk.Button(root, text="Линейное уменьшение", command=start_linear,
bg="#4CAF50", fg="white").grid(row=3, column=0, columnspan=2, padx=10,
pady=5)
root.mainloop()
```