# РЕФЕРАТ

Отчет 33 с., 5 рис., 6 табл., 7 источн., 1 прил.

**Ключевые слова:** *задача о рюкзаке; задача о ранце; knapsack problem; комбинаторная оптимизация; метод ветвей и границ;*

В отчете представлены результаты курсового проекта на тему “Методы решения задачи о ранце со многими ограничениями“, выполненного на основании учебного плана подготовки бакалавров по направлению 09.03.04 "Программная инженерия" и утвержденной академическим руководителем темы курсового проекта.

**Объект исследования** – задача о ранце со многими ограничениями .

**Предмет исследования** – алгоритмы решения задачи о ранце со многими ограничениями.

**Цель исследования** – разработка эффективных методов решения задачи о булевом ранце со многими ограничениями, исследование эффективности разработанных алгоритмов.

**Задачи исследования:**

* Составление обзора существующих подходов к решению задачи о ранце со многими ограничениями.
* Выбор алгоритма для оптимизации и программная реализация выбранного алгоритма на языке C++.
* Оптимизация выбранного решения или разработка новых методов решения задачи.
* Параллельная реализация полученных алгоритмов на языке C++.
* Проведение сравнительного тестирования полученных методов.
* Анализ результатов тестирования.

**Методы исследования:**

* Изучение публикаций и статей.
* Реализация существующих и разработанных решений и оценка их эффективности

**Научная новизна:**

* Предложена модификация алгоритма, основанного на методе ветвей и границ.
* Проведен сравнительный анализ эффективности реализованных алгоритмов.

**Достоверность научных результатов** подтверждена результатами экспериментальных исследований на базе разработанной программной реализации.

**Практическая значимость.** В отличие от одномерной задачи о ранце, многомерный вариант позволяет описывать широкие классы прикладных задач, в частности задачи, связанные с планированием и управлением транспортными и производственными процессами.

**Результаты работы**

* Изучены существующие методы решения многомерной задачи о ранце.
* Предложена модификация алгоритма, основанного на методе ветвей и границ.
* Разработана программная параллельная реализация предложенных алгоритмов.
* Проведены вычислительные эксперименты для сравнения эффективности реализованных алгоритмов.

ОГЛАВЛЕНИЕ

[РЕФЕРАТ 2](#_Toc83063971)

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 5](#_Toc83063972)

[ВВЕДЕНИЕ 6](#_Toc83063973)

[ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ 7](#_Toc83063974)

[1.1 Постановка задачи 7](#_Toc83063975)

[1.2 Методы решения 7](#_Toc83063976)

[1.2.1 Точные методы решения 7](#_Toc83063977)

[1.2.2 Теоретический анализ 8](#_Toc83063978)

[1.2.3 Эвристика 9](#_Toc83063979)

[ГЛАВА 2. АЛГОРИТМ ШИХА И ЕГО МОДИФИКАЦИЯ 10](#_Toc83063980)

[2.1 Алгоритм Шиха 10](#_Toc83063981)

[2.1.1 Вычисление верхней границы 10](#_Toc83063982)

[2.1.2 Выбор узла и переменной для ветвления 11](#_Toc83063983)

[2.1.3 Переназначение индексов переменных 11](#_Toc83063984)

[2.1.4 Схема алгоритма метода ветвей и границ 12](#_Toc83063985)

[2.1.5 Пример 14](#_Toc83063986)

[2.2 Модификация метода ветвей и границ 19](#_Toc83063987)

[2.2.1 Метод ветвей и границ: запоминание лучшего решения 19](#_Toc83063988)

[2.3 Распараллеливание метода ветвей и границ 20](#_Toc83063989)

[2.3.1 Распараллеливание алгоритма Шиха 20](#_Toc83063990)

[2.3.2 Распараллеливание МВГ с запоминанием лучшего решения 21](#_Toc83063991)

[ГЛАВА 3. ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ 22](#_Toc83063992)

[3.1 Реализация алгоритмов 22](#_Toc83063993)

[3.2 Планирование вычислительного эксперимента 22](#_Toc83063994)

[3.3 Тестирование алгоритмов и анализ результатов 23](#_Toc83063995)

[ЗАКЛЮЧЕНИЕ 28](#_Toc83063996)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc83063997)

[ПРИЛОЖЕНИЕ 30](#_Toc83063998)

# 

# ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Классическая задача о ранце (КЗР) – NP-полная задача комбинаторной оптимизации. Свое название получила от конечной цели: уложить как можно большее число ценных вещей в рюкзак в рюкзак, при условии, что вместимость рюкзака ограничена.

Многомерная задача о ранце (МЗР) – обобщение классической задачи о ранце. Вместо веса дано несколько разных ресурсов. На каждый ресурс есть собственное ограничение.

Класс NP – класс задач, верифицируемых(проверяемых) за полиномиальное время.

Метод ветвей и границ (МВГ) – общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации. Является развитием метода полного перебора с основным отличием – отсевом подмножеств решений, заведомо не содержащих оптимальных решений.

Эвристический алгоритм – алгоритм решения задачи, включающий практический метод, не являющийся гарантированно точным или оптимальным, но достаточный для решения поставленной задачи.

ВВЕДЕНИЕ

Классическая задача о ранце является одной из самых известных задач дискретной оптимизации. Описать задачу о ранце можно следующим образом: нужно сложить в рюкзак набор предметов максимальной суммарной ценности при условии, что каждый предмет обладает собственным весом, а вес, который мы можем переносить в рюкзаке ограничен.

Сформулирована задача о рюкзаке была в прошлом веке американским математиком Джорджем Данцигом [3]. Задача быстро привлекла интерес научного сообщества, так как имеет массу приложений. Основная область применения – задачи планирования и управления различными системами (транспортными, экономическими, производственными).

На практике же КЗР встречается не так часто. Причина заключается в том, что в каждой системе обычно находятся несколько людей, каждый из которых оценивает принимаемые решения по-своему. Более того, каждое решение чаще всего оценивается не по одному критерию. Для описания таких моделей подходит задача о ранце со многими ограничениями (МЗР).

Основное отличие от классической задачи о ранце состоит в наличии нескольких параметров у каждого предмета (не только веса), для каждого параметра существует свое ограничение. Для двумерного случая можно придумать следующую аналогию: нужно собрать в рюкзак набор предметов максимальной стоимости, но помимо ограничения на вес, который мы можем поднять, есть так же ограничение на суммарный объем (в рюкзак не поместится много предметов большого объема).

Задача о рюкзаке является NP-полной, т. е. не имеет полиномиального решения. Алгоритм полного перебора имеет сложность O(2N), что очевидно, так как каждый предмет имеет только два состояния: используется или не используется, т. е. решение можно представить как двоичное слово длины N, а всего таких слов 2N.

Цель данной работы – исследование существующих решений многомерной задачи о рюкзаке с целью модификации одного из предложенных алгоритмов или разработки своего решения.

# ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

## 1.1 Постановка задачи

Сформулируем математическую постановку задачи:

где – число предметов, – число ограничений, – бит, который отвечает за включение предмета под номером *,* – стоимость -го предмета, – вес -го предмета в -ом ограничении,– -ое ограничение. Задача – найти такой поднабор предметов, что его суммарная ценность будет максимальной при условии, что ни одно ограничение не будет нарушено. Без потери общности, будем считать, что

Несмотря на то, что многомерная задача о ранце это просто обобщение классической задачи о ранце, ситуация сильно изменяется, когда появляются дополнительные ограничения.

## 1.2 Методы решения

Обзор существующих подходов к решению многомерной задачи о ранце был представлен в [1].

### 1.2.1 Точные методы решения

Разработка точных алгоритмов началась в шестидесятых годах прошлого века. Первые предложенные решения были основаны на динамическом программировании. Предлагались различные гибридные алгоритмы, которые использовали внутри себя метод ветвей и границ, эвристику, устранения недопустимых решений во время выполнения динамики. Ни один из предложенных методов не оказался эффективным. Основная проблема заключалась в том, что динамическое программирование требует очень много дополнительной памяти, поэтому получить оптимальное решение при большом числе предметов или ограничений не представлялось возможным

Другой подход к решению заключался в получении преимущества от специальной структуры МЗР. Ших представил первый алгоритм [2], который использовал линейное программирование. Данный алгоритм основано на методе ветвей и границ. Вычисление верхней границы и последующее ветвление узла основывалось на решении одномерных рюкзаков (каждое ограничение было выделено в отдельный рюкзак) с помощью линейного программирования. Из полученных решений выделялось наименьшее и считалось верхней границей данного узла. Переменная для ветвления определялась следующим образом: рассматривался одномерный ранец, решение которого впоследствии становилось верхней границей узла. Методом Данцига [2] решалась задача линейного программирования, ветвление осуществлялось по дробной переменной. Если дробной переменной нет, и данное решение было целочисленным ветвление прекращалось.

Алгоритм Шиха показал впечатляющие результаты. Время его работы было на порядок меньше предшественников. Впоследствии, алгоритм Шиха был выбран для реализации и его модификации. Главным недостатком данного метода является большая потребность памяти для хранения узлов при большом количестве предметов.

Так же стоит уделить внимание алгоритмам с использованием метода ветвей и границ, основанных на Лагранжевой/суррогатной релаксации. Данные релаксации позволяют получить преимущество, за счет более быстрого вычисления границ в узлах.

### 1.2.2 Теоретический анализ

Теоретический анализ позволяет нам понять огромную разницу в сложности при переходе от КЗР к МЗР. Задача о ранце не является NP-полной в сильном смысле. Это значит, что у нее может существовать псевдополиномиальный алгоритм. Известно, что для обеих задач существует решение динамическим программированием, работающее за .

Так же, известно, что существуют схема полиномиальной аппроксимации. Схемы полиномиальной аппроксимации связаны с анализом наихудшего случая, что обеспечивает гарантию на максимальную величину, на которую эвристический алгоритм отклонится от оптимальности для любого случая. Коэффициент наихудшего случая производительности это наибольшее , для которого, где *I* – объект МЗР, – значение полученное при решении данного рюкзака эвристическим алгоритмом *H*. Полиномиальная аппроксимация позволяет показать полиномиальное время для эвристики со значением *r,* сколь угодно близким к 1.

Много работ было посвящено вероятностному анализу МЗР. Большинство работ исследовало зависимость между численным решением рюкзака и емкостью ограничений или количеством предметов *n*.

### 1.2.3 Эвристика

Решения, основанные на методе ветвей и границ сделали возможным решение экземпляров МЗР среднего размера. Тем не менее эвристические алгоритмы остаются конкурентоспособными, особенно когда число ограничений велико.

Как уже упоминалось, во многих работах были разработаны эвристические подходы к МЗР, среди которых можно найти все эвристические подходы, представленные в комбинаторной оптимизации. Классифицируем эти подходы на 3 группы: жадные алгоритмы, методы математического программирования и метаэвристические методы.

Жадные алгоритмы быстры и, как правило, просты в реализации. Ранние подходы успешно обобщили решение КЗР, к коэффициентам, определяемыми как отношение стоимости предмета к весу. В многомерном случае предметы выбирались исходя из коэффициентов , где это заданные неотрицательные веса.

# ГЛАВА 2. АЛГОРИТМ ШИХА И ЕГО МОДИФИКАЦИЯ

## Алгоритм Шиха

Рассмотрим упомянутый ранее алгоритм Шиха [2] более подробно. Алгоритм основан на методе ветвей и границ. Данный метод заключается в принятии двух ключевых правил: вычисления верхней границы в узле, а также выбор специального критерия для определения переменной, по которой будет происходить ветвление.

### 2.1.1 Вычисление верхней границы

Для вычисления верхней границы рассмотрим *m* одномерных задач о ранце. Для каждой задачи найдем ее оптимальное дробное решение. Данциг показал в [3], что оптимальное дробное решение для одномерной задачи о рюкзаке,

где индексы отсортированы по удельной ценности задается системой:

где это последнее целое (), для которого

Если такого не существует, то все , и если , тогда итоговое целочисленное решение является оптимальным. Данное решение подразумевает, что мы добавляем в рюкзак как можно больше предметов, начиная с предметов с наивысшей удельной ценностью, до тех пор, пока суммарный вес предметов в точности не достигнет ограничения.

Воспользуемся методом Данцига для вычисления границы в узле МЗР. Рассмотрим классических задач о ранце, каждая из которые соответствует какому-то ограничению в МЗР. Вычислим оптимальное дробное решение для каждой задачи методом Данцига. Минимальное полученное значение и будет верхней границей узла. Действительно, оптимальное решение МЗР в данном узле должно удовлетворять каждому ограничению, следовательно, ни одно из значений, полученных при решении одномерных задач, не может быть превышено*.* Следовательно, наименьшее из этих значений соответствует верхней границе в данном узле.

### 2.1.2 Выбор узла и переменной для ветвления

Узлом, выбранным для последующего ветвления будет такой узел, что его верхняя граница будет наибольшей среди всех доступных на данный момент узлов при условии, что решении связанное с этой верхней границей не является допустимым (т. е. решение не является целочисленным или нарушаются другие ограничения). Переменной для ветвления будет ненулевая переменная с наименьшей удельной стоимостью в ограничении, связанным с верхней границей узла.

Вычислительное преимущество выбора переменной с наименьшей, а не наибольшим значением удельной стоимостью в качестве переменной ветвления было показано Гринбергом и Хегерихом для классической задачи о ранце [4].

### 2.1.3 Переназначение индексов переменных

Для применения алгоритма необходимо переназначить каждую переменную с помощью вспомогательных индексов вместо одного индекса .

Для каждого ограничения *,* вычислим значение коэффициента

удельной стоимости для каждой из переменных и отсортируем их в порядке уменьшения. В случае, если переменные имеют одинаковое значение коэффициента для некоторого ограничения, расположим их в порядке увеличения номера переменной.

Рассмотрим рангов этих коэффициентов для каждой переменной как вспомогательных индексов этой переменной. Несмотря на то, что переменная может получить одинаковые ранги в различных ограничениях, например, никакие две переменные не будут иметь одинаковые последовательности индексов. Более того, для идентификации переменной может быть использован любой из ее вспомогательных индексов, а не вся последовательность. Действительно, для того чтобы идентифицировать переменную нам достаточно узнать ее номер в ранжировании в любом из ограничений.

Теперь эти вспомогательных индексов будут использоваться для замены индекса в переменных , использованных в первоначальной постановке задачи. Переменная будет представлена как , где . указывается на порядок величины удельной стоимости предмета в ограничении . Таким же образом переназначим индексы значений стоимости и веса предметов. Тогда, стоимость предмета будет представлена как , а вес предмета в -ом ограничении как .

Тогда переформулируем поставленную задачу с использованием вспомогательных индексов:

Для идентификации переменной и ее коэффициентов будем использовать только индекс , а всю последовательность вспомогательных индексов будем использовать для выбора предмета, который будет добавлен в рюкзак в данном узле.

### 2.1.4 Схема алгоритма метода ветвей и границ

Опишем схему метода ветвей и границ, учитывая предположение о том, что каждый предмет может быть использован в единственном экземпляре без нарушения ограничений, а также о том, что оптимальное решение не является тривиальным, то есть каждый предмет не может быть использован.

Для описания алгоритма будем использовать следующие обозначения, введенные Колесаром [5]:

предметы, которые заведомо включены в ранец в -ом узле (т. е. переменные с фиксированным значением 1 в узле ).

предметы, которые заведомо исключены из ранца в -ом узле (т. е. переменные с фиксированным значением 0 в узле ).

объединение множеств и (т. е. переменные с фиксированным значением в узле ).

дополнение множества . Свободные предметы в -ом узле (т. е. переменные, которые могут принимать любое значение в узле ).

верхняя граница в -ом узле.

суммарная стоимость оптимального дробного решения, которое удовлетворяет только -ому ограничению.

множество свободных предметов, которые включены в рюкзак в оптимальном дробном решении, которое удовлетворяет -ому ограничению. Все переменные в кроме последней добавленной имеют значение 1, а последняя добавленная переменная лежит в промежутке .

***Шаг 1: Создание начального узла***

Рассмотрим каждое ограничение раздельно, начиная с первого. Будем последовательно добавлять самые ценные предметы в данном ограничении (обращаясь к вспомогательным индексам) до тех пор, пока ограничение не будет достигнуто, т. е.

для -го ограничения. Повторим эту процедуру для всех ограничений и получим различных оптимальных дробных решений. Найдем суммарную стоимость загруженных предметов для каждого ограничения и найдем минимальное значение . Тогда для начального узла положим и .

***Шаг 2: Выбор узла и переменной ветвления***

Выберем узел с наибольшей верхней границей . Проверим решение, связанное с этой границей на допустимость. Решение допустимо если все использованные в нем переменные целочисленные и никакие ограничения не нарушены. Это решение представлено переменными содержащимися в и Если решение допустимо, то оно является оптимальным. Если нет, то выберем в такую переменную для ветвления, что ее номер в индексе, который соответствует ограничению, суммарная стоимость оптимального дробного решения которого совпадает с верхней границей, будет максимальным. Это значит, что будет выбран свободный предмет, удельная ценность которого в данном ограничении минимальна среди всех используемых предметов (т. е. последний добавленный предмет).

***Шаг 3: Ветвление и вычисление верхней границы***

Опишем процесс ветвления узла с переменной ветвления . После ветвления образуются два новых узла с номерами и . Тогда для первого потомка , , а для второго потомка , .

Вычислим верхние границы у потомков. Проверим допустимость частичного решения, представленного множеством , в узле, т. е. проверим не нарушаются ли ограничения:

Если какое-то ограничение было нарушено, не будем рассматривать полученный узел (т.е. просто удалим его из дерева), так как частичное решение с фиксированными переменным в данном узле недопустимо. Заметим, что первый потомок всегда будет давать допустимое частичное решение, представленное множеством , при условии, что решение предка было допустимо. Действительно, множество переменных , фиксированных в единице, не изменилось, а значит, никакое ограничение не может быть нарушено.

Для каждого ограничения загрузим все фиксированные в единице переменные() в используемый набор, а затем будем добавлять переменные из множества в порядке увеличения индекса, связанного с данным ограничением, до тех пор, пока ограничение не будет достигнуто, т.е. получим оптимальное дробное решение для каждого ограничение в узле. Верхней границей будет минимальная суммарная стоимость среди этих решений:

,

.

### 2.1.5 Пример

Рассмотрим числовой пример:

Вычислим удельную стоимость предметов в каждом ограничении:

Таблица 1. Удельная стоимость предметов

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

Используя полученные значения перейдем к вспомогательным индексам:

Таблица 2. Вспомогательные индексы переменных

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

*Стартовый узел:*

Найдем верхнюю границу стартового узла . Для этого вычислим суммарную стоимость оптимального дробного решения для каждого ограничения и .

Верхняя граница стартового узла равна , переменная ветвления .

*Узел 2 :*

*Узел 3 :*

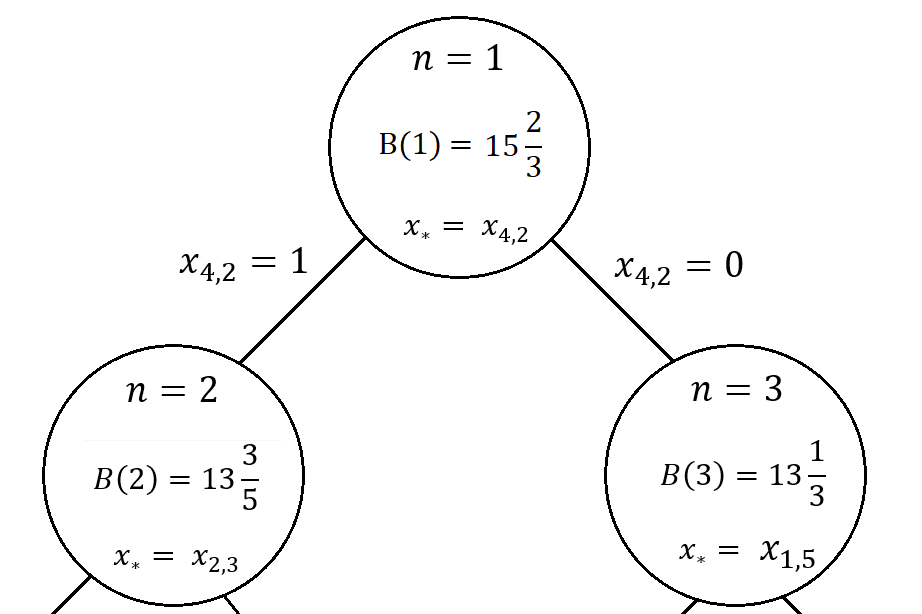
**

Рисунок 1. Дерево узлов (3 узла)

Граница второго узла больше границы третьего узла > , значит ветвление будет происходить по второму узлу.

*Узел 4 :*

*Узел 5 :*

Так как , будем ветвиться по переменной , так как она относится к ограничению с меньшим номером.

Среди трех доступных узлов наибольшая граница у узла 4.

*Узел 6 :*

Набор предметов, заведомо включенных в ранец в данном узле, нарушает оба ограничения, поэтому данный узел недопустим.

*Узел 7 :*

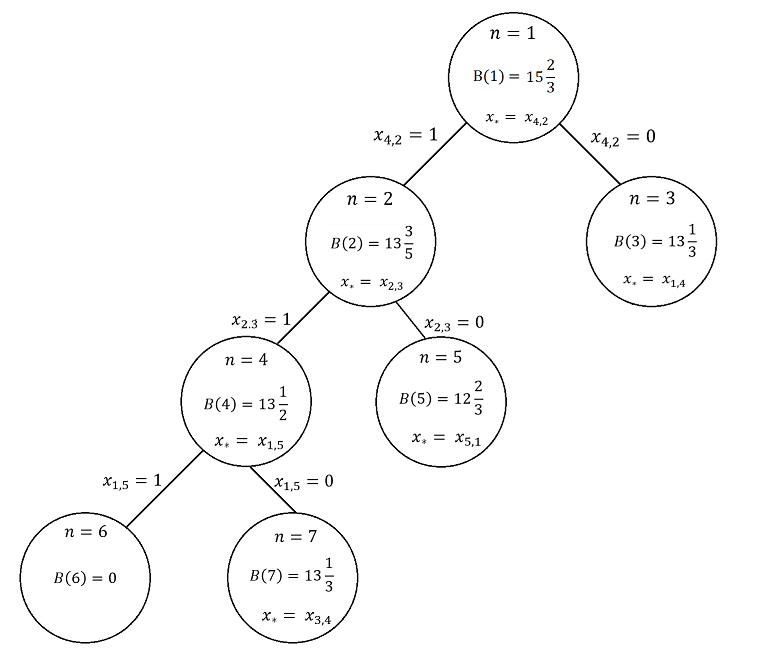
**

Рисунок 2. Дерево узлов (7 узлов)

Следующий узел для ветвления – узел 3.

*Узел 8 :*

*Узел 9 :*

Решение целочисленное, . Узел ветвления – 7.

*Узел 10 :*

Набор предметов, заведомо включенных в ранец в данном узле, нарушает оба ограничения, поэтому данный узел недопустим.

*Узел 11 :*

Следующим выбранным для ветвления узлом будет узел 9. Так как решение, связанное с этим узлом целочисленное, алгоритм заканчивается. Данное решение и будет оптимальным.

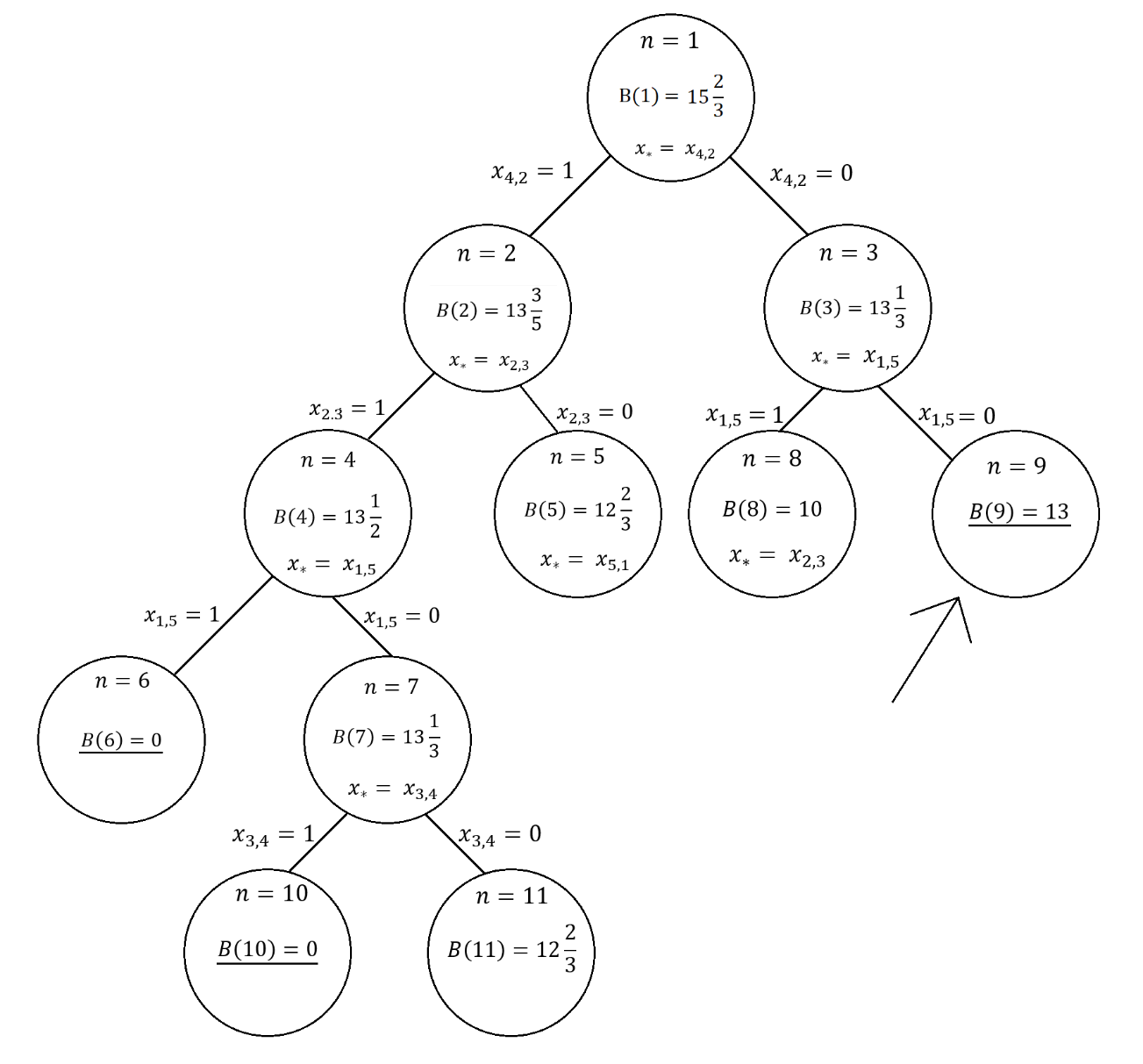


Рисунок 2. Итоговое дерево узлов

## Модификация метода ветвей и границ

Время работы алгоритма Шиха зависит от скорости вычисления верхней границы узла, количества узлов, которые будут рассмотрены до получения оптимального целочисленного решения и скорости выбора узла.

При вычислении верхней границы работа с каждым ограничением происходит раздельно. Первоначально мы загружаем все фиксированные в единицы предметы из множества , а затем среди неиспользуемых предметов из мы загружаем предметы с максимальной удельной ценностью в данном ограничении пока это возможно. Вспомогательные индексы, присвоенные заранее, позволяют упорядочить неиспользуемые переменные по удельной ценности для каждого ограничения, не производя дополнительных вычислений. Таким образом, единственные вычисления, которые происходят при нахождении суммарной стоимости оптимального дробного решения это суммирование стоимостей предметов из множества и сравнение суммарной стоимости предметов с ограничением . Поэтому, значительное ускорение вычисления верхней границы узла не представляется возможным.

Количество рассмотренных узлов зависит от двух ключевых моментов: выбор переменной ветвления и выбор узла для ветвления. Как уже упоминалось ранее, Гринберг и Хегерих [4] показали с помощью вычислительного эксперимента, что выбор для ветвления переменной с наименьшей удельной стоимостью в ограничении, связанном с верхней границей узла, является оптимальным. В качестве узла для ветвления в алгоритме Шиха выбирается узел с наибольшей верхней границей. Несмотря на то, что первый рассмотренный целочисленный узел и будет оптимальным и число рассмотренных вершин будет минимально, постоянный выбор узла с наибольшей границей может занимать много времени, ввиду необходимости хранения списка доступных вершин в отсортированном виде.

### Метод ветвей и границ: запоминание лучшего решения

Рассмотрим другую процедуру выбора и проверки узлов в методе ветвей и границ, основанную на запоминании лучшего целочисленного решения.

Зададим переменную , которая равно номеру узла, содержащего лучшее целочисленное решение на данный момент и зададим переменную , равную границе узла с лучшим решением. Для ветвления будем выбирать узел с наименьшим номером из доступных узлов на данный момент.

Теперь опишем измененный процесс ветвления. Помимо проверки на допустимость и вычисления границ образованных узлов, будем так же сравнивать полученную границу с . Если граница не превосходит суммарную стоимость лучшего сохраненного решения ( удалим данный узел, так как не имеет смысл рассматривать часть дерева с границей не превосходящей (если решение данного узла или его потомков будет целочисленным, то его суммарная стоимость все равно не превосходит стоимость сохраненного решения, следовательно, оно не является оптимальным). Если же граница больше, чем , то проверим, является ли решение данного узла целочисленным. В случае, если является, обновим лучшее решение , и удалим узел, хранящий старое лучшее решение.

Будем продолжать процедуру до тех пор, пока не останется один доступный узел. Решение, соответствующее этому узлу и будет оптимальным.

## 2.3 Распараллеливание метода ветвей и границ

Основная сложность при распараллеливании МВГ заключается в непредсказуемости метода. Структура дерева ветвления является динамической и строится в процессе решения., что существенно затрудняет распараллеливание.

Опишем алгоритм распараллеливания метода ветвей и границ, использованного в алгоритме Шиха и метода с запоминанием лучшего решения.

### Распараллеливание алгоритма Шиха

Так как метод ветвей и границ в алгоритме Шиха основан на постоянном выборе узла с наибольшей границей, в любой момент времени работы алгоритма не представляется возможным оценить последовательность узлов, которые будут выбраны для ветвления в дальнейшем. Несмотря на это, среди всех доступных узлов на определенном этапе работы алгоритма можно оценить какие узлы с большей вероятностью будут рассмотрены в последствии. Это будут узлы с наибольшей границей. Более того, чем больше граница узла, тем больше вероятность того, что узел будет выбран для ветвления.

Изменим структуру алгоритма Шиха. Вместо того что, чтобы каждый раз искать вершину с наибольшей границей среди всех доступных, будем хранить все узлы упорядоченно. Это позволит выбирать переменную для ветвления не производя поиск, а просто обращаясь к первому или последнему элементу упорядоченного списка. Более того, мы заранее будем знать какие вершины более вероятно будут включены в рюкзак впоследствии.

Опишем параллельный алгоритм. На каждом шаге алгоритма будем параллельно ветвиться по узлам с наибольшей границей, где это число потоков. После выхода из параллельной секции будем вставлять все образовавшиеся узлы в упорядоченный список и переходить к следующему шагу. Когда наибольший по границе узел окажется в вершине списка алгоритм прекращает работу – этот узел соответствует оптимальному решению.

Сложно оценить оптимальное число узлов, которые стоит рассматривать параллельно на каждом шаге, так как границы наибольших узлов могут сильно отличаться. Оптимальное число потоков будет определено с помощью вычислительного эксперимента.

### Распараллеливание МВГ с запоминанием лучшего решения

Для распараллеливания метода ветвей и границ с запоминанием лучшего воспользуемся другим подходом. Так как метод основан на последовательном рассматривании всех узлов, имеет смысл на каждом шаге алгоритма рассматривать все доступные на данный момент узлы. Будем параллельно проходить по всем доступным вершинам и ветвиться с соблюдением условия. В каждом потоке создадим переменную которая будет соответствовать лучшему решению для данного потока. Первоначально, присвоим ей значение лучшего решения, известного на данный момент. В каждом потоке будем действовать так же, как и в классическом алгоритме, только вместо общей переменной для лучшего решения будем работать с собственной переменной каждого потока. Затем, синхронизируем потоки и сравним лучшие решения каждого потока, чтобы найти общее лучшее решение. После этого рассмотрим все образовавшиеся узлы и удалим те, чья граница меньше суммарной стоимости общего лучшего решения, и перейдем к следующему шагу.

Заметим, что ветвление может произойти по узлам, которые в непараллельном варианте были бы просто удалены. Действительно, если в каком-то потоке обновилось лучшее решение, то в классическом варианте все последующие узлы с меньшей границей были бы просто удалены. Тем не менее, так как после синхронизации потоков мы обновляем общее лучшее решение и еще раз проверяем все образовавшиеся узлы, потомки узлов, по которым не должно было происходить ветвление, будут удалены перед следующим шагом.

# ГЛАВА 3. ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ

## 3.1 Реализация алгоритмов

В качестве средства разработки использовался *C++* 11-го стандарта. Параллельные вычисления производились при помощи стандартной директивы *OpenMP .* Число потоков, используемых кодом в параллельном варианте МВГ с запоминанием лучшего решения, равняется числу логических ядер процессора. Число потоков в параллельной реализации алгоритма Шиха будет варьироваться от двух до числа логических ядер процессора с целью определения оптимального числа потоков.

Узлы представлены как объекты реализованного класса *Node*, который хранит информацию о границе узла, номере переменной ветвления и множествах заведомо использованных переменных и неиспользованных . Вспомогательные индексы переменных хранятся в векторе векторов. При реализации алгоритма Шиха доступные для ветвления узлы хранятся в коллекции *set<Node>*, а при реализации МВГ с запоминанием лучшего решения доступные узлы хранятся в векторе *vector<Node>.*

## 3.2 Планирование вычислительного эксперимента

Для сравнительного тестирования разработанных алгоритмов была использована библиотека тестов *ORLib*, представленная Чу и Бизли. Библиотека содержит 9 наборов больших коррелированных задач с фиксированными значениями и ( и не имеет известных оптимальных решений. Веса предметов в каждом ограничении определялись как случайные целые числа из диапазона (0,1000). Ограничения определялись как усредненная суммарная стоимость весов, умноженная на коэффициент плотности , где . Для каждого значения коэффициента сгенерировано 10 задач. Стоимости предметов определялись как усредненная суммарная стоимость весов во всех предметах суммированная со случайным целом числом из диапазона (0,500): . Было показано, что, в общем, коррелирующие задачи решаются сложнее чем не коррелирующие.

Таким образом, в данной библиотеке представлено 270 задач (по 30 для каждого набора). Первоначально планируется тестирования многопоточного алгоритма Шиха с различным значением параметра , где это число потоков, с целью определения оптимального числа потоков на 30 задачах из первого набора примеров (. Затем будет произведен сравнительный анализ четырех реализованных алгоритмов (алгоритм Шиха, параллельный алгоритм Шиха с оптимальным числом потоков, МВГ с запоминанием лучшего решения и его параллельный вариант).

Правильность работы алгоритмов была проверена на небольших задачах с известным оптимальным решением из библиотеки *SAC94*. Использованные библиотеки были представлены Д. Дрейком [7].

## 3.3 Тестирование алгоритмов и анализ результатов

Вычисления производились на системе с процессором AMD Ryzen 5 3600, который обладает 12 логическими ядрами. Поэтому многопоточный алгоритм Шиха был протестирован 11 с использование числа потоков от двух до двенаднцати.

Внизу приведена таблица времени работы алгоритма Шиха для различного числа потоков (SA – классический алгоритм Шиха, SA2 – многопоточный алгоритм Шиха с числом потоков 2 и т. д.) для набора 30 задач со значением и . Для каждого коэффициента плотности было рассмотрено 10 сгенерированных задач и посчитано среднее время работы программы. Среднее время указано в секундах с точностью до десятых.

Таблица 3. Время работы алгоритма Шиха для различного числа потоков.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | SA | SA2 | SA3 | SA4 | SA5 | SA6 |
| 100 | 5 | 0.25 | 61.5 | 32.2 | 23.7 | 19.2 | 17.3 | 15.4 |
| 0.50 | 205.7 | 103.6 | 71.1 | 54.9 | 46.4 | 40.2 |
| 0.75 | 61.9 | 33.8 | 24.7 | 20.2 | 18.2 | 16.6 |
| Avg. | 109.7 | 56.5 | 39.8 | 31.4 | 27.3 | 24.1 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | SA7 | SA8 | SA9 | SA10 | SA11 | SA12 |
| 100 | 5 | 0.25 | 14.8 | 14.2 | 13.7 | 13.4 | 13.2 | 13.1 |
| 0.50 | 36.5 | 33.5 | 31.0 | 30.0 | 28.0 | 26.7 |
| 0.75 | 16.0 | 15.9 | 15.5 | 15.3 | 15.2 | 15.1 |
| Avg. | 22.4 | 21.2 | 20.1 | 19.6 | 18.8 | 18.3 |

Как видно из таблицы, увеличение числа потоков ускоряет выполнение алгоритма. Тем не менее, прирост скорости с увеличением потоков постепенно уменьшался. Два потока ускоряли работу алгоритма примерно в 2 раза, четыре потока чуть больше чем в 3 раза, а 12 потоков давали ускорение примерно в 4 раза. Это связано с тем, что при увеличении числа потоков на каждом шаге мы начинаем рассматривать все больше и больше узлов, ветвление по которым не должно было происходить. Предполагается, что при последующем увеличении числа потоков время работы алгоритма стало бы увеличиваться.

Рисунок 4. График среднего времени работы параллельного алгоритма Шиха с различным параметром числа потоков.

В процессе выполнения вычислений был обнаружен интересный факт. Чем меньше время решения конкретной задачи, тем раньше замечалось увеличение времени работы алгоритма при увеличении числа потоков и, наоборот, чем больше выполнялась программа, тем дольше сохранялось эффективность многопоточных вариантов при увеличении числа алгоритмов. Рассмотрим два примера – пятую и девятую задачу из первой выборки ().

Можно заметить, что у задачи, которая решилась, быстро началось постоянное увеличение время работы начиная с 7-го потока, при чем время работы начинало расти все быстрее и быстрее при увеличении числа потоков. В то же время, в девятой задаче увеличение скорости было постоянным и замедлилось только при достижении 12 потоков.

Так же стоит учитывать, что все тесты были коррелированными. При некоррелированных тестах структура дерева ветвления могла бы сильно отличаться от тех, что были построены в процессе решения данных задач.

Таблица 4. Время работы алгоритма Шиха для различного числа потоков для двух конкретных задач. – номер задачи в заданной выборке.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | SA | SA2 | SA3 | SA4 | SA5 | SA6 |
|  | 1.96 | 1.39 | 1.10 | 1.00 | 0.99 | 0.97 |
|  | 47.97 | 25.32 | 19.27 | 15.91 | 14.24 | 13.26 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | SA7 | SA8 | SA9 | SA10 | SA11 | SA12 |
|  | 1.03 | 1.05 | 1.07 | 1.09 | 1.17 | 1.19 |
|  | 12.74 | 12.42 | 12.27 | 12.00 | 11.98 | 11.98 |

Так же стоит учитывать, что все тесты были коррелированными. При некоррелированных тестах структура дерева ветвления могла бы сильно отличаться от тех, что были построены в процессе решения данных задач.

По результатам произведенного эксперимента оптимальным числом потоков будет считаться максимально возможное значение на используемой системе – 12. Для сравнительного анализа четырех реализованных алгоритмов(Алгоритм Шиха – SA, Параллельный алгоритм Шиха с использованием 12 потоков – SA12, МВГ с запоминанием лучшего решения – BSS(best solution saving) и его параллельный вариант - BSSP) было произведено тестирование на следующем наборе примеров: все возможные наборы тестов со ста предметами ( и ) и 15 задач с использованием набора из 250 переменных и 10 ограничений ( и ), по 5 случайно выбранных задач для каждого значения коэффициента плотности,0.50,0.75. Результаты тестирования приведены в таблице 5 (N – число рассмотренных из набора тестов с заданными параметрами), среднее время указано в секундах с точностью до десятых.

Как можно заметить, модификация алгоритма Шиха в среднем работает сильно быстрее классического алгоритма. Это связано с тем, что при разрастании древа ветвления на больших примеров число доступных узлов может достигнуть нескольких тысяч. Так как алгоритм предполагает хранение доступных узлов в упорядоченном виде, при большом числе узлов вставка новых вершин в упорядоченный список начинает занимать много времени. Несмотря на то, что число рассматриваемых вершин в алгоритме с запоминанием лучшего решения намного больше, чем в алгоритме Шиха, вычеркивание заведомо неподходящих вершин сильно уменьшает суммарное число рассмотренных узлов. А тот факт, что данный алгоритм не предполагает хранение узлов в упорядоченном виде, позволяет понять причину получившихся результатов. Стоит заметить, что во многих задачах время работы алгоритма Шиха было на порядок меньше. Трудно заранее определить какой алгоритм будет более эффективным для конкретной задачи, ведь эффективность зависит от структуры дерева ветвления и вершины дерева, где хранится оптимальный узел. В классическом алгоритме структура дерева совершенно непредсказуема, в то время как спуск по дереву в модификации происходит равномерно с вычеркиванием заведомо невыгодных ветвей.

Таблица 5. Время работы реализованных алгоритмов.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | N | SA | SA12 | BSS | BSSP |
| 100 | 5 | 0.25 | 10 | 61.5 | 13.1 | 21.3 | 17.2 |
| 0.50 | 10 | 205.7 | 26.7 | 24.7 | 19.9 |
| 0.75 | 10 | 61.9 | 15.1 | 99.9 | 85.1 |
| 100 | 10 | 0.25 | 10 | 42.0 | 10.3 | 19.0 | 13.7 |
| 0.50 | 10 | 251.9 | 42.2 | 71.3 | 52.2 |
| 0.75 | 10 | 2001.1 | 232.9 | 146.6 | 110.8 |
| 100 | 30 | 0.25 | 10 | 25.5 | 11.3 | 54.0 | 34.3 |
| 0.50 | 5 | 432.1 | 89.6 | 164.7 | 109.6 |
| 0.75 | 5 | 297.2 | 79.6 | 326.6 | 230.1 |
| 250 | 5 | 0.25 | 5 | 65.4 | 14.3 | 21.8 | 17.7 |
| 0.50 | 5 | 654.6 | 71.2 | 29.1 | 23.5 |
| 0.75 | 5 | 109.1 | 23.4 | 34.8 | 28.5 |
| Avg. | | 350.7 | 52.5 | 84.5 | 61.9 |

Так как сравнение алгоритма Шиха и его параллельной реализации было проведено ранее на меньшей выборке тестов, полученный результат был предсказуем. Сравним результаты алгоритмов BSS и BSSP. В среднем параллельный вариант алгоритма при работе на 12 потоках дает выигрыш в 30%. Такой незначительный прирост по сравнению с алгоритмом Шиха и его параллельным вариантом можно объяснить несколькими ключевыми моментами. Во-первых, как уже упоминалось ранее, каждый поток сохраняет свое лучшее решение, что приводит к множественному ветвлению ненужных узлов. Во-вторых, синхронизация потоков с целью определения общего лучшего решения и повторный проход по всем рассмотренным узлам на каждом шаге с вычеркиванием ненужных узлов может занимать много времени. В-третьих, несмотря на то что каждый поток работает со своим набором узлов, все эти узлы хранятся в общей памяти, что может немного замедлять работу потоков. Первоначально рассматривался вариант, где для каждого потока была создана своя коллекция с узлами и последующим объединением этих коллекций в критической секции, но из-за плохой эффективности на малых задачах этот вариант был отброшен.

Если сравнивать реализованные параллельные алгоритмы, то они получились достаточно конкурентноспособными. Это связано с тем, что классический алгоритм Шиха в среднем работает медленнее BSS, в то время как его параллельная реализация дает намного больший прирост. Заранее определить оптимальный алгоритм для конкретной задачи тяжело – все зависит от многих факторов, таких как способ генерации тестов, корреляция сгенерированных значений, структура дерева ветвления и местоположения узла с оптимальным решением.

Рисунок 5. Среднее время работы реализованных алгоритмов.

Результаты тестирования для каждого рассмотренного примера содержатся в приложении.

# ЗАКЛЮЧЕНИЕ

В рамках данной исследовательской работы:

* осуществлен обзор существующих решений многомерной задачи о ранце;
* реализован один из алгоритмов, создана его модификация и разработаны параллельные варианты реализованных алгоритмов на языке C++;
* произведен теоретический анализ реализованных алгоритмов;
* проведены экспериментальные исследования эффективности предложенных методов решения задачи с целью сравнительного анализа;
* выявлены оптимальные алгоритмы для наборов больших задач.

В качестве дальнейших направлений работы по усовершенствованию предложенных алгоритмов можно рассмотреть модификацию алгоритмов, не связанную с усовершенствованием метода ветвей и границ, например, анализ выбора оптимального узла для ветвления или ускорение вычисления границы узла с помощью многопоточности.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Fréville A. The multidimensional 0–1 knapsack problem: An overview //European Journal of Operational Research. – 2004. – Т. 155. – №. 1. – С. 1-21.

2. W. Shih, A branch and bound method for the multiconstraint zero–one knapsack problem //Journal of the Operations Research Society. – 1979. – Т. 30.– С. 369-378.

3. G.B. Dantzig, Discrete-Variable Extremum Problems //European Journal of Operational Research. – 1957. – Т. 5. – №. 2. – С. 266-277.

4. H. Greenberg and R. L. Hegerich, A Branch Search Algorithm for the Knapsack Problem //Management Science. – 1970. – Т. 16. – №. 5. – С. 327-322.

5. Kolesar, P. J., A Branch and Bound Algorithm for the Knapsack Problem //Management Science. – 1967. – Т. 13. – С. 723-735.

6. P.C. Chu & J.E. Beasley, A Genetic Algorithm for the Multidimensional Knapsack Problem //Journal of Heuristics. – 1998. – Т. 4. – С. 63-86.

7. J. H. Drake, Benchmark instances for the Multidimensional Knapsack Problem, [Электронный ресурс] — URL: Режим доступа: <https://www.researchgate.net/publication/271198281_Benchmark_instances_for_the_Multidimensional_Knapsack_Problem>, свободный. (дата обращения. 16.09.2021)

# ПРИЛОЖЕНИЕ

Таблица 6. Время работы алгоритмов для каждой задачи

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | k | SA | SA12 | BSS | BSSP |
| 100 | 5 | 0.25 | 1 | 146.583 | 27.233 | 30.68 | 24.732 |
| 2 | 14.901 | 4.997 | 11.015 | 8.862 |
| 3 | 65.764 | 14.945 | 18.287 | 14.503 |
| 4 | 35.357 | 9.491 | 28.24 | 23.194 |
| 5 | 65.338 | 15.072 | 21.346 | 16.911 |
| 6 | 1.96 | 1.185 | 1.493 | 1.157 |
| 7 | 39.138 | 9.682 | 40.832 | 32.71 |
| 8 | 191.834 | 34.029 | 40.875 | 33.026 |
| 9 | 47.973 | 11.981 | 13.55 | 11.046 |
| 10 | 6.313 | 2.668 | 6.77 | 5.434 |
| 0.50 | 1 | 6.61 | 3.388 | 18.749 | 15.271 |
| 2 | - | - | 104.747 | 85.008 |
| 3 | 2.314 | 1.933 | 5.868 | 4.766 |
| 4 | 79.667 | 18.195 | 15.767 | 12.764 |
| 5 | 0.021 | 0.029 | 0.021 | 0.029 |
| 6 | 0.272 | 0.926 | 2.118 | 1.778 |
| 7 | 7.968 | 3.701 | 4.762 | 3.827 |
| 8 | 1721.82 | 199.796 | 72.7 | 58.481 |
| 9 | 16.784 | 6.659 | 8.399 | 6.692 |
| 10 | 15.644 | 5.568 | 14.129 | 11.323 |
| 0.75 | 1 | 46.052 | 12.866 | 31.283 | 26.295 |
| 2 | 13.529 | 5.732 | 28.717 | 23.98 |
| 3 | 84.926 | 18.017 | 97.828 | 80.146 |
| 4 | 79.651 | 17.636 | 61.208 | 50.484 |
| 5 | 87.396 | 18.878 | 45.87 | 36.794 |
| 6 | 11.763 | 7.293 | 6.242 | 4.874 |
| 7 | 65.219 | 17.033 | 91.493 | 75.746 |
| 8 | 69.262 | 17.278 | 69.426 | 57.002 |
| 9 | 72.126 | 17.272 | 30.054 | 24.038 |
| 10 | 89.448 | 19.388 | 537.46 | 471.909 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 10 | 0.25 | 1 | 86.281 | 21.134 | 81.267 | 59.467 |
| 2 | 7.548 | 3.74 | 5 | 3.632 |
| 3 | 40.308 | 12.141 | 19.131 | 13.462 |
| 4 | 2.623 | 1.622 | 6.835 | 5.005 |
| 5 | 13.462 | 5.348 | 17.412 | 12.653 |
| 6 | 18.597 | 7.123 | 10.494 | 7.513 |
| 7 | 1.172 | 0.993 | 2.067 | 1.516 |
| 8 | 6.102 | 3.039 | 4.317 | 3.13 |
| 9 | 4.38 | 2.492 | 4.271 | 3.141 |
| 10 | 239.571 | 45.447 | 39.267 | 27.655 |
| 0.50 | 1 | 12.558 | 6.075 | 17.001 | 12.297 |
| 2 | 74.109 | 21.888 | 24.16 | 17.431 |
| 3 | 28.458 | 10.828 | 27.32 | 19.725 |
| 4 | 284.857 | 61.968 | 81.35 | 58.352 |
| 5 | 1537.98 | 198.266 | 110.954 | 78.076 |
| 6 | 394.53 | 68.76 | 98.735 | 70.958 |
| 7 | 10.408 | 5.017 | 7.214 | 5.193 |
| 8 | 68.126 | 20.398 | 22.514 | 16.098 |
| 9 | 3.647 | 2.435 | 35.663 | 25.651 |
| 10 | 104.824 | 26.63 | 287.846 | 218.551 |
| 0.75 | 1 | 1643.71 | 224 | 123.81 | 90.795 |
| 2 | 87.882 | 28.4 | 59.207 | 44.524 |
| 3 | 33 | 12.725 | 57.474 | 44.018 |
| 4 | 1932.74 | 253.958 | 128.299 | 93.444 |
| 5 | 8421.38 | 934.704 | 392.888 | 310.213 |
| 6 | 681.84 | 99.071 | 66.966 | 49.288 |
| 7 | 491.251 | 74.887 | 87.427 | 65.839 |
| 8 | 3988.38 | 404.94 | 208.682 | 152.583 |
| 9 | 2637.16 | 271.765 | 245.539 | 182.093 |
| 10 | 93.235 | 25.16 | 95.283 | 75.846 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 30 | 0.25 | 1 | 14.974 | 7.672 | 12.24 | 7.589 |
| 2 | 3.167 | 2.886 | 86.951 | 54.925 |
| 3 | 14.812 | 8.304 | 30.132 | 19.099 |
| 4 | 56.067 | 23.225 | 50.339 | 32.203 |
| 5 | 33.365 | 15.589 | 50.575 | 31.944 |
| 6 | 0.787 | 1.452 | 0.886 | 0.604 |
| 7 | 25.09 | 12.515 | 69.289 | 44.582 |
| 8 | 10.1 | 5.878 | 12.498 | 8.147 |
| 9 | 5.107 | 3.548 | 17.689 | 11.643 |
| 10 | 91.361 | 32.602 | 209.804 | 132.138 |
| 0.50 | 1 | 20.708 | 12.239 | 90.743 | 60.651 |
| 2 | 23.464 | 13.493 | 24.021 | 16.696 |
| 3 | 1852.88 | 329.491 | 496.467 | 328.569 |
| 4 | 50.28 | 23.537 | 50.926 | 34.249 |
| 5 | 213.037 | 69.326 | 161.53 | 107.788 |
| 0.75 | 1 | 405.324 | 108.486 | 170.666 | 116.011 |
| 2 | 52.205 | 26.392 | 76.85 | 54.629 |
| 3 | 36.378 | 19.248 | 266.685 | 184.478 |
| 4 | 873.169 | 193.642 | 334.489 | 223.315 |
| 5 | 118.879 | 50.262 | 784.223 | 572.048 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 250 | 5 | 0.25 | 1 | 146.841 | 27.067 | 30.629 | 24.618 |
| 2 | 14.774 | 5.005 | 10.903 | 8.892 |
| 3 | 64.977 | 14.783 | 18.154 | 14.58 |
| 4 | 35.424 | 9.429 | 28.201 | 23.281 |
| 5 | 64.995 | 14.979 | 21.358 | 16.939 |
| 0.50 | 1 | 6.477 | 3.364 | 18.77 | 15.239 |
| 2 | 3185.26 | 332.556 | 104.872 | 84.701 |
| 3 | 2.325 | 1.94 | 5.886 | 4.728 |
| 4 | 78.888 | 18.305 | 15.809 | 12.675 |
| 5 | 0.022 | 0.03 | 0.021 | 0.029 |
| 0.75 | 1 | 47.366 | 12.879 | 31.249 | 26.364 |
| 2 | 13.866 | 5.748 | 28.788 | 23.989 |
| 3 | 190.254 | 43.589 | 61.666 | 50.252 |
| 4 | 281.703 | 47.301 | 46.013 | 36.977 |
| 5 | 12.139 | 7.406 | 6.231 | 4.951 |