**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФГБОУ ВО “СЕВЕРО-КАВКАЗСКИЙ ГОРНО-МЕТАЛЛУРГИЧЕСКИЙ ИНСТИТУТ   
(ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)**

**Факультет** Информационных технологий и электронной техники

**Кафедра** Информатика и вычислительная техника

**Направление подготовки** Информатика и вычислительная техника

**Профиль** АСОИиУ

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ**

**на тему:** Экспериментальный анализ эффективности детерминированных композитных алгоритмов, совмещающих на каждой итерации локальный перебор, вызванный введением в базис h переменных единовременно, со спуском по дереву ветвлений в лучшем направлении применительно к поиску локально оптимальных решений задачи о ранце.

**Студентка** Джиоева Лаура Казбековна

**Руководитель проекта д.т.н., проф. Гроппен В. О.**

**Проект рассмотрен кафедрой и допущен к защите в ГЭК**

**Заведующий кафедрой проф. Гроппен В. О.**

**г. Владикавказ 2019 г.**

# РЕФЕРАТ

Пояснительная записка 100 с., 13 рис., 12 табл., 8 ист., 2 прил.

ЛОКАЛЬНО ОПТИМАЛЬНОЕ РЕШЕНИЕ, БУЛЕВЫ ПЕРЕМЕННЫЕ, КОМПОЗИТНЫЕ АЛГОРИТМЫ, ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ, С#, СПУСК ПО ДЕРЕВУ ВЕТВЛЕНИЙ В ЛУЧШЕМ НАПРАВЛЕНИИ, ЭКСПЕРЕМЕНТАЛЬНЫЙ АНАЛИЗ, ЗАДАЧА О РАНЦЕ.

**Объект разработки:** композитная версия алгоритма решения задачи о ранце.

**Цель выпускной работы:** оценка сравнительной эффективности композитной и традиционной версий поиска локально оптимального решения задачи о ранце, детерминированным спуском по дереву ветвлений в лучшем направлении.

**Использованное прикладное и системное программное обеспечение:**

* Операционная система Windows 10 PRO – 64 bit.
* Среда разработки Visual Studio 2013.

**Характеристики вычислительной системы:**

Intel Core i5-24500M 2500 МГц nVidia GeForce GT 620M, ОЗУ 16 Гб, ОС Windows 10 PRO – 64 bit.

**Полученные результаты:**

В рамках работы был проведен аналитический обзор предметной области, выбрана математическая модель, разработана композитная версия алгоритма, использующаяся для решения задачи о ранце. Также создан программный комплекс, реализующий данный композитный алгоритм. Была проведена серия экспериментов, подтверждающая адекватность выбранной математической модели и эффективность программного комплекса.

Оглавление

[РЕФЕРАТ 2](#_Toc516222521)

[ВВЕДЕНИЕ 5](#_Toc516222522)

[Глава 1. АНАЛИТИЧЕСКИЙ ОБЗОР 7](#_Toc516222523)

[1.1 Постановка задачи 7](#_Toc516222524)

[1.2 Задача о ранце 7](#_Toc516222526)

[1.2.1 Классификация задач о ранце 8](#_Toc516222527)

[1.2.2 NP-полнота­­ 10](#_Toc516222527)

[1.3 Комбинаторная оптимизация 15](#_Toc516222534)

[1.4 Приближенные методы решения задачи о ранце 17](#_Toc516222534)

[1.4.1 Жадный алгоритм 17](#_Toc516222535)

[1.4.2 Метод муравьиной колонии 19](#_Toc516222536)

[1.4.3 Генетический метод 24](#_Toc516222536)

[1.5 Точные методы решения задачи о ранце. 27](#_Toc516222537)

[1.5.1 Полный перебор 27](#_Toc516222536)

[1.5.2 Динамическое программирование 29](#_Toc516222536)

[1.5.3 Метод ветвей и границ 30](#_Toc516222536)

[1.6 Детерминированный композитный алгоритм 31](#_Toc516222538)

[1.7 Заключение 32](#_Toc516222538)

[Глава 2. Содержательная и формальная постановка задачи 33](#_Toc516222539)

[Глава 3. Выбор и обоснование алгоритма 35](#_Toc516222539)

[3.1 Выбор и обоснование алгоритма 35](#_Toc516222540)

[3.2 Детерминированный композитный алгоритм 35](#_Toc516222540)

[3.3 Классический алгоритм 41](#_Toc516222541)

[3.4 Заключение 42](#_Toc516222545)

[Глава 4. ПРОГРАММНЫЙ КОМПЛЕКС 43](#_Toc516222546)

[4.1 Выбор ОС и среды разработки 43](#_Toc516222547)

[4.2 Язык программирования C# 43](#_Toc516222548)

[4.3 Работа программного комплекса 44](#_Toc516222549)

[3.4 Заключение 50](#_Toc516222550)

[Глава 5. Экспериментальная часть 51](#_Toc516222551)

[4.1 Эксперимент 1. 52](#_Toc516222552)

[4.2 Эксперимент 2. 60](#_Toc516222553)

[4.3 Эксперимент 3. 65](#_Toc516222554)

[4.4 Эксперимент 4. 68](#_Toc516222554)

[ЗАКЛЮЧЕНИЕ 70](#_Toc516222556)

[Список использованных источников 71](#_Toc516222557)

[Приложение 1. ГРАФИЧЕСКАЯ ЧАСТЬ 72](#_Toc516222558)

[Приложение 2. Листинг программы 83](#_Toc516222559)

# ВВЕДЕНИЕ

Практика порождает все новые и новые задачи оптимизации, причем их сложность растет. Реальные прикладные задачи дискретной оптимизации очень сложны. Современные методы оптимизации далеко не всегда справляются с решением реальных задач без помощи человека. Требуются новые математические модели и методы, которые позволяют учитывать наличие многих критериев и проводят глобальный поиск оптимума. Следует отдавать предпочтение таким методам, которыми проще управлять в процессе решения задачи.

Классическая задача о ранце (КЗР) относится к числу широко известных задач дискретной оптимизации. Задача о ранце является актуальной и достаточно востребованной с точки зрения ее приложения в реальной жизни. Задача о загрузке (о ранце) и её модификации часто возникают в экономике, прикладной математике, криптографии, генетике и логистике для нахождения оптимальной загрузки транспорта (самолёта, поезда, трюма корабля) или склада. В частности, следует отметить формулируемые в рамках КЗР задачи объемного планирования для предприятий с единичным и мелкосерийным характером производства и задачи загрузки транспортных средств.

Комбинаторная оптимизация – это широкая и бурно развивающаяся область математического программирования и дискретной математики, исследующая структурные и оптимизационные задачи на объектах, имеющих выраженный комбинаторный смысл.

Рассматриваемая нами задача является NP–полной, то есть для нее не существует полиномиального алгоритма, решающего ее за разумное время.

Существует несколько модификаций задачи.

1. Каждый предмет можно брать только один раз.
2. Каждый предмет можно брать сколько угодно раз.
3. Каждый предмет можно брать определенное количество раз
4. На размер рюкзака имеется несколько ограничений.
5. Некоторые вещи имею больший приоритет, чем другие

Цель данной работы – оценка сравнительной эффективности композитной и традиционной версий поиска локально оптимального решения задачи о ранце, детерминированным спуском по дереву ветвлений в лучшем направлении.

Провести аналитический обзор предметной области, выбрать математическую модель, разработать композитную версию алгоритма, использующуюся для решения задачи о ранце. Также создать программный комплекс, реализующий данный детерминированный композитный алгоритм. Провести серию экспериментов, подтверждающих адекватность выбранной математической модели и эффективность программного комплекса.

## АНАЛИТИЧЕСКИЙ ОБЗОР

### Постановка задачи

Целью моей дипломной работы является, оценка сравнительной эффективности композитной и традиционной версий поиска локально оптимального решения задачи о ранце, детерминированным спуском по дереву ветвлений в лучшем направлении.

Исходя из поставленной цели, были выделены следующие задачи:

1. Провести аналитический обзор предметной области.
2. Выбрать математическую модель
3. Разработать композитную версию алгоритма, использующуюся для решения задачи о ранце.
4. Создать программный комплекс, реализующий разработанный алгоритм.
5. Фактически проверить адекватность мат. модели и разработанный программный комплекс.

### Задача о ранце

Задача о ранце (рюкзаке) — одна из NP-полных задач комбинаторной оптимизации. Своё название задача получила от оптимизационной задачи укладки как можно большего числа ценных вещей в рюкзак при условии, что общий объём (или вес) всех предметов, способных поместиться в ранец, ограничен. Задача о ранце является актуальной и достаточно востребованной с точки зрения ее приложения в реальной жизни. Задача о загрузке (о рюкзаке) и её модификации часто возникают в экономике, прикладной математике, криптографии, генетике и логистике для нахождения оптимальной загрузки транспорта (самолёта, поезда, трюма корабля) или склада.

Пусть имеется набор предметов, каждый из которых имеет два параметра – вес и ценность. Имеется ранец c некоторым заданным значением вместимости. Задача заключается в том, чтобы собрать ранец с максимальной ценностью предметов внутри, соблюдая при этом весовое ограничение ранца. Математически постановка задачи формулируется следующим образом: пусть имеется n предметов. Для каждого i-го предмета задан его вес 0 > pi и стоимость (ценность) 0 > ci , 1,2,…,n. Задано ограничение на максимальный вес ранца ‒ P . Каждый xi может принимать только одно из двух значений: xi = 1, если i-й предмет упаковывают в ранец, или xi = 0, в противном случае.

Требуется выбрать из заданного множества предметов набор с максимальной суммарной стоимостью при одновременном соблюдении ограничения на суммарный вес найденного набора [2]

#### Классификация задач о ранце

Существует несколько разновидностей задачи о ранце, отличия между которыми заключаются в условиях, наложенных на ранец, предметы или их выбор:

1. Ранец 0-1 или Классическая задача;
2. Ограниченный вес ранца;
3. Неограниченный вес ранца;
4. Непрерывный ранец;
5. Ранец с мультивыбором;
6. Мультипликативный ранец;
7. Многомерный ранец.[2]
8. Ранец 0-1

Это самая распространенная разновидность ранца. Пусть принимает два значения: , если груз упакован, и в противном случае, где . Задача: максимизировать при наличии ограничения на вместимость ранца. [3]

1. Ограниченный вес ранца

Каждый предмет может быть выбран ограниченное число раз. Задача: максимизировать так, чтобы выполнялось условие на вместимость и для всех . Число называют границей. [4]

1. Неограниченный вес ранца

Каждый предмет может быть выбран неограниченное число раз. Задача: максимизировать так, чтобы выполнялось условие на вместимость и для всех [4]

1. Непрерывный ранец

Вариант задачи, в котором возможно брать любую дробную часть от предмета, при этом удельная стоимость сохраняется. Задача выбрать часть каждого предмета так, чтобы максимизировать общую стоимость и чтобы выполнилось условие совместности: , где дробное, для всех [5]

1. Ранец с мультивыбором

Все предметы разделяют на классов . Обязательным является условие выбора только одного предмета из каждого класса. принимает значение только 0 и 1. Задача: максимизировать так, чтобы выполнялось условие на вместимость и для всех

1. Мультипликативный ранец

Пусть у нас есть предметов и ранцев . У каждого предмета есть вес и ценность , у каждого ранца соответственно своя вместимость при . Задача: максимизировать так, чтобы выполнялось условие для всех , для всех [3]

1. Многомерный ранец

Если есть более одного ограничения на ранец, например объем и вес, задачу называют m-мерной задачей о ранце. Например, для не ограниченного варианта: максимизировать так, чтобы

, и для всех [4]

#### NP-полнота

В [теории алгоритмов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2) классом NP (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) non-deterministic polynomial) называют множество [задач разрешимости](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B5%D1%88%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B8), решение которых возможно проверить на [машине Тьюринга](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0) за время, не превосходящее значения некоторого [многочлена](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD) от размера входных данных, при наличии некоторых дополнительных сведений (так называемого сертификата решения).

Эквивалентно класс NP включает задачи, которые можно за [полиномиальное время](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0#%D0%9F%D0%BE%D0%BB%D0%B8%D0%BD%D0%BE%D0%BC%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D1%8F) решить на [недетерминированной машине Тьюринга](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B4%D0%B5%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0).

Задачи, имеющие полиномиальные по времени алгоритмы решения, можно решать с помощью компьютера значительно быстрее, чем путём прямого перебора, время которого [экспоненциально](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D1%81%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%B0). Это обусловливает практическое значение проблемы о [равенстве классов P и NP](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B2%D0%B5%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2_P_%D0%B8_NP).

NP-полная задача — в [теории алгоритмов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2) [задача с ответом «да» или «нет»](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B5%D1%88%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B8) из [класса NP](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_NP), к которой можно свести любую другую задачу из этого класса за [полиномиальное время](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BD%D0%BE%D0%BC%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D1%8F) (то есть при помощи операций, число которых не превышает некоторого полинома в зависимости от размера исходных данных). Таким образом, NP-полные задачи образуют в некотором смысле подмножество «типовых» задач в классе NP: если для какой-то из них найден «полиномиально быстрый» алгоритм решения, то и любая другая задача из класса NP может быть решена так же «быстро».

Алфавитом называется всякое конечное множество символов (напрмер, {0, 1} или {a, b, c}). Множество всех возможных слов (конечных строк, составленных из символов этого алфавита) над некоторым алфавитом ∑ обозначается . Языком L нал алфавитом ∑ называется всякое подмножество множества , то есть L ⸦ . Задачей распознавания для языка L называется определение того, принадлежит ли данное слово языку L. Пусть L1 и L2 – два языка над алфавитом ∑. Язык L1 называется сводимым (по Карпу) к языку L2 , если существует функция, , вычислимая за полиноминальное время, обладающая следующим свойством: тогда и только тогда, когда . Сводимость по Карпу обозначается как .

Язык {\displaystyle L\_{2}}Д называется NP-трудным, если любой язык из класса NP сводится к нему. Язык называют NP-полным, если он NP-труден, и при этом сам лежит в классе NP. Неформально говоря, то что задача  A {\displaystyle A} Ф сводится к задаче {\displaystyle B}B, означает, что задача {\displaystyle A}A «не сложнее» задачи B{\displaystyle B}BBBBb (так как, если мы можем решить {\displaystyle B}B, то можем решить и {\displaystyle A}A). Таким образом, класс NP-трудных задач включает NP-полные задачи и задачи, которые «сложнее» их (то есть те задачи, к которым могут быть сведены NP-полные задачи). Класс NP включает NP-полные задачи и задачи, которые «легче» их (то есть те задачи, которые сводятся к NP-полным задачам).

Из определения следует, что, если будет найден алгоритм, решающий некоторую (любую) NP-полную задачу за полиномиальное время, то все NP-задачи окажутся в [классе P](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_P), то есть будут решаться за полиномиальное время.

Задача называется NP-полной в сильном смысле, если у неё существует подзадача, которая:

1. не является [задачей с числовыми параметрами](https://ru.wikipedia.org/w/index.php?title=%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D1%81_%D1%87%D0%B8%D1%81%D0%BB%D0%BE%D0%B2%D1%8B%D0%BC%D0%B8_%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%B0%D0%BC%D0%B8&action=edit&redlink=1) (то есть максимальное значение величин, встречающихся в этой задаче, ограничено сверху полиномом от длины входа),
2. принадлежит классу NP,
3. является NP-полной.

Класс таких задач называется NPCS. Если [гипотеза P ≠ NP](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B2%D0%B5%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2_P_%D0%B8_NP) верна, то для NPCS-задачи не существует [псевдополиномиального алгоритма](https://ru.wikipedia.org/wiki/%D0%9F%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D0%BF%D0%BE%D0%BB%D0%B8%D0%BD%D0%BE%D0%BC%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC). [6]

Задача о ранце является NP-полной, т.е. для нее не существует полиномиального алгоритма, решающего её за разумное время. Если перебирать всевозможные подмножества данного набора из n предметов, то получится решение сложности не менее чем O(2n). В настоящее время неизвестен (и, скорее всего, вообще не существует) алгоритм решения этой задачи, сложность которого является многочленом от n.

В статье «Изучение различных постановок задачи о рюкзаке и методов их решениях» [2] был рассмотрен практический пример задачи о ранце с максимальной вместимостью ранца P = 30 и количеством предметов n = 20 , остальные данные приведены в Таблице 1.1.



Таблица 1.1 Предметы для практической задачи ранца

С помощью перечисленных методов были найдены решения всех семи вариантов задачи о ранце, а также выполнено сравнение методов по скорости и точности нахождения решения. Проведенное сравнение показано в Таблице 1.2.

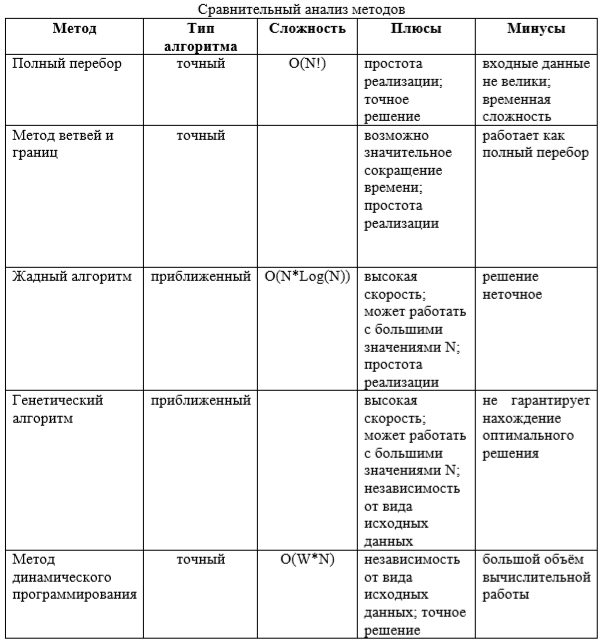


Таблица 1.2 Сравнительный анализ методов решения задачи о ранце

При выборе алгоритма решения приходится выбирать между точными алгоритмами, которые не применимы для ранцев большой размерности, и приближенными, которые работают быстро, но не обеспечивают оптимального решения задачи. Выбор использования того или иного метода является спорным вопросом. Всё зависит от постановки задачи, а также от того, какие цели поставлены. Если требуется найти точное решение, то необходимо использовать точные методы. В ситуации небольшого набора входных данных (до 10-20 предметов), лучше применять методы перебора или ветвей и границ, в силу простоты реализации. Если же точность решения не играет большой роли, или входные данные таковы, что ни один из точных методов не работоспособен, то для решения задачи можно использовать лишь приближенные алгоритмы. [2]

### Комбинаторная оптимизация

Существует обширный класс задач, имеющих важное фундаментальное и/или прикладное значение, в рамках которых возникает необходимость в минимизации или максимизации функций различного вида (говоря другими словами, в отыскании их экстремумов). Данный раздел математики называется оптимизацией и включает в себя большое число задач различного вида и подходов к их решению.

Оптимизационные задачи могут быть разделены на два существенно различных класса: задачи непрерывной (гладкой) и задачи дискретной оптимизации. Их основным отличием являются типы функций, для которых производится отыскание экстремумов. В первом случае это, как правило, гладкие дифференцируемые функции (в некоторых случаях с рядом особенностей), определенные на множестве вещественных или комплексных значений аргументов (как правило, бесконечном или несчетном); во втором – как правило, функции от какого-либо дискретного аргумента или дискретной математической структуры (например, счетного множества, графа или матроида). Задачи непрерывной оптимизации известны с античных времен и возникают в ряде областей науки (физика, астрономия, теоретическая механика, экономика и пр.), и их решению посвящено достаточно большое количество литературы. Задачи дискретной оптимизации получили активное распространение с конца XIX – начала XX века в связи с развитием дискретной прикладной математики и ее составляющих, таких как теория графов, теория игр, исследование операций, математическое целочисленное программирование, теория расписаний. В результате развития соответствующих направлений науки в XVII – XX вв. в работах Готфрида Лейбница (G. Leibniz), Леонарда Эйлера (L. Euler) и др. известных ученых зародилось новое направление, именуемое комбинаторикой, в рамках которого изучаются дискретные объекты, их свойства и отношения между ними. Примерами комбинаторных объектов, изучаемых в рамках соответствующего направления, являются счетные множества и мультимножества, графы и гиперграфы, расписания различного вида (например, расписание занятий школы или университета, расписание движение поездов или самолетов, сетевой график работ и т.д.), логические игры (игральные карты, шахматы, пятнашки и т.д.), различные математические структуры (например, перестановки, группы, латинские квадраты и т.д.).

Как правило, комбинаторные задачи подразумевают организацию перебора того или иного множества решений с выполнением каких либо дополнительных действий, например, подсчета числа решений или выбора одного из них. В некоторых частных случаях они допускают аналитическое решение, например, методами теории вероятностей, математической статистики, теории систем массового обслуживания и т.д. [1]

### Приближенные методы решения задачи о ранце

Как и для большинства NP-полных задач, не всегда необходимо получать точное решение, так как решения, близкие к оптимальным, могут применяться в прикладных задачах. К приближенным методам можно отнести:

1. Жадный алгоритм;
2. Метод муравьиной колоний;
3. Генетический метод.
4. Жадный алгоритм

Жадный алгоритм — алгоритм, заключающийся в принятии локально [оптимальных решений](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5) на каждом этапе, допуская, что конечное решение также окажется оптимальным. Известно, что если структура задачи задается [матроидом](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%BE%D0%B8%D0%B4), тогда применение жадного алгоритма выдаст глобальный оптимум.

Если глобальная оптимальность алгоритма имеет место практически всегда, его обычно предпочитают другим методам оптимизации, таким как [динамическое программирование](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Как и для большинства NP-полных задач, не всегда необходимо получать точное решение, так как решения, близкие к оптимальным, могут применяться в прикладных задачах.

Для решения задачи [жадным алгоритмом](https://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), необходимо отсортировать вещи по их удельной ценности (то есть отношению ценности предмета к его весу), и поместить в рюкзак предметы с наибольшей удельной ценностью.

Время работы данного алгоритма складывается из времени [сортировки](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8) и времени укладки. [Сложность](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) сортировки предметов составляет {\displaystyle O(N\log(N))}. Далее происходит вычисление того, сколько предметов поместится в ранец за общее время {\displaystyle O(N)}. Итоговая сложность {\displaystyle O(N\log(N))} {\displaystyle O(N\log(N))} при необходимости сортировки и {\displaystyle O(N)} при уже отсортированных данных.

Пример. Пусть вместимость рюкзака {\displaystyle W=80}W=80, вес и цена предметов приведены а Таблице 1.3. Предметы уже отсортированы по удельной ценности. Применим жадный алгоритм.

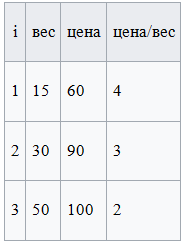


Таблица 1.3 Данные для решения задачи о ранце жадным алгоритмом

Кладём в ранец первый предмет, а за ним второй. Третий предмет в ранец не влезет. Суммарная ценность вещей в ранце равна 150. Если бы были взяты второй и третий предметы, то суммарная ценность составила бы 190. Таким образом, мы получили некоторое неоптимальное решение.

Следует понимать, что жадный алгоритм может привести к ответу сколь угодно далёкому от оптимального. Например, если один предмет имеет вес 1 и стоимость 2, а другой — вес W и стоимость W, то жадный алгоритм наберёт итоговую стоимость 2 при оптимальном ответе W. При этом тот же алгоритм для неограниченной задачи о ранце приведёт к ответу, составляющему не менее 50% от ценности оптимального.

Впервые жадный алгоритм был предложен [Джорджем Данцигом](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D1%86%D0%B8%D0%B3,_%D0%94%D0%B6%D0%BE%D1%80%D0%B4%D0%B6) для решения задачи о неограниченном ранце. [5]

Жадный алгоритм является однокритериальным алгоритмом, поэтому он неприменим для многомерного ранца. Однако следует отметить, что для задачи непрерывного ранца именно этот алгоритм находит наиболее оптимальное решение. [2]

1. Метод муравьиной колонии

Использование комбинации случайного перебора и взвешивающих эвристик обеспечивает на практике ощутимую выгоду, заключающуюся в возможности быстрого получения решений неплохого качества, что не обеспечивается методами на базе случайной или жадной стратегий по отдельности. Однако при итеративном повторении поиска решений опыт прошлых итераций не учитывается, плохие или хорошие решения фактически отбрасываются, запоминается лишь лучшее из них (рекорд). Исправить данную ситуацию можно с использованием метода муравьиной колонии (англ. Ant Colony optimization, сокр. AC), в котором старые решения запоминаются путем специальной пометки ветвей дерева комбинаторного перебора и соответствующих им элементов решения. Данная пометка выражается в использовании специальных дополнительных слагаемых или множителей в составе эвристики, на основе которой принимается решение о выборе направления движения. Метод муравьиной колонии был предложен итальянским ученым Марко Дориго (M. Dorigo) в 1992 г. как результат наблюдения за поведением муравьев в природе, где их целью является нахождение кратчайшего пути между муравейником и источником пищи. [1]

С помощью муравьиных алгоритмов были решены такие задачи комбинаторной оптимизации, как задача коммивояжера, квадратическая задача о назначениях, задача маршрутизации грузовиков, задача календарного планирования, задача раскраски графа, а также задача о ранце.

Сформулируем многомерную задачу о ранце:

Предполагается, что

Основу «социального» поведения муравьев составляет самоорганизация – множество динамических механизмов, обеспечивающих достижение системой глобальной цели в результате низкоуровневого взаимодействия ее элементов. Самоорганизация является результатом взаимодействия следующих четырех компонентов:

- случайность;

- многократность;

- положительная обратная связь;

- отрицательная обратная связь.

Рассмотрим, как реализовать четыре составляющие самоорганизации муравьев при решении задачи о многомерном ранце. Многократность взаимодействия реализуется итерационным поиском оптимального набора предметов в ранце одновременно несколькими муравьями. При этом каждый муравей рассматривается как отдельный, независимый, решающий свою задачу. За одну итерацию алгоритма каждый муравей набирает полный ранец.

Положительная обратная связь реализуется как имитация поведения муравьев типа “оставление следов на предметах”. Для задачи о ранце положительная обратная связь реализуется следующим стохастическим правилом: вероятность положить предмет муравьем в ранец пропорциональна количеству феромона на этом предмете.

Применение такого вероятностного правила обеспечивает реализацию другой составляющей самоорганизации – случайности. Количество откладываемого муравьем феромона на предмете пропорционально его важности.

Использование только положительной обратной связи приводит к преждевременной сходимости решений – к случаю, когда все муравьи выбирают одни и те же предметы. Для избежания этого используется отрицательная обратная связь – испарение феромона. Время испарения не должно быть слишком большим и слишком малым.

Для каждого муравья решение взять предмет зависит от трех составляющих: памяти муравья (tabu list), значимости и виртуального следа феромона.

Tabu list – это список взятых муравьем предметов, которые брать еще раз нельзя. В этот список должны быть включены те предметы, взяв которые мы нарушим одно из ограничений задачи. Обозначим через *Jk*  список предметов, которые муравью *k*  можно взять на итерации *t.*

Значимость(ηj) – это локальная статическая информация, выражающая эвристическое желание муравья взять предмет. Данная мера зависит от некоторых характеристик предмета, таких как масса, объем, цена, и является величиной постоянной для каждой конкретной задачи.

В этом случае мы суммируем коэффициенты по каждому , где – число ограничений задачи.

Виртуальный след феромона на предмете *j* представляет подтверждение муравьиным опытом желание взять предмет *j*. Количество виртуального феромона на предмете *j* на итерации *t* обозначим через j(*t*)*.*

Важную роль в муравьиных алгоритмах играет вероятностно-пропорциональное правило, определяющее вероятность для каждого *k*-го муравья взять предмет *j* на итерации *t* :

где α и β – два регулируемых параметра, задающие все следа феромона и значимость при выборе маршрута.

Вероятности выбора того или иного предмета зависят от трех параметров: значимости предметов, которые являются постоянными величинами в ходе решения задачи, следа феромона, значение которого постоянно обновляется, и tabu list – чем больше предметов в этот список входят, тем больше вероятность у оставшихся предметов быть выбранными.

Способ изменения следа феромона зависит от модели муравьиного алгоритма.

Ant-density – в этой модели количество феромона, оставляемого муравьем на предмете, является постоянной величиной:

Ant-cycle – в этой модели количество феромона, оставляемого муравьем на предмете, зависит от общей ценности этого набора предметов:

где – набор предметов у муравья *k* на итерации  *t*; - ценность этого набора; – регулируемый параметр.

Для исследования всего пространства решений необходимо обеспечить испарение феромона – уменьшение во времени количество отложенного на предыдущих итерациях феромона. Обозначим коэффициент испарения феромона через . Тогда правило обновления феромона примет вид:

где , *K –* количество муравьев в колонии.

В начале работы алгоритма оптимизации количество феромена принимается равным небольшому положительному числу 0. Общее количество муравьев в колонии остается постоянным на протяжении выполнения алгоритма. Число муравьев можно назначить равным произведению числа предметов и числа ограничений.

Последовательность шагов алгоритма муравьиных колоний выглядит следующим образом.

Подготовительные процедуры:

1. Инициализация параметров алгоритма муравьиных колоний: , β, *p* и *q*, *m*, 0 , *N* – число итераций алгоритма;
2. Задание следа феромона 0 на предметах, расчет значимости предметов ηji.

Основной цикл(повторяется *N* раз). Начало:

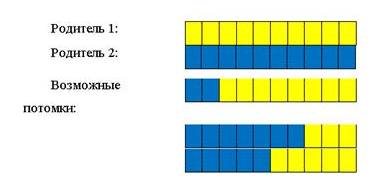
1. Выбор каждым муравьем колонии набора предметов по вероятностно-пропорциональному правилу (1). Муравей набирает предметы, пока его tabu list не будет включать список всех предметов. Как только все муравьи колонии набрали свои ранцы, вычисляем ценность и переходим к шагу 4;
2. Обновление следа феромона на предметах по заранее выбранной схеме: ant-density или ant-cycle. Испарение феромона;
3. Поиск лучшего решения *f*maxитерации, сравнение с наилучшим *f*\* : если *f*max >  *f*\* то *f*\* = *f*max и переходим к шагу 3.

Конец: полученное значение *f*\* является решением задачи.

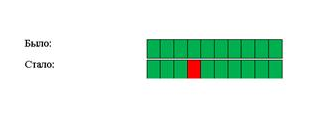
По сравнению с точными методами, например динамическим программированием или методом ветвей и границ, муравьиный алгоритм находит близкие к оптимальному решения за значительно меньшее время даже для задач небольшой размерности (n>30).[7]

1. Генетический метод

Генетический алгоритм ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) genetic algorithm) — это [эвристический алгоритм](https://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D1%80%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных [естественному отбору](https://ru.wikipedia.org/wiki/%D0%95%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BE%D1%82%D0%B1%D0%BE%D1%80) в природе. Является разновидностью [эволюционных вычислений](https://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D0%BE%D0%BB%D1%8E%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как [наследование](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%B1%D0%B8%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F)), [мутации](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D1%82%D0%B0%D1%86%D0%B8%D1%8F), [отбор](https://ru.wikipedia.org/wiki/%D0%95%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BE%D1%82%D0%B1%D0%BE%D1%80) и [кроссинговер](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%B8%D0%BD%D0%B3%D0%BE%D0%B2%D0%B5%D1%80). Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Пример: на Рисунке 1.1 представлен возможные результаты скрещивания.  
  


1. Возможные результаты скрещивания

Мутация — случайное событие, которое случайным образом меняет одно значение на противоположное (учитывая переполнение ранца). На Рисунке 1.2 представлен пример мутации:  
 

1. Пример мутации

Как и для других NP-трудных задач оптимизации, для решения задачи о ранце применяются [генетические алгоритмы](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC). Они не гарантируют нахождения оптимального решения за полиномиальное время и не дают оценку близости решения к оптимальному, но обладают хорошими временными показателями, позволяя найти достаточно хорошее решение быстрее других известных детерминированных или эвристических методов.[5]

Коэффициент полезности — некоторое значение, которое отображает, насколько данный ген приближен к ответу на задачу. В нашем случае это суммарная ценность одного гена (при условии, что ранец не переполняется).

Итак, суть алгоритма в следующем:

1.  Создаём начальную популяцию с помощью ДСЧ.

2.  Вычисляем коэффициент полезности для каждой особи.

3.  Специальным образом выбираем 2 особи для скрещивания (повторяем 5 раз). Специальный способ заключается в следующем, вероятность выбора каждого существа соответствует его коэффициенту полезности. Т. е. чем больше коэффициент полезности, тем выше вероятность того что это существо будет участвовать в скрещивании.

4.  Получаем несколько особей от каждого скрещивания (пять в нашем случае). Потом сортируем всех потомков и у случайных особей происходит мутация. После чего вновь считается коэффициент полезности для всех особей. Собираем всех существ (и родителей, и потомков) сортируем по коэффициенту полезности. Десять первых существ будут основой новой популяции. Далее повторяем со следующей популяцией действия с шага 1.

Подчеркнём некоторые принципы работы по этому методу:

·     На любом шаге следим, чтобы суммарный вес ранца был не больше максимального.

·     Продолжительность алгоритма зависит от ограничения по времени, но чем больше времени выделено на работу алгоритма, тем ближе результат алгоритма будет к искомому.  
[8]

Генетический алгоритм является одним из самых быстрых алгоритмов. Но ограничением этого алгоритма является то, что его хромосомы кодируются дискретно. Соответственно, его нельзя использовать для решения непрерывных и бесконечных задач. Недостатком генетического алгоритма также является то, что он не гарантирует нахождение оптимального решения в некоторых ситуациях (находит локальный экстремум вместо глобального). Происходит это вследствие того, что алгоритм может заканчиваться не только при достижении оптимального решения, но и следующих условиях:

• пройдено максимальное заданное число итераций;

• прошло максимальное время, заданное для выполнения алгоритма;

• при переходе к новому поколению не происходит существенных изменений.[2]

### Точные методы решения задачи о ранце

Задача о ранце относится к классу [NP-полных](https://ru.wikipedia.org/wiki/NP-%D0%BF%D0%BE%D0%BB%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0), и для неё нет [полиномиального алгоритма](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BD%D0%BE%D0%BC%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), решающего её за разумное время. Поэтому при решении задачи о ранце необходимо выбирать между точными алгоритмами, которые неприменимы для «больших» ранцев, и приближенными, которые работают быстро, но не гарантируют оптимального решения задачи.

К точным методам решения задачи о ранце относятся:

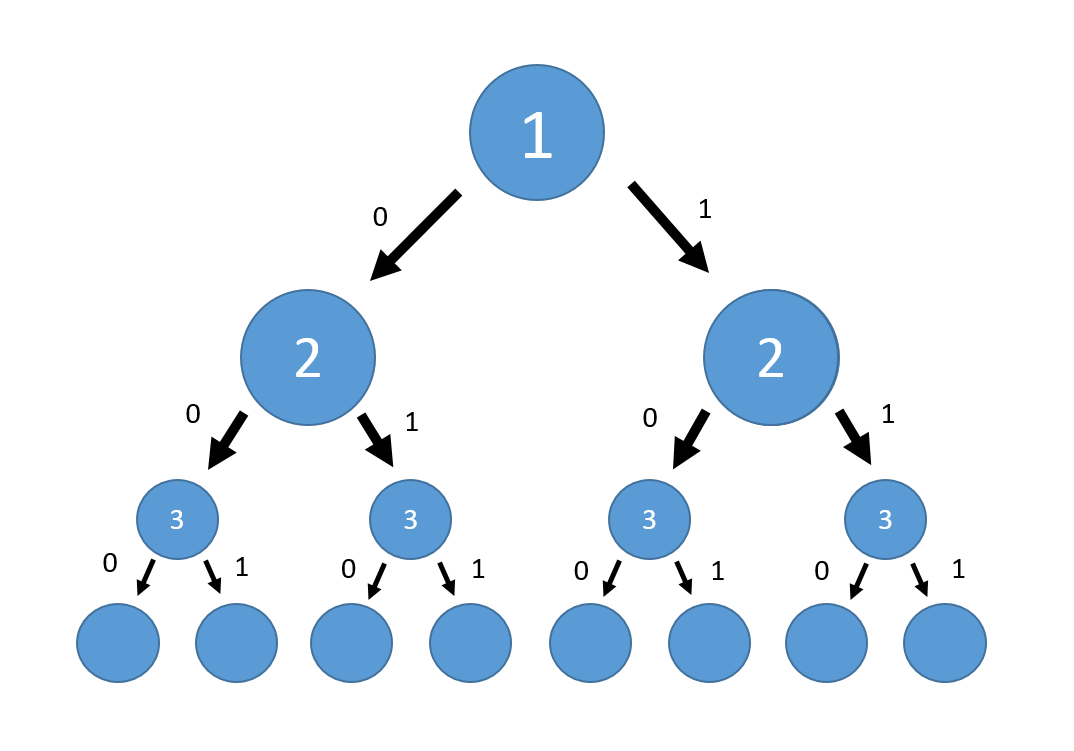
1. Полный перебор;
2. Методы динамического программирования;
3. Метод ветвей и границ.
4. Полный перебор

Как и для других [дискретных задач](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), задачу о ранце можно решить, [полностью перебрав](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D0%BE%D1%80) все возможные решения. Допустим, имеется {\displaystyle N}N предметов, которые можно укладывать в ранец. Нужно определить максимальную стоимость груза, вес которого не превышает {\displaystyle W}W.

Для каждого предмета существует 2 варианта: предмет либо кладётся в ранец, либо нет. Тогда перебор всех возможных вариантов имеет [временную сложность](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) O(2N) ,что позволяет его использовать лишь для небольшого количества предметов. С ростом числа предметов задача становится неразрешимой данным методом за приемлемое время.

На Рисунке 1.3 показано дерево перебора для трёх предметов. Каждый [лист](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)) соответствует некоторому подмножеству предметов. После составления дерева необходимо найти лист с максимальной ценностью среди тех, вес которых не превышает {\displaystyle W}W.

В каждом узле определяется, будет ли данный предмет уложен в ранец. Цифра в узле соответствует номеру предмета. Цифры на рёбрах: 0 означает, что предмет не был взят, 1 — что был. [5]



1. Дерево перебора задачи о ранце для трех предметов

Поскольку метод полного перебора очень трудоемкий, то размер ранца и количество предметов при его использовании должны быть ограничены. Следовательно, этот метод неприменим для неограниченного и непрерывного ранцев. При решении перебором задач мультипликативного и ограниченного ранцев необходимо уменьшить количество предметов до 10, чтобы решение было найдено. [2]

1. Динамическое программирование

Динамическое программирование – один из наиболее мощных методов оптимизации. С задачами принятия рациональных решений, выбора наилучших вариантов, оптимального управления имеют дело специалисты разного профиля. Среди методов оптимизации динамическое программирование занимает особое положение. Этот метод исключительно привлекателен благодаря простоте и ясности своего основного принципа – принципа оптимальности. Сфера приложения принципа оптимальности чрезвычайно широка, круг задач, к которым он может быть применен, до настоящего времени еще полностью не очерчен. Динамическое программирование с самого начала выступает как средство практического решения задач оптимизации.

Словосочетание «динамическое программирование» впервые было использовано в [1940](https://ru.wikipedia.org/wiki/1940-%D0%B5_%D0%B3%D0%BE%D0%B4%D1%8B)-х годах [Р. Беллманом](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D1%87%D0%B0%D1%80%D0%B4_%D0%91%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD) для описания процесса нахождения решения задачи, где ответ на одну задачу может быть получен только после решения задачи, «предшествующей» ей. В [1953](https://ru.wikipedia.org/wiki/1953) г. он уточнил это определение до современного. Первоначально эта область была основана, как [системный анализ](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7) и инжиниринг, которая была признана [IEEE](https://ru.wikipedia.org/wiki/IEEE).

Динамическое программирование представляет собой многошаговый процесс принятия решений, направленных на достижение единой цели. При этом на каждом шаге этого процесса решается задача меньшей размерности, чем исходная. Оптимальная стратегия обладает тем свойством, что независимо от начального состояния и начального решения задачи, последующие решения должны составлять оптимальную стратегию лишь в рассматриваемый момент времени. Иными словами оптимальная стратегия в каждый момент времени определяется лишь состоянием системы, но не ее предысторией.

Главным недостатком метода является, говоря словами Беллмана, «проклятие размерности» – его сложность катастрофически возрастает с увеличением размерности задачи.

1. Метод ветвей и границ

[Метод ветвей и границ](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B2%D0%B5%D1%82%D0%B2%D0%B5%D0%B9_%D0%B8_%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%86) является вариацией метода полного перебора с той разницей, что мы исключаем заведомо неоптимальные ветви дерева полного перебора. Как и метод полного перебора, он позволяет найти оптимальное решение и поэтому относится к точным алгоритмам.

Оригинальный алгоритм, предложенный Питером Колесаром ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Peter Kolesar) в 1967 году, предлагает отсортировать предметы по их удельной стоимости (отношению ценности к весу) и строить дерево [полного перебора](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D0%BE%D1%80). Его улучшение заключается в том, что в процессе построения дерева, для каждого узла мы оцениваем [верхнюю границу](https://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D1%80%D1%85%D0%BD%D1%8F%D1%8F_%D0%B8_%D0%BD%D0%B8%D0%B6%D0%BD%D1%8F%D1%8F_%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D1%8B) ценности решения, и продолжаем строить дерево только для узла с максимальной оценкой. Когда максимальная верхняя граница оказывается в листе дерева, алгоритм заканчивает свою работу.

Способность метода ветвей и границ уменьшать количество вариантов перебора сильно опирается на входные данные. Его целесообразно применять только в том случае, когда удельные ценности предметов отличаются значительно. [5]

Очевидным недостатком алгоритма метода ветвей и границ при решении задач большой размерности является необходимость перебрать слишком большое количество вариантов перед тем, как будет найден оптимальный. Несомненно, метод ветвей повторяет перебор. Однако, благодаря отсечению "бесполезных" решений, может самостоятельно ограничить бесконечное пространство решений. Поэтому его можно применить для неограниченного ранца, но нельзя применить для непрерывного ранца. [2]

### Детерминированный композитный алгоритм

Основная идея композитных алгоритмов – совмещение различных стратегий оценки планов:

* Использование технологий отсечения «плохих» планов, развитых для динамического программирования, применительно к методам типа ветвей и границ.
* Использование технологий оценки качества планов, развитых для методов типа ветвей и границ, применительно к алгоритмам динамического программирования.

Если алгоритмы А и В позволяют получить одинаковые результаты, но затраченные ресурсы (время поиска решения, объем использованной оперативной памяти и т.п.) затраченные алгоритмом А всегда меньше, чем аналогичные затраты ресурсов алгоритмом В на решение тех же задач, то интеллект алгоритма А выше, чем интеллект В.

Композитные алгоритмы обладают более высоким интеллектом, чем «классические» методы поиска решений экстремальных задач с дискретными переменными т.к. они требуют меньше ресурсов (время счета, объем используемой памяти). Сравнительная эффективность композитных алгоритмов возрастает с ростом сложности решаемых задач.

### Заключение

В данной главе были описаны алгоритмы решения задачи о ранце, показаны их достоинства и недостатки, рассказано про работу данных алгоритмов.

Из рассмотренных в первой главе алгоритмов был выбран детерминированный композитный алгоритм, включающий в себя метод ветвей и границ и динамическое программирование, так как этот алгоритм был предложен в задании к выпускной работе.

В третьей главе будет более подробно описан детерминированный композитный алгоритм решения задачи о ранце.

## СОДЕРЖАТЕЛЬНАЯ И ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Ниже рассматриваются задача о ранце с вектором Z булевых переменных вида:

где – константы, причем далее полагаем, что все константы неотрицательны.

Далее верхняя оценка величины *F* при условии, что в базис введено подмножество переменных, значениям которых соответствует вершина дерева поиска, ниже определяется выражением:

где – обозначает множество индексов введенных в базис булевых переменных, значения которых определяются положением на дереве поиска;

– множество индексов всех булевых переменных решаемой задачи.

Аналогично определяется нижняя оценка величины *F* задачи (2), отвечающая вершине дерева поиска:

Еще одной характеристикой вершины дерева поиска является ресурс :

Таким образом, каждой вершине дерева поиска, построенного композитной версией алгоритма, можно поставить в соответствие вектор опираясь на который можно сформулировать правило отсечения подмножеств “неперспективных” векторов переменных. Так, если на дереве поиска существует две вершины , такие, что справедливо хотя бы одно из приводимых ниже условий а, б:

а) (3)

или

б) (4)

то подмножество векторов переменных, отвечающих вершине , можно исключить из дальнейшего рассмотрения.

## выбор и обоснование алгоритма

### Выбор и обоснование алгоритма

В первой главе были разобраны несколько алгоритмов решения задачи о ранце, каждый из которых имеет как и преимущества так и недостатки.

Из рассмотренных алгоритмов был выбран детерминированный композитный алгоритм, так как этот алгоритм был предложен в задании к выпускной работе. Преимущество этого алгоритма перед остальным в том, что его эффективность значительно возрастает с ростом размерности задачи о ранце.

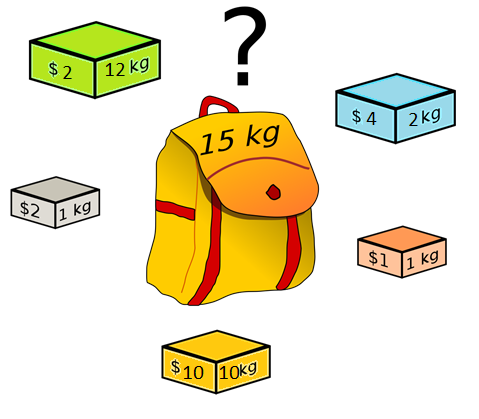
### Детерминированный композитный алгоритм

Существуют детерминированные и рандомизированные алгоритмы. Эти типы алгоритмов являются противоположными друг другу. Детерминированный алгоритм выдает уникальный и предопределенный результат для заданных входных данных, а рандомизированный случайный результат. При решении задачи о ранце, детерминированность проявляется в том, что при локальном переборе мы считаем наилучшей вершиной той, которая удовлетворяет конкретным условиям. В нашем случае это максимальное значение целевой функции относящейся к данной вершине. Преимуществом детерминированных алгоритмов в том, что результат всегда зависит от входных данных и состоянием системы.

Композитные алгоритмы отличаются от классических тем, что они включают в себя совмещение нескольких алгоритмов. Применение композитных алгоритмов может существенно сократить затраты ресурсов (время поиска решения, использование оперативной памяти и т.п.).

При решении задачи о ранце я использовала детерминированный композитный алгоритм, который включает в себя спуск по дереву ветвлений в лучшем направлении и применение динамического программирования для отсечения “плохих” направлений спуска по дереву ветвлений. Далее описан подробно данный алгоритм.

Суть задачи о ранце в том, чтобы уложить как можно большее число ценных вещей в ранец при условии, что вместимость ранца ограничена. На Рисунке 3.1. проиллюстрировано вышеописанное. Вместимость нашего ранца 15 кг и мы можем уложить в него предметы, вес которых в сумме не превышает 15 кг. Но наша цель уложить наиболее ценные вещи, поэтому наиболее выгоднее положить желтый предмет чем зеленый, так как его вес меньше, а цена выше.



1. Иллюстрация сути задачи о ранце

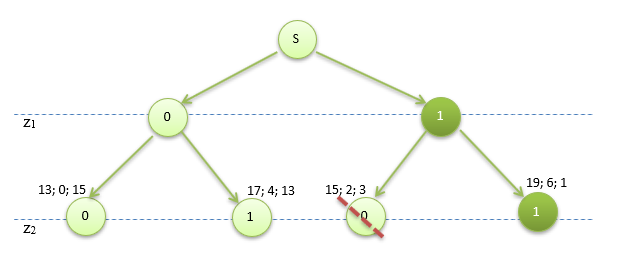
Представим условие задачи о ранце на Рисунке 3.1 в виде формальной постановки.

Будем вводить в базис 2 переменные.

Вводим переменные и и строим дерево со всеми возможными ветвлениями. При каждом ветвлении находим три значения:

1. Верхняя оценка - значение целевой функции. Если известное не равно 0, то значение коэффициента при нем добавляется к сумме коэффициентов неизвестных ;
2. Нижняя оценка – значение целевой функции. Сумма коэффициентов при известных и не равных 0;
3. Ресурс – остаток от ресурса при известных .

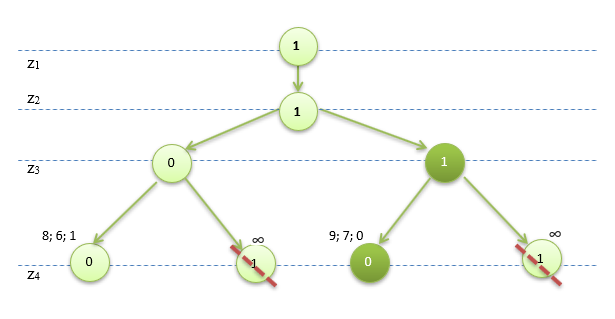
Первая итерация изображена на рисунке 3.2. Мы применяем динамическое программирование и вычеркиваем ветвление {1; 0} так как его нижняя оценка и ресурс меньше нижней оценки и ресурса ветвления {0; 1}. И выбираем наилучшее ветвление – это {1; 1}, так как его нижняя оценка самая наибольшая.



1. Первая итерация решения задачи о ранце детерминированный композитным алгоритмом

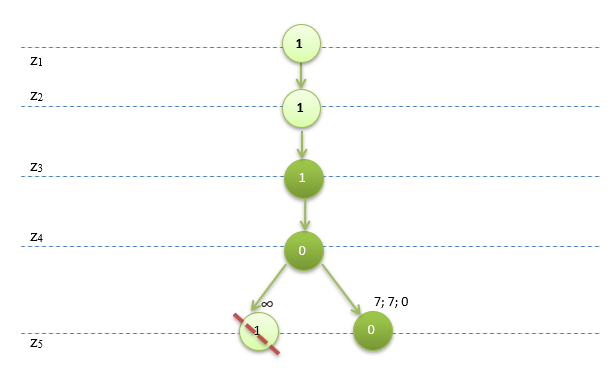
Мы определили, что и . Далее вводим и (Рисунок 3.3).

Ветвления {1; 1; 0; 1} и {1; 1; 1; 1} вычеркиваются, так как они приводят к переполнению ранца. Наилучшее ветвление из оставшихся {1; 1; 1; 0}, так как оно имеет наибольшую нижнюю оценку. Найдены значения и .



1. Вторая итерация решения задачи о ранце детерминированным композитным алгоритмом

Так как количество предметов не кратно количеству переменных введенных в базис, на последней итерации вводится только одна переменная (Рисунок 3.4.). Ветвление {1; 1; 1; 0; 1} приводит к переполнению ранца, поэтому оно вычеркивается и остается только {1; 1; 1; 0; 0}.



1. Третья итерация решения задачи о ранце детерминированным композитным алгоритмом

Мы нашли решение данной задачи:

Возможно при других значениях результат будет лучше, но нам необходимо было найти локально оптимальное решение.

Пошаговое описание детерминированного композитного алгоритма

1 шаг: Вводим сразу h (количество переменных введенных в базис) переменных и строим всевозможные ветвления.

2 шаг: Для каждой висячей вершины *xk* вычисляются компоненты ∆(*xk*), δ(*xk*) и μ(*xk*) и удаляются ветвления, у которых μ(*xk*) выходит за ограничение.

3 шаг: Если на множестве висячих вершин существует вершина *xk*  для которой на том же множестве вершин существует вершина *xq* такая, что справедлива система (3) или (4), то вершина *xk* вычеркивается.

4 шаг: На множестве не вычеркнутых висячих вершин выбирается вершина *xk*  с наибольшей δ(*xk*) компонентой. Если введены не все переменные, то вводим новые h переменных и строим из вершины *xk* всевозможные ветвления и переходим к шагу 2, иначе к шагу 5.

5 шаг: Алгоритм закончен. Вектор переменных Z, соответствующий выбранной на шаге 4 вершине *xk* является решением.

### Классический алгоритм

Для достижения поставленной цели и задач необходимо реализовать классический алгоритм, чтобы сравнить с ним детерминированный композитный алгоритм.

Пошаговое описание классического алгоритма приведено ниже:

1 шаг: Вводим сразу h (количество переменных введенных в базис) переменных и строим всевозможные ветвления.

2 шаг: Для каждой висячей вершины *xk* вычисляются компоненты ∆(*xk*) и μ(*xk*) и удаляются ветвления, у которых μ(*xk*) выходит за ограничение.

3 шаг: На множестве не вычеркнутых висячих вершин выбирается вершина *xk*  с наибольшей ∆(*xk*) компонентой. Если введены не все переменные, то вводим новые h переменных и строим из вершины *xk* всевозможные ветвления и переходим к шагу 2, иначе к шагу 4.

4 шаг: Алгоритм закончен. Вектор переменных Z, соответствующий выбранной на шаге 3 вершине *xk* является решением.

### Заключение

В данной главе была обоснована причина выбора детерминированного композитного алгоритма и подробно рассмотрен сам алгоритм. Данный алгоритм был реализован в программном комплексе. Его структура описана в четвертой главе.

## ПРОГРАММНЫЙ КОМПЛЕКС

1. Выбор ОС и среды разработки

Программный продукт разрабатывался для операционной системы семейства Windows. Основные причины для выбора данной ОС:

* широкая распространенность данной операционной системы среди пользователей персонального компьютера;
* система достаточна проста в использовании;
* для Windows выпускается огромное количество различных приложений и дополнений.

Программный комплекс был написан в среде разработки Visual Studio 2013. Причинами выбора данной среды разработки были:

* Обеспечение единой IDE (интегрированная среда разработки), независимо от языка программирования или типа приложения;
* Поддерживает несколько языков и различные типы проектов;
* Поддерживает отладку из первоначального кода.

1. Язык программирования C#

C# — объектно-ориентированный язык программирования. Он относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

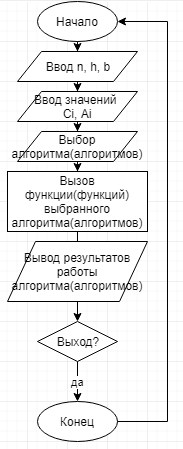
Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Причины выбора языка программирования C#:

* возможность легкого создания интерфейса программы;
* в нем имеется «сборка мусора», обработка исключений, безопасность типов, что улучшает работу программы и повышает безопасность кода;
* простота синтаксиса: код на C# выглядит просто и лаконично;
* удобство отладки.

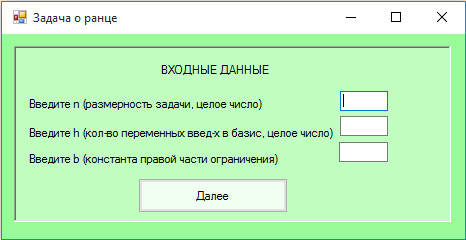
1. Работа программного комплекса

Работа программного комплекса описана следующей блок-схемой:



1. Блок-схема программного комплекса

Главное окно показано на Рисунке 4.2

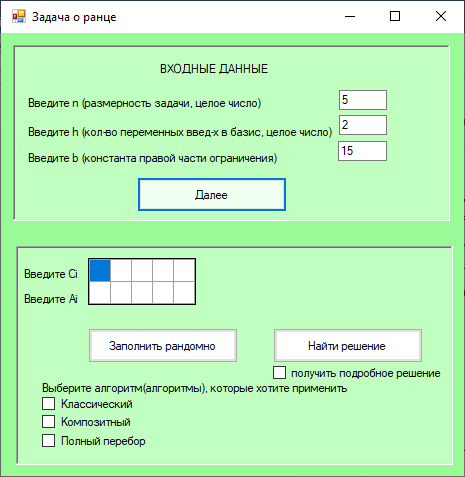


1. Главное окно

В главное окно необходимо ввести входные данные:

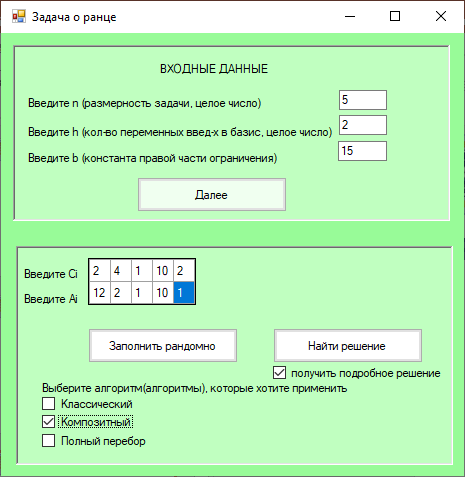
1. n – размерность задачи (обязательно целое положительное число);
2. h – количество переменных введенных в базис (любое целое положительное число, которое не больше n;
3. b – константа правой части ограничения (любое положительное число).

После ввода данных и нажатия на кнопку «Далее» появляется матрица размерностью 2xN (Рисунок 4.3) для ввода значений Ci (коэффициенты при z в целевой функции) и Ai (коэффициенты при z в функции ограничения).



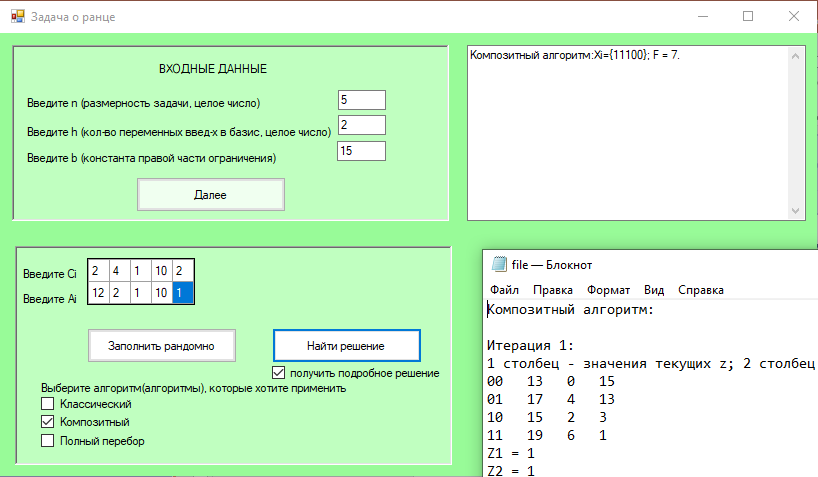
1. Ячейки для коэффициентов

Матрицу можно заполнить вручную или нажать на кнопку «Заполнить рандомно» после чего она будет заполнена случайными положительными числами и можно выбрать алгоритм или алгоритмы, которые хотим применить (классический, композитный, полный перебор) (Рисунок 4.4).

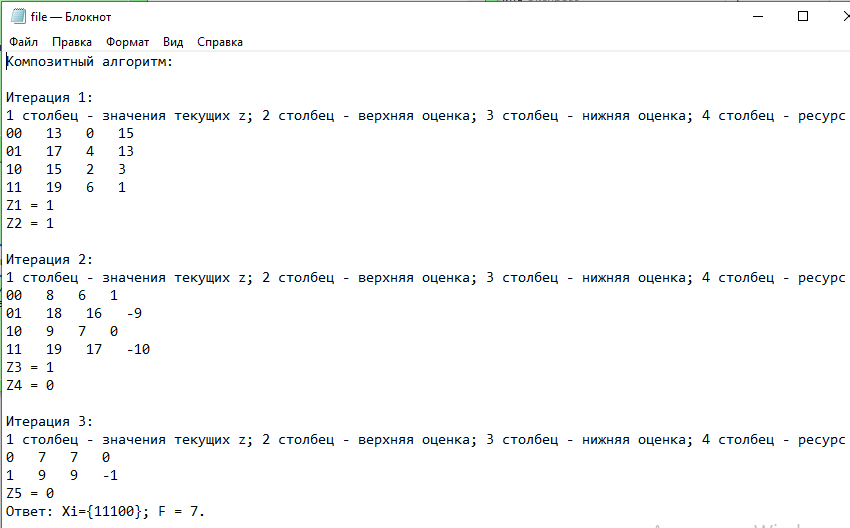


1. Заполнение матрицы

После нажатия на кнопку «Найти решение» выводятся результаты (Рисунок 4.5) решения задачи о ранце выбранным алгоритмом. А если поставить галочку «получить подробное решение», то в текстовом файле будет вывод подробного решения (Рисунок 4.6).



1. Результат работы программы



1. Текстовый файл, с подробным решением

Рассмотрим работу программного комплекса более подробно.

Программный комплекс содержит два класса – VertexComp и VertexSimple;

В функции реализующей классический алгоритм при каждом введении в базис новых переменных создается массив типа VertexSimple размерностью 2h хранящий в себе все вершины. Соответственно в функции реализующей композитный алгоритм создается массив типа VertexComp.

Класс VertexComp состоит из четырех полей:

* public string perestanovka\_0\_1 – значения введенных z;
* public double top\_ocenka – верхняя оценка при данных значениях z;
* public double bottom\_ocenka – нижняя оценка при данных значениях z;
* public double resurs – остаток ресурсов при данных значениях z.

И из двух методов:

* public VertexComp[] Del(int i\_del, VertexComp[] vertex) – удаляет из массива vertex вершину под i\_del-ой позицией;
* public int Search\_Max(VertexComp[] vertex) – ищет позицию вершины с максимальным значением целевой функции в массиве vertex.

А класс VertexSimple из трех полей:

* public string perestanovka\_0\_1 – значения введенных z;
* public double top\_ocenka – верхняя оценка при данных значениях z;
* public double resurs – остаток ресурсов при данных значениях z.

И тоже из двух методов:

* public VertexSimple[] Del(int i\_del, VertexSimple[] vertex) – удаляет из массива vertex вершину под i\_del-ой позицией;
* public int Search\_Max(VertexSimple[] vertex) – ищет позицию вершины с максимальным значением целевой функции в массиве vertex.

### Заключение

В данной главе были рассмотрены причины выбора операционной системы и среды разработки, выбран язык программирования, на котором создавался программный комплекс и была рассмотрена работа программного комплекса.

## Экспериментальная часть

Для проведения сравнительного анализа, проверки адекватности выбранной математической модели и эффективности разработанного программного комплекса была проведена серия экспериментов.

Для проведения экспериментов использовался ПК со следующими характеристиками:

|  |  |
| --- | --- |
| Производитель процессора | Intel |
| Тип процессора | Core i5-2450M |
| Количество ядер процессора | 2 |
| Баз. такт. частота | 2.50 Ггц |
| Кэш память | 3 Мб |
| Оперативная память (RAM) | 16 Гб |
| Тип памяти | DDR3 1066/1333 |
| Производитель видеокарты | NVIDIA |
| Графический контроллер | GeForce GT 620М |
| Жесткий диск (HDD) | 500 ГБ |
| ОС | Windows 10 PRO (64 bit) |

Таблица 5.1 Характеристики ПК

Главной исследуемой характеристикой работы программного комплекса является время его работы.

Для замера времени работы использовался класс Stopwatch. Чтобы измерить время работы участка кода следует перед этим участком вызвать функцию Stopwatch.Start(). В конце участка кода также вызывается функция Stopwatch.Stop(). Время записано в Stopwatch.Elapsed.

1. Эксперимент 1

**Цели эксперимента:**

* Построить графическую зависимость, отражающую среднее время поиска решения классическим алгоритмом в зависимости от n и h;
* Определить аналитическую зависимость, отражающую среднее время поиска решения классическим алгоритмом в зависимости от n и h.

**Ход эксперимента:**

Размерность n решаемых задач меняется в диапазоне 120 ≤ n ≤ 324 с шагом 12. При каждом фиксированном n решается 100 задач о ранце классическим алгоритмом со случайно генерируемыми коэффициентами. Константа правой части ограничения равна половине суммы коэффициентов левой части. Каждая такая задача решается при величинах h, пробегающих значения 1, 2, 3, 4.

Для проведения эксперимента была написана отдельная программа, которая замеряла время выполнения классического алгоритма и выводила среднее время.

Получились следующие данные:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n/h | 1 | 2 | 3 | 4 |
| 120 | 0,0003 | 0,0003 | 0,0004 | 0,0006 |
| 132 | 0,0003 | 0,0003 | 0,0004 | 0,0007 |
| 144 | 0,0004 | 0,0004 | 0,0005 | 0,0009 |
| 156 | 0,0005 | 0,0005 | 0,0006 | 0,001 |
| 168 | 0,0005 | 0,0005 | 0,0007 | 0,0011 |
| 180 | 0,0006 | 0,0006 | 0,0009 | 0,0013 |
| 192 | 0,0008 | 0,0008 | 0,0011 | 0,0016 |
| 204 | 0,0011 | 0,001 | 0,0013 | 0,0022 |
| 216 | 0,0011 | 0,0012 | 0,0015 | 0,0023 |
| 228 | 0,0013 | 0,0013 | 0,0017 | 0,0027 |
| 240 | 0,0014 | 0,0014 | 0,0018 | 0,0028 |
| 252 | 0,0015 | 0,0015 | 0,002 | 0,003 |
| 264 | 0,0016 | 0,0017 | 0,0022 | 0,0034 |
| 276 | 0,0019 | 0,0018 | 0,0024 | 0,0037 |
| 288 | 0,002 | 0,002 | 0,0028 | 0,0041 |
| 300 | 0,0021 | 0,0021 | 0,0029 | 0,0043 |
| 312 | 0,0025 | 0,0024 | 0,0033 | 0,0048 |
| 324 | 0,0025 | 0,0025 | 0,0033 | 0,005 |

Таблица 5.2 Среднее время поиска решения классическим алгоритмом, полученные экспериментально

График 5.1 Зависимость времени работы классического алгоритма от размерности ранца, полученные экспериментально *Ткл*, где n – размерность задачи, h – количество переменных, вводимых в базис

Используя метод наименьших квадратов, аппроксимируем эти данные функцией вида для каждого *h* отдельно, где *x* – это *n* , а *y* – функция времени работы классического алгоритма при данном *n*.

Найдем коэффициенты *а* для всех *h* отдельно, при котором

принимает наименьшее значение.

Находим частную производную функции

по переменной *a* и приравниваем эту производную к нулю.

Для начала найдем значение коэффициента *a* при *h* = 1:

Подставляя в уравнение (5) данные из Таблицы 5.2 получим:

Далее проделываем все тоже самое для остальных *h*.

При *h* = 2:

При *h* = 3:

При *h* = 4:

Получили следующие данные:

|  |  |
| --- | --- |
| h | a |
| 1 | 0,0000000239152 |
| 2 | 0,0000000237629 |
| 3 | 0,0000000318861 |
| 4 | 0,0000000481989 |

Таблица 5.3 Значения коэффициентов а в зависимости от значений h

Так же, методом наименьших квадратов, аппроксимируем эти данные функцией, найденной методом подбора, которая имеет вид , где *x*– это значение *h,* а *y* – значения *a.* Составляется и решается система из двух уравнений с двумя неизвестными.

Находим частные производные функции по переменным  *b* и *c,* приравниваем эти производные к нулю.

Решаем полученную систему методом Крамера, используя данные из Таблицы 5.3.

Найдем значения коэффициентов *b* и *c*, используя формулы Крамера:

Подставив в функцию найденные коэффициенты, получим .

Подставим в вместо коэффициента *а,* подобранную функцию и получим функцию , которая описывает экспериментально полученные данные, с помощью классического алгоритма.

Данные полученные с помощью найденной функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n/h | 1 | 2 | 3 | 4 |
| 120 | 0,000329246 | 0,0003635 | 0,000453 | 0,0006942 |
| 132 | 0,000398388 | 0,0004398 | 0,000548 | 0,00084 |
| 144 | 0,000474115 | 0,0005235 | 0,000652 | 0,0009997 |
| 156 | 0,000556427 | 0,0006143 | 0,000765 | 0,0011732 |
| 168 | 0,000645323 | 0,0007125 | 0,000888 | 0,0013606 |
| 180 | 0,000740805 | 0,0008179 | 0,001019 | 0,001562 |
| 192 | 0,000842871 | 0,0009306 | 0,001159 | 0,0017772 |
| 204 | 0,000951522 | 0,0010505 | 0,001309 | 0,0020063 |
| 216 | 0,001066759 | 0,0011778 | 0,001467 | 0,0022492 |
| 228 | 0,00118858 | 0,0013123 | 0,001635 | 0,0025061 |
| 240 | 0,001316986 | 0,001454 | 0,001811 | 0,0027768 |
| 252 | 0,001451977 | 0,0016031 | 0,001997 | 0,0030615 |
| 264 | 0,001593553 | 0,0017594 | 0,002192 | 0,00336 |
| 276 | 0,001741714 | 0,001923 | 0,002395 | 0,0036724 |
| 288 | 0,00189646 | 0,0020938 | 0,002608 | 0,0039986 |
| 300 | 0,002057791 | 0,0022719 | 0,00283 | 0,0043388 |
| 312 | 0,002225706 | 0,0024573 | 0,003061 | 0,0046928 |
| 324 | 0,002400207 | 0,00265 | 0,003301 | 0,0050608 |

Таблица 5.4 Среднее время поиска решения классическим алгоритмом, полученные аналитически

График 5.2 Зависимость времени работы классического алгоритма от размерности ранца, полученные аналитически, где n – размерность задачи, h – количество переменных вводимых в базис

Существует показатель, характеризующий тесноту линейной связи между *x* и *y*, который называется коэффициент корреляции и рассчитывается по формуле:

А так же средняя ошибка аппроксимации – среднее отклонение полученных значений от фактических:

Для каждого *h* найдем коэффициент корреляции по формуле (8).

Для *h* = 1:

Для *h* = 2:

Для *h* = 3:

Для *h* = 4:

А среднюю ошибку аппроксимации по формуле (9).

*r* для всех *h >* 0,7 , следовательно связь между изучаемыми показателями сильная. А средняя ошибка аппроксимации < 15%, значит хорошо подобрана модель уравнения.

1. Эксперимент 2

**Цели эксперимента:**

* Построить графическую зависимость, отражающую среднее время поиска решения композитным алгоритмом в зависимости от n и h;
* Определить аналитическую зависимость, отражающую среднее время поиска решения композитным алгоритмом в зависимости от n и h.

**Ход эксперимента:**

Размерность n решаемых задач меняется в диапазоне 120 ≤ n ≤ 324 с шагом 12. При каждом фиксированном n решается 100 задач о ранце композитным алгоритмом со случайно генерируемыми коэффициентами. Константа правой части ограничения равна половине суммы коэффициентов левой части. Каждая такая задача решается при величинах h, пробегающих значения 1, 2, 3, 4.

Для проведения эксперимента была написана отдельная программа, которая замеряла время выполнения композитного алгоритма и выводила среднее время.

Получились следующие данные:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n/h | 1 | 2 | 3 | 4 |
| 120 | 0,0003 | 0,0003 | 0,0005 | 0,0008 |
| 132 | 0,0004 | 0,0004 | 0,0006 | 0,001 |
| 144 | 0,0005 | 0,0005 | 0,0007 | 0,0012 |
| 156 | 0,0006 | 0,0006 | 0,0008 | 0,0013 |
| 168 | 0,0006 | 0,0007 | 0,0009 | 0,0016 |
| 180 | 0,0008 | 0,0008 | 0,0011 | 0,0018 |
| 192 | 0,0009 | 0,0009 | 0,0013 | 0,0021 |
| 204 | 0,0012 | 0,0012 | 0,0017 | 0,0029 |
| 216 | 0,0014 | 0,0014 | 0,0019 | 0,0031 |
| 228 | 0,0015 | 0,0015 | 0,0021 | 0,0034 |
| 240 | 0,0016 | 0,0016 | 0,0023 | 0,0036 |
| 252 | 0,0018 | 0,0018 | 0,0025 | 0,0039 |
| 264 | 0,002 | 0,002 | 0,0027 | 0,0043 |
| 276 | 0,0022 | 0,0022 | 0,003 | 0,0047 |
| 288 | 0,0023 | 0,0024 | 0,0033 | 0,0051 |
| 300 | 0,0025 | 0,0025 | 0,0035 | 0,0055 |
| 312 | 0,0029 | 0,003 | 0,004 | 0,0064 |
| 324 | 0,003 | 0,0031 | 0,0041 | 0,0065 |

Таблица 5.5 Среднее время поиска решения композитным алгоритмом, полученные экспериментально

График 5.3 Зависимость времени работы композитного алгоритма от размерности ранца *Тком*, полученные экспериментально, где n – размерность задачи, h – количество переменных вводимых в базис

Так же как и в предыдущем эксперименте используем МНК.

Так как виды функций используются одни и те же, то просто подставим данные из Таблицы 5.5 в уравнение (5).

Получили следующие значения коэффициента *а*:

|  |  |
| --- | --- |
| h | a |
| 1 | 0,0000000282555 |
| 2 | 0,000000028739 |
| 3 | 0,0000000392623 |
| 4 | 0,000000062381 |

Таблица 5.6 Значения коэффициентов а в зависимости от значений h

Так же, методом наименьших квадратов, аппроксимируем эти данные функцией, найденной в предыдущем эксперименте, которая имеет вид .

Находим коэффициенты *b* и *c,* используя формулы (7) и данные из Таблицы 5.6.

Найдем значения коэффициентов *b* и *c*, используя формулы Крамера:

В итоге получили функцию , которая описывает экспериментально полученные данные, с помощью композитного алгоритма.

Данные полученные с помощью найденной функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n/h | 1 | 2 | 3 | 4 |
| 120 | 0,000329247 | 0,000436698 | 0,000561171 | 0,000897563 |
| 132 | 0,000470629 | 0,000528405 | 0,000679016 | 0,001086052 |
| 144 | 0,000560087 | 0,000628845 | 0,000808086 | 0,001292491 |
| 156 | 0,000657324 | 0,00073802 | 0,000948378 | 0,001516882 |
| 168 | 0,000762341 | 0,000855929 | 0,001099894 | 0,001759224 |
| 180 | 0,000875136 | 0,000982571 | 0,001262634 | 0,002019517 |
| 192 | 0,00099571 | 0,001117947 | 0,001436597 | 0,002297762 |
| 204 | 0,001124064 | 0,001262058 | 0,001621783 | 0,002593958 |
| 216 | 0,001260196 | 0,001414902 | 0,001818193 | 0,002908105 |
| 228 | 0,001404107 | 0,001576481 | 0,002025826 | 0,003240204 |
| 240 | 0,001555797 | 0,001746793 | 0,002244682 | 0,003590253 |
| 252 | 0,001715267 | 0,001925839 | 0,002474762 | 0,003958254 |
| 264 | 0,001882515 | 0,002113619 | 0,002716065 | 0,004344206 |
| 276 | 0,002057542 | 0,002310134 | 0,002968592 | 0,00474811 |
| 288 | 0,002240348 | 0,002515382 | 0,003232342 | 0,005169965 |
| 300 | 0,002430933 | 0,002729364 | 0,003507316 | 0,005609771 |
| 312 | 0,002629298 | 0,00295208 | 0,003793513 | 0,006067528 |
| 324 | 0,002835441 | 0,00318353 | 0,004090933 | 0,006543237 |

Таблица 5.7 Среднее время поиска решения композитным алгоритмом, полученные аналитически

График 5.4 Зависимость времени работы композитного алгоритма от размерности ранца, полученные аналитически, где n – размерность задачи, h – количество переменных вводимых в базис

Для каждого *h* найдем коэффициент корреляции по формуле (8).

Для *h* = 1:

Для *h* = 2:

Для *h* = 3:

Для *h* = 4:

А среднюю ошибку аппроксимации по формуле (9).

*r* для всех *h >* 0,7 , следовательно связь между изучаемыми показателями сильная. А средняя ошибка аппроксимации < 15%, значит хорошо подобрана модель уравнения.

1. Эксперимент 3

**Цели эксперимента:**

* Построить графическую зависимость, отражающую сравнительную эффективность классического и композитного алгоритма в зависимости от n и h;
* Определить наилучшее значение h в зависимости от n.

**Ход эксперимента:**

Определим данные найденные по формуле и построим по этим данным график.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n/h | 1 | 2 | 3 | 4 |
| 120 | 1 | 1 | 0,8 | 0,75 |
| 132 | 0,75 | 0,75 | 0,6667 | 0,7 |
| 144 | 0,8 | 0,8 | 0,7143 | 0,75 |
| 156 | 0,8333333 | 0,8333 | 0,75 | 0,7692 |
| 168 | 0,8333333 | 0,7143 | 0,7778 | 0,6875 |
| 180 | 0,75 | 0,75 | 0,8182 | 0,7222 |
| 192 | 0,8888889 | 0,8889 | 0,8462 | 0,7619 |
| 204 | 0,9166667 | 0,8333 | 0,7647 | 0,7586 |
| 216 | 0,7857143 | 0,8571 | 0,7895 | 0,7419 |
| 228 | 0,8666667 | 0,8667 | 0,8095 | 0,7941 |
| 240 | 0,875 | 0,875 | 0,7826 | 0,7778 |
| 252 | 0,8333333 | 0,8333 | 0,8 | 0,7692 |
| 264 | 0,8 | 0,85 | 0,8148 | 0,7907 |
| 276 | 0,8636364 | 0,8182 | 0,8 | 0,7872 |
| 288 | 0,8695652 | 0,8333 | 0,8485 | 0,8039 |
| 300 | 0,84 | 0,84 | 0,8286 | 0,7818 |
| 312 | 0,862069 | 0,8 | 0,825 | 0,75 |
| 324 | 0,8333333 | 0,8065 | 0,8049 | 0,7692 |

Таблица 5.8 Значения

График 5.5 Зависимость отношения от числа переменных n и шага h

Из Таблицы 5.8 и Графика 5.5 видно, что классический алгоритм работает быстрее композитного алгоритма. Это происходит из за того, что в композитном алгоритме добавляется дополнительное действие, удаление «неперспективных» ветвлений, которого нет в классическом, следовательно времени на первый алгоритм требуется больше. При использовании композитного алгоритма был бы выигрыш по времени, если бы мы использовали глобальный перебор, а не локальный.

Из Таблицы 5.2 и Таблицы 5.5 видно, что при увеличении h, время работы классического и композитного алгоритмов увеличивается. Следовательно, рекомендуется решать задачу с использованием минимального h.

1. Эксперимент 4

**Цели эксперимента:**

* Построить графическую зависимость, отражающую вероятности получения наилучшего решения композитным алгоритмом в зависимости от n и h;

**Ход эксперимента:**

Размерность n решаемых задач меняется в диапазоне 4 ≤ n ≤ 24 с шагом 2. При каждом фиксированном n решается 10 задач о ранце композитным алгоритмом и методом полного перебора со случайно генерируемыми коэффициентами. Константа правой части ограничения равна половине суммы коэффициентов левой части. Каждая такая задача решается при величинах h, пробегающих значения 1, 2, 3, 4. Это было сделано для того, чтобы определить вероятности совпадения решения задачи о ранце композитным алгоритмом, с глобально оптимальным решением. Были получены следующие данные.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n\h | 1 | 2 | 3 | 4 |
| 4 | 0,6 | 0,8 | 0 | 0,9 |
| 6 | 0,4 | 0,5 | 0,6 | 0,6 |
| 8 | 0,5 | 0,6 | 0,6 | 0,7 |
| 10 | 0,4 | 0,4 | 0,1 | 0,4 |
| 12 | 0,3 | 0,3 | 0,4 | 0,4 |
| 14 | 0,3 | 0,3 | 0,3 | 0,3 |
| 16 | 0,3 | 0,3 | 0 | 0,4 |
| 18 | 0,4 | 0,4 | 0,4 | 0,4 |
| 20 | 0,3 | 0,4 | 0,4 | 0,4 |
| 22 | 0,5 | 0,5 | 0 | 0,5 |
| 24 | 0,5 | 0,5 | 0,5 | 0,5 |

Таблица 5.9 Вероятности получения наилучшего результата в зависимости от величин n и h

График 5.6 Вероятности получения наилучшего результата в зависимости от величин n и h

Из Таблицы 5.9 и Графика 5.6 видно, что при увеличении h вероятность получения глобально оптимального решения с помощью композитного алгоритма увеличивается.

# ЗАКЛЮЧЕНИЕ

Данная выпускная работа посвящена сравнительной оценки эффективности композитной и традиционной версий поиска локально оптимального решения задачи о ранце, детерминированным спуском по дереву ветвлений в лучшем направлении. В рамках работы был проведен аналитический обзор, была разработана композитная версия алгоритма решения задачи о ранце. Кроме этого разработан программный комплекс, реализующий разработанный алгоритм. Также проведена серия экспериментов, подтверждающая адекватность выбранной математической модели и эффективность программного комплекса.

Поставленную цель считаю достигнутой, а задачи выполненными.

# Список использованных источников

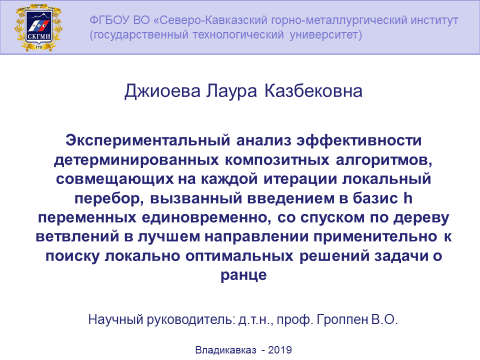
1. *Ватутин Э.И., Титов В.С., Емельянов С.Г.* Основы дискретной комбинаторной оптимизации - М : Аргамак-Медиа, 2016. – 270с.
2. *Додонова* *М.М.* Изучение различных постановок задачи о рюкзаке и методов их решения // Молодежь и наука: сборник материалов Х Юбилейной Всероссийской научно-технической конференции студентов, аспирантов и молодых ученых с международным участием, посвященной 80-летию образования Красноярского края [Электронный ресурс]. — Красноярск: Сибирский федеральный ун-т, 2014.
3. *Silvano Martelo, Paolo Toth*. Knapsack problems. — Wiley, 1990. — 306 с.
4. *David Pisinger.* [Knapsack problems](http://www.diku.dk/users/pisinger/95-1.pdf). — 1995.
5. <https://neerc.ifmo.ru/wiki/index.php?title=Задача_о_рюкзаке>

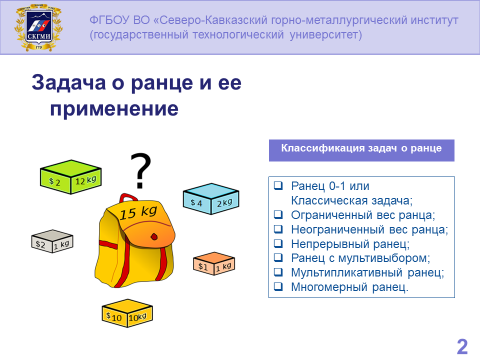
6. <https://ru.wikipedia.org/wiki/NP-полная_задача>

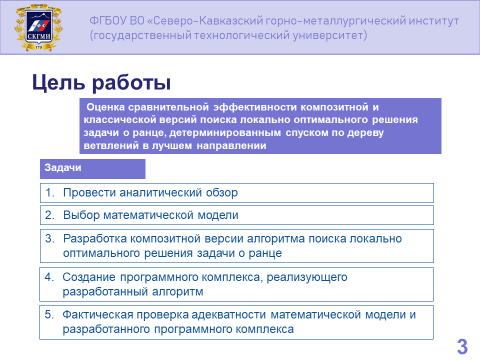
7. *Кагиров Р.Р.* Многомерная задача о рюкзаке: новые методы решения// Математика, механика, информатика. – С.16-20.

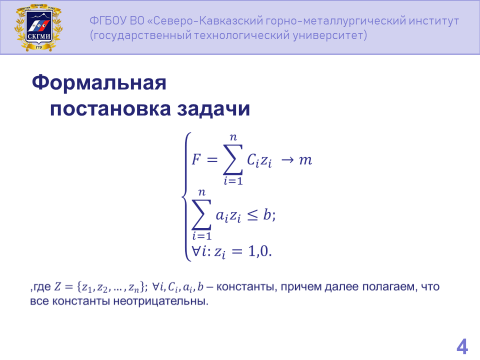
8. *Фалетов К.Э, Воронина Т.В.* Методы решения задачи о рюкзаке// [XX Студенческая международная заочная научно-практическая конференция «Молодежный научный форум: технические и математические науки»](https://nauchforum.ru/studconf/tech/xx) - 2015.

1. Графическая часть



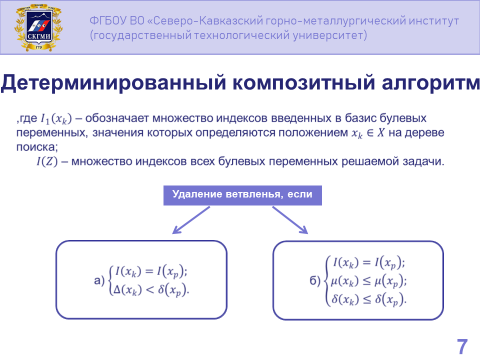


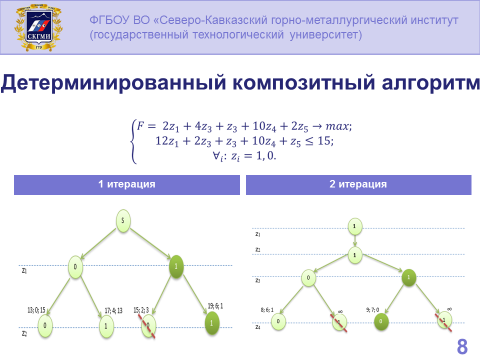




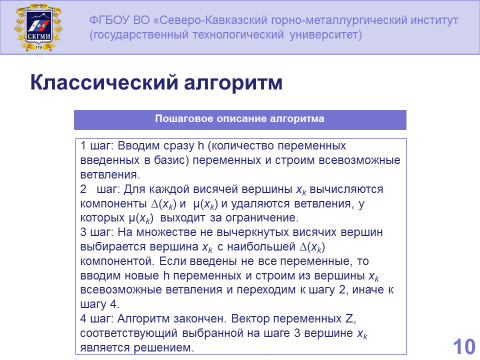


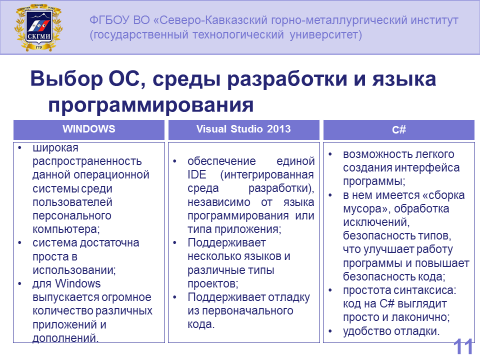


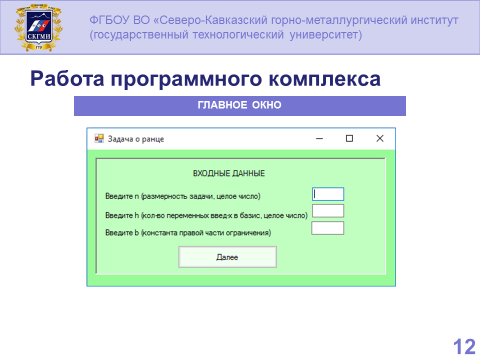


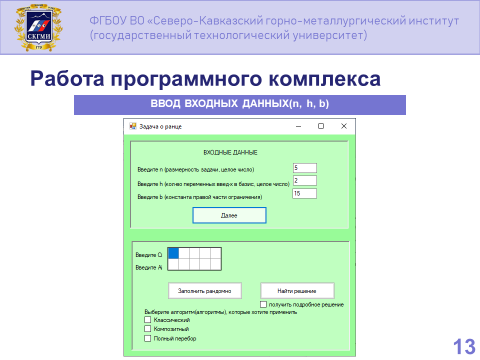


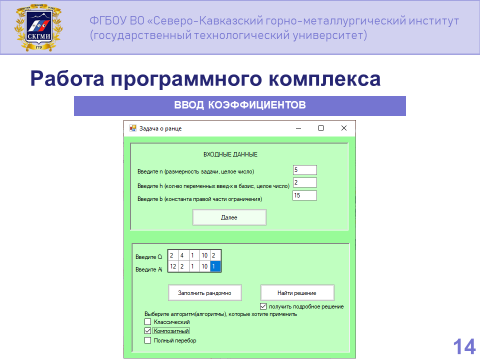


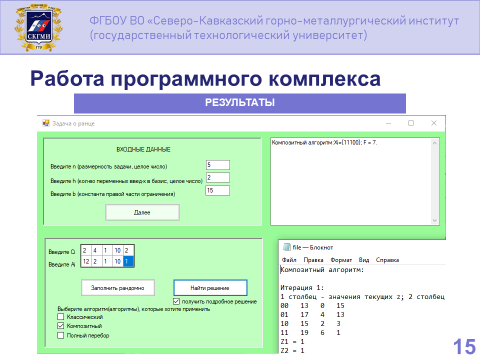


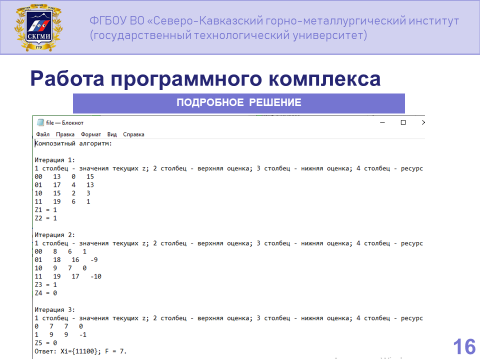




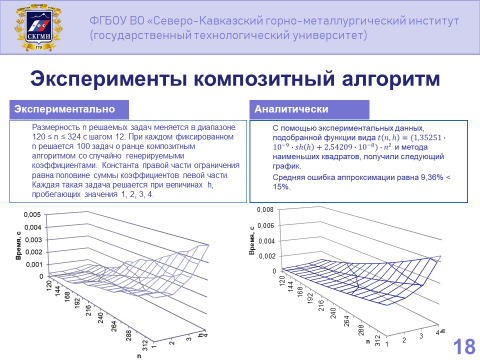






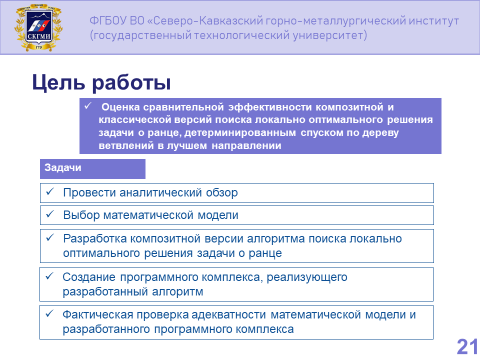


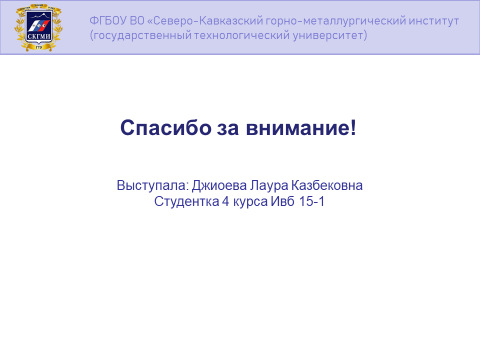






****

****



1. Листинг программы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Diagnostics;

using System.IO;

namespace diplom\_program

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

public struct VertexComp

{

public string perestanovka\_0\_1;

public double top\_ocenka;

public double bottom\_ocenka;

public double resurs;

public VertexComp[] Del(int i\_del, VertexComp[] vertex)

{

VertexComp[] vertex\_del = new VertexComp[vertex.Length - 1];

for (int i = 0, j = 0; i < vertex.Length; i++)

if (i != i\_del)

{

vertex\_del[j] = vertex[i];

j++;

}

return vertex\_del;

}

public int Search\_Max(VertexComp[] vertex)

{

int ind\_max = 0;

double max\_zh = vertex[0].bottom\_ocenka;

for (int i = 1; i < vertex.Length; i++)

if (vertex[i].bottom\_ocenka > max\_zh)

{

max\_zh = vertex[i].bottom\_ocenka;

ind\_max = i;

}

return ind\_max;

}

}

public struct VertexSimple

{

public string perestanovka\_0\_1;

public double top\_ocenka;

public double resurs;

public VertexSimple[] Del(int i\_del, VertexSimple[] vertex)

{

VertexSimple[] vertex\_del = new VertexSimple[vertex.Length - 1];

for (int i = 0, j = 0; i < vertex.Length; i++)

if (i != i\_del)

{

vertex\_del[j] = vertex[i];

j++;

}

return vertex\_del;

}

public int Search\_Max(VertexSimple[] vertex)

{

int ind\_max = 0;

double max\_zh = vertex[0].top\_ocenka;

for (int i = 1; i < vertex.Length; i++)

if (vertex[i].top\_ocenka > max\_zh)

{

max\_zh = vertex[i].top\_ocenka;

ind\_max = i;

}

return ind\_max;

}

}

static string[] Perestanovka(int hh)

{

double n = Math.Pow(2, (double)hh);

string[] mas\_str\_per = new string[(int)n];

int[,] mas\_per = new int[(int)n, hh];

for (int i = 1; i < (int)n; i++)

{

for (int j1 = 0; j1 < hh; j1++)

mas\_per[i, j1] = mas\_per[i - 1, j1];

int j = hh - 1;

while (mas\_per[i, j] != 0)

{

mas\_per[i, j] = 0;

j--;

}

mas\_per[i, j] = 1;

}

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

for (int j = 0; j < hh; j++)

mas\_str\_per[i] += mas\_per[i, j];

return mas\_str\_per;

}

static string Simple\_Algoritm()

{

string otvet="";

string[] mas\_per = new string[(int)Math.Pow(2, h)];

mas\_per = Perestanovka(h);

int index\_st = 0;

int[] mas\_x = new int[n];

int iter = 1;

while (((index\_st + h <= n) && (n % h == 0)) || ((index\_st + h <= n + (n - (n / h) \* h)) && (n % h != 0))) ////Нахождение значений z

{

if (chbFlag)

{

writer.WriteLine();

writer.WriteLine("Итерация " + iter + ":");

iter++;

writer.WriteLine("1 столбец - значения текущих z; 2 столбец - верхняя оценка; 3 столбец - ресурс");

}

if ((n % h != 0) && (index\_st >= (n / h) \* h)) ///// Если количество х-ов в последнем ветвлении меньше чем в предыдущих

{

VertexSimple[] vertex = new VertexSimple[(int)Math.Pow(2, (n % h))];

for (int i = 0, i\_per = 0; i < vertex.Length; i++, i\_per++)

{

mas\_per = new string[(int)Math.Pow(2, (n % h))];

mas\_per = Perestanovka(n - (n / h) \* h);

vertex[i].perestanovka\_0\_1 = mas\_per[i\_per];

vertex[i].resurs = b;

int j = 0;

for (int ind\_z = 0; ind\_z < n; ind\_z++) ////Заполнение значений z при текущей перестановке

{

if ((ind\_z >= index\_st) && (ind\_z < index\_st + h))

{

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

vertex[i].resurs -= mas\_kof\_ogr\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

j++;

}

if (ind\_z < index\_st)

{

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z] \* mas\_x[ind\_z];

vertex[i].resurs -= mas\_kof\_ogr\_f[ind\_z] \* mas\_x[ind\_z];

}

if (ind\_z >= index\_st + h)

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z];

}

if(chbFlag) writer.WriteLine(vertex[i].perestanovka\_0\_1 + " " + vertex[i].top\_ocenka + " " + vertex[i].resurs);

if (vertex[i].resurs < 0)

{

vertex = vertex[i].Del(i, vertex);

i--;

}

}

///search max

int ind\_max = vertex[0].Search\_Max(vertex);

string per\_z = vertex[ind\_max].perestanovka\_0\_1;

for (int i = index\_st, j = 0; i < n; i++, j++)

{

mas\_x[i] = per\_z[j] - '0';

if (chbFlag) writer.WriteLine("Z" + (i + 1) + " = " + mas\_x[i]);

}

}

else

{

VertexSimple[] vertex = new VertexSimple[(int)Math.Pow(2, h)];

for (int i = 0, i\_per = 0; i < vertex.Length; i++, i\_per++)

{

vertex[i].perestanovka\_0\_1 = mas\_per[i\_per];

vertex[i].resurs = b;

int j = 0;

for (int ind\_z = 0; ind\_z < n; ind\_z++) ////Заполнение значений z при текущей перестановке

{

if ((ind\_z >= index\_st) && (ind\_z < index\_st + h))

{

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

vertex[i].resurs -= mas\_kof\_ogr\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

j++;

}

if (ind\_z < index\_st)

{

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z] \* mas\_x[ind\_z];

vertex[i].resurs -= mas\_kof\_ogr\_f[ind\_z] \* mas\_x[ind\_z];

}

if (ind\_z >= index\_st + h)

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z];

}

if (chbFlag) writer.WriteLine(vertex[i].perestanovka\_0\_1 + " " + vertex[i].top\_ocenka + " " + vertex[i].resurs);

if (vertex[i].resurs < 0)

{

vertex = vertex[i].Del(i, vertex);

i--;

}

}

///search max

int ind\_max = vertex[0].Search\_Max(vertex);

string per\_z = vertex[ind\_max].perestanovka\_0\_1;

for (int i = index\_st, j = 0; i < index\_st + h; i++, j++)

{

mas\_x[i] = per\_z[j] - '0';

if (chbFlag) writer.WriteLine("Z" + (i + 1) + " = " + mas\_x[i]);

}

}

index\_st += h;

}

double F = 0;

otvet += "Xi={";

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

{

otvet += mas\_x[i];

F += mas\_x[i] \* mas\_kof\_cel\_f[i];

}

otvet += "}; F = " + F+".";

if (chbFlag) writer.WriteLine("Ответ: " + otvet);

return otvet;

}

static string Compos\_Algoritm()

{

string otvet = "";

string[] mas\_per = new string[(int)Math.Pow(2, h)];

mas\_per = Perestanovka(h);

int index\_st = 0;

int[] mas\_x = new int[n];

int iter = 1;

while (((index\_st + h <= n) && (n % h == 0)) || ((index\_st + h <= n + (n - (n / h) \* h)) && (n % h != 0))) ////Нахождение значений z

{

if (chbFlag)

{

writer.WriteLine();

writer.WriteLine("Итерация " + iter + ":");

iter++;

writer.WriteLine("1 столбец - значения текущих z; 2 столбец - верхняя оценка; 3 столбец - нижняя оценка; 4 столбец - ресурс");

}

if ((n % h != 0) && (index\_st >= (n / h) \* h)) ///// Если количество х-ов в последнем ветвлении меньше чем в предыдущих

{

VertexComp[] vertex = new VertexComp[(int)Math.Pow(2, (n % h))];

for (int i = 0, i\_per = 0; i < vertex.Length; i++, i\_per++)

{

mas\_per = new string[(int)Math.Pow(2, (n % h))];

mas\_per = Perestanovka(n - (n / h) \* h);

vertex[i].perestanovka\_0\_1 = mas\_per[i\_per];

vertex[i].resurs = b;

int j = 0;

for (int ind\_z = 0; ind\_z < n; ind\_z++) ////Заполнение значений z при текущей перестановке

{

if ((ind\_z >= index\_st) && (ind\_z < index\_st + h))

{

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

vertex[i].bottom\_ocenka += mas\_kof\_cel\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

vertex[i].resurs -= mas\_kof\_ogr\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

j++;

}

if (ind\_z < index\_st)

{

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z] \* mas\_x[ind\_z];

vertex[i].bottom\_ocenka += mas\_kof\_cel\_f[ind\_z] \* mas\_x[ind\_z];

vertex[i].resurs -= mas\_kof\_ogr\_f[ind\_z] \* mas\_x[ind\_z];

}

if(ind\_z>=index\_st+h)

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z];

}

if (chbFlag) writer.WriteLine(vertex[i].perestanovka\_0\_1 + " " + vertex[i].top\_ocenka + " " + vertex[i].bottom\_ocenka+" "+ vertex[i].resurs);

if (vertex[i].resurs < 0)

{

vertex = vertex[i].Del(i, vertex);

i--;

}

}

///del din

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

for (int j = 0; j < vertex.Length; j++)

if (i != j)

if ((vertex[i].bottom\_ocenka <= vertex[j].bottom\_ocenka) && (vertex[i].resurs <= vertex[j].resurs)||(vertex[i].top\_ocenka<vertex[j].bottom\_ocenka))

{

vertex = vertex[i].Del(i, vertex);

i--;

break;

}

///search max

int ind\_max = vertex[0].Search\_Max(vertex);

string per\_z = vertex[ind\_max].perestanovka\_0\_1;

for (int i = index\_st, j = 0; i < n; i++, j++)

{

mas\_x[i] = per\_z[j] - '0';

if (chbFlag) writer.WriteLine("Z" + (i + 1) + " = " + mas\_x[i]);

}

}

else

{

VertexComp[] vertex = new VertexComp[(int)Math.Pow(2, h)];

for (int i = 0, i\_per = 0; i < vertex.Length; i++, i\_per++)

{

vertex[i].perestanovka\_0\_1 = mas\_per[i\_per];

vertex[i].resurs = b;

int j = 0;

for (int ind\_z = 0; ind\_z < n; ind\_z++) ////Заполнение значений z при текущей перестановке

{

if ((ind\_z >= index\_st) && (ind\_z < index\_st + h))

{

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

vertex[i].bottom\_ocenka += mas\_kof\_cel\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

vertex[i].resurs -= mas\_kof\_ogr\_f[ind\_z] \* (vertex[i].perestanovka\_0\_1[j] - '0');

j++;

}

if (ind\_z < index\_st)

{

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z] \* mas\_x[ind\_z];

vertex[i].bottom\_ocenka += mas\_kof\_cel\_f[ind\_z] \* mas\_x[ind\_z];

vertex[i].resurs -= mas\_kof\_ogr\_f[ind\_z] \* mas\_x[ind\_z];

}

if(ind\_z>=index\_st+h)

vertex[i].top\_ocenka += mas\_kof\_cel\_f[ind\_z];

}

if (chbFlag) writer.WriteLine(vertex[i].perestanovka\_0\_1 + " " + vertex[i].top\_ocenka + " " + vertex[i].bottom\_ocenka + " " + vertex[i].resurs);

if (vertex[i].resurs < 0)

{

vertex = vertex[i].Del(i, vertex);

i--;

}

}

///del din

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

for (int j = 0; j < vertex.Length; j++)

if (i != j)

if ((vertex[i].bottom\_ocenka <= vertex[j].bottom\_ocenka) && (vertex[i].resurs <= vertex[j].resurs) || (vertex[i].top\_ocenka < vertex[j].bottom\_ocenka))

{

vertex = vertex[i].Del(i, vertex);

i--;

break;

}

///search max

int ind\_max = vertex[0].Search\_Max(vertex);

string per\_z = vertex[ind\_max].perestanovka\_0\_1;

for (int i = index\_st, j = 0; i < index\_st + h; i++, j++)

{

mas\_x[i] = per\_z[j] - '0';

if (chbFlag) writer.WriteLine("Z" + (i + 1) + " = " + mas\_x[i]);

}

}

index\_st += h;

}

double F = 0;

otvet += "Xi={";

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

{

otvet += mas\_x[i];

F += mas\_x[i] \* mas\_kof\_cel\_f[i];

}

otvet += "}; F = " + F + ".";

if (chbFlag) writer.WriteLine("Ответ: " + otvet);

return otvet;

}

static int n, h, b;

static double[] mas\_kof\_cel\_f;

static double[] mas\_kof\_ogr\_f;

static FileStream file;

static StreamWriter writer;

static bool chbFlag;

private void btnOtvet\_Click(object sender, EventArgs e)

{

chbFlag = false;

if (chbPodr.Checked) chbFlag = true;

tbOtvet.Visible = true;

n = Convert.ToInt32(tbN.Text);

h = Convert.ToInt32(tbH.Text);

b = Convert.ToInt32(tbB.Text);

mas\_kof\_cel\_f = new double[n];

mas\_kof\_ogr\_f = new double[n];

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

mas\_kof\_cel\_f[i] = Convert.ToDouble(dtgvZnX[i, 0].Value);

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

mas\_kof\_ogr\_f[i] = Convert.ToDouble(dtgvZnX[i, 1].Value);

tbOtvet.Text = "";

file = new FileStream("C:/Users/Public/file.txt", FileMode.Create);

writer = new StreamWriter(file);

if (chbFlag) writer.WriteLine("Классический алгоритм:");

tbOtvet.Text += "Классический алгоритм:" + Simple\_Algoritm() + Environment.NewLine;

if (chbFlag) writer.WriteLine();

if (chbFlag) writer.WriteLine("Композитный алгоритм:");

tbOtvet.Text += "Композитный алгоритм:" + Compos\_Algoritm();

writer.Close();

if (chbFlag) Process.Start("C:/Users/Public/file.txt");

}

private void btnStart\_Click(object sender, EventArgs e)

{

if ((Convert.ToDouble(tbN.Text) % 1 != 0) || (Convert.ToDouble(tbH.Text) % 1 != 0))

{

MessageBox.Show("Данные введены некоректно!");

tbN.Text = "";

tbH.Text = "";

}

else

{

chbPodr.Visible = true;

pnlEnterKonst.Visible = true;

n = Convert.ToInt32(tbN.Text);

h = Convert.ToInt32(tbH.Text);

b = Convert.ToInt32(tbB.Text);

dtgvZnX.ColumnCount = n;

dtgvZnX.RowCount = 2;

dtgvZnX.Size = new Size(30 \* n, 47);

}

}

private void btnRandom\_Click(object sender, EventArgs e)

{

Random rnd = new Random();

n = Convert.ToInt32(tbN.Text);

for (int i = 0; i < 2; i++)

for (int j = 0; j < n; j++)

dtgvZnX[j, i].Value = rnd.Next(0, 50);

}

}

}

Билет № **1**