**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФГБОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ГОРНО-МЕТАЛЛУРГИЧЕСКИЙ ИНСТИТУТ**

**(ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)»**

**Факультет** Информационных технологий

**Кафедра** Автоматизированной обработки информации

**Направление подготовки** Информатика и вычислительная техника

**Профиль** Автоматизированные системы управления

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К ВЫПУСКНОЙ РАБОТЕ

**на тему:** Экспериментальный анализ эффективности композитной версии методов типа ветвей и границ, осуществляющей, фронтальный спуск по дереву ветвлений применительно к задачам с не булевыми переменными.

**Студент** Тобоев Леонид Олегович

**Руководитель проекта** проф. Гроппен Виталий Оскарович

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

**Заведующий кафедрой** проф. Гроппен Виталий Оскарович

**г. Владикавказ 2018 г.**

**РЕФЕРАТ**

Пояснительная записка: 51 стр., 10 рис., 2 гр., 6 ист., 1 прил.

Метод ветвей и границ, Метод динамического программирования, МВГ, МДП, Композитные алгоритмы, фронтальный спуск по дереву ветвлений, вычисление границ, оптимизационный алгоритм, Delphi 7.

Объект разработки: Прикладное программное обеспечение, представляющее решение задач с не булевыми переменными классической и композитной версий методов типа ветвей и границ, осуществляющей фронтальный спуск по дереву ветвлений.

Цель: Экспериментальный анализ эффективности композитной версии методов типа ветвей и границ, осуществляющей, фронтальный спуск по дереву ветвлений.

Использованное прикладное и системное программное обеспечение:

* Операционная система Microsoft 10 64-bit;
* Среда разработки Microsoft Visual Studio 2017.

Типы используемых вычислительных средств:

при написании дипломного проекта использовался персональный компьютер со следующими характеристиками:

AMD FX(tm)-4100 Quad-Core Processor 3,6 GHz, NVIDIA GeForce GTX 650, ОЗУ 8, ОС Windows 10 – 64 bit.

Полученные результаты: В ходе этой выпускной работы было разработано прикладное программное обеспечение, был проведен теоретический анализ сравнительной эффективности все алгоритмов, а затем была проведена серия экспериментов.

**Дата выдачи задания:** «29» декабря 2017 г.

**Научный руководитель**

Проф., Зав. кафедры АОИ Гроппен Виталий Оскарович

**Задание принял к исполнению** Тобоев Л. О.

СОДЕРЖАНИЕ

[Введение 5](#_Toc515463904)

[Глава 1. Аналитический обзор 7](#_Toc515463905)

**1.1 Метод типа ветвей и границ**

**1.1.1 Идея методов типа ветвей и границ**………………………………………………………………………………7

**1.1.2 Идея фронтального спуска по дереву ветвлений**…………………………………………………………7

**1.1.3** **Содержательное описание алгоритма**………………..…………………………………………………………7

**1.1.4 Достоинства и недостатки фронтального спуска по дереву ветвлений**……………………..7

**1.2 Метод динамического программирования**……………………………………………………………………………..8

**1.2.1 Идея динамического программирования**…………………………………………………………………….8

**1.2.2 Достоинства и недостатки метода динамического программирования**

[Глава 2. Композитная реализация метода типа ветвей и границ 21](#_Toc515463919)

[**2.1** **Формальная постановка задачи** 21](#_Toc515463920)

[**2.2** **Алгоритм** 22](#_Toc515463921)

[**2.3** **Пример решения задачи вручную** 22](#_Toc515463922)

[**2.4** **Пример решения задачи с помощью программы** 26](#_Toc515463923)

[Глава 3. Выбор и обоснование программной реализации 27](#_Toc515463924)

[**3.1** **C Sharp C#** 27](#_Toc515463925)

[**3.1.1** **Описание языка** 27](#_Toc515463926)

[**3.1.2** **История развития языка** 27](#_Toc515463929)

[**3.1.3** **Синтаксис языка** 31](#_Toc515463930)

[**3.2** **Пошаговый алгоритм работы программы** 34](#_Toc515463933)

[Глава 4. Результаты эксперимента 35](#_Toc515463934)

[Заключение 37](#_Toc515463935)

[Список литературы 38](#_Toc515463938)

[Приложение 1. Листинг программы 39](#_Toc515463939)

[Приложение 2. Руководство пользователя 62](#_Toc515463940)

[Приложение 3. Графический материал 64](#_Toc515463941)

# Введение

Рост числа различных приложений систем искусственного интеллекта повышает требования к используемым в рамках этой технологии оптимизационным алгоритмам. Это особенно важно применительно к задачам дискретной оптимизации, для которых отсутствуют эффективные процедуры, гарантирующие в общем случае глобально оптимальное решение. К классическим алгоритмам поиска глобально оптимальных решений экстремальных задач дискретной оптимизации можно отнести методы, развитые во второй половине прошлого века: полный перебор, методы типа ветвей и границ и динамическое программирование. До последнего времени повышение их интеллекта было связано с распараллеливанием вычислений и совершенствованием способов вычисления оценок, адаптируя их применительно к специфике решаемых задач. Ниже анализируется интеллект композитных алгоритмов, ориентированных на решение задач дискретной оптимизации с булевыми переменными. По аналогии с композитными материалами, под композитными алгоритмами ниже понимаются процедуры, в которых присутствуют компоненты не менее двух классических алгоритмов. Так, ниже процедура выбора направления спуска по дереву ветвлений в методах типа ветвей и границ дополняется технологией отсечения «плохих» планов, развитой в динамическом программировании, а процедуры отсечения, используемые в динамическом программировании, расширяются за счет применения оценок, присущих методам типа ветвей и границ.

# Глава 1. Аналитический обзор

# 1.1 Метод типа ветвей и границ

# Метод, известный как метод ветвей и границ, исследует древовидную модель пространства решений, и применим для широкого круга дискретных комбинаторных задач. Алгоритмы ветвей и границ ориентированы в большей степени на оптимизацию.

# В этом методе определяется некая функция, как весовая числовая функция для различных вариантов решения задачи и вычисляется численные значения этой функции для различных вариантов решения. Цель — найти конфигурацию, на которой эта функция достигает максимального или минимального значения. Сам же метод отыскания и построения такой функции получил название метод ветвей и границ. Такое название возникло в результате того, что в нем используется ветвление и вычисляются границы.

# Ветвление - это процесс разбиения всех вариантов решения задачи на множества, таким образом, что какая-либо конкретная подзадача принадлежит только одному из этих множеств. В этом случае говорят, что ветвление происходит на этой подзадаче. Если пользоваться терминологией древовидной структуры, то говорят, что ветвление происходит на определенной ветви древовидной структуры.

# 1.1.1 Идея методов типа ветвей и границ

# Все множества планов решаемой задачи разбиваются на ряд подмножеств.

# Для планов каждого подмножества вычисляется наилучшая оценка.

# На основании оценок отбрасываются те подмножества планов, которые заведомо не могут содержать наилучшего решения, а оставшиеся исследуются.

# 1.1.2 Идея фронтального спуска по дереву ветвлений

# Три основных шага построения дерева ветвлений фронтальным спуском:

# - На множестве висячих вершин построенной части дерева выбирается вершина с наилучшей оценкой.

# - Ветвление осуществляется из вершины, выбранной на предыдущем шаге.

# - Если выбранной вершине отвечает случай, когда в базис введены все переменные, то алгоритм закончен – оптимальный план найден.

# 1.1.3 Содержательное описание алгоритма

# Шаг 1. На построенной части дерева ветвлений выбирается вершина с наилучшей оценкой, принадлежащая i-у ярусу.

# Шаг 2. Если i=n, где n – число переменных, то перейти к шагу 4, в противном случае – к шагу 3.

# Шаг 3. В базис частичного плана, соответствующего выбранной вершине, вводится (i+1)-я переменная и вычисляются соответствующие оценки. Перейти к шагу 1.

# Шаг 4. Конец алгоритма. Оценки выбранной на предыдущем шаге вершины являются оптимальным значением целевой функции.

# 1.1.4 Достоинства и недостатки фронтального спуска по дереву ветвлений

# Достоинства: шанс на неполный перебор, первый же полный допустимый план является глобально оптимальным.

# Недостатки: по мере спуска по дереву ветвлений растут:

# число оценок, хранимых в памяти;

# 2) затраты времени на их сравнение при выборе направления спуска.

# 1.1.5 Алгоритм

# 1. На множестве висячих вершин построенной части дерева ветвлений G(X,U) где Х – множество вершин, U – множество дуг, выбирается вершина xj с наилучшей оценкой. Если это осуществляется на первой итерации, то такой вершиной a priori считается корневая вершина дерева.

# 2. Если выбранной вершине отвечает равенство I1= I (I- множество индексов всех переменных), то перейти к шагу 5, в противном случае – к следующему шагу.

# 3. Ветвление осуществляется из выбранной на шаге 1 последней итерации вершины xj. Новое множество висячих вершин дерева вновь обозначаем Х1.

# 4. Вычисляются оценки, отвечающие висячим вершинам построенного на предыдущем шаге «куста». Для этого можно воспользоваться:

# Δ(xk) = ,

# если целевая функция системы является максимизируемой и

# δ(xk) =

# – если целевая функция минимизируется. Перейти к шагу 1.

# 5. Алгоритм закончен. Вектор переменных, соответствующий выбранной вершине, является оптимальным.

# Пример 1. Решение задачи с не булевыми переменными методом фронтального спуска по дереву ветвлений.

# Пусть задана задача вида:

# (3)

# 0 1 2 x1

# x2

# 

# x3

# x4

# x5

# Xopt = {1,0,0,1,0};

# R = 9;

# Висячих вершин – 22.

# 1.2 Динамическое программирование

# Метод динамического программирования это способ решения сложных задач путем разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. В таком случае время вычисления, по сравнению с «наивными» методами, можно значительно сократить.

# 1.2.1 Идея динамического программирования

# Ключевая идея в динамическом программировании достаточно проста. Как правило, чтобы решить поставленную задачу, требуется решить отдельные части задачи (подзадачи), после чего объединить решения этих подзадач в одно общее решение. Часто многие из подзадач одинаковы. Подход динамического программирования состоит в том, чтобы решить каждую подзадачу только один раз, сократив тем самым количество вычислений. Это особенно полезно в случаях, когда число повторяющихся подзадач экспоненциально велико.

# 1.2.2 Достоинства и недостатки метода динамического программирования

# Динамическое программирование можно использовать как для решения задач, связанных с динамикой процесса или системы, так и для статических задач, связанных, например, с распределением ресурсов. Это значительно расширяет область применения динамического программирования для решения задач управления. А возможность упрощения процесса решения, которое достигается за счет ограничения области и количества, исследуемых при переходе к очередному этапу вариантов, увеличивает достоинства этого комплекса методов.

# Вместе с тем динамическому программированию свойственны и недостатки. Прежде всего, в нем нет единого универсального метода решения. Практически каждая задача, решаемая этим методом, характеризуется своими особенностями и требует проведения поиска наиболее приемлемой совокупности методов для ее решения. Кроме того, большие объемы и трудоемкость решения многошаговых задач, имеющих множество состояний, приводят к необходимости отбора задач малой размерности либо использования сжатой информации.

# 1.2.3 Алгоритм

# 1. На множестве висячих и не помеченных вершин построенной части дерева ветвлений G(X,U) выбирается вершина xj. Если таковой вершины нет, то перейти к шагу 4. Если выбор осуществляется на первой итерации, то такой вершиной a priori считается корневая вершина дерева. 2. Осуществляется ветвление из выбранной на шаге 1 последней итерации вершины xj и вычисляются компоненты каждого вектора R(xk), отвечающего всем висячим вершинам построенного «куста» (величина h=1). Для определения первой компоненты этого вектора используется δ(xk) = , остальные компоненты вычисляются по формуле:

# rj(xk) =

# где xk – вершина, принадлежащая множеству висячих вершин куста, в которую заходит дуга из выбранной на шаге 1 последней итерации вершины xj.

# 3. Все вершины куста, построенного на шаге 2 последней итерации, помечаются, перейти к шагу 1. 4. Убираются все пометки подмножества висячих вершин дерева ветвлений.

# 

# 5. Если на множестве висячих вершин существует вершина xk, для которой справедливо хотя бы одно из условий:

# а) I1(xk) = I1(xq);

# б) δ(xk) «хуже» чем δ(xq);

# в) rj(xq) ≥ rj(xk),

# то эта вершина вычеркивается.

# 6. Если число введенных в базис переменных висячих вершин дерева ветвлений равно числу переменных решаемой задачи, то перейти к следующему шагу, в противном случае перейти к шагу 1. 7. На множестве не вычеркнутых висячих вершин выбирается вершина xk с наилучшей первой компонентой соответствующего ей вектора R(xk). 8. Алгоритм закончен. Вектор переменных, соответствующий выбранной на шаге 7 вершине xk, является оптимальным.

# Пример 2. Решение задачи с не булевыми переменными методом динамического программирования

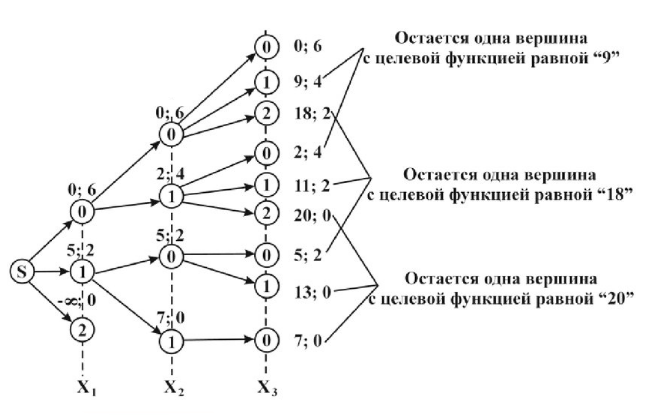
# Пусть задана задача вида:

# (4)

# Первые две итерации:



# Третья итерация:



# Четвертая итерация:

# 

# Глава 2. Композитная реализация метода типа ветвей и границ с не булевыми переменными

## **Формальная постановка задачи**

Далее приводятся комбинация методов типа ветвей и границ, методика отсечения в которых включает технологии, используемые в методе динамического программировании, а также модификации алгоритма, реализующего динамическое программирование с привлечением оценок, используемых методами типа ветвей и границ.

где: Сi и bi,j – коэффициенты, aj- константы, причем полагаем, что Сi ≥0.

## **2.2 Алгоритм**

1. На множестве не вычеркнутых висячих вершин построенной части дерева ветвлений G(X,U) выбирается вершина xj с «наилучшей» первой компонентой вектора V(xj). Если это осуществляется на первой итерации, то такой вершиной a priori считается корневая вершина дерева.

2. Если выбранной вершине отвечает равенство I1 = I, то перейти к шагу 8, в противном случае – к следующему шагу.

3. Ветвление осуществляется из выбранной на шаге 1 последней итерации вершины xj. Новое множество висячих вершин дерева вновь обозначаем

4. Для каждой висячей вершины построенного на предыдущем шаге «куста» xj формируется вектор V(xj). Для формирования первой компоненты этого вектора используется процедура, применявшаяся для вычисления соответствующей оценки в методе типа ветвей и границ, для определения второй компоненты вектора V(xj) используется (5), остальные компоненты вычисляются по формуле (6).

5. Если на множестве висячих вершин существует вершина xk, для которой справедливо хотя бы одно из условий:



а) I1(xk) = I1(xq);

б) δ(xk) «хуже» чем δ(xq);

в) rj(xq) ≥ rj(xk),

то эта вершина вычеркивается.

6. Если существуют вершины и , для которых справедливо:

а) I1(xj) = I1(xq);

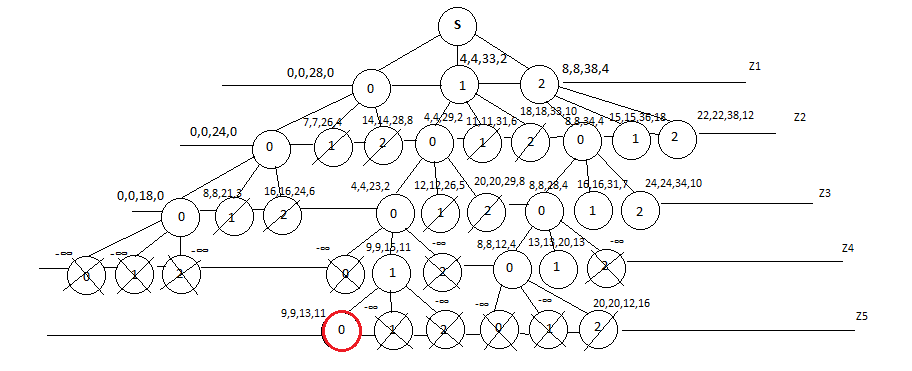
б) δ(xj) «лучше» чем Δ(xj),

то вершина xj вычеркивается.

7. Перейти к шагу 1.

8. Алгоритм закончен. Вектор переменных, соответствующий выбранной вершине, является оптимальным.

## **2.3 Пример решения задачи вручную**



# Xopt = {1,0,0,1,0};

# R = 9;

# Висячих вершин – 6.

# Однократно перечеркнутые вершинные отвечают условию 5, а дважды перечеркнутые вершины не подходят под условия задачи.

## **2.4 Пример решения задачи с помощью программы**

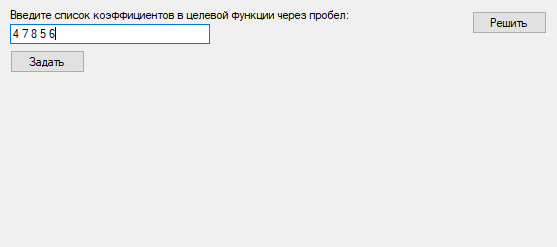


Рисунок 1. Ввод коэф. целевой функции

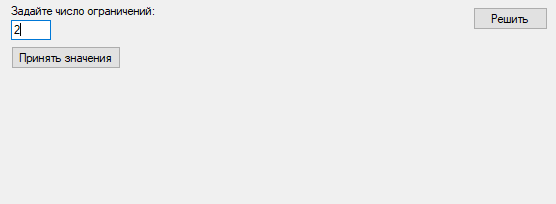


Рисунок 2. Число ограничений

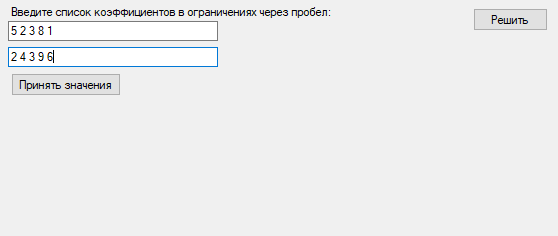


Рисунок 3. Ввод коэф. для каждого ограничения

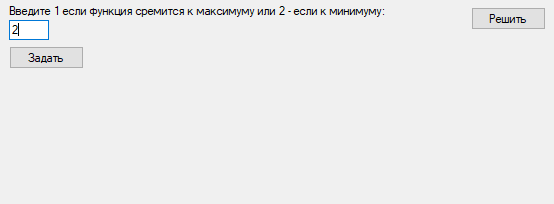


Рисунок 4. К max или к min стремится целевая функция

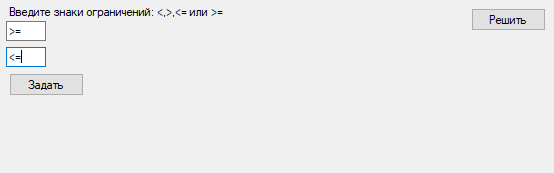


Рисунок 5. Знаки ограничений

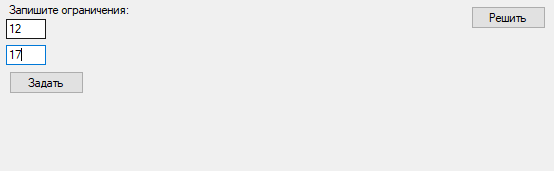


Рисунок 6. Бордеры

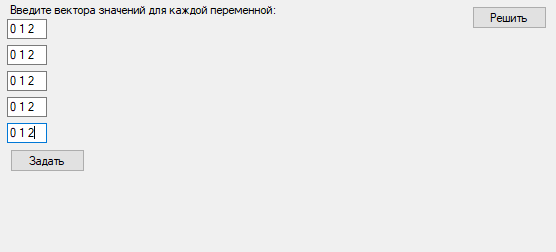


Рисунок 7. Ввод векторов значений переменных

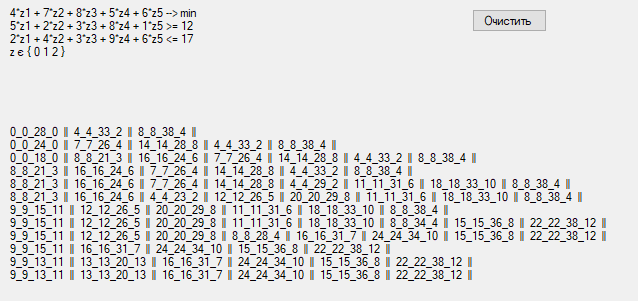


Рисунок 8. Вывод результатов

# Глава 3. Выбор и обоснование программной реализации

## **C Sharp**

C# в настоящее время набирает очень большой темп, и нет такого простого и многофункционального языка, как Си шарп. В нем сосредоточены все достоинства разных языков программирования.

### **Описание языка**

## C Sharp является объектно-ориентированным языком программирования. Появился на свет и был разработан в 1998 – 2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET. Компилятор с C# входит в стандартную установку самой .NET.

## Синтаксис Си шарпа близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов, делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Заимствуя многое от своих предшественников – языков C++, Java, Delphi, Модула и Smalltalk – C Sharp, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных средств: так например, C# не поддерживает множественное наследование классов, а С++ поддерживает.

### **История развития языка**

**Версия 1.0**

Проект Си Шарп был начат в декабре 1998 года и получил кодовое название COOL (C-style Object Oriented Language). Версия 1.0 была анонсирована вместе с платформой .NET в июне 2000 г., тогда же появилась и первейшая общедоступная бета-версия; C# 1.0 окончательно вышел вместе с Microsoft Visual Studio .NET в феврале 2002 года.

Первая версия Си Шарпа по своим возможностям напоминала Java 1.4, немного их расширяя: так, в C# имелись свойства (похожие в коде как поля объекта, а на деле вызывающие при обращении к ним методы класса), индексаторы (подобные свойствам, но принимающие параметр как индекс массива), события, делегаты, циклы foreach, структуры, автоматическое преобразование встроенных типов в объекты при необходимости (boxing), атрибуты, встроенные средства взаимодействия с неуправляемым кодом (DLL, COM) и т. д.

Дополнительно, в C# были перенесены некоторые возможности C++, отсутствовавшие в Java: беззнаковые типы, перегрузку операторов (с некоторыми ограничениями, в отличие от C++), передача параметров в метод по ссылке, методы с переменным числом параметров, оператор goto (с ограничениями). Также в C# оставили ограниченную возможность работы с указателями – в местах кода, специально обозначенных словом unsafe и при указании специальной опции компилятору.

**Версия 2.0**

Проект спецификации C# 2.0 впервые был опубликован Microsoft в октябре 2003 года; в 2004 году выходили бета-версии (проект с кодовым названием Whidbey), C# 2.0 вышел окончательно 7 ноября 2005 года вместе с Visual Studio 2005 и .NET 2.0.

Новые возможности в версии 2.0

* Частичные типы (разделение реализации класса более чем на один файл).
* Новая форма итератора, позволяющая создавать сопрограммы с помощью ключевого слова yield.
* Обобщённые, или параметризованные типы (generics). В отличие от шаблонов С++, они поддерживают некоторые дополнительные возможности и работают на уровне виртуальной машиной. Вместе с тем, параметрами обобщённого типа не могут быть выражения, они не могут быть полностью или частично специализированы, не поддерживают шаблонных параметров по умолчанию, от шаблонного параметра нельзя наследоваться, и т. д.
* Оператор '??': return obj1 ?? obj2; означает (в нотации C# 1.0) return obj1!=null ? obj1 : obj2;.
* Анонимные методы, обеспечивающие функциональность замыкания.
* Поддержка 64-разрядных вычислений, что кроме всего прочего, позволяет увеличить адресное пространство и использовать 64-разрядные примитивные типы данных.
* Возможность создавать хранимые процедуры, триггеры и даже типы данных на .Net языках (в том числе и на C#).
* Обнуляемые ('nullable') типы-значения (обозначаемые вопросительным знком, например, int? i = null;), представляющие собой те же самые типы-значения, способные принимать также значение null. Такие типы позволяют улучшить взаимодействие с базами данных через язык SQL.

**Версия 3.0**

На сайте Microsoft в июне 2004 года Андерс Хейлсберг впервые рассказал о планируемых расширениях языка в C#3.0. В сентябре 2005 года вышел проект спецификации C# 3.0 и проект бета-версия C# 3.0, устанавливаемая в виде дополнения к существующим Visual Studio 2005 и .NET 2.0. И окончательно эта версия языка вошла в Visual Studio 2008 и .NET 3.5.

Новые возможности в версии 3.0

* В C# 3.0 появились следующие радикальные добавления к языку: ключевые слова select, from, where, позволяющие делать запросы из SQL,XML, коллекций и т. п.
* Деревья выражений – лямбда-выражения теперь могут представляться в виде структуры данных, доступной для обхода во время выполнения, тем самым позволяя транслировать строго типизированные C#-выражения в другие домены (например, выражения SQL).
* Инициализация объекта вместе с его свойствами.
* Вывод типов локальной переменной: var x = "hello"; вместо string x = "hello";
* Безымянные типы: var x = new { Name = "James" };
* Методы-расширения – добавление метода в существующий класс с помощью ключевого слова this при первом параметре статической функции.
* Автоматические свойства: компилятор сгенерирует закрытое (private) поле и соответствующие аксессор и мутатор для кода вида: public string Name { get; private set; }

C# 3.0 совместим с C# 2.0 по генерируемому MSIL-коду; улучшения в языке – чисто синтаксические и реализуются на этапе компиляции. Например, многие из интегрированных запросов LINQ можно осуществить, используя безымянные делегаты в сочетании с предикатными методами над контейнерами вроде List.FindAll и List.RemoveAll.

**Версия 4.0**

Превью C# 4.0 было представлено в конце 2008 года, вместе с CTP-версией Visual Studio 2010.

Visual Basic 10.0 и C# 4.0 были выпущены в апреле 2010 года, одновременно с выпуском Visual Studio 2010.

Новые возможности в версии 4.0

Возможность использования позднего связывания, для использования:

* + с языками с динамической типизацией (Python, Ruby)
  + с COM-объектами
  + отражения (reflection)
  + объектов с изменяемой структурой (DOM). Появляется ключевое слово dynamic.
* Новые возможности COM interop
* Именованные и опциональные параметры
* Ковариантность и контравариантность
* Контракты в коде (Code Contracts)

**Версия 5.0**

C# 5.0 вышла в августе 2012 года вместе с Visual Studio 2012.

Новые возможности в версии 5.0

* Асинхронные методы
* Компилятор как сервис

**Версия 6.0**

Microsoft выпустила предварительную версию Visual studio 2015 и .Net 4.6. В новом C# 6.0 несколько новых возможностей, которые могут облегчить кодинг.

Новые возможности в версии 6.0

* Инициализация свойств со значениями
* Интерполяция строк
* Использование лямбда-выражений
* Импорт статических классов
* Null-условный оператор
* nameof оператор
* Await в catch и finally блоках
* Фильтры исключений
* Инициализация Dictionary

### **Синтаксис языка**

Синтаксис C# очень выразителен, к тому же прост в изучении. Все, кто знаком с языками C, C++ или Java с легкостью узнают синтаксис с фигурными скобками, характерный для языка C#.  Синтаксис Си Шарп делает проще то, что было сложно в C++, и обеспечивает мощные возможности, такие как типы значений Nullable, перечисления, делегаты, лямбда-выражения и прямой доступ к памяти, чего нет в Java. C# поддерживает универсальные методы и типы, обеспечивая более высокий уровень безопасности и производительности, а также итераторы, позволяющие при реализации коллекций классов определять собственное поведение итерации, которое может легко использоваться в клиентском коде. Выражения LINQ – они делают строго типизированный запрос очень удобной языковой конструкцией.

### **C# - объектно-ориентированный язык**

Как объектно-ориентированный язык, C# поддерживает понятия инкапсуляции, наследования и полиморфизма. Все переменные и методы, включая метод Main – точку входа приложения – инкапсулируются в определения классов. Класс может наследовать непосредственно из одного родительского класса, но может реализовывать любое число интерфейсов. Для методов, которые переопределяют виртуальные методы в родительском классе, необходимо ключевое слово override, чтобы исключить случайное повторное определение. В языке C# структура похожа на облегченный класс: это тип, распределяемый в стеке, реализующий интерфейсы, но не поддерживающий наследование.

В дополнение к основным описанным объектно-ориентированным принципам, язык C# упрощает разработку компонентов программного обеспечения благодаря нескольким инновационным конструкциям языка, в число которых входят следующие:

* Инкапсулированные сигнатуры методов, называемые делегатами, которые поддерживают типобезопасные уведомления о событиях.
* Свойства, выступающие в роли методов доступа для закрытых переменных-членов.
* Атрибуты с декларативными метаданными о типах во время выполнения.
* Встроенные комментарии XML-документации.
* LINQ, предлагающий встроенные возможности запросов в различных источниках данных.

### **Основные черты .NET**

* Поддержка нескольких языков. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому мы можем сделать отдельные модули одного приложения на отдельных языках.
* Кроссплатформенность. .NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент .NET Framework 4.5.1. поддерживается на большинстве современных ОС Windows (Windows 8.1/8/7/Vista). А благодаря проекту Mono можно создавать приложения, которые будут работать и на других ОС семейства Linux, в том числе на мобильных платформах Android и iOS.
* Мощная библиотека классов. .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# - текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов .NET.
* Разнообразие технологий. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF. Для создания веб-сайтов - ASP.NET и т.д.

Также еще следует отметить такую особенность языка C# и фреймворка .NET, как автоматическая сборка мусора. А это значит, что нам в большинстве случаев не придется, в отличие от С++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

## **Пошаговый алгоритм работы программы**

Алгоритм работы программы представлен в виде последовательности шагов, целью которых является: показать эффективность композитной версии.

Алгоритм:

* 1. Ввод исходных данных:
  2. Построение дерева ветвлений
  3. Вычисление оценок методом типа ветвей и границ, отсечение «плохих» вершин, методика которых включает технологии, используемые в динамическом программировании.
  4. Вывод результатов.

# Глава 4. Результаты эксперимента

Эксперимент проводился с целью сравнительной эффективности алгоритмов – классической версии метода ветвей и границ и композитным алгоритмом.

Эксперимент проводился на компьютере c процессором AMD FX(tm)-4100 Quad-Core Processor 3,6 GHz, ОЗУ 8, ОС Windows 10 – 64 bit.

**Эксперимент №1**

Для проведения эксперимента были использованы функции размерностью:

Таблица 1. Исходные данные для эксперимента №1

По графику можно сделать вывод о том, при “маленьких” и “средних” размерностях задач производительность композитного алгоритма не показывает высокой эффективности в сравнении с методом ветвей и границ. Это можно объяснить лишним количеством операций, совершающихся для поиска и удаления “плохих” вершин, что, в свою очередь, не дает значимого эффекта в долгосрочной перспективе.

Однако на графике видно, что при больших размерностях задач, **эффективность производительности** композитного алгоритма **улучшается в разы**.

Полученный график функций значительно отличается от первого, но не противоречит ему. Значительные изменения в производительности на “больших” задачах можно объяснить тем, что **противоречивые оценки** отсекают большую часть решения. В такой ситуации **композитный алгоритм** хоть и **дает более оптимальное решение**, но не значительное.

Так же не стоит забывать про то, что большую роль в нахождении глобального оптимального решения композитным алгоритмом играют входные данные (условия задачи). Этот “недостаток” композитный алгоритм унаследовал от метода ветвей и границ.

Так или иначе, результаты проведенных экспериментов позволяют сделать вывод – в сравнении с методом ветвей и границ, **композитный алгоритм** дает **более оптимальное** глобальное **решение** экстремальных задач дискретной оптимизации.

# Заключение

В результате выполнения выпускной квалификационной работы была полностью достигнута поставленная цель работы – Экспериментальный анализ эффективности композитной версии методов типа ветвей и границ, осуществляющей, фронтальный спуск по дереву ветвлений применительно к задачам с не булевыми переменными:

* Проведен разбор и анализ классических алгоритмов решения экстремальных задач дискретной оптимизации;
* Рассмотрена композитная версия метода ветвей и границ применительно к задачам с не булевыми переменными осуществляющим фронтальный спуск по дереву ветвлений;
* Осуществлена программная реализация системы на языке C Sharp;
* Проведен сравнительный анализ эффективности классической и композитной версии метода типа ветвей и границ.
* Проведены тестирование и отладка.

# Список литературы

1. Гроппен В.О. Интеллект композитных алгоритмов поиска глобально оптимальных решений экстремальных задач с булевыми переменными.
2. Будаева А.А., Гроппен В.О. Принятие решений: теория, технология, приложения. Владикавказ: Фламинго, 2009.
3. Кочегурова Е.А. Теория и методы оптимизации. Изд-во Томского политехнического университета,2012.
4. Гроппен В.О. Принципы оптимизации программного обеспечения ЭВМ. Изд. Ростовского университета. 1993 г.
5. C# 4.0: Полное руководство, Герберт Шилдт,2011.
6. Шилд Герберт. С++. Руководство для начинающих. Вильямс, Москва, 2005

# Приложение 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.Windows.Forms;

namespace Dip\_Forms

{

public partial class Form1 : Form

{

TextBox[] MasBox;

int n; //Число ограничений

int m; //Число переменных

int[] F; //Коэффичиенты в целевой ф-ции

int[,] ogr; //Коэффициенты в ограничениях

int predel;

string[] znak;

int[] border;

int[] z;

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

string[] split = textBox1.Text.Trim().Split(' ');

m = split.Length;

F = new int[m];

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

F[i] = int.Parse(split[i]);

textBox1.Text = "";

textBox1.Size = new Size(40, 20);

label1.Text = "Задайте число ограничений:";

button2.Visible = true;

button1.Visible = false;

}

private void button2\_Click(object sender, EventArgs e)

{

n = int.Parse(textBox1.Text);

MasBox = new TextBox[n];

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

{

MasBox[i] = new TextBox() { TabIndex = 4 + i, Size = new Size(210, 20), Location = new Point(12, 25 + i \* 26) };

this.Controls.Add(MasBox[i]);

}

button3.Location = new Point(15, MasBox[n - 1].Location.Y + 26);

button2.Visible = false;

label1.Text = "Введите список коэффициентов в ограничениях через пробел:";

textBox1.Visible = false;

button3.Visible = true;

textBox1.Text = "";

}

private void button3\_Click(object sender, EventArgs e)

{

ogr = new int[n, m];

string[] split = new string[m];

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

{

split = MasBox[i].Text.Trim().Split(' ');

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

{

ogr[i, j] = int.Parse(split[j]);

}

}

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

{

this.Controls.Remove(MasBox[i]);

}

label1.Text = "Введите 1 если функция сремится к максимуму или 2 - если к минимуму:";

textBox1.Visible = true;

button3.Visible = false;

button4.Visible = true;

}

private void button4\_Click(object sender, EventArgs e)

{

predel = int.Parse(textBox1.Text);

textBox1.Visible = false;

button4.Visible = false;

button5.Visible = true;

textBox1.Text = "";

label1.Text = "Введите знаки ограничений: <,>,<= или >=";

znak = new string[n];

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

{

MasBox[i] = new TextBox() { TabIndex = 4 + i, Size = new Size(40, 20), Location = new Point(12, 25 + i \* 26) };

this.Controls.Add(MasBox[i]);

}

button5.Location = new Point(15, MasBox[n - 1].Location.Y + 26);

}

private void button5\_Click(object sender, EventArgs e)

{

for (int i = 0; i < n; i++) znak[i] = MasBox[i].Text.Trim();

button5.Visible = false;

button6.Visible = true;

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

{

this.Controls.Remove(MasBox[i]);

}

label1.Text = "Запишите ограничения:";

border = new int[n];

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

{

MasBox[i] = new TextBox() { TabIndex = 4 + i, Size = new Size(40, 20), Location = new Point(12, 25 + i \* 26) };

this.Controls.Add(MasBox[i]);

}

button6.Location = new Point(15, MasBox[n - 1].Location.Y + 26);

}

private void button6\_Click(object sender, EventArgs e)

{

for (int i = 0; i < n; i++) border[i] = int.Parse(MasBox[i].Text);

button6.Visible = false;

button7.Visible = true;

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

{

this.Controls.Remove(MasBox[i]);

}

label1.Text = "Введите вектор значений переменных:";

textBox1.Size = new Size(100, 20);

textBox1.Visible = true;

button7.Location = new Point(15, 51);

}

private void button7\_Click(object sender, EventArgs e)

{

string[] split = new string[textBox1.Text.Trim().Split(' ').Length];

split = textBox1.Text.Trim().Split(' ');

z = new int[split.Length + 2];

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

z[i+1] = int.Parse(split[i]);

button7.Visible = false;

button8.Visible = true;

label1.Text = "";

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

label1.Text += F[i] + "\*z" + (i + 1) + " + ";

label1.Text = label1.Text.Remove(label1.Text.Length - 2, 2);

label1.Text += (predel == 1) ? "-->" + "max" : "--> " + "min";

label1.Text += "\n";

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

{

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

label1.Text += ogr[i, j] + "\*z" + (j + 1) + " + ";

label1.Text = label1.Text.Remove(label1.Text.Length - 2, 2);

label1.Text += znak[i] + " " + border[i];

label1.Text += "\n";

}

label1.Text += "z є " + "{ " + z[1];

for (int i = 2; i < split.Length + 1; i++)

label1.Text += " " + z[i];

label1.Text += " }";

label2.Location = new Point(12, (1 + n + m) \* 16);

label2.Text = "";

label2.Visible = true;

button8.Visible = true;

textBox1.Visible = false;

}

private void button8\_Click(object sender, EventArgs e)

{

string st, str;

int[,] masskomb = new int[100002, 100]; //массив бинарного счетчика

int[] helpmain = new int[100];

int[] helpkpzmain = new int[100];

int[,] helpogr = new int[100, 100];

string[] helpkpzsravn = new string[100];

string[] vverkom = new string[100002]; //для регистрации не нужных вершин

int[,] mtree = new int[100002, 100]; // массив дерево [кол ветвей],[текущее число на ветви, ступень ветви, попадание в ограничения...]

int kombin; // количество комбинаций

int kolz = z.Length - 2; // количество значений вектора

int MaxElz; //максимальный елемент вектора

int MinElz; //минимальный елемент вектора

int extr = 0;

byte kolkpzsravn;

int tkextr; // текущий экстремум

int mainsum, mainkpzsum, ogrsum;

int hlp, hlpit, hlprzr;

bool konec, kpzdel;

string Sravn;

kombin = 1;

for (int i = 1; i <= m; i++) masskomb[kombin, i] = 1;

hlp = 0;

while (hlp < m \* kolz)

{

for (int i = m; i > 0; i--)

if (masskomb[kombin, i] == kolz + 1)

{

masskomb[kombin, i - 1]++;

masskomb[kombin, i] = 1;

}

str = "";

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

str = str + masskomb[kombin, i];

/\*

//---------------------

Console.WriteLine(str);

//---------------------

\*/

hlp = 0;

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

hlp = hlp + masskomb[kombin, i];

kombin++;

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

masskomb[kombin, i] = masskomb[kombin - 1, i];

masskomb[kombin, m]++;

}

kombin--;

/\*

//---------------------

Console.WriteLine(kombin + "-" + m);

//---------------------

\*/

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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

{

mtree[i, 2] = 0;

vverkom[i] = "p";

}

MaxElz = z[1];

MinElz = z[1];

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

{

if (MaxElz < z[i])

MaxElz = z[i];

if (MinElz > z[i])

MinElz = z[i];

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

if (predel == 1) extr = 0;

if (predel == 2) extr = 10000;

for (int i = 1; i <= kombin; i++) mtree[i, 1] = extr;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

konec = false;

Sravn = "";

while (konec == false)

{

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

if (mtree[i, 1] == extr && vverkom[i] == "p")

mtree[i, 2]++;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

st = "";

hlpit = -1;

hlprzr = 0;

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

{

for (int j = 1; j <= m; j++)

{

if (predel == 1) helpmain[j] = MaxElz;

if (predel == 2) helpmain[j] = MinElz;

helpkpzmain[j] = MinElz;

}

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

for (int g = 1; g <= m; g++)

{

if (znak[j-1] == ">=" || znak[j-1] == ">")

helpogr[j, g] = MaxElz;

if (znak[j-1] == "<=" || znak[j-1] == "<")

helpogr[j, g] = MinElz;

}

for (int j = 1; j <= mtree[i, 2]; j++)

{

helpmain[j] = z[masskomb[i, j]];

helpkpzmain[j] = z[masskomb[i, j]];

}

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

for (int g = 1; g <= mtree[i, 2]; g++)

helpogr[j, g] = z[masskomb[i, g]];

mainsum = 0;

mainkpzsum = 0;

for (int j = 1; j <= m; j++)

{

mainsum = mainsum + helpmain[j] \* F[j-1];

mainkpzsum = mainkpzsum + helpkpzmain[j] \* F[j-1];

}

mtree[i, 1] = mainsum;

mtree[i, 3] = mainkpzsum;

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

{

ogrsum = 0;

for (int g = 1; g <= m; g++)

ogrsum = ogrsum + helpogr[j, g] \* ogr[j-1, g-1];

mtree[i, j + 3] = ogrsum;

if (znak[j-1] == ">=")

if (mtree[i, j + 3] <= border[j-1])

{

vverkom[i] = "n";

mtree[i, 1] = 0;

}

if (znak[j-1] == ">")

if (mtree[i, j + 3] > border[j-1])

{

vverkom[i] = "n";

mtree[i, 1] = 0;

}

if (znak[j-1] == "<=")

if (mtree[i, j + 3] >= border[j-1])

{

vverkom[i] = "n";

mtree[i, 1] = 0;

}

if (znak[j-1] == "<")

if (mtree[i, j + 3] < border[j-1])

{

vverkom[i] = "n";

mtree[i, 1] = 0;

}

}

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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

{

str = "";

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

str = str + "\_" + mtree[i, l + 3];

/\*

//---------------------

Console.WriteLine(mtree[i, 1] + "\_" + mtree[i, 3] + str);

Console.WriteLine(mtree[i, 2]);

Console.WriteLine(vverkom[i]);

//---------------------

\*/

if ((hlpit != z[masskomb[i, mtree[i, 2]]] || hlprzr != mtree[i, 2]) && vverkom[i] != "n")

{

hlpit = z[masskomb[i, mtree[i, 2]]];

hlprzr = mtree[i, 2];

st = st + mtree[i, 1] + "\_" + mtree[i, 3] + str + " || ";

}

kolkpzsravn = Convert.ToByte(n + 3);

for (int j = 1; j <= kombin; j++)

{

if (mtree[j, 2] == mtree[i, 2])

{

helpkpzsravn[2] = "+";

if (predel == 1)

{

if (mtree[j, 1] < mtree[i, 1]) helpkpzsravn[1] = "+";

if (mtree[j, 1] == mtree[i, 1]) helpkpzsravn[1] = "=";

if (mtree[j, 1] > mtree[i, 1]) helpkpzsravn[1] = "-";

if (mtree[j, 3] < mtree[i, 3]) helpkpzsravn[3] = "+";

if (mtree[j, 3] == mtree[i, 3]) helpkpzsravn[3] = "=";

if (mtree[j, 3] > mtree[i, 3]) helpkpzsravn[3] = "-";

}

if (predel == 2)

{

if (mtree[j, 1] > mtree[i, 1]) helpkpzsravn[1] = "+";

if (mtree[j, 1] == mtree[i, 1]) helpkpzsravn[1] = "=";

if (mtree[j, 1] < mtree[i, 1]) helpkpzsravn[1] = "-";

if (mtree[j, 3] > mtree[i, 3]) helpkpzsravn[3] = "+";

if (mtree[j, 3] == mtree[i, 3]) helpkpzsravn[3] = "=";

if (mtree[j, 3] < mtree[i, 3]) helpkpzsravn[3] = "-";

}

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

{

if (znak[g-1] == ">=" || znak[g-1] == ">")

{

if (mtree[j, g + 3] < mtree[i, g + 3]) helpkpzsravn[g + 3] = "+";

if (mtree[j, g + 3] == mtree[i, g + 3]) helpkpzsravn[g + 3] = "=";

if (mtree[j, g + 3] > mtree[i, g + 3]) helpkpzsravn[g + 3] = "-";

}

if (znak[g-1] == "<=" || znak[g-1] == "<")

{

if (mtree[j, g + 3] > mtree[i, g + 3]) helpkpzsravn[g + 3] = "+";

if (mtree[j, g + 3] == mtree[i, g + 3]) helpkpzsravn[g + 3] = "=";

if (mtree[j, g + 3] < mtree[i, g + 3]) helpkpzsravn[g + 3] = "-";

}

}

kpzdel = true;

for (int g = 1; g <= n + 3; g++)

if (helpkpzsravn[g] == "-" || helpkpzsravn[g] == "=")

kpzdel = false;

if (kpzdel == true)

{

mtree[j, 1] = 0;

vverkom[j] = "n";

}

}

}

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

label2.Text += st;

label2.Text += "\n";

if (predel == 1)

{

extr = 0;

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

if (mtree[i, 1] > extr && vverkom[i] == "p")

{

extr = mtree[i, 1];

tkextr = i;

}

}

if (predel == 2)

{

extr = 1000;

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

if (mtree[i, 1] <= extr && vverkom[i] == "p")

{

extr = mtree[i, 1];

tkextr = i;

}

}

if (st == Sravn)

konec = true;

else

Sravn = st;

}

button8.Visible = false;

button9.Visible = true;

button9.Location = button8.Location;

}

private void button9\_Click(object sender, EventArgs e)

{

label1.Text = "Введите список коэффициентов в целевой функции через пробел:";

label2.Text = "";

label2.Visible = false;

textBox1.Visible = true;

textBox1.Size = new Size(200, 20);

textBox1.Text = "";

button1.Visible = true;

button9.Visible = false;

}

}

}

# Приложение 2. Руководство пользователя