Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет «Высшая школа экономики»

Московский институт электроники и математики им. А.Н. Тихонова

Образовательная программа
«Системный анализ и математические технологии»
Уровень образования – Магистратура

О Т Ч Е Т
по проектной работе
Научно-исследовательский проект № 1123
«Исследование подходов к реализации задачи поиска оптимальных сочетаний партий сырья»

Выполнил студент гр. МСМТ-211
Дырова Элина Руслановна

_____
(подпись)

Руководитель проекта:
Ведущий научный сотрудник Международной лаборатории САММА НИУ ВШЭ, к.т.н., Семенов Александр Сергеевич

_____ _____
(оценка) *(подпись)*

_____
(дата)

Москва 2022

Техническое задание

Цели и задачи проекта

Целью выполнения проекта является исследование подходов к решению задачи объединения партий сырья в группы для получения пригодных для переработки материалов.

Задачи проекта:

1. Исследование и выбор программных технологий для решения задачи.
2. Решение поставленной задачи.
3. Исследование способов решения задачи за ограниченное время.

Планируемый результат

1. Обзор программных средств решения задачи оптимизации.
2. Программная реализация решения задачи.
3. Разработанная методика по получению решения задачи за ограниченное время при помощи применения эвристик и возможность использования параллельных вычислений.
4. Выступление на конференции в случае получения интересных научных результатов.

Фактический результат

1. Обзор программных средств решения задачи оптимизации при помощи SCIP.
2. Программная реализация решения модельной задачи при помощи языка ZIMPL.
3. Проведено исследование влияния различных эвристик SCIP и параллельного выполнения модельной программы на время достижения заданного значения разрыва между наилучшим полученным решением и решением соответствующей двойственной задачи.
4. Разработаны рекомендации по самому быстрому получению решения задачи за ограниченное время при помощи применения эвристик SCIP и возможности использования параллельных вычислений.

Методы исследования

Методы исследования подходов к решению задачи заключаются в изучении алгоритмических методов целочисленной оптимизации, эвристических алгоритмов, их сравнительное оценочное тестирование с использованием параллельных вычислений.

Содержание

Введение

Методы целочисленной оптимизации позволяют максимизировать выгоду и минимизировать затраты при решении задач, связанных с практической деятельностью человека, таких как транспортные задачи, составление расписаний, промышленное производство.

Целью выполнения проекта является исследование подходов к решению модельной задачи оптимизации объединения партий сырья в группы для получения пригодных для переработки материалов.

Важным ограничением проекта является возможность использования только программных open-source средств. В настоящее время одним из лучших инструментом для решения задач оптимизации с возможностью некоммерческого использования является разработанный в Германии SCIP, изучению возможностей которого посвящен проект.

1. Обзор программных средств решения задачи целочисленной оптимизации при помощи SCIP

1.1. Алгоритмы и методы оптимизации

Линейное программирование - это набор методов оптимизации линейной целевой функции с учетом ограничений линейных равенств и линейных неравенств. Его допустимая область представляет собой выпуклый многогранник, который представляет собой множество, определенное как пересечение конечного числа полупространств, каждое из которых определяется линейным неравенством. Его целевая функция – это вещественная аффинная (линейная) функция, определенная на этом многограннике. Алгоритм линейного программирования находит точку многогранника, в которой эта функция имеет наименьшее (или наибольшее) значение, если такая точка существует.

Общей задачей линейного программирования (ОЗЛП) называют задачу, в которой функция цели, которую надо оптимизировать, представляет собой линейную комбинацию известных коэффициентов c_j (i =1..n) и неизвестных переменных x_j (j =1..n) вида:

$$f = \sum_{j=1}^{n} c_j x_j$$

Ограничения, накладываемые на область возможных решений, имеют вид линейных неравенств или равенств:

$$Ax \leq b$$

Допустимое решение задачи линейного программирования – любая совокупность неотрицательных переменных, удовлетворяющих условиям. Оптимальное решение – это допустимое решение, которое обращает целевую функцию в максимум (минимум) [1].

Задача целочисленного линейного программирования (ЗЦЛП) – это задача математической оптимизации, в которой некоторые или все переменные должны быть целыми числами. Иными словами, ЗЦЛП является

ОЗЛП с добавлением ограничения на целочисленность переменных: $x_j \in Z$, j = 1..n.

1.1.1. Релаксация линейного программирования

Задачу линейного программирования возможно решить за полиномиальное время. Такой метод называется линейной релаксацией. Оптимальное значение ОЗЛП гарантированно будет нижней границей (если задача на максимум – верхней) для задачи целочисленного линейного программирования, поскольку устранение ограничений может только расширить допустимый набор. Если оптимальное решение удовлетворяет интегральным ограничениям, то оно гарантированно будет оптимальным для исходной задачи. В задачах линейного программирования нижнюю (верхнюю) границу называют двойственной границей (dual bound) [7].

LP-допустимым решением называют решение (набор $\bar{x}_j$), удовлетворяющее всем ограничениям задачи за исключением целочисленности переменных.

1.1.2. Метод ветвей и границ

Метод ветвей и границ (branch-and-bound) основан на двух процедурах: разбиение множества допустимых решений на подмножества (ветвление) и оценка целевой функции на этих подмножествах (вычисление границ) [3].

Общая идея метода может быть описана на примере поиска минимума функции *f(x)* на множестве допустимых значений переменной *x*.

Процедура ветвления состоит в разбиении множества допустимых значений переменной *x* на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое деревом поиска или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти (подмножества множества значений переменной *x*). Иными словами, каждый

дочерний элемент дерева является подзадачей для своего родителя, а корневой элемент – основная полная задача.

Процедура нахождения оценок заключается в поиске верхних или нижних границ для решения задачи на подобласти допустимых значений переменной *x*.

В основе метода ветвей и границ лежит следующая идея (для поиска минимума): если нижняя граница значений функции на подобласти A дерева поиска больше, чем допустимое решение на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (правило отсева). Обычно минимальное из полученных допустимых решений записывают в глобальную переменную m; любой узел дерева поиска, нижняя граница которого больше значения m, может быть исключен из дальнейшего рассмотрения.

1.1.3. Параллельная версия метода ветвей и границ

Основная сложность распараллеливания метода ветвей и границ заключается в том, что он ведет себя непредсказуемо, т. е. структура дерева поиска заранее неизвестна и строится в процессе решения. Эта структура, по существу, является динамической, и это существенно затрудняет распараллеливание.

Выделенный процессор, называемый мастер-процессором, управляет равномерным распределением задач по остальным процессорам, называемых рабочими.

При этом он может как производить ветвления, так и управлять остальными процессорами. Каждый рабочий процессор получает подзадачу или набор подзадач. После выполнения определенного количества ветвлений или получения определенного количества подзадач, процессор посылает сообщение мастеру, что предел достигнут. Мастер находит свободный процессор и пересылает ему часть подзадач. В случае если свободного

процессора нет, задачи хранятся на мастере до тех пор, пока один из процессоров не сообщит об окончании решения всех своих подзадач [2].

1.2. Эвристические методы

Первичная эвристика (primal heuristic) - это метод, разработанный для более быстрого решения проблемы, когда классические методы слишком медленные, или для нахождения приближенного решения, когда классические методы не могут найти точное решение. Это достигается путем обмена оптимальности, полноты или точности на скорость.

Эвристическая функция ранжирует альтернативы в алгоритмах поиска на каждом шаге ветвления на основе доступной информации, чтобы решить, какой ветке следовать. Например, она может приближать точное решение. Цель эвристики состоит в том, чтобы в разумные сроки найти решение, достаточное для решения поставленной задачи. Это решение может быть не лучшим из всех решений этой проблемы, или оно может просто приближаться к точному решению. Любое такое найденное решение называется первичной границей (primal bound).

При запуске алгоритма ветвей и границ отслеживают две основные характеристики: двойственную границу, отвечающую за оптимальность решения, и первичную границу, отвечающую за допустимость решения. Разницу между двумя границами называют разрывом (gap). Он может быть абсолютным (1) и относительным (2).

$$gap = |global\ primal\ bound - global\ dual\ bound| \quad (1)$$

$$gap = \frac{|global\ primal\ bound - global\ dual\ bound|}{min(global\ primal\ bound; global\ dual\ bound)} \quad (2)$$

1.2.1. Эвристики округления

Эвристики округления (rounding heuristics) основаны на построении целочисленного решения из дробного решения после линейной релаксации (LP-допустимого) путем округления переменных. Эти эвристики могут ускорить метод ветвей и границ, пытаясь найти лучшие действующие

решения в каждом узле дерева поиска. Все эвристики округления стремятся сходиться к интегральному решению за определенное число итераций, в которых округляются переменные (фиксируются переменные x_j либо равными 0, либо 1), после чего повторно ищется двойственная граница. Эвристики округления не всегда приводят к допустимому целочисленному решению [8].

Переменная x_j называется тривиально округляемой в меньшую сторону, если все коэффициенты a_{ij} соответствующего столбца матрицы A неотрицательны, то есть, $A_{\cdot j} \geq 0$.

Переменная x_j называется тривиально округляемой в большую сторону, если все коэффициенты a_{ij} соответствующего столбца матрицы A являются неположительными, то есть, $A_{\cdot j} \leq 0$.

Число отрицательных коэффициентов a_{ij} столбца $A_{\cdot j}$ называется числом нижних блокировок переменной x_j.

Число положительных коэффициентов a_{ij} столбца $A_{\cdot j}$ называется числом верхних блокировок переменной x_j.

В рамках проекта были использованы следующие эвристики округления, реализованные в SCIP: Simple Rounding, ZI-Rounding, Rounding (Табл. 1).

Таблица 1. Эвристики округления в SCIP

Название эвристики	Описание
Simple Rounding	Simple Rounding перебирает набор дробных переменных LP-допустимого решения, и, если возможно, округляет их до целого значения, оставаясь при этом LP-допустимым. Если число нижних блокировок равно нулю, то переменная округляется в меньшую сторону, если число верхних блокировок равно нулю – в большую. Алгоритм прекращается, если находит переменную, которая не является тривиально округляемой, или при округлении всех переменных x_j.
ZI Round	ZI Round постепенно избавляется от

	нецелочисленности LP-допустимого решения путем смещения дробных значений в сторону целых значений, но не обязательно округляя их. Для каждой дробной переменной x_j эвристика вычисляет границы для двух возможных направлений округления $\bar{x}_j$ таким образом, чтобы полученное решение оставалось LP-выполнимым. Затем эвристика сдвигает x_j на соответствующую границу в направлении, которое максимально увеличивает целочисленность решения $\min \{\bar{x}_j - \lfloor \bar{x}_j \rfloor, \lceil \bar{x}_j \rceil - \bar{x}_j\}$. ZI Round может обрабатывать набор дробных переменных несколько раз. Эвристика завершается либо возможным решением, либо прерывается, если дробность больше не может быть уменьшена, или если эвристика достигает предопределенного предела итерации.
Rounding	В отличие от двух предыдущих эвристик, Rounding также выполняет округления, которые потенциально приводят к нарушению некоторых линейных ограничений, а затем пытается исправить это нарушение путем дальнейших округлений позже. Подобно Simple Rounding, эвристика Rounding учитывает блокировки вверх и вниз целочисленной переменной с дробным значением решения $\bar{x}_j$. До тех пор, пока линейные ограничения не нарушены, алгоритм выполняет итерацию по дробным переменным $\bar{x}_j$ и применяет округление в направлении меньшего количества блокировок, обновляя $A\hat{x}$ после каждого шага, при этом $\hat{x}$ является частично округленным решением. Если существует нарушенное линейное ограничение, эвристика попытается найти дробную переменную для округления в направлении, которое уменьшает нарушение ограничения. Если никакое округление не может уменьшить нарушение ограничения, процедура прерывается.

1.2.2. Эвристики погружения

Эвристики погружения (diving heuristics) имитируют частичный поиск в глубину в дереве ветвей и границ и являются важным дополнением к алгоритму. Эвристика погружения исследует единственный пробный путь вниз по дереву поиска. В отличие от обычного поиска, здесь поиск выполняется во вспомогательном дереве, исходящем из узла фокуса основного дерева поиска, где была вызвана эвристика. Погружение отличается от эвристик округления тем фактом, что решения после линейной релаксации повторно оптимизируется после ограничения или фиксации одной или нескольких переменных. Эвристику погружения можно понимать как эвристический поиск в дереве ветвей и границ на основе линейной релаксации: поиск погружается от корня к листу в дереве ветвей и границ, эвристически выбирая ветвь в каждом узле, как показано на рисунке 1. Правило ветвления, используемое в таком контексте, обычно сильно отличается от того, которое используется в точном подходе: в эвристике погружения речь идет не о балансировке дерева, а о быстром нахождении хороших оптимальных решений [10].

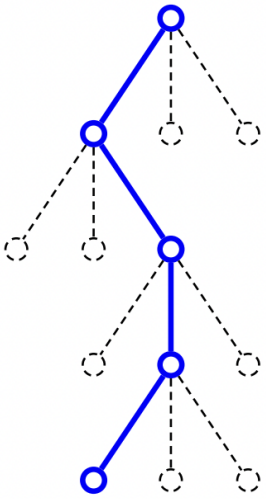

Рисунок 1. Эвристика погружения в дереве поиска

Алгоритмы эвристик погружения основаны на фракционности (дробности) переменных. Фракционностью (дробностью) переменной x_j называется функция $\varphi(x_j) = \left| x_j - \lfloor x_j + 0.5 \rfloor \right|$.

В рамках проекта были использованы следующие эвристики погружения, реализованные в SCIP: coefdiving, fracdiving и veclendiving (Табл. 2).

Таблица 2. Эвристики погружения в SCIP

Название эвристики	Описание
coefdiving	Для каждой переменной направление округления определяется только на основе блокировок переменных. Переменная x_j с минимальным положительным числом блокировок округляется в направлении, в котором достигается этот минимум. Если существует более одной переменной, в которой достигается минимальное количество блокировок, выбирается переменная с минимальной фракционностью $\varphi(\bar{x}_j)$.
fracdiving	Переменная x_j с наименьшей фракционностью $\varphi(\bar{x}_j)$ округляется до ближайшего целого числа.
veclendiving	Vector length Diving округляет переменную, для которой частное между $([x_j]-x_j) * c_j$ и $\lvert\{a_{ij} \mid a_{ij} \neq 0\}\rvert$ является минимальным. В этом случае $[\cdot]$ обозначает округление в направлении, где целевая функция потенциально ухудшается, – вверх, если $c_j \geq 0$, и вниз в противном случае. Это соотношение обеспечивает изменение целевой функции для каждой строки, которое осуществляется путем округления.

1.2.3. LNS эвристики

LNS (Large neighborhood search) эвристики исследуют окрестности одной начальной точки (или небольшого набора начальных точек), таких как

существующее решение или оптимальное решение релаксации LP. Они создают подзадачу исходной задачи ЛП обычно путем привязки некоторых переменных к значениям, взятым из заданных точек. Для задач только с двоичными переменными добавляются линейные ограничения, которые ограничивают количество переменных, отличных от заданной точки. Также добавляется ограничение отсечения для обеспечения того, чтобы решение, найденное для подзадачи, было лучше, чем начальное существующее решение. Это особенно целесообразно, если начальное существующее решение было осуществимо для подзадачи (чтобы избежать повторного нахождения одного и того же решения).

Очевидно, что поиск хорошего определения окрестности является важнейшей частью эвристики LNS. Окрестность должна содержать высококачественные решения, эти решения должны быть легко доступны, а окрестность должна быть легко обрабатываемой. Часто эти три цели на практике противоречат друг другу [6].

В рамках проекта были использованы следующие эвристики LNS, реализованные в SCIP: RINS, mutation, crossover (Табл. 3).

Таблица 3. LNS эвристики

Название эвристики	Описание
RINS	RINS (Relaxation Induced Neighborhood Search), это эвристика поиска в больших окрестностях, и для нее требуется известное допустимое решение. Она решает подзадачу, которая создается путем фиксации переменных, которые принимают одно и то же значение после линейной релаксации и допустимого решения текущего узла.
Crossover	Эвристика crossover основана на генетических алгоритмах и требует более одного возможного решения. Crossover стремится зафиксировать переменные в значениях, которые совпадают (по крайней мере) в двух решениях. Обоснование кроссовера заключается в том, что часто хорошие

	выполнимые решения имеют много общих значений переменных. Реализация crossover в SCIP требует, чтобы его эталонные решения не были найдены все с помощью одной и той же эвристики на одном и том же узле. В противном случае Crossover, скорее всего, оптимизировал бы то же или, по крайней мере, аналогичное пространство поиска, что и другая эвристика.
Mutation	Эвристика mutation основана на генетических алгоритмах и требует заранее известного допустимого решения. Она случайным образом фиксирует переменные к их значению в действующем решении и решает подзадачу, состоящую из оставшихся переменных.

1.2.4. Возможности параллелизма в SCIP

SCIP позволяет запускать несколько экземпляров программы параллельно в отдельных потоках в рамках одного вычислительного узла. Они могут совместно использовать возможные решения и границы глобальных переменных на протяжении всего процесса решения. Частота связи между потоками зависит от величины gap в определенный момент времени. Потоки собирают информацию в процессе решения определенной части задачи, а затем, при достижении детерминированных точек (они установлены разработчиками SCIP, обычно это получение решения хотя бы одним из потоков), обмениваются данными, используя общую структуру данных. Недостаток такого подхода заключается в том, что когда какой-либо поток оказывается в этой детерминированной точке, ему приходится простаивать и ждать остальные потоки [5].

Также SCIP имеет возможность запуска на нескольких хостах (распределенного запуска).

2. Программная реализация модельной задачи оптимизации

В данной работе рассматривается модельная задача оптимизации сочетаний партий сырья.

2.1. Описание задачи и цель

На складе имеется N партий сырья. Известна масса m_j , $j = 1..N$ каждой партии , каждая партия состоит из различных примесей I_{jk} в разных заданных соотношениях. Заданы ограничения объединения партий на размер группы (от 2 до 6), а также по каждой из примесей I_k^{min}, I_k^{max}.

Необходимо, не нарушая ограничений объединения, объединить партии в группы при оптимизации трех критериев: вовлечь как можно больше партий, вовлечь как можно больше партий с большой массой, вовлечь как можно больше партий с высоким содержанием примесей.

2.2. Математическая постановка задачи

Представим условия задачи в виде квадратной матрицы $N \times N$, содержащей значения x_{ij}. Строки – партии, столбцы – группы партий; $x_{ij} = 1$, если партия входит в группу, $x_{ij} = 0$, если нет. При этом должны быть соблюдены следующие ограничения:

1. Каждая партия входит только в одну группу, либо никуда не входит:

$$\sum_{i=1}^{N} x_{ij} \leq 1,\ j = 1,..,N$$

2. Ограничение на размер группы:

$$2 \leq \sum_{j=1}^{N} x_{ij} \leq 6 \text{ или } \sum_{j=1}^{N} x_{ij} = 0,\ i = 1,..,N$$

3. Общая масса каждой примеси в группе ограничена:

$$I_k^{min} * \sum_{j=1}^{N} x_{ij} * m_j \leq \sum_{j=1}^{N} x_{ij} * I_{jk} \leq I_k^{max} * \sum_{j=1}^{N} x_{ij} * m_j$$

$$\text{или } \sum_{j=1}^{N} x_{ij} * I_{jk} = 0,\ k = 1,..,K,\ i = 1,..,N\ ,$$

где I_k^{min}, I_k^{max} – минимальное и максимальное процентное содержание примеси k от общей массы группы соответственно, m_j – масса партии.

Критерии оптимизации задачи (по приоритетности):

1. Максимизация количества партий:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} x_{ij} \rightarrow max$$

2. Максимизация веса вовлеченных партии

$$\sum_{i=1}^{N} \sum_{j=1}^{N} m_j * x_{ij} \rightarrow max$$

3. Максимизация веса суммы вовлеченных примесей

$$\sum_{i=1}^{N} \sum_{j=1}^{N} MV_j * x_{ij} \rightarrow max$$

2.3. Программная реализация

Zimpl - это специализированный язык (Domain Specific Language) для перевода математической модели задачи в линейную или (смешанную) целочисленную математическую программу, выраженную в формате файла lp, который может быть подан на вход SCIP [9]. Исходный код модельной задачи на языке Zimpl представлен в Приложении.

Основные используемые понятия языка Zimpl следующие. Для обозначения констант используется ключевое слово param. Константа может хранить в себе числовое значение или строковое. Массив (вектор или

матрица) констант определяется словом set. Переменные (искомые значения x_{ij}) обозначены с помощью ключевого слова var.

В реализации модельной задачи первое ограничение из п. 2.2. описано внутри конструкции subto c1, второе – внутри конструкций subto c21, subto c22, третье – внутри subto c31 и subto c32.

Для того, чтобы при решении задачи не выводились все допустимые решения, были искусственно добавлены ограничения criterion1, criterion2, criterion3 (на основе полученных ранее решений). Эти ограничения отсекают большую часть неоптимальных решений, так как SCIP выводит все возможные решения задачи.

3. Результаты исследования

3.1. Тестовый набор данных

В работе использовался тестовый набор данных, в котором N = 123 переменных (партий) и K = 11 примесей.

3.2. Вычислительная система и программное обеспечение

В процессе работы использовалась персональная ЭВМ со следующими характеристиками:

- Процессор: Intel Core i7-10700K
- Число ядер: 8
- Число потоков: 16 (технология Hyper Threading)
- Объем ОЗУ: 16 ГБ
- ОС: Ubuntu 20.04
- Версия SCIP: 8.0.0

Из практического удобства исследования установлено ограничение на максимальное время работы одного запуска - 24 часа.

3.3. Тестирование параллельной версии программы

Первый этап тестирования реализации решения задачи оптимизации сочетаний партий сырья на тестовом наборе данных проведен на разном числе потоков без использования эвристик. Результаты тестирования приведены в Табл. 4 и на Рис. 2. Результаты тестирования указаны во времени (в минутах) до достижения определенного значения gap (разрыва) при запуске основной программы на 1, 2, 4, 8 и 16 потоках. Знак «-» означает отсутствие сведений о времени достижения разрыва в связи с превышением ограничения по времени. График на Рис. 2 в графическом виде отображает информацию из Табл. 4.

Необходимо отметить, что минимальное время решения задачи при достижении наименьшего gap достигается при 8 потоках - максимальному количеству ядер на вычислительной системе. Использование технологии Hyper Threading ухудшает результаты.

Наибольшую скорость решения задачи имеет версия программы, запущенная на 8 потоков, поэтому все дальнейшие исследования были проведены именно с такими условиями.

Таблица 4. Результаты выполнения (в минутах) программы в зависимости от количества потоков вычислительной системы

gap	**1 thread**	**2 threads**	**4 threads**	**8 threads**	**16 threads**
INF	0	0	0	0	0
90%	398	186	147	15	178
80%	-	2350	165	67	235
65%	-	-	211	78	316
50%	-	-	249	112	371
30%	-	-	270	180	402
10%	-	-	292	200	415
5%	-	-	294	220	416
3%	-	-	296	230	419
1%	-	-	444	298	915
0,50%	-	-	1630	532	-

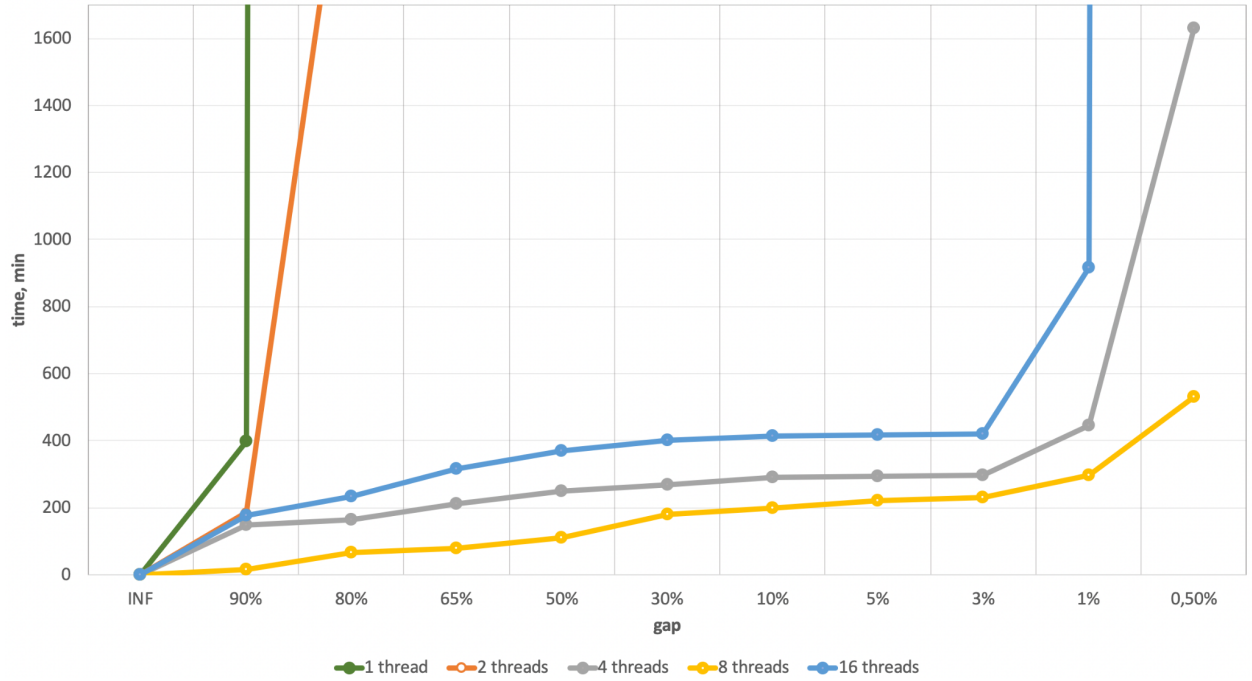

Рисунок 2. Результаты выполнения (в минутах) программы в зависимости от количества потоков

3.4. Тестирование программы с использованием эвристик

В Табл. 5 и на Рис. 3 приведены данные о тестировании программы с использованием эвристик округления.

Таблица 5. Результаты применения (в минутах) эвристик округления

gap	SimpleRounding	ZI Round	Rounding	Без эвристик
INF	0	0	0	0
4%	72	110	112	220
3%	144	139	112	230
2%	168	155	133	262
1%	202	172	147	298
0,8%	216	268	241	344

0,6%	229	293	307	498
0,50%	263	647	385	532

При использовании эвристики Simple Rounding было получено 3 допустимых решения исходной задачи. Применение эвристики ZI Round позволило получить 14 допустимых решений, а Rounding – 18. Все найденные решения здесь и далее в рамках исследования указаны при значении gap = 0.5%.

Наибольшую скорость среди используемых эвристик округления имеет эвристика Simple Rounding (Рис. 3).

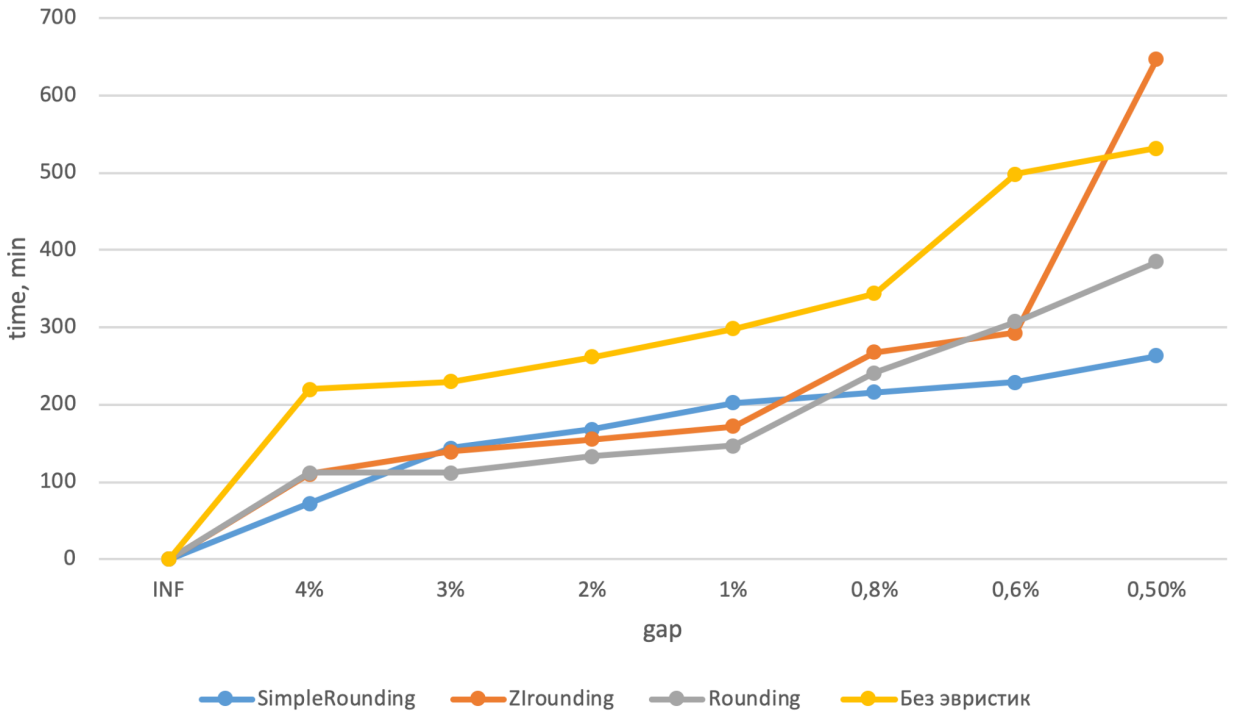

Рисунок 3. Сравнение (в минутах) эвристик округления

В Табл. 6 и на Рис. 4 приведены данные о тестировании программы с использованием эвристик погружения.

Таблица 6. Результаты применения (в минутах) эвристик погружения

gap	**CoefDiving**	**FracDiving**	**VecLenDivin**	**Без**

			g	эвристик
INF	0	0	0	0
4%	130	79	88	220
3%	133	127	143	230
2%	138	332	171	262
1%	150	350	188	298
0,8%	182	358	203	344
0,6%	204	380	215	498
0,50%	729	844	219	532

При использовании эвристики coefdiving было получено 19 допустимых решения исходной задачи. Применение эвристики fracdiving позволило получить 11 допустимых решений, а veclendiving – 12.

Наибольшую скорость среди используемых эвристик округления имеет эвристика veclendiving (Рис. 4).

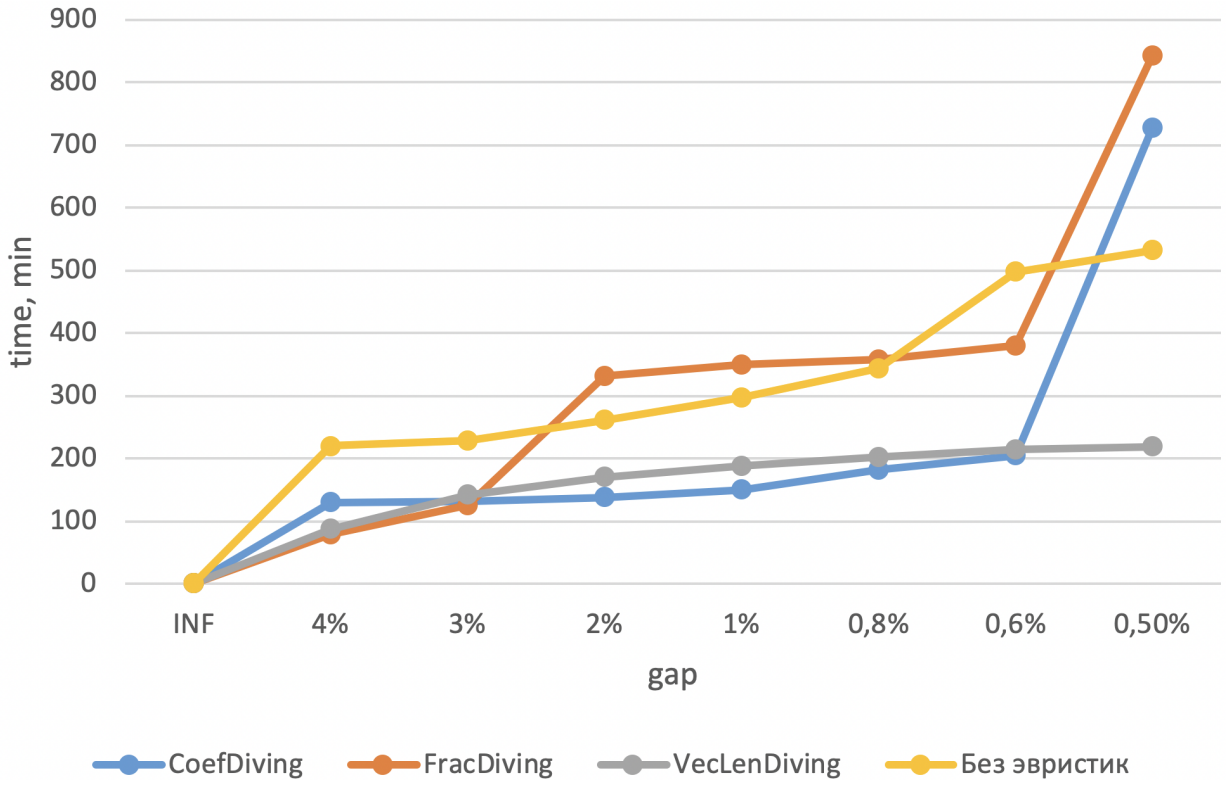

Рисунок 4. Сравнение (в минутах) эвристик погружения

В Табл. 7 и на Рис. 5 приведены данные о тестировании программы с использованием LNS эвристик.

Таблица 7. Результаты применения (в минутах) LNS эвристик

gap	Crossover	Rins	Mutation	Без эвристик
INF	0	0	0	0
4%	109	110	109	220
3%	128	128	117	230
2%	138	147	149	262
1%	146	163	189	298
0,8%	165	166	195	344
0,6%	175	199	198	498
0,50%	201	203	879	532

При использовании эвристики crossover было получено 48 допустимых решения исходной задачи. Применение эвристики rins позволило получить 32 допустимых решений, а mutation – 6.

Наименьшую скорость среди используемых эвристик округления имеет эвристика mutation (Рис. 5), две другие LNS эвристики имеют примерно одинаковую скорость преодоления определенного разрыва.

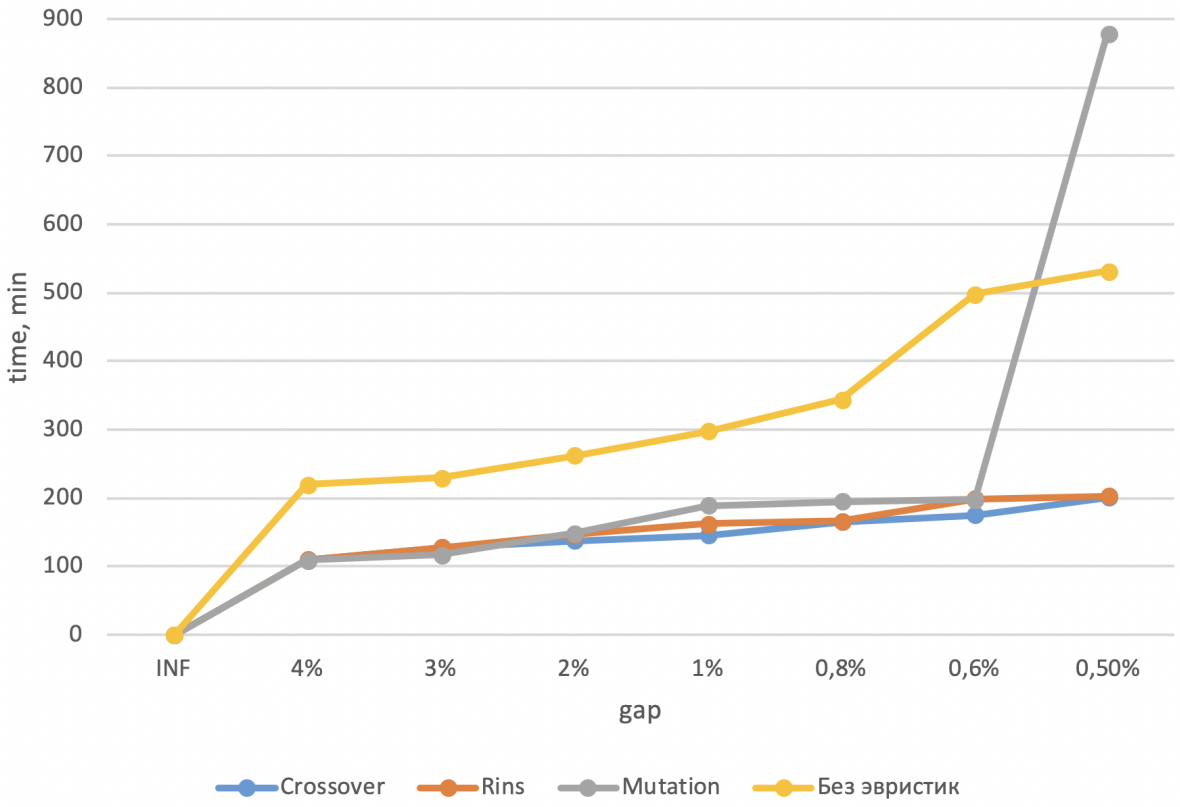

Рисунок 5. Сравнение LNS эвристик

На Рис. 6 приведены сравнения скорости достижения определенного значения gap для программы с эвристиками и без. Для каждого типа эвристик взято среднее значение результата работы программы трех эвристических алгоритмов, относящихся к этому типу. Параллельная версия – основная программа, распределенная на 8 потоков без применения эвристик. На графике видно, что применение эвристик значительно увеличивает скорость выполнения программы.

Сравнение эвристик по количеству найденных решений (при уровне gap=0.5%) приведено в Табл. 8. При этом указано время, которое затрачено для достижения заданного уровня gap. Следует отметить, что найденные решения - это все найденные решения, для которых gap больше, либо равен заданного значения; то есть среди найденных решений одни могут быть лучше, другие - хуже. Также следует отметить, что для некоторые эвристики реализуют рандомизированный подход, и от запуска к запуску найденные решения и их количество может отличаться.

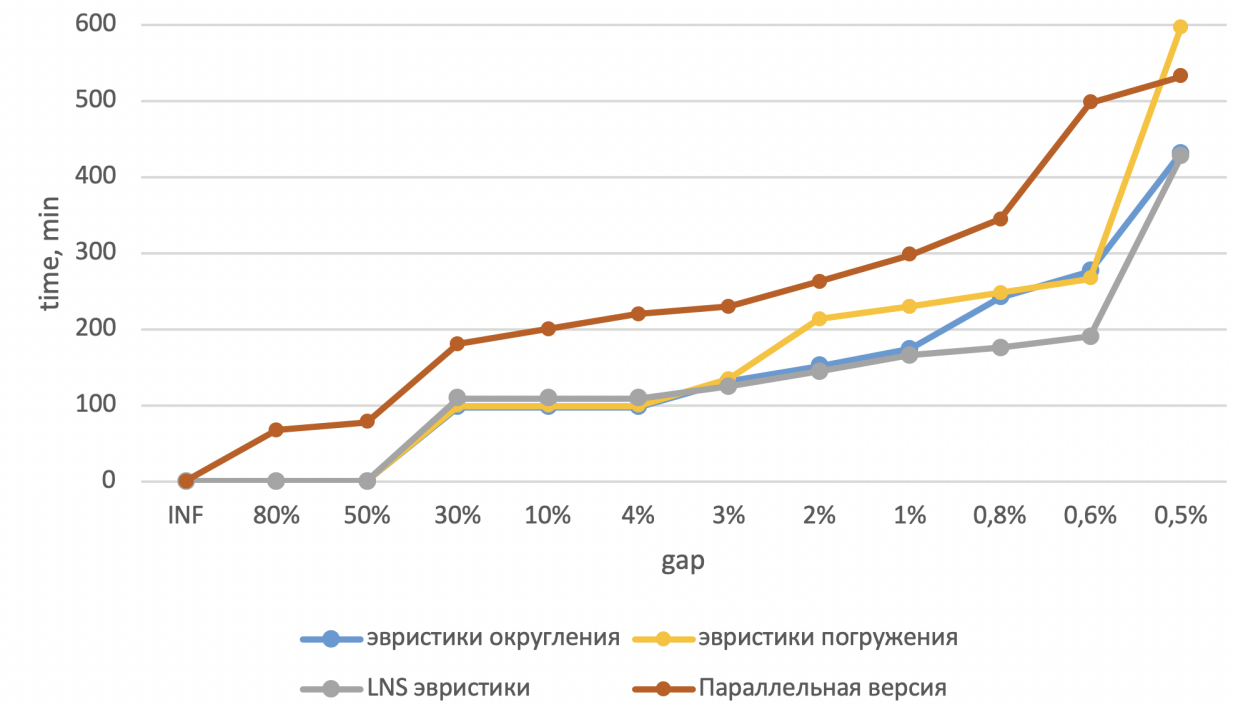

Рисунок 6. Сравнение скорости работы программы с эвристиками и без

Таблица 8. Количество найденных допустимых решений и затраченное время (при значении gap=0.5%)

Алгоритм	**Количество решений**	**Время (мин.)**
Без эвристик	9	532
SimpleRounding	3	263
ZI Round	14	647
Rounding	18	385
CoefDiving	19	729
FracDiving	11	844
VecLenDiving	12	219
Crossover	48	201
Rins	32	203

Mutation	6	879

Оптимальное решение ЗЦЛП было найдено при применении эвристики Crossover. В сравнении с запусками программы, где использовались другие эвристики, алгоритм ветвей и границ с задействованием Crossover нашел наибольшее число решений. Это связано с тем, что в исходных данных имелось множество схожих значений по параметрам (вес партии, процент примесей в составе), которые могли быть взаимозаменяемыми, что привело к близким по вычислениям целевых функций решениям. Crossover объединяет по крайней мере два известных заранее решения и сравнивает их между собой.

Примечания:

1. С использованием дефолтных установок SCIP для достижения gap=0.5% требуется более 24 (= 1440 минут) часов. Поэтому можно считать, что в результате достигнуто ускорение более, чем в 7 раз = (1440/201).

4 самых лучших решения найдены за 30 часов на запуске с дефолтными параметрами SCIP.

3.5. Рекомендации по решению модельной задачи за ограниченное время

Исследование показало, что для задачи оптимизации сочетаний при ограничении времени решения лучше всего использовать параллелизм до максимального количества ядер вычислительной системы с общей памятью, а также LNS эвристики, в частности Crossover. Такие эвристические алгоритмы дают наибольшее ускорение достижения минимального gap (разрыва) и находят большее число решений по сравнению с другими.

Используемые параметры запуска для получения решений (обозначаются в файле scip.set):

1. # the minimum number of threads used during parallel solve
   # [type: int, advanced: FALSE, range: [0,64], default: 1]
   parallel/minnthreads = 8

2. # priority of heuristic <crossover>
   # [type: int, advanced: TRUE, range: [-536870912,536870911], default: -1104000]
   heuristics/crossover/priority = 1000000

В зависимости от потребности можно изменять уровень gap (параметр запуска limits/gap, например, limits/gap = 0.5), чтобы удобно управлять качеством получаемых решений. При достижении заданного gap выполнение решателя завершается.

Заключение

В данной работе для решения задачи целочисленного линейного программирования рассмотрено программное средство SCIP. В обзоре рассмотрены все основные эвристики, имеющиеся в SCIP: округления, погружения и LNS эвристики, а также возможности параллельного выполнения программ.

В работе рассматривается модельная задача поиска оптимальных сочетаний партий сырья. Для задачи дан тестовый набор данных, состоящий из 123 переменных (партий) и 11 примесей.

Проведено исследование влияния эвристик и применения возможностей параллельных вычислений на время до достижения заданного значения разрыва (gap) между наилучшим полученным решением и решением двойственной задачи целочисленного линейного программирования.

Разработаны рекомендации по самому быстрому получению решения задачи при общем ограничении времени, которые включают в себя использование LNS-эвристики Crossover, а также использование параллельных вычислений.

Литература

1. Гераськин М.И., Клентак Л.С. Линейное программирование: учеб. пособие; – Самара: Изд-во СГАУ, 2014. – 104 с.
2. Сигал И.Х., Бабинская Я.Л., Посыпкин М.А. Параллельная реализация метода ветвей и границ в задаче коммивояжера на базе библиотеки BNB-Solver // Труды ИСА РАН. – 2006 – Т. 25. – С. 26-36.
3. Тюхтина А.А. Методы дискретной оптимизации: Часть 1: Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2014. – 62 с.
4. Achterberg T., Berthold T., Koch T., Martin A., and Wolter. K. SCIP (Solving Constraint Integer Programs), documentation. URL: https://scipopt.org/doc/html/
5. Gottwald, R.L., Maher, S.J., & Shinano, Y. (2017). Distributed Domain Propagation. SEA.
6. Hendel, G. Adaptive large neighborhood search for mixed integer programming. Math. Prog. Comp. 14, 185–221 (2022). URL: https://doi.org/10.1007/s12532-021-00209-7
7. Nair, V., Bartunov, S., Gimeno, F., Glehn, I.V., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., Addanki, R., Hapuarachchi, T., Keck, T., Keeling, J., Kohli, P., Ktena, I., Li, Y., Vinyals, O., & Zwols, Y. (2020). Solving Mixed Integer Programs Using Neural Networks. ArXiv, abs/2012.13349.
8. Timo Berthold. Heuristic algorithms in global MINLP solvers. PhD thesis, TU Berlin, 2014. URL: https://www.zib.de/berthold/Berthold2014.pdf
9. Thorsten Koch. (2020). Zimpl User Guide for Version 3.3.9. URL: https://zimpl.zib.de/download/zimpl.pdf
10. Witzig, J., & Gleixner, A.M. (2021). Conflict-Driven Heuristics for Mixed Integer Programming. INFORMS J. Comput., 33, 706-720.