Федеральное государственное бюджетное учреждение науки
Институт проблем передачи информации им. А.А. Харкевича
Российской академии наук (ИППИ РАН)

На правах рукописи

Кокорев Денис Сергеевич

МОДЕЛИРОВАНИЕ ПРОЦЕССА ВПИСЫВАНИЯ МНОГОГРАННЫХ 3D-ОБЪЕКТОВ

Специальность 05.13.18 –
математическое моделирование, численные методы и комплексы программ

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель:
доктор физико-математических наук, профессор
Афанасьев Александр Петрович

Москва – 2017

Оглавление

Введение

Актуальность темы. При решении многих прикладных задач возникает необходимость определять взаиморасположение различных объектов. Чаще всего необходимо либо расположить внутри одного объекта один или несколько других объектов, либо расположить максимальное количество заданных объектов вокруг другого так, чтобы они его касались. Наибольшее количество результатов в данной геометрической области относятся к случаям, когда одним из объектов является сфера. Например, гипотеза Кеплера, контактное число шаров, задача Таммеса, вписанная и описанная сфера около простейших объектов. Есть также доказанные результаты для простейших случаев взаиморасположения многогранников, например, Теорема Г. Хадвигера про контактное число плоских, выпуклых фигур и решение задачи Кеплера о максимальной копии правильного многогранника внутри другого правильного многогранника. В случае же взаиморасположения произвольных многогранников задача оказывается очень сложной, и не существует теоретических подходов, позволяющих подступиться к этой задаче в общем виде. Это обуславливает актуальность исследований в области взаимодействия трехмерных многогранников широкого класса.

Одной из задач в этой области является задача нахождения в трехмерном многограннике произвольной формы выпуклого многогранника наибольшего объема с заданными геометрическими свойствами. Кроме своей теоретической значимости данная задача имеет широкую область применения: ювелирная промышленность; обработка дорогостоящих материалов; приложения, решающие задачи упаковки и раскроя в трехмерном пространстве; программы, отвечающие за передвижение роботов на пресеченной местности; компьютерное моделирование взаимодействия трехмерных объектов; компьютерные игры. Важным фактором является то, что большая часть перечисленных применений требуют решения поставленной задачи за ограниченное время.

Так как не существует теоретического полиномиального алгоритма для решения этой геометрической задачи, эффективным методом является возможность свести ее к задаче нелинейного программирования и решить численными методами оптимизации. Для этого необходимо разработать и исследовать различные математические модели процесса вписывания выпуклого многогранника в невыпуклый многогранник, и выбрать модель, которая наиболее эффективно решается с помощью численных методов. Подобные исследования очень слабо освещены в литературе.

Таким образом, проблема моделирования процесса вписывания многогранных 3d-объектов является актуальной.

Предметом исследования является задача вписывания трехмерных многогранников, и алгоритмы, решающие эту задачу.

Целью работы является создание эффективной математической модели процесса вписывания выпуклого многогранника с заданными свойствами в невыпуклый многогранник, и разработка комплекса программ для численного решения прикладных задач вписывания многогранников, на основе созданной модели.

Задачи исследования:

- Разработать и сравнить различных математических моделей процесса вписывания выпуклого многогранника в невыпуклый многогранник.
- Провести исследование различных численных методов и выбрать наиболее эффективный для решения задачи вписывания выпуклого многогранника в невыпуклый многогранник.
- Разработать комплекс программ, предназначенный для численного решения прикладных задач вписывания выпуклого многогранника в невыпуклый многогранник.

Научная новизна заключается в разработке эффективной модели процесса вписывания выпуклого многогранника с заданными свойствами в невыпуклый многогранник, представляющей многогранники в виде набора вершин и граней одновременно. Определено новое, более широкое понятие вписывания многогранника в многогранник.

Практическая ценность и внедрение результатов работы. Разработанные методы моделирования и комплекс программ оказались эффективными и могут быть применены для решения широкого круга прикладных задач. С помощью них удалось создать алгоритм поиска оптимальной круглой огранки в алмазе, превосходящий по качеству все существующие подобные алгоритмы. А также создать несколько уникальных модификаций алгоритма, имеющих практическую ценность в огранке бриллиантов, и не имеющих промышленных аналогов.

Алгоритм внедрен под названием «Automatic Asymmetrical Smart Recut» в коммерческий продукт «HPOxygen» кампании «OctoNus Software Ltd» и уже используется на ограночных фабриках в Индии, России и Бельгии.

Методы исследования. В основу решения геометрической задачи вписывания многогранников легло нелинейное программирование. При решении вычислительных задач использовались численные методы нелинейной оптимизации. При разработке программного обеспечения использовались языки C++, AMPL. Система оценки производительности алгоритмов написана на языке Python.

Оценка достоверности результатов исследования выявила:

Математическая модель построена на основе обобщения известных подходов описания многогранников с использованием либо вершин, либо набора плоскостей.

Решение задачи вписывания многогранников основано на использовании известных алгоритмов, находящихся в открытом доступе

Выводы об эффективности алгоритмов получены путем проведения многочисленных воспроизводимых вычислительных экспериментов с применением современных методов и технологий и согласуются с результатами, полученными ранее по этой тематике другими авторами.

На защиту выносятся:

- Предложена математическая модель процесса вписывания выпуклого многогранника с заданными свойствами в невыпуклый многогранник, представляющая многогранники в виде набора вершин и граней одновременно, что позволяет быстрее решать задачу вписывания методом внутренней точки.
- На основании численных исследований установлено, что алгоритм на базе метода внутренней точки является наиболее эффективным для решения задачи вписывания выпуклого многогранника в невыпуклый многогранник.
- Разработанный комплекс программ для численного решения прикладных задач вписывания выпуклого многогранника с заданными свойствами в невыпуклый многогранник позволяет значительно сократить время разработки прикладного приложения за счет создания основы для задачи вписывания.
- Алгоритм поиска оптимального круглого бриллианта в алмазе, созданный на базе разработанного комплекса программ, позволяющий увеличивать массу бриллианта в среднем на 4%.

Апробация работы. Результаты диссертационной работы докладывались и обсуждались на: 55 и 56 научных конференциях МФТИ (Долгопрудный, 2012, 2013); на VI сессии научной школы-практикума молодых ученых и специалистов «Технологии высокопроизводительных вычислений и компьютерного моделирования: технологии eScience»(Санкт-Петербург, 2013); на 38-ой конференции школы ИППИ РАН Информационные технологии и системы (Нижний Новгород, 2014); на «Национальном Суперкомпьютерном Форуме» (Переславль-Залесский, 2014, 2015); на 7 международной конференции «Распределённые вычисления и Грид-технологии в науке и образовании» (Дубна,

2016); на 5th International Young Scientists Conference in HPC and Simulation (Краков, 2016).

Личный вклад автора. Все результаты диссертации, вынесенные на защиту, получены автором самостоятельно. Постановка задач и обсуждение результатов проводилось совместно с научным руководителем.

В опубликованных работах [К5] и [К8], написанных в соавторстве, содержательная часть статьи написана автором. Вклад соавторов заключался в оформлении статьи под требования издания.

Публикации. По теме диссертации опубликовано 9 работ, две из них [К2, К7] в изданиях из списка ВАК и две [К8, К9] в изданиях, индексируемых Scopus.

Структура и объем диссертации. Общий объем диссертации составляет 108 страниц, включая 8 рисунков и 4 таблицы. Работа состоит из введения, четырех глав, заключения и библиографии (41 наименование).

Глава 1. Основные направления исследований в геометрических задачах, касающихся взаиморасположения объектов

Диссертация посвящена решению задачи нахождения в многограннике произвольной формы выпуклого многогранника наибольшего объема с заданными свойствами. Многогранник, в который требуется найти другой, может быть невыпуклым. Эта геометрическая задача решается с помощью сведения ее к задаче нелинейного программирования, с последующим вычислением оптимального решения методами численной оптимизации. В такой формулировке задача является достаточно узким частным случаем. Поэтому очень мало статей, которые касаются непосредственно направления исследования диссертации. Наиболее близкой по содержанию является статья «Computing maximal copies of polyhedra contained in a polyhedra» [1]. Ее обзор приведен в разделе 1.2. Кроме нее разделе 1.2 рассмотрены также доказанные результаты для простейших случаев взаиморасположения многогранников, например, Теорема Г. Хадвигера про контактное число плоских, выпуклых фигур. В случае же взаиморасположения произвольных многогранников задача, судя по литературе, изучена очень слабо.

В более общем виде геометрическую составляющую диссертации можно сформулировать как упаковка некоторого ограниченного пространства набором фигур с заданными свойствами. В общем виде задача является очень сложной. Известных случаев решения подобных задач в истории довольно небольшое количество: восемнадцатая проблема Гильберта, задачи упаковки и раскроя, гипотеза Кеплера, задачи поиска контактных чисел шаров и других фигур. Восемнадцатая проблема Гильберта затрагивает размещение одинаковых фигур в произвольном пространстве так, чтобы они покрывали наибольший возможный объем, а также рассматривает задачу нахождения таких многогранников, которыми можно покрыть пространство полностью без пробелов. Задачи упаковки и раскроя

решают проблему размещения набора каких-то фиксированных фигур в ограниченном пространстве. Вписывание многогранника является частный случай задачи раскроя для одной фигуры. Гипотеза Кеплера и сильная проблема тринадцати сфер касаются оптимального расположения в трехмерном пространстве сфер. Задачи поиска контактных чисел затрагивают размещение объектов не в пространстве, а вокруг другого объекта, равного им, объекта. Более подробно исторические и методологические аспекты каждой из этих задач описаны в разделе 1.1.

1.1. Задачи упаковки пространства

В этом разделе описаны и проанализированы наиболее значимые достижение науки в области упаковки пространства. Большая часть описанных проблем решены только частично, так как в общей постановке являются крайне сложными, особенно если речь идет об упаковках какими-то сложными фигурами или о задачах в пространствах произвольной размерности. Поэтому в этой области есть куда стремиться, особенно если учитывать, что многие из рассмотренных задач имеют применение в промышленном производстве, новейших способах передачи информации и других современных технологиях

1.1.1. Проблемы Гильберта

Перечень своих проблем 38-ий немецкий профессор Давид Гильберт сформулировал, будучи уже известным ученым в областях теории инвариантов и теории алгебраических чисел. Это произошло 8 августа 1900 года на совместном заседании секций «истории и библиографии математики» и «преподавания и

методологии математики» Второго Международного Конгресса Математиков. Конгресс проходил с 6 по 12 августа в Париже. На Конгрессе присутствовали представители множества стран: Франции, Германии, Соединенных Штатов, Италии, Бельгии, России, Австрии, Швейцарии, Англии, Швеции, Турции, Греции, Норвегии, Канады, Японии и Мексики. Председателем Конгресса был назначен Анри Пуанкаре. Гильберт выступил с докладом «Математические проблемы» и произвел необычайный фурор среди математиков того времени.

Доклад Гильберта интересен с трех разных точек зрения. Во-первых, довольно парадоксальной является его вводная часть, в которой он рассуждает о математической строгости и уже доказанных фундаментальных теоремах, но при этом сам способ его повествования является захватывающим, лаконичным, практически литературным произведением, изобилующим массой риторических вопросов и другими совершенно не математическими способами передачи информации.

Во-вторых, в исторический период, когда математика стала достаточно сильно развита и поделилась на множество различных направлений, никто кроме него не ставил перед собой задачи, охватывающей математику в целом, а не какую-то ее отдельную отрасль. Но благодаря своему широкому кругозору Гильберт смог охватить множество областей сразу и сформулировать довольно сложные задачи в каждой из них. Как говорит Болибрух А.А. в статье «Проблемы Гильберта (100 лет спустя)» [2] эти проблемы делятся по областям математики следующим образом (Таблица 1.):

Область математики	№№ проблем
Основания математики	1, 2
Алгебра	13, 14, 17
Теория чисел	7, 8, 9, 10, 11, 12

Геометрия	3, 4, 18
Топология	16
Алгебраическая геометрия	12, 13, 14, 15, 16, 22
Группы Ли	5, 14, 18
Вещественный и комплексный анализ	13, 22
Дифференциальные уравнения	16, 19, 20, 21
Математическая физика и теория вероятностей	6
Вариационное исчисление	23

Таблица 1.

Конечно, в итоге оказалось, что часть из этих задач уже решены частично или полностью, но невозможно следить за исследованиями во всем мире, особенно в те времена, когда еще не существовало интернета.

И в-третьих, поражает результат, который был достигнут. Гильберт пытался сформулировать проблемы, которые были наиболее значимые и интересные для математиков двадцатого века. При составлении задач он пользовался принципом «все гениальное - просто», то есть понятная формулировка и решаемость задачи, но при этом выбирал достаточно сложные задачи, чтобы их было интересно решать. Как говорил Бернулли: “Как показывает опыт, ничто с такой силой не побуждает высокие умы к работе над обогащением знания, как постановка трудной и в то же время полезной задачи”. И Гильберт сформулировал именно такие задачи и добился своего,- его задачи на протяжении практически полувека являлись объектом для умственных усилий множества ученых из совершенно различных областей математики. То есть, он своим докладом, возможно, повернул развитие математики в совершенно другое русло.

18-ая проблема Гильберта[3], которая непосредственно имеет отношение к диссертации, содержит в себе три вопроса:

- Существует ли в n-мерном евклидовом пространстве только конечное число существенно различных типов групп движений с фундаментальной областью
- Существуют ли, кроме того, такие многогранники, которые не являются фундаментальными областями групп движений и с помощью которых все же возможно заполнить все пространство без пробелов соответствующим укладыванием конгруэнтных экземпляров этих многогранников
- Как можно наиболее плотным образом расположить в пространстве бесконечное множество одинаковых тел заданной формы, например, шаров заданного радиуса или правильных тетраэдров с данным ребром

Все эти три вопроса на сегодняшний день являются решенными или частично решенными. Решением первой части 18-ой проблемы Гильберта считается теорема Бибербаха[4, 5, 6]:

- Всякая n-мерная кристаллографическая группа Г содержит n линейно независимых параллельных переносов; группа G линейных частей преобразований конечна.
- Две кристаллографические группы эквивалентны тогда и только тогда, когда они изоморфны как абстрактные группы.
- При любом n имеется лишь конечное число n-мерных кристаллографических групп, рассматриваемых с точностью до эквивалентности.

Данная теорема является основой всей кристаллографической геометрии. Ее доказательство для больших размерностей является очень трудоемким. Но из нее тривиальным образом следуют множество других полезных теорем. Этот факт еще раз подчеркивает вклад проблем Гильберта в развитие математики.

На второй вопрос 18-ой проблемы Гильберта дал ответ К. Рейнгардт. Он доказал существование многогранников, которые не являются фундаментальными

областями групп движений, и укладыванием конгруэнтных экземпляров которых можно заполнить без пробелов все пространство. Но Рейнгардт не ответил на вопрос, как искать такие многогранники. Позже Б. Делоне и Н. Сандакова сделали конечный алгоритм, который находил все такие многогранники, но только дающие нормальные разбиения пространства, для заданной размерности пространства. Можно сказать, что таким образом они находят только многогранники, которые проще всего описать машинным кодом.

Третий вопрос 18-ой проблемы Гильберта считает нерешаемым в общем виде. Но существует решение данного вопроса для сфер. Оно называется гипотеза Кеплера.

1.1.2. Гипотеза Кеплера

Изначально задача, ответ на которую дает гипотеза Кеплера была придумана не самим Кеплером. Ее сформулировал сэр Уолтэр Рэли и она имела совершенно не математический, а прикладной характер. Звучала она следующим образом: разработайте формулу, которая позволяет определить, сколько пушечных ядер лежит в груде. На те времена эта задача имела как военное значение, так как знание запаса своих ядер помогало корректировать тактику боя, так и экономическое, - знание примерного количества ядер в большой куче позволяло их закупать не пересчитывая. Ответ на данную задачу дал Хэрриот, сказав, что для достаточно большой кучи самой плотной упаковкой является кубическая гранецентрированная упаковка и посчитал ее плотность – $\frac{\pi}{3\sqrt{2}} \cong 0.74048$.

Уже после этого Кеплер опубликовал аналогичную гипотезу в трактате «Новогодний подарок, или о шестиугольных снежинках» [7]. Этот трактат являлся подарком на новый 1611 год его другу Вакгеру фон Вакгенфельсу, который являлся

королевским советником. В этом трактате Кеплер рассматривает ряд занимательных вопросов:

- Почему соты имеют шестиугольную форму?
- Почему зерна граната имеет форму икосаэдра?
- Почему лепестки цветов чаще всего группируются по 5?
- Почему снежники имеют такую причудливую форму?

И в этом же скорее развлекательном, чем научном трактате Кеплер рассматривает задачу, как расположить на плоскости равные круги так, чтобы они не пересекались и заполняли наибольшую площадь. Он предлагает 2 раскладки кругов, в одной из них каждый круг соприкасается с четырьмя другими кругами и центры этих кругов образуют квадраты. В другой раскладке каждый круг соприкасается с шестью кругами и их центры образуют правильные шестиугольники. Кеплер считал, что вторая раскладка является более плотной. Элементарные вычисления плотности показывают, что плотность второй раскладки больше, чем первой, и приблизительно равна 0.9069. Более того, Кеплер предполагает, что это самая плотная из возможных раскладок. Далее, пользуясь полученным результатом, он рассматривает аналогичную задачу для шаров в пространстве. Он расставляет первый слой шаров так, что их сечения образуют в плоскости вторую двухмерную раскладку кругов. А далее накладывает аналогичные слои в выемки первого слоя сверху и снизу. В итоге получается, что по четырем направлениям в пространстве секущие плоскости дают картинку двухмерного случая. Полученная упаковка является кубической гранецентрированной упаковкой, которую ранее предложил Хэрриот. Общепринятая формулировка гипотезы Кеплера звучит следующим образом: «Никакая упаковка шаров равного размера в пространстве не имеет среднюю плотность больше, чем у гранецентрированной кубической упаковки и упаковок, равных ей по плотности» [8].

Но Кеплер не делал даже попыток доказать свою гипотезу. На протяжении трех столетий после него было сделано масса попыток найти другие укладки кругов на плоскости, имеющие большую плотность, но не одна из них не увенчалась успехом. Первым, кто попытался дать осмысленное доказательство двухмерного случая гипотезы Кеплера (для кругов на плоскости) был норвежский математик, занимающийся геометрическими методами в теории чисел, Адольф Туэ. Его первое доказательство не было опубликовано, но было представлено в 1892 году на конгрессе математиков скандинавских стран. Но оно содержало в себе значительные пробелы. Позже в 1910 году Адольф Туэ попытался опубликовать более полное доказательство, но и оно было не достаточно строгим. И только в 1940-1945 годах независимо друг от друга появилось сразу три полных доказательства этой гипотезы, найденные Ласло Фейешем Тотом, Беньямино Сегре и Куртом Малером.

Но доказательства для трехмерного случая с шарами так и не было найдено с помощью традиционных математических методов. Главная сложность доказательства гипотезы Кеплера заключается в элементарности ее формулировке. Ее очень сложно разбить на подзадачи или как-то проанализировать. Но при этом она полезна и сложна, что полностью удовлетворяло требованиям Давида Гильберта при формулировке его проблем. Значительные продвижения в доказательстве гипотезы Кеплера начались в середине XX века. Ученые двигались в двух направления в попытках отыскать доказательство. Одно направление – это сужение допустимой области задачи. Другими словами, попытаться ограничить возможную плотность упаковки сверху и постепенно довести эту верхнюю границу до желанной 0.74048. В этом направлении преуспели Рэнкин – в 1947 году доказал ограничение в ~0.83, Роджерс – в 1958 году опустил границу до ~0.78, и наконец последний результат был достигнут в 1993 году Мадером и равнялся ~0.77. Но в этом направлении так и не было получено полное доказательство. Начало второму пути доказательства положил венгерский математик Фейес-Тоз в 1953 году. Он смог свести доказательство гипотезы Кеплера к задаче оптимизации, в которой

нужно найти максимум сложной нелинейной функции от большого числа переменных. Первые компьютеры того времени не могли справиться с такими сложными задачами, но начало успешного доказательства было положено. Полным же успехом увенчалось исследование американца Томаса Хейлса. Он составил функцию от порядка 150 переменных, которую необходимо было оптимизировать, и написал свою библиотеку интервальной арифметики для обработки многочисленных ограничений. Первые результаты Т. Хейлс получил в 1994 году[9], а достиг конечной цели только 19 августа 1998 года вместе со своим учеником Сэмьюэлем Фергюсоном. То есть на то, чтобы доказать гипотезу, которая не вызывала ни у кого сомнений даже в XVII веке, потребовалось почти четыре столетия и довольно большие компьютерные мощности.

Гипотеза Кеплера является основополагающей в разделе геометрии, называемом дискретной геометрией. Не смотря на свою простоту формулировки, ее многомерный вариант имеет ряд очень важных приложений в теории связи и вычислительной математике. Это еще раз подтверждает неоценимость вклада Д. Гильберта в развитие математики, ведь, если бы он не включил ее в список своих проблем, ученые XX века могли бы и не вспомнить о столь очевидном для всех факте.

1.1.3. Контактное число шаров

Близкой по смыслу к гипотезе Кеплера является задача поиска контактных чисел шаров. Контактное число шаров размерности n – это максимальное количество шаров единичного радиуса, которые могут одновременно касаться одного такого же шара в n-мерном евклидовом пространстве, при этом не пересекающиеся друг с друга[10]. Обозначается контактное число обычно k(n), так как его английское название kissing numbers. Данная задача в общем виде является нерешенной математической задачей.

Одномерный и двухмерный случай данной задачи являются довольно очевидными, тогда как для больших размерностей задача является трудно доказуемой. В одномерном случае шаром является отрезок, у которого, как известно, есть только два конца, а соответственно и касаться его могут только два таких же отрезка. А значит k(1) = 2. В двухмерном случае шаром является круг. Каждая круг, который касается центрального круга, виден из его центра под углом 60 градусов. Разделив 360 на 60, мы получаем 6, то есть больше 6 кругов расположить нельзя, так как они должны быть видны под не пересекающимися углами. А пример, как расположить 6 кругов вокруг одного, является очевидным.

Задачу нахождения контактного числа шаров для размерности 3 называют еще проблемой тринадцати сфер[11]. Есть две красивых расстановки 12 шаров – одна из них кубическая гранецентрированная, когда в одной плоскости расставляется шесть шаров вокруг центрального, а в плоскостях снизу и сверху кладутся по 3 шара в выемки. У второй красивой расстановки центры двенадцати шаров лежат в вершинах правильного икосаэдра. То есть k(3) больше 12. Вопрос о том, каким же точно является k(3) возник очень давно. Он стал темой известной научной дискуссии между Дэвидом Грегори и Исааком Ньютоном в 1694 году. Дэвид Грегори специально прибыл в Кембридж, чтобы подискутировать с самым известным ученым того времени – Ньютоном. Ньютон утверждал, что невозможно расположить больше двенадцати шаров. В то время как Грегори считал, что можно расположить 13 шаров. Его главным аргументом было то, что каждая касающаяся сфера занимает у центрального шара сферический сектор радиусов в тридцать градусов. Если разделить площадь поверхности сферы на площадь таких секторов, то получается ~14.92, то есть у шаров есть довольно много свободного пространства. Значительно позже канадский математик британского происхождения Гарольд Скотт Макдональд Коксетер доказал, что «для 12 шаров так много пространства, что шары можно перекатывать, не отрывая от центрального, и таким образом можно переставить любые два шара местами». Первым, кто посчитал, что решил задачу тринадцати сфер, был Рейнольд Хоппе.

Это произошло в 1874 году, но доказательство было крайне сложным и запутанным, и многие ученые считали, что оно содержит ошибки. Несостоятельность этого доказательства опубликовал Томас Хейлс в статье "The status of the Kepler conjecture". Корректное же доказательство первыми дали Карл Шютте и Баартель Леенберт ван дер Варден в 1953 году. Позже более простое доказательство попробовал дать Джон Лич в 1956 году, которое позже было опубликовано в знаменитом сборнике авторов M. Aigner, G. Ziegler «Proofs from THE BOOK», но и оно оказалось довольно сложным и громоздким, поэтому во втором издании книги его убрали.

Для четырехмерного случая давно было известно расположение 24 сфер. Если взять центр внутренней сферы за (0,0,0,0). То центры 24 сфер будут лежать на векторах, у которых 2 компоненты нулевые, одна +1 и одна -1, на расстоянии, соответствующем двум радиусам сфер. Первым такое расположение для задачи поиска контактного числа предложил Госсет в статье «On the regular and semi-regular figures in space of n dimensions». В 1963 Коксетер доказал, что k(4) не больше 26, в своем доказательстве он использовал предположение, что для сферических секторов равного размера наиболее плотна упаковка, у которой триангуляция Делоне центров секторов является симплексной. Это предположение было доказано через 15 лет Берецким. В 1993 году В.-И. Цианг опубликовал доказательство того, что k(4) = 24, но оно до сегодняшнего дня не является признанным как полностью корректное. Решен же четырехмерный случай был только в 2003 году российским математиком Олегом Мусиным. В своей статье «Проблема двадцати пяти сфер» [12] он доказал, что k(4) = 24. Его доказательство опирается на модификацию метода Дельсарта.

Для больших размерностей задача решена только в размерностях 8 и 24. k(8) = 240 и k(24) = 196560. Шары располагаются в узлах решеток E8 и решетки Лича соответственно. Доказали эти факты с помощью метода Делсарта в 1979 году одновременно две группы ученых – Левенштейн В.И. в работе «О границах для упаковок в n-мерном евклидовом пространстве» и A. M. Odlyzko с N. J. A. Sloane в

работе «New bounds on the number of unit spheres that can touch a unit sphere in n dimensions». Для других размерностей существуют только доказанные верхние и нижние границы, их можно видеть в Таблице 2:

Размерность	Нижняя граница	Верхняя граница	Размерность	Нижняя граница	Верхняя граница
1	2	2	13	1154	2069
2	6	6	14	1606	3183
3	12	12	15	2564	4866
4	24	24	16	4320	7355
5	40	44	17	5346	11072
6	72	78	18	7398	16572
7	126	134	19	10688	24812
8	240	240	20	17400	36764
9	306	364	21	27720	54584
10	500	554	22	49896	82340
11	582	870	23	83150	124416
12	840	1357	24	196569	196560

Таблица 2.

1.1.4. Сферические коды

Задача расположения одинаковых шаров вокруг центрального имеет повсеместное на сегодняшний день применение в передаче и хранении данных. Впервые использовать в теории кодирования упаковки единичной сферы сферическими секторами предложил К.Е.Шэннон в 1948 году в статье «Математическая теория коммуникаций» [13]. Главная проблема, которая

решается с помощью сферического кодирования – это искажение информации при передаче данных на очень большие расстояния, например, с Земли на спутник. Каждой сфере, которая касается внутренней, соответствуют теневой сектор на внутренней сфере и точка касания между сферами. Расположения этих точек касания является символами алфавита. В каждый момент времени необходимо передать слово, состоящее из этих символов на другую сферу. Как известно при передаче информации, особенно на большие расстояния, появляются искажения. Но так как сектора имеют довольно большой размер, то даже после искажения точки касания попадают в нужный сектор, но не в его центр. Соответственно по сектору, в который попала точка, можно понять, что за символ передавался, а по всем переданным символам восстановить слово целиком. Объемы передаваемой информации постоянно растут, поэтому возникает потребность передавать слова, из большего возможного набора, то есть необходимо расширять алфавит. Для этого нужно увеличить количество круглых секторов на сферах, уменьшив их радиус до возможной погрешности при передаче. При этом нужно знать максимальный возможный размер секторов при их заданном количестве. В связи с этим в современном технологическом мире становится крайне актуальной задача Таммеса.

1.1.5. Задача Таммеса

Эту задачу сформулировал, еще до появления сферических кодов, голландский ботаник Р.М.Л.Таммес в своем труде «On the origin number and arrangement of the places of exits on the surface of pollengrains». Он изучал места выхода на поверхности сферических пыльцевых зерен и нашел некоторую закономерность их расположения. Эти места выхода представляют собой сектора сферы пыльцевых зерен, которые не могут пересекаться и имеют разный размер у разных растений. Соответственно, размер этих секторов ограничивает

максимальное количество выходов на каждом пыльцевом зерне. Таммес предположил, что у каждого растения эти сектора расположены так, чтобы их было максимальное возможное количество. В связи с этим он сформулировал вопрос: «Сколько сферических секторов, определенного размера может быть на сфере?» [14]. Но это исключительно прикладной вопрос. Если его обобщить как математическую задачу, то получается следующая формулировка: «Как должны быть упакованы N сферических секторов с равными радиусами, чтобы угловые радиусы были максимальны? И чему равен этот максимальный радиус?» Но обычно математики используют третью формулировка: «Найти расположение N точек на единичной сфере такое, чтобы минимальное расстояние между любыми двумя точками было максимально возможным». Если рассматривать угловые величины, то это расстояние соответствует расстоянию между центрами секторов, то есть угловому диаметру сектора.

Задача Таммеса решена для $N \leq 13$ и $N = 24$. Для $N = 3, 4, 6, 12$ задачу решил Л. Фейеш Тот в 1943 году. Для $N = 5,7,8,9$ - К. Шютте и Б. Л. ван дер Варден в 1951 году, для $N = 11$ первым задачу решил К. Бёрёцки в 1983, после чего Л. Данцер в 1986 году дал доказательство для $N = 10$ и 11, и для $N = 24$ задачу решил Р. М. Робинсон 1961 году. Задачу Таммеса для $N = 13$ решили О.Р. Мусин и А.С. Тарасов в 2010 году с помощью машинного перебора[15,16] и для $N = 16$ в 2015 году[17]. Их доказательство основано на построении всех возможных неприводимых контактных графов и отсев перебором графов, которые не подходят для задачи. Отсев происходит на разных уровнях приближения, чтобы оптимизировать время работы алгоритма, на каждом из уровней проверяются необходимые для решения свойства и граф отбрасывается, если какое-то из них не выполняется. Похожий алгоритм сейчас применяется для решения задач большего размера

1.1.6. Задачи раскроя и упаковки

Самыми полезными в прикладном значении задачами в данной области являются задачи раскроя и упаковки. Эти задачи уже давно используются в таких областях как: перевозка грузов, машиностроение, металлургия, деревообрабатывающая промышленность, целлюлозо-бумажная индустрия, швейная промышленность. В условиях конкурентного рынка крайне важно экономить расходные материалы, чтобы понизить себестоимость товара или увеличить прибыль. При этом постоянное появление все новых и новых технических деталей обусловливает постоянную актуальность этих задач. Упрощенно задачу раскроя можно сформулировать так: разместить определенное количество заготовок определенной формы в кусках материала так, чтобы расход материала был минимальный. А задачу упаковки так: разместить определенное количество грузов определенного размера по набору контейнеров. Их формулировки очень похожи, поэтому их часто объединяют в одну группу задач. Если их обобщить, то можно получить следующую формулировку: установите соответствие между двумя группами объектов так, чтобы какие-то свойства этого соответствия были оптимальными. Но, не смотря на свою схожесть, задачи имеют разную смысловую и прикладную нагрузку, поэтому они рознятся для конкретных прикладных случаев. От поставленной в диссертации задачи они отличаются тем, что заготовки/грузы имеют фиксированную форму и размер, а в моей задаче задается только форму, а размер нужно увеличивать до максимально возможного.

Задачи раскроя и упаковки касаются очень широкого спектра задач, у каждой из которых есть свои нюансы и прикладное значение. В статье Валиахметовой и Филипповой[18] перечислены основные задачи подобного рода:

- раскрой запаса материала (при раскрое на определенные заготовки минимизация исходного материала);

- плотное размещение геометрических объектов в заданной области (размещение грузов, товаров на складе и т.п.);
- загрузка рюкзака (выбор множества элементов с максимальной суммарной ценностью, объем которых не превосходит объема рюкзака);
- упаковка контейнеров (размещение предметов в ограниченную область);
- загрузка транспорта (размещение товаров в транспортном средстве с учетом технологичности);
- выбор ассортимента (выбор при заказе одного размера из стандартных);
- планировка помещений (максимизация полезных областей при планировании с учетом технологичных требований);
- обеспечение ритмичности производственного процесса (задачи о составлении расписания, составление расписания многопроцессорных систем);
- распределение производственных мощностей (распределение памяти вычислительной машины);
- задача о поставе в лесопилении (расположение пил при пилении бревна на доски разной толщины, выбор числа пил для максимизации выхода);
- раскрой древесного ствола по длине (максимизация продукции при раскрое ствола на круглые сортименты);
- раскрой досок (раскрой на заготовки, более близкие к окончательной продукции; обойти пороки и максимизировать суммарную кубатуру или суммарную коммерческую цену);
- раскрой листового материала на случайные заготовки (раскрой материала с учетом опережения–запаздывания производства заготовок);
- максимальное (минимальное) геометрическое покрытие (расстановка минимального количества единиц оборудования на заданной области)

Все эти задачи могут быть классифицированы по нескольким признакам. Во-первых, может меняться размерность задач. В прикладных целях чаще всего

размерности этих задач могут меняться от одного до трех, но в научных целях встречаются и такие задачи как упаковка N-мерных параллелотопов в N-мерную полу бесконечную полосу. Во-вторых, можно выделить задачи с разными способами разреза. Могут быть произвольный способ разреза, когда материал может разрезаться как угодно, и гильотинный, когда возможными являются только сквозные разрезы, параллельные кромкам материала, что значительно сокращает свободу решения. В-третьих, задачи отличаются формой деталей. Чаще всего детали имеют форму параллелотопов определенной размерности, так как это значительно упрощает сложность задачи и чаще нужно в прикладном плане. Иногда задача более общего вида сводится именно к такой. Например, при перевозке грузов на корабле клиент предоставляет товар, упакованный с ящики, и именно ящики распределяются по кораблю, а не сам товар произвольной формы. Но встречают и задачи, когда детали имеют сложную геометрическую форму, их называют проблемами нестинга. Часто встречаются детали, имеющие частично сферическую форму или прямоугольно-ориентированные заготовки. В-четвертых, задачи можно поделить по форме материала. Самые стандартные случаи – это когда материал имеет форму полосы (например, плоская лента или длинный брус), иногда предполагается, что полоса является бесконечной; или форму контейнера (область, ограниченная плоскими стенками); или форму открытой области, то есть какие-то края могут быть неровными. Также встречаются задачи, в которых материал имеет дефекты, на которые не должны попадать детали. Итак, как можно видеть задачи раскроя и упаковки имеют огромную вариативность и область применения.

Теория, строящаяся вокруг задач раскроя и упаковки, начала развиваться в 40-х года XX века. Первым ее описанным применением на практике стала статья Фельдмана Х.Л. « Система максимальных поставов на распиловку », в которой предложен математический метод, позволяющий в теории, уменьшить количество отходов при раскрое пиловочного сырья. Но это была только первая попытка улучшить производство такими математическими методами, и она не имела

большого успеха, потому что модель была чисто теоретическая и не учитывала множество факторов, возникающих на реальном производстве. Метод Фельдмана был улучшен в 1956 году Залгаллером В.А., его графический метод позволил приблизить теоретическую задачу к практическому применению.

Но основоположником этой ветви математической науки принято считать другого человека – Канторовича Леонида Витальевича. Канторович Л.В. – советский математик и экономист, один из создателей линейного программирования, лауреат нобелевской премии по экономике 1975 года за вклад в теорию оптимального распределения ресурсов. В 1939 году в работе «Математические методы организации и планирования производства» Канторович предложил метод разрешающих множителей, который позволил найти оптимальные решения для задач массового раскроя и многих других прикладных задач. Но применение данного метода вручную было очень трудоемким и практически не возможным даже для относительно легких задач, необходимо было разработать новый математический аппарат, который бы позволил решать производственные задачи. Этому и были посвящены следующие статьи Канторовича – «Рациональные методы раскроя металла» в 1942 году и «Расчет рационального раскроя материалов» в 1951 году. Вторая статья была написана в соавторстве с В. А. Залгаллером, он разработал способ подбора целочисленных индексов, также предложил использовать вспомогательную линейную задачу для случая плоского раскроя, и методы решения для задачи раскроя с материалами разной длины. В книге «Расчет рационального раскроя материалов» активно рассматривается решение задачи раскроя методами линейного программирования, когда данные задаются в неявном виде. Хорошая совместимость на практике задач раскроя и линейного программирования дала толчок развития обеим этим областям науки, так как с появлением ЭВМ ученые стремились развивать линейное программирование и делали это именно на этих задачах, так как они были необходимы в прикладных целях. В 60-ые годы было проделано множество исследований в этой области: реализация метода шкалы индексов на ЭМВ для

задач линейного раскроя Яковлевой М.А. и Булавского В.А., сеточный метод генерирования гильотинных раскроев с максимальной оценкой Мухачевой Э.А., метод склейки Романовского И.В. и другие менее значимые результаты.

Задачи упаковки и раскроя являются NP-трудными, поэтому большое внимание уделяется приближенным методам или точным методам с эффективными алгоритмами, сокращающими полный перебор. Для сокращения перебора наиболее часто используют метод ветвей и границ, предназначенный для отсекания веток перебора, на которых не может быть достигнут результат лучше, чем уже найден. Помимо него были предложены и другие метода: метод дихотомии для решения дискретных задач И.В. Романовского и Н.П. Христовой, опубликованный в 1973 году; метод поиска нижних границ Картака В.М., основанный на линейной релаксации; метод перехода к подмножеству допустимых решений, предложенный в 1986 году Яковлевым С.В. и Стоян Ю.Г.

В 70-ых, 80-ых годах было написано множество статей по использованию задач раскроя в разных отраслях промышленности, но в них в основном затрагиваются индивидуальные нюансы прикладных задач, и нет никаких выдающихся математических достижений, позволяющих решать широкий круг задач.

Для решения задач раскроя и упаковки часто используются методы целочисленного линейного программирования. Эти методы хорошо решают простые случаи либо одномерного раскроя, либо гильотинного раскроя. Для них постоянно пытаются придумать новые модели и методики ветвления, но для более сложных случаев этих методов пока что не достаточно. При решении задач раскроя больших размерностей часто применяют чисто комбинаторные методы, основанные на методах искусственного интеллекта, но и они до сих пор недостаточно эффективны, чтобы охватить весь спектр задач.

Для задач негильотинного раскроя первые серьезные результаты, позволяющие найти максимум за конечное число операций, были получены в 1985

Липовецким А.И.. Эти результаты описаны в его работе «К оптимизации свободного размещения прямоугольников». В ней задача упаковки записана в виде проблемы комбинаторной оптимизации и предложен «метод зон» для ее решения. В статье доказано, что оптимальное решение существует в классе упаковок, построенных на ступенчатых границах. И предложены методы отсечения вариантов, которые отбрасывают заведомо плохие решения и решения, похожие на уже полученные.

Также для решения NP-трудных комбинаторных задач часто используются эвристические методы. Одним из самых эффективных эвристических методов для решения задач раскроя и упаковки является метод последовательного уточнения оценок, базирующий на идее объективно-обусловленных оценок, предложенных Канторовичем. Этот метод опирается на упорядочивание элементов перебора, которое происходит по экономическому смыслу двойственных переменных. Впервые метод описан в 1993 году Мухачевой и Залгаллером. Он позволяет решить задачи гильотинного и линейного раскроя в двухмерном и трехмерном пространстве. Как оказалось позже метод последовательного уточнения оценок может быть улучшен внесением в него элементов случайности. Например, использование для поиска локального минимума таких алгоритмов как: поиск с запретами, эволюционные алгоритмы. Описание и анализ этих алгоритмов можно прочитать в труде Кочетова Ю.А. «Вероятностные методы локального поиска для задач дискретной оптимизации» 2000 года выпуска.

В XXI веке ученые стали замечать, что прикладное применение задач раскроя намного шире, чем казалось сначала. Оказалось, что к задачам раскроя можно свести практически любую задачу, в которой необходимо определить какое-то оптимальное множество объектов, покрывающее какое-либо допустимое или необходимое пространство, с учетом каких-то специальных для задач условий. Можно сказать, что это подняло новую волну исследований в данной отрасли. Задачи раскроя все еще представляют большой интерес для современных ученых, постоянно разрабатываются новые методики их решения и улучшаются старые.

1.2. Задачи взаиморасположения многогранников

1.2.1. Проблема Кеплера и Крофта

Наиболее близкие к теме диссертации результаты, рассмотренные в литературе, относятся к решению проблемы, сформулированной Кеплером и Крофтом. Эта проблема идет под пунктом B3 в книге нерешенный проблем геометрии 1991 года [19] и касается вписывания в правильный многогранник раздутых копий других правильных многогранников. Самое раннее упоминания этой проблемы припоставил Кеплер в 1619 году [20]. Он привел описание наибольшего правильного тетраэдра в кубе и наибольшего куба в додекаэдре, но не делал попытки доказать максимальность. Наибольший вклад в решение этой проблемы внес Крофт в 1980 [21]. Он привел решение для 14 из 20 случаев.

Решения для оставшихся случаев были предложены в статье 2015 года [1]. В этой статье используются методы аналогичные используемым в диссертации, но только для решения проблемы Кеплера и Крофта, то есть для правильных многогранников. В статье предлагается оптимизационная модель с квадратичными ограничениями для произвольных пар правильных многогранников. Это модель решается численно, а затем, если решение алгебраическое, точный оптимум находится с помощью алгоритмов соотношения.

Сначала описывается математическую модель, с помощью которой решается задача. Ключевым фактом в описании задачи является то, что многогранники представляются не только набором вершин, но также и набором граней. Это значительно упрощает формулировку задачи, но добавляет дополнительные переменные, относящиеся к граням. К аналогичному результату привело и диссертационное исследование. В качестве оптимизационной функции авторы предлагают брать коэффициент гомотетии, то есть во сколько раз удлиняется ребро

вписываемого многогранника по сравнению с начальным. Главными ограничениями в задачи являются условия вписывания. Считается, что один многогранник лежит в другом, если все его вершины лежат внутри пересечения граней другого. Эти условия легко записываются в случае, когда многогранники выпуклые, что выполняется для правильных фигур. Но их недостаточно для невыпуклого случая. Также ограничения значительно упрощаются, если начало координат лежит внутри обоих многогранников. Помимо этих условий необходимо записать, что многогранник сохраняет свою правильную форму, то есть расстояния между соседними вершинами остались попарно равны.

Затем. автор предлагают усовершенствовать предложенную формулировку, чтобы сократить количество переменных и квадратичных уравнений. Для того чтобы сократить количество переменных предлагается из вершин вписываемого многогранника выбрать аффинный базис, а все остальные вершины представить через линейную комбинацию базисных вершин. Подомная операция уменьшает количество переменных, но увеличивает количество линейных уравнений. Для уменьшения количества квадратичных уравнений также используется аффинный базис. После его введения нет необходимости проверять, что расстояния между всеми вершинами вписываемого многогранника остаются равными. Необходимо только проверить, что расстояния между вершинами аффинного базиса увеличилось в коэффициент гомотетии раз по сравнению с начальным. Также авторы предлагают использовать соотношение Кеплера для того, чтобы задать верхнюю границу коэффициента гомотетии и ускорить оптимизацию.

Поставленная задача численно решается на SCIP. SCIP – это программное обеспечение для решения задач смешанного численного нелинейного программирования. Этот решатель использует метод ветвей и границ. Полученное решение является первым приближением точного решения.

На самом деле система уравнений должна задаваться алгебраически и получаемое решение также должно быть алгебраическим. Но такую систему практически невозможно решить вручную или на компьютере. Поэтому для

получения точного решения автор использует следующую трёхшаговую процедуру:

1) На основании уже полученных результатов автор дополняет начальную систему условиями, что некоторые вершины вписываемого решения лежат строго на гранях внешнего многогранника. Для случая правильных тел количество таких вершин может быть вычислено. После этого такую расширенную систему численно решают с очень большой точностью, применяя метод ньютона, используя при этом предыдущее решение в качестве начальной точки.

2) Затем для каждой полученной вершины угадываются алгебраические числа, близкие к полученному численному значению, с помощью алгоритмов соотношений. Эту процедуру возможно сделать, если алгебраические числа состоят из многочленов с маленькой степенью и маленькими коэффициентами.

3) И затем полученные алгебраические числа подставляются в настоящую систему, чтобы проверить результат.

С помощью описанного в статье метода, автор подтверждает результаты, полученные Крофтом для 14 случаев, и находит решения для 6 нерешенных ранее случаев. Для вычислений был использован SCIP версии 3.1.0. с заданной точность 10^{-10}. Вычисления проводились на одном ядре Xeon CPU 3GHz, с использованием меньше 8Gb оперативной памяти. В зависимости от случая вычисления длились от нескольких минут до нескольких часов. В разделе 3. Автор приводит таблицы с полученными коэффициентами гомотетии и подробно останавливается на 6 ранее не решенных случаях.

Существуют также доказанные результаты для задач, похожих на проблему Кеплера и Крофта другой размерности. Задача вписывания правильных многоугольников рассматривается в статьях [22] и [23]. Максимальный коэффициент гомотетии известен только, если количества углов рассматриваемых

многоугольников не являются взаимно простыми. Для взаимно простого случая известен только предполагаемый результат [24].

Похожая проблема нахождения наибольшего, не обязательно простого, симплекса размерности i в кубе размерности j связана с матрицами Хардамара и рассматривается в статье [25]. В некоторых случаях оказывается, что решением является плавильный симплекс [26]

1.2.2. Контактные числа многогранников

Вполне естественно, что задача о контактных числах возникла не только для сфер, но и для других фигур. В общей постановке она звучит следующим образом: пусть F – произвольная фигура, чему равно наибольшее количество K(F) равных F фигур, которые можно расположить так, чтобы они касались F и не пересекались друг с другом. Обозначения взяты из книги «Проблема тринадцати шаров»[11]. Эта формулировка является обобщением задачи контактного числа шаров. Так как речь идет о произвольной размерности пространства и о произвольной фигуре, то эта задача является очень сложной и не существует подходов, способных решить ее в общем виде. Одно из возможных упрощений этой задачи – это ограничить фигуры тем, что они должны быть параллельны центральной фигуре, то есть получаются из F не произвольным движением, а параллельным переносом. Это значительно упрощает задачу и переводит ее в область афинной геометрии, а не евклидовой, что позволяет использовать более простые методы для ее решения. Будем обозначать контактное число в таком случае k(F). Так же можно ограничить задачу другим движением, то есть фигуры получаются из F не параллельным переносом, а каким-то другим движением G, в таком случае контактное число обозначается k(F,G). Очевидно, что $k(F) \leq K(F)$, так как любое решение более легкой задачи является также и решением более сложной. Так же понятно, что задача о контактном числе сфер входит в категории упрощенной задачи, так как равные сферы всегда

получаются друг из друга с помощью параллельного переноса. Вторым естественным упрощением задачи является ограничение на выпуклость фигуры F. С выпуклыми фигурами значительно проще работать, и они обычно чаще нужны в прикладных целях.

Первыми шагом в нахождении контактных чисел не для сфер стала теорема Гуго Хадвигера. В 1957 году он доказал, что для плоских выпуклых фигур верно, что $6 \leq k_2(F) \leq 8$, при этом он предположил, что $k_2(F) = 8$ только для параллелограммов. И его гипотеза оказалась верной, позже в 1961 году эту гипотезу независимо друг от друга доказали Гельмут Грёмер и Бранко Грюнбаум. Причем Б. Грюнбаум решил задачу точно, он доказал, что $k_2(F) = 8$ верно для параллелограммов, а для остальных фигур $k_2(F) = 6$. Также в 1959 году было получено нижнее ограничение и для невыпуклых фигур, математики К.Дж.Хальберг-младший, Е. Левин и Е.Г.Страус доказали, что $k_2(F) \geq 6$. В своем доказательство теоремы Хадвигер по сути еще сильнее упрощает задачу. Он доказывает ее для случая, когда фигура является центрально симметричной, после чего приводит лемму, о том, что для любой плоской выпуклой фигуры F существует центрально симметричная фигура F*, что $k_2(F) = k_2(F^*)$

Для задачи нахождения $K_2(F)$ результаты были получены результаты только для правильных n-угольников. Первым без доказательств их опубликовал Л.Фейеша Тот в 1967 году, он предположил, что для правильных n-угольников $K_2(My_n) = 12$ при $n = 3$, равна 8 при $n = 4$, и равна 6 при $n > 4$. Все эти утверждения доказал венгерский геометр К. Бёрёцки в 1971 году, кроме случая $n = 5$. Доказательства этих предположений базируются на проведении специальной ломаной линии на расстоянии 0.5 от центральной фигуры и вычислении какова длина части этой ломаной, которую может отсечь прикладываемая фигура. Поиск точного значения $K_2(F)$ для фигур общего вида не имеет смысла, потому что для очень длинных фигур $K_2(F)$ может быть сколь угодно велико. Но существует достижение даже в этом направлении. Л. Фейеш Тот установил максимальную границу для $K_2(F)$ для плоских выпуклых фигур, она равна $(2\pi + 4)c + \frac{1}{c} + 2$, где c –

это соотношение максимального опорного расстояния к минимальному опорному расстоянию.

Для пространств размерности больше, чем 2, задача является намного более сложной, и ею мало кто занимался. Каких-либо результатов по нахождению $K_n(F)$ нет. Самый же сильный результат в $k_n(F)$ принадлежит Б. Грюнбаумону. Он доказал следующую оценку для n-мерных выпуклых тел: $n^2+n \leq k_n(F) \leq 3^n - 1$. Причем нижнее значение n^2+n достигается для n-мерного симплекса, а верхнее значение $3^n - 1$ достигается для n-мерного параллелотопа.

Глава 2. Численное решение задачи вписывания выпуклого многогранника в невыпуклый многогранник

Первоначальным этапом алгоритма, решающего задачу вписывания многогранника, является нахождение стартовой точки - начального приблизительного расположения внутреннего многогранника. В качестве стартовой точки могут быть взяты данные работы других неточных алгоритмов. Одним из примеров таких алгоритмов является гомотетичное раздувание под разными углами и с центром масс в разных точках искомого объекта до пересечения с внешним многогранником и нахождение, таким образом, стартовой точки с максимальным объемом, но строго зафиксированной формой, подобной некоторому эталону. Далее необходимо, шевеля вершины на заданное расстояние, найти положение с максимальным объемом и удовлетворяющее заданным геометрическим ограничениям на форму.

2.1. Используемая терминология

Определение 1.1. *Комплексом* называется конечная совокупность K многогранников в E_d, удовлетворяющая условиям: 1) наряду с каждым многогранником M из семейства K в K входит также и любая грань многогранника M; 2) пересечение любых двух многогранников из K является гранью каждого из них[27].

Определение 1.2. Максимальная размерность многогранников из K называется *размерностью комплекса*, k-мерный комплекс называется k-комплексом.

Определение 1.3. Пусть M — d-многогранник (размерности d) в E_d , и пусть целое число k удовлетворяет условию $0 < k < d$. Множество всех граней многогранника M размерности, не превышающей k, является комплексом, который называется *k-скелетом многогранника M*.

Определение 1.4. (d - 1)-скелет многогранника M будем обозначать символом $F(M)$ и называть *граничным комплексом многогранника*.

Определение 1.5. Два комплекса K и K' называются *изоморфными комплексами*, если между ними существует взаимно однозначное отображение φ, сохраняющее операцию включения:

$$F_1 \subset F_2 \Leftrightarrow \varphi(F_1) \subset \varphi(F_2)$$

Определение 1.6. Будем говорить, что два многогранника обладают одинаковой *комбинаторной структурой* тогда и только тогда, когда изоморфны их граничные комплексы. Такие многогранники называются *комбинаторно эквивалентными*. Пример таких многогранников на Рисунке 1.

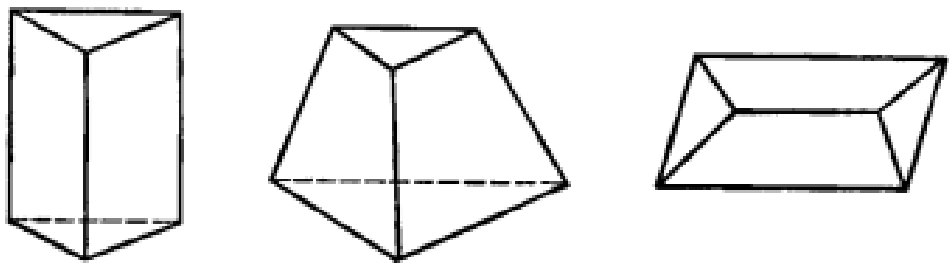

Рисунок 1.

Определение 2.1. *Вписыванием* многогранника A в многогранник B будем называть нахождение многогранника, комбинаторно эквивалентного многограннику A, имеющего максимальный объем и содержащегося в многограннике B.

Определение 2.2. Многогранник A будем называть внутренним

Определение 2.3. Многогранник B будем называть внешним

Определение 3. *Задачей нелинейного программирования (НП-задача)* называется оптимизационная задача следующего вида:

$f(x) \rightarrow min$

при ограничениях:

$gL \leq g(x) \leq gU$

$xL \leq x \leq xU$

где $x \in \mathbb{R}^n$ – оптимизационные переменные с верхними и нижними ограничениями:

$xL \in (\mathbb{R} \cup \{-\infty\})^n$, $xU \in (\mathbb{R} \cup \{+\infty\})^n$;

$f : \mathbb{R}^n \rightarrow \mathbb{R}$ - целевая функция;

g: $\mathbb{R}^n \rightarrow \mathbb{R}$ - общие нелинейные ограничения с верхними и нижними границами:

$gL \in (\mathbb{R} \cup \{-\infty\})^m$, $gU \in (\mathbb{R} \cup \{+\infty\})^m$;

Определение 4. *AMPL* [28] (*A Modeling Language for Mathematical Programming*) — язык программирования высокого уровня для описания и решения сложных задач оптимизации и теории расписаний. Главным преимуществом AMPL является подобие его синтаксиса математической записи задач оптимизации, что позволяет дать очень краткое и легко читаемое определение задач математического программирования.

Например, задача:

$f(x) = x_1 + x_2 \rightarrow min$

при ограничениях:

$4 \leq x_1 * x_2 \leq +\infty$

$0 \leq x_1 , x_2 \leq 3$

будет выглядеть на AMPL следующим образом:

var x1 >= 0, <= 3; #задание первой переменной

var x2 >= 0, <= 3; #задание второй переменной

minimize f: x1 + x2; #целевая функция

*con1: x1 * x2 >= 4; #ограничение с именем con1*

Для решения задач, написанных на AMPL, используются вычислительные солверы.

Определение 5. *Нелинейный солвер* – программа для решения задач нелинейного программирования, использующая один из методов оптимизации.

Определение 6. *Ipopt* [29](Interior Point Optimizer)– открытый программный пакет для решения НП-задач большой размерности. Этот солвер предназначен для поиска локального оптимума в задаче нелинейного программирования методом внутренней точки[30].

Определение 7. *Выпуклой оболочкой* множества X называется наименьшее выпуклое множество, содержащее X.

Определение 8. *Смешанным произведением* $(\vec{a}, \vec{b}, \vec{c})$ векторов $\vec{a}, \vec{b}, \vec{c}$ называется скалярное произведение вектора $\vec{a}$ на векторное произведение векторов $\vec{b}$ и $\vec{c}$.

$$(\vec{a}, \vec{b}, \vec{c}) = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix}$$

Ориентированный объём симплекса в трехмерном евклидовом пространстве можно определить по формуле:

$V = \frac{1}{6}\, det(v_1 - v_0, v_2 - v_0, v_3 - v_0)$, где v_i – вектора координат вершин симплекса.

2.2. Математическая модель процесса вписывания многогранников

2.2.1. Метод колебания вершин

Даны два многогранника – внешний и начальное приблизительное расположение внутреннего. Начальное расположение внутреннего многогранника получено алгоритмом вписывания, который не умеет менять форму внутреннего многогранника. Оно содержится внутри внешнего многогранника и имеет объем близкий к максимальному. Требуется, изменяя положение вершин, увеличить внутренний многогранник так, чтобы он осталась вписанной во внешний многогранник, сохранил свою комбинаторную структуру, и удовлетворял заданным геометрическим ограничениям.

Поставленную геометрическую задачу необходимо записать в терминах задачи нелинейного программирования. Целевой функцией *f(x)* является объем искомого внутреннего многогранника. Он вычисляется через координаты вершин следующим образом. Берется произвольная внутренняя точка многогранник. Грани внутреннего многогранника разбиваются на треугольники. После чего объем представляется в виде суммы объемов симплексов, натянутых на эти треугольники и внутреннюю точку. Объем симплексов считается с помощью смешанного произведения и является многочленом третьей степени.

Переменные в задаче делятся на 2 типа. Первый тип – параметры граней внутреннего многогранника y_{ap} , y_{bp} ,y_{cp} , y_{dp} ,где p – номер грани внутреннего многогранник. В качестве ограничений (xU и xL) на переменные первого типа берутся начальные значения параметров граней плюс/минус некоторая величина. Размер этой величины зависит от того, насколько разрешается изменять углы нормалей граней заданной комбинаторной структуры. Параметры граней внешнего

многогранника не меняются, поэтому рассматриваются как константы. Второй тип переменных – это параметры, отвечающие за расположение вершин внутреннего многогранника в пространстве, $x_{ji} \in (-\infty, +\infty)$, где j – номер вершины, i – ось координат. Если в качестве начальной точке берется решение довольно точного алгоритма, то для ускорения работы на координаты вершин можно наложить ограничения такие, чтобы вершины не удалялись от начального состояния дальше, чем на некоторую величину *dx*.

Есть несколько основных групп ограничений *g(x)* для данной задачи. Первая группа - ограничения на сохранение комбинаторной структуры внутреннего многогранника:

$$y_{ap}x_{j1} + y_{bp}x_{j2} + y_{cp}x_{j3} - y_{dp} = 0$$

где p – номер грани внутреннего многогранника, j – номер вершины, индексы *1,2,3* соответствуют трем осям координат. Рассмотренная группа ограничений записывается для всех пар вершина-грань, для которых верно, что вершина лежит на грани. Перечень всех таких пар получается из начального представления граней внутреннего многогранника. Или напрямую из комбинаторной структуры, если она задана в каком-то явном виде.

Вторая группа - ограничения на выпуклость искомого внутреннего многогранника. Необходимо записать для всех пар вершина-грань внутреннего многогранника, для которых вершина не принадлежит грани:

$$y_{ap}x_{j1} + y_{bp}x_{j2} + y_{cp}x_{j3} - y_{dp} \leq 0$$

Можно записывать эти ограничения только для всех пар соседних граней. Потому что попарная выпуклость всех соседних граней является достаточным условием выпуклости всего многогранника.

Последняя группа обязательных условий связана с тем, что внутренний многогранник должен остаться вписанным во внешний многогранник. Эти условия записываются в следующем виде:

$$A_r x_{j1} + B_r x_{j2} + C_r x_{j3} - D_r \leq 0$$

где r - номер грани внешнего многогранника, а j – номер вершины внутреннего многогранника. Стоит заметить, что A_r ,B_r ,C_r ,D_r являются константами, а не переменными; а значит, эта группа ограничений является линейными и их количество меньше влияет на время работы солвера. Тем не менее все равно стоит, исходя из начального положения и dx, рассчитывать какие вершины внутреннего многогранника теоретически не могут во время оптимизации выйти за грани внешнего многогранника и не писать лишние ограничения.

Для случая, когда внешний многогранник является невыпуклым, необходимы дополнительные ограничения, отделяющие внутренний многогранник от невыпуклых частей. Сначала строится выпуклая оболочка внешнего многогранника. Для описанных ранее ограничений используются плоскости этой выпуклой оболочки. Далее необходимо вычислить какие грани внешнего многогранника являются невыпуклыми. Это делается, используя критерий, что грань является невыпуклой, если в многограннике есть две вершины, лежащие по разные стороны от этой грани. Далее для каждой такой невыпуклой грани необходимо создать свою разделяющую плоскостью. Разделяющая плоскость задается свободными переменными y_a, y_b, y_c, y_d. На эту разделяющую плоскость накладываются ограничения, что все вершины невыпуклой плоскости, привязанной к ней, лежат по одну сторону от нее. А все вершины внутреннего многогранника лежат по другую сторону от нее. Такие уравнения задаются для каждой невыпуклой грани внешнего многогранника. Если для вершин внутреннего многогранника используется максимальное отклонение от начального положения dx, то можно отделять разделяющей плоскостью не все вершины вписываемого многогранники, а только вершины тех граней, которые могут пересечься с невыпуклой гранью внешнего многогранника. Зная значение dx не сложно вычислить набор таких граней для каждой невыпуклой грани внешнего многогранника. Для упрощения работы солвера можно дополнительно задать ограничение, что длина нормали разделяющей плоскости близка к 1.0.

Помимо перечисленных ограничений нужно также описать все необходимые геометрические ограничения на форму внутреннего многогранника.

2.2.2. Альтернативный метод колебания вершин

Был также исследован альтернативный метод представления многогранников в виде уравнений. Он может быть использован для более простых геометрических задач. Преимуществом данного метода является минимальное возможное количество используемых переменных. Это делает матрицы производных меньшего размера, но при этом они являются менее разреженными. Переменными являются только координаты вершин многогранника, их достаточно, чтобы записать все базовые ограничения. Данный подход не применим, если для описания геометрических ограничений на форму внутреннего многогранника требуются переменные, отвечающие за уравнения граней.

Если использовать в качестве переменных только координаты вершин, то ограничения на сохранение комбинаторной структуры и выпуклость необходимо записывать по-другому.

Если в грани было три вершины, то комбинаторная структура для нее не может изменить. Если же больше, то требуется наложить ограничения. Условие того, что четыре точки лежат в одной плоскости равносильно тому, что объем натянутого на них симплекса равен нулю, то есть в нерасписном виде условие выглядит так:

$$\det (x_1 - x_0, x_2 - x_0, x_3 - x_0) = 0$$

где x_0 , x_1, x_2, x_3 – точки, лежащие на одной грани. Для лучшей точности это условие необходимо записать для всех четверок вершин во всех гранях, но это сильно замедлит время вычислений солвера, поэтому можно оптимизировать работу исходя из того, что некоторые наборы точек лежат именно в одной грани.

Поэтому можно в каждой грани брать фиксированные три точки и добавлять к ним все остальные по-очереди.

При данном подходе единственным способом задания ограничений на выпуклость многогранника является выписывание условий локальной выпуклости для каждой из вершин. Для этого необходимо для каждой вершины брать тройки смежных с ней по ребрам вершин и смотреть объем натянутого на них симплекса. Если вектора к этой тройке вершин образуют правую декартову систему координат и начальная вершина является выпуклой, то объем симплекса будет больше нуля, если же точка не выпукла, то объем отрицательный. Исходя из этого, получаются следующие условия:

$$\det(x_1 - x_0, x_2 - x_0, x_3 - x_0) \geq 0$$

только теперь уже x_1, x_2, x_3 – точки смежные по ребрам с x_0. Если было введено ограничение на сдвиг вершин dx, то для ускорения по времени можно вычислить какие вершины не могут стать невыпуклыми и не писать для них ограничения

Не смотря на сокращение количества переменных этот метод является неэффективным, потому что возникает очень много сложных ограничений третьего порядка, что значительно замедляет работу солвера.

2.2.3. Оптимизационный подход поиска начальной точки

Были проведены исследования по разработке алгоритма, работающего на базе НП-задачи, не нуждающегося в начальной точке. Первым этапом исследования стало написание алгоритма, который находит оптимальное расположение для внутреннего многогранника, подобного эталонному. То есть находит начальную точку для метода колебания вершин.

Есть начальные координаты вершин эталонного многогранника v_i. Они только задают форму будущего решения, сам многогранник может быть любого размера

и иметь любое расположение по отношению к внешнему многограннику. Конечные координаты $v1_i$ записываются в виде ограничений $g(x)$ следующим образом:

$$\overline{v1_\iota} = \mathbf{R} * \bar{v}_\iota * zoom + \overline{trans}$$

Где $zoom$ – это произвольный коэффициент увеличения, $\overline{trans}$ – это произвольный вектор трансляции, **R** – произвольная матрица поворота, заданная через кватернионы.

$$\mathbf{R} = \begin{bmatrix} 1 - 2q_j^2 - 2q_k^2 & 2(q_i q_j - q_k q_r) & 2(q_i q_k + q_j q_r) \\ 2(q_i q_j + q_k q_r) & 1 - 2q_i^2 - 2q_k^2 & 2(q_j q_k - q_i q_r) \\ 2(q_i q_k - q_j q_r) & 2(q_j q_k + q_i q_r) & 1 - 2q_i^2 - 2q_j^2 \end{bmatrix}$$

Кроме этих уравнений необходимо задать, что сумма квадратов кватернионов равна единицы.

Таким образом, мы задаем, что эталонный многогранник может подвергаться повороту, увеличению и трансляции, и занимать любое расположение в пространстве. Также необходимо задать ограничения на нахождение внутреннего многогранника внутри внешнего. Целевой функцией является объем, выраженный через переменные $v1_i$, аналогично методу колебания вершин.

На исследуемых данных этот алгоритм дает объем в среднем на 0.4% больше, чем его аналоги, используемыt для получения начальных точек в прикладных задачах. То есть методы численной оптимизации являются более надежными для получения начальной точки для метода колебания вершин. Но алгоритм работает в среднем 41 секунду, что является слишком большим временем для прикладных целей. В то время как аналоги работают в среднем 10-15 секунд.

В дальнейшем планируется протестировать другие способы задания преобразования пространства, например, использовать матрицу поворота, выраженную через углы Эйлера или через базисные вектора. Также планируется

настроить солвер Ipopt специально под конкретную задачу, это может дать существенное ускорение в несколько раз. Если получится улучшить алгоритм, работающий с подобными многогранниками, до необходимых рабочих характеристик, то будут проведены работы по скрещиванию его с методов колебания вершин для получения полностью автономного алгоритма для задачи вписывания многогранника.

2.3. Клиент-серверный и локальный подходы в реализации численных методов

Алгоритм был реализован в клиент-серверном виде в исследовательских целях и локальном виде в коммерческих целях. Для клиент-серверной реализации алгоритма были использованы сторонние программные ресурсы, организованные с помощью RESTful-веб-сервисов [31]. Такой подход позволяет не тратить время на разработку сложных математических алгоритмов и интерфейсов взаимодействия, а использовать уже готовые решения, тем самым сконцентрироваться на поставленной задаче. О таком подходе вы можете узнать из статьи [32].

2.3.1. Клиент-серверное приложение

Клиент-серверное приложение состоит из четырех основных этапов работы, они показаны на Рисунке 2. оранжевыми блоками:

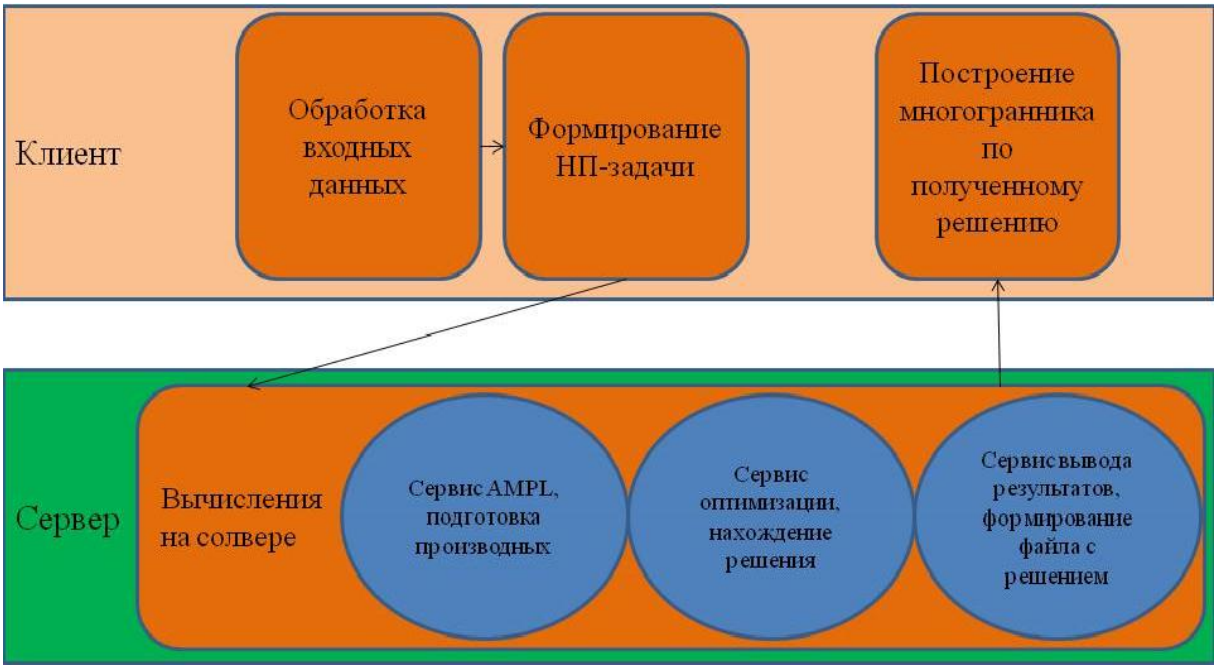

Рисунок 2.

Первый этап заключается в преобразовании входных данных, которые могут иметь разные форматы, в необходимый для дальнейшего вычисления формат. На втором этапе исходная геометрическая задача формулируется в терминах нелинейного программирования по методу колебания вершин. После чего на третьем этапе нелинейный солвер решает поставленную НП-задачу. Третий этап происходит на сервере и является для пользователя черным ящиком. На самом же деле на нем последовательно запускаются несколько программ, организованных с помощью RESTful-веб-сервисов. После завершения работы солвер выдает набор переменных, определяющих оптимальное решение, по которым на четвертом этапе строится искомый многогранник.

Для вычислений сформулированной НП-задачи в качестве сервиса оптимизации мной был использован солвер IPOPT[29](Internal Point OPTimization). Этот солвер предназначен для поиска локального оптимума в задаче нелинейного

программирования методом внутренней точки. Для вычисления солверу необходимо предоставить функции обратного вызова, вычисляющие следующие величины:

- значения целевой функции *f(x)*;
- градиента целевой функции $\nabla f(x)$;
- вектора значений вектор-функции ограничений *g(x)*;
- якобиана вектор-функции ограничений $\nabla g(x)^{\mathrm{T}}$
- гессиана расширенной функции Лагранжа $\sigma_f \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x)$

Если для формулировки НП-задачи использовать язык программирования AMPL[28], то он сам предоставляет эти функции при получении *f(x)* и *g(x)*. Таким образом, нужно позаботиться только о формулировке НП-задачи в формате AMPL.

Плюсом клиент-серверного подхода является то, что можно одновременно обрабатывать сколько угодно задач при наличии соответствующих ресурсов. Кроме того, лицензия на коммерческое использование AMPL и библиотек Ipopt на единый вычислительный сервер выйдет дешевле, чем лицензии на большое количество локальных приложений.

2.3.2. Локальное приложение

Локальное приложение или другими словами коробочная версия программы, которая устанавливается на персональный компьютер, нужна, потому что большая часть клиентов программы находятся в малоразвитых странах, где нет стабильного доступа к серверу через интернет. AMPL является платным и дорогостоящим для коммерческого использования в локальных приложениях, поэтому был разработан и реализован на C++ интерфейс для работы с солвером Ipopt напрямую, без использования AMPL. Были написаны соответствующие callback-функции,

упомянутые выше, для узкого множества функций ограничений. Эти функции имеют вид многочленов третьего порядка:

$$\sum c_i x_{i1} x_{i2} x_{i3} + \sum c_j x_{j1} x_{j2} + \sum c_k x_k$$

Также были разработаны интерфейсы для функций sin и cos

Матрицы производных для таких ограничений записываются легко, в отличие от производных для задачи общего вида. Для базового алгоритма без дополнительных ограничений достаточно такого представления для целевой функции и функций ограничений. Более того, большинство геометрических ограничений могут быть аппроксимированы в таком виде с достаточно хорошей точностью.

Целевая функция *f(x)* и ограничения *g(x)* хранятся в специальном виде и Ipopt на каждой итерации обращается к callback-функциям, написанным на C++, которые быстро вычисляют необходимые производные в текущей точке (Рисунок 3). Важно минимизировать алгоритмическую сложность вычисления callback-функций, так как эти функции вызываются очень много раз.

Рисунок 3.

Ipopt довольно требователен к процессорному времени. Поэтому одновременно на компьютере можно обсчитывать количество задач равное количеству ядер (удвоенное количество на ядрах с multithreading). Если одновременно поставить больше задач, результаты будут приходить значительно медленнее. Можно настроить Ipopt, чтобы одна задача решалась одновременно на нескольких процессорах, но это дает очень маленький коэффициент производительности, поэтому в условиях большого количества маленьких задач это не имеет смысла.

2.3.3. Тестирование различных сборок Ipopt

В своей работе Ipopt использует различные библиотеки для решения систем линейных уравнений и BLAS'ы (библиотеки базовых операций линейной алгебры). На раннем этапе развития алгоритма были проведены эксперименты по запуску задач вписывания многогранников без геометрических ограничений на разных сборках Ipopt, включающих в себя разные комбинации библиотек. Некоторые из этих библиотек могут работать в многопоточном режиме, поэтому отличается еще и количество потоков. Результаты приведены в Таблице 3. Как можно видеть, лучше всего себя показали варианты, использующие библиотеку MA57.

BLAS	Библ. СЛУ	Число потоков	Среднее общее время решения	Среднее число итераций
mkl_par	ma27	1	17,72	68
mkl_par	ma27	4	17,87	68
mkl_par	ma27	8	17,91	68
mkl_par	ma57_1	1	4,01	68

mkl_par	ma57_1	4	4,12	68
mkl_par	ma57_1	8	4,05	68
mkl_par	ma57_2	1	4,14	68
mkl_par	ma57_2	4	4,22	68
mkl_par	ma57_2	8	4,15	68
mkl_par	ma86	1	195,82	560
mkl_par	ma86	4	118,09	527
mkl_par	ma86	8	124,92	577
mkl_par	mumps	1	249,03	128
mkl_par	mumps	4	436,39	261
mkl_par	mumps	8	436,66	261
mkl_par	wsmp	1	7,14	68
mkl_par	wsmp	4	6,26	68
mkl_par	wsmp	8	6,74	68
mkl_seq	ma27	1	17,7	68
mkl_seq	ma27	4	17,63	68
mkl_seq	ma27	8	17,73	68
mkl_seq	ma57_1	1	4,04	68
mkl_seq	ma57_1	4	4,01	68
mkl_seq	ma57_1	8	4	68
mkl_seq	ma57_2	1	4,15	68

mkl_seq	ma57_2	4	4,17	68
mkl_seq	ma57_2	8	4,18	68
mkl_seq	ma86	1	194,69	560
mkl_seq	ma86	4	8,47	68
mkl_seq	ma86	8	135,83	642
mkl_seq	mumps	1	249,01	128
mkl_seq	mumps	4	249,43	128
mkl_seq	mumps	8	248,93	128
mkl_seq	wsmp	1	7,33	68
mkl_seq	wsmp	4	6,29	68
mkl_seq	wsmp	8	6,65	68
netlib	ma27	1	17,69	68
netlib	ma57_1	1	4,94	68
netlib	ma57_2	1	4,89	68
netlib	ma86	1	26,82	87
netlib	mumps	1	419,69	132
openblas	ma27	1	18,43	68
openblas	ma27	4	19,17	68
openblas	ma27	8	19,37	68
openblas	ma57_1	1	4,25	68
openblas	ma57_1	4	4,69	68

openblas	ma57_1	8	4,6	68
openblas	ma57_2	1	4,51	68
openblas	ma57_2	4	4,83	68
openblas	ma57_2	8	4,9	68
openblas	ma86	1	188,34	516
openblas	ma86	4	138,48	570
openblas	ma86	8	155,12	627
openblas	mumps	1	102,78	73
openblas	mumps	4	124,52	81
openblas	mumps	8	124,29	81
openblas	wsmp	1	7,74	68
openblas	wsmp	4	7,02	68
openblas	wsmp	8	7,56	68

Таблица 3.

Библиотека MA57 при коммерческом использовании платна. Причем лицензия довольно дорогая и штучная на каждое рабочее место, что делает ее использование не всегда возможным. Поэтому отдельно был проведен эксперимент по сравнению MA57 с бесплатными в коммерческом использовании MA27 и MUMPS при использовании бесплатного OPENBLAS и одном потоке.

Исследования проводились на двух примерах. В первом примере (пометка /1 на горизонтальной оси) начальное положение внутреннего сногогранника значительно ближе к оптимальному решению, чем во втором (пометка /2 на горизонтальной оси). Алгоритм запускался на разных настройках, которые отличаются количеством дополнительных геометрических ограничений. На

графике 1. на горизонтальной оси отмечено количество ограничений в НП-задаче, сформулированной на AMPL.

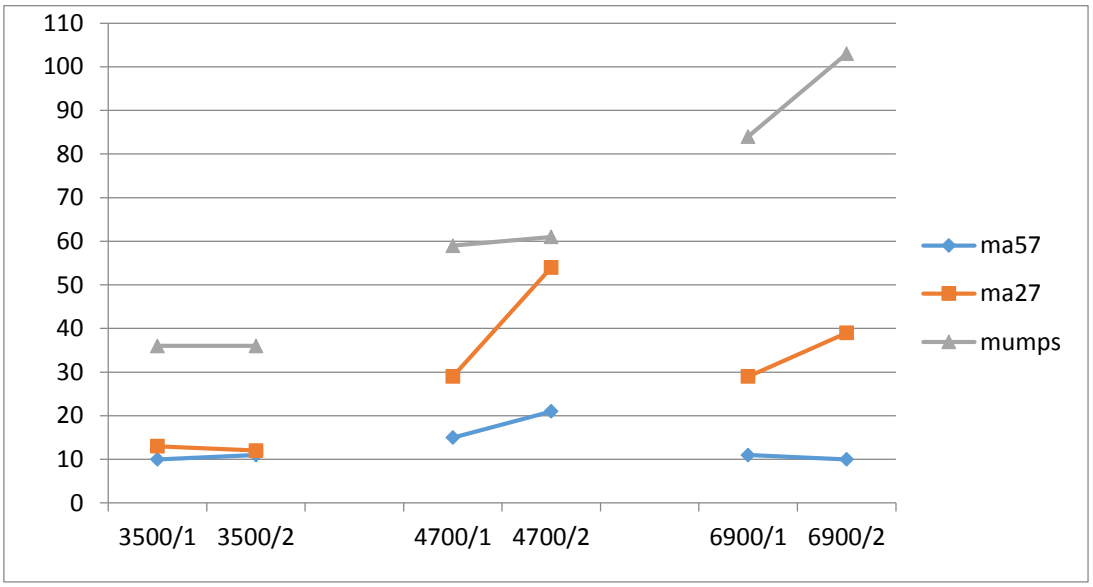

График 1.

На одних и тех же задачах разные модификации солвера дают одинаковые численные результаты, но довольно значительно отличается время работы. Как можно видеть из графика, модификация солвера с библиотекой MA27 на данной задаче работает быстрее, чем с MUMPS, но медленнее, чем с MA57. Также можно заметить, что на примере, в котором начальная точка далека от оптимума, MA27 может давать довольно плохие результаты по времени, близкие к времени с использованием MUMPS.

2.4. Принципы задания геометрических свойств внутреннего многогранника

Сложность составления ограничений заключается в том, что понятная физическая величина не всегда тривиальным образом представляется в виде уравнений. Зачастую для вычисления величины нужны условные операторы или

циклы, поэтому иногда приходится прибегать к аппроксимации или нетривиальному представлению. Иногда численный метод решения позволяется задавать вычисление каких-то величин не стандартным методом, что часто приводит к ускорению оптимизации. Численные методы позволяют программисту задавать не то, как надо вычислять величину. А то, каким свойствам эта величина должна удовлетворять в результате.

Кроме того, в арсенале исследования есть только многочлены третьего порядка, sin и cos. Можно писать функции обратного вызова и для других функций. Но, как показывает практика, Ipopt наиболее быстро работает с уравнениями в виде многочленов. Причем, чем меньше порядок, тем лучше. В случае с геометрическими объектами многие характеристики легко представляются в необходимом виде.

Приведенные ниже, казалось бы, элементарные аппроксимации позволяют работать с достаточно большой для прикладных задач точностью, при этом используя только уравнения третьего порядка, что значительно сокращает время работы оптимизации по сравнению с использованием уравнений высокого порядка, корней и обратных тригонометрических функций.

2.4.1. Обход простейших сложностей

Не смотря на то, что этого лучше избегать, может возникнуть необходимость в уравнения большего, чем третий порядок. Данная проблема решается методом введения дополнительных переменных, равных произведению двух или трех других, что позволяет понизить порядок уравнений.

Также достаточно часто возникает ситуации, когда нужно работать с соотношением двух переменных “x / y”. Для таких ситуаций необходимо завести переменную z, записать уравнение $z*y - x = 0$ и использовать z там, где нужно

соотношение. Желательно также записать уравнения *y > 0* или *y < 0,* если известно, что *y* является знакопостоянной величиной.

Достаточно часто может потребоваться операция взятия корня от числа. Не смотря на то, что функции корня в явном виде нет, его достаточно легко получить с помощью введения дополнительной переменной. Важно понимать, что уравнения решаются численно, а не путем последовательных вычислений. Рассмотрим на примере квадратичного корня. Недоступное уравнение $y = \sqrt{x}$ эквивалентно следующей системе

*y*y = x*

y >= 0

x >= 0

Аналогично можно реализовать и корни другой степени. Важно не забывать указывать как переменные, участвующие в операции взятия корня, должны соотноситься с нулем. Иначе система может быть не однозначна или не совместна

2.4.2. Основные типы ограничений

Самые распространенные геометрические ограничения можно разбить на два типа: ограничение на среднее значение группы объектов и ограничения на разброс значений в группе объектов. Ограничения на среднее значение вырождаются в ограничения на отдельный объект, если группа состоит из одного объекта. Ограничения на разброс в частности отвечают за симметричность и за равенство каких-либо величин.

1) Ограничения на разброс значений в группе объектов

Необходимо ограничивать разброс значений в группе *A* на величину *dev*, где *A* - какой-то перечень значений. Это тождественно ограничению на *max{A}* -

min{A}. Можно ввести переменные *min* и *max* и для всех c_i записать следующие уравнения:

$min < A_i$; $max > A_i$; $max - min < dev$

Записанные так ограничения позволяют не искать точные *min* и *max,* а также избежать необходимости сравнивать числа попарно. И то, и другое плохо влияет на время работы оптимизации

2) Ограничение на среднее значение группы

Необходимо ограничить среднее значение в группе *A* в диапазоне [*dev_min, dev_max*]. Тут нет никакой сложности, нужно только помножить на n обе части, *n* – заранее известный константный размер группы

$avg * n = \sum_{i=0}^{n} Ai$

$dev\_min <= avg <= dev\_max$

Можно также привести ограничения для среднеквадратичного:

$avg * avg * n = \sum_{i=0}^{n} (Ai * Ai)$

$avg >= 0$

$dev\_min <= avg <= dev\_max$

2.4.3. Точная нижняя и верхняя грань множества

Может возникнуть необходимость посчитать точную, например, верхнюю грань множества значений в группе *A*. Это задача для целочисленного программирования, но ее формально можно записать и в терминах нелинейного программирования. Но это очень сильно увеличивает время работы оптимизации. У этой системы существует единственное решение, при этом в переменной sup получается именно точная верхняя граница

$sup >= A_i$

$\sup = \sum_{i=0}^{n}(Ai * Bi)$

$1 = \sum_{i=0}^{n} Bi$

$0 <= Bi <= 1$

2.4.4. Основные геометрические объекты

Чаще всего, когда нужны дополнительные геометрические ограничения, эти ограничения касаются либо соотношений расстояний, либо соотношений углов. Все базовые геометрические объекты (измеряемые величины) можно разбить по типам на взаимодействия вершина с вершиной, вершина с гранью, грань с грань. К взаимодействиям вершина с вершиной относится расстояние между вершинами. Для получения расстояния нужно использовать замену корня на эквивалентную систему:

$r > 0$

$r^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$

Таким образом в переменной r, находится искомое расстояние, которое можно использовать в дальнейших ограничениях.

Рассмотрим взаимодействия с гранью. Грани заданы в виде параметров плоскостей y_{ap}, y_{bp}, y_{cp}, y_{dp}, где p – номер грани внутреннего многогранник, а также ограничениями на комбинаторную структуру. Остановимся на взаимодействиях с гранью, как на взаимодействиях с ее плоскостью или нормалью. Рассмотрим взаимодействия с гранью 0, заданную параметрами y_{a0}, y_{b0}, y_{c0}, y_{d0}. Для дальнейшего удобства для всех граней внутреннего многогранника, или хотя бы для всех граней, участвующих в дополнительных ограничениях стоит записать ограничения на единичную нормаль:

$y_{ap} * y_{ap} + y_{bp} * y_{bp} + y_{cp} * y_{cp} = 1$

Рассмотрим основные взаимодействия вершины с плоскостью. Если выбрана какая-то плоскость, то у вершины появляется понятие высоты или расстояния до этой плоскости:

$h = y_{a0}x_1 + y_{b0}x_2 + y_{c0}x_3 - y_{d0}$

Через высоту можно построить проекцию вершины на плоскость. Если для дальнейших вычислений нужна только проекция вершины, а высота является промежуточной величиной, то лучше не вводить дополнительную переменную h, а сразу подставить ее в выражение для проекции:

$x_{proj1} = x_1 - h * y_{a0}$

$x_{proj2} = x_2 - h * y_{b0}$

$x_{proj3} = x_3 - h * y_{c0}$

Если есть две A и B, соединенные вектором a, и есть плоскость 0 с единичной нормалью n_0. То с помощью скалярного произведения можно посчитать разницу высот этих точек по отношению к плоскости 0. Или, что тоже самое, проекцию вектора a на направление n_0

$dh = (a, n_0)$

Рассмотрим основные взаимодействия плоскости с плоскостью. По-прежнему есть плоскость 0 с единичной нормалью n_0, заданная параметрами y_{a0}, y_{b0}, y_{c0}, y_{d0}. Относительно нее любая плоскость 1 имеет две основные характеристики, - это угол между нормалями и азимут проекции нормали 1 на плоскость 0.

Для ограничений на углы между нормалями проще всего использовать косинусы углов, вычисляемые, через скалярное произведение при условии, что нормали единичные.

$cosa = (n_0, n_1)$

Большую часть ограничений на углы можно записать в виде ограничений на их косинусы. Это позволит работать только с многочленами, не применяя функцию *cosa = cos(a)* для нахождения самого угла *a*, которая замедляет работу оптимизации. Но для очень маленьких углов необходимо использовать синусы углов вместо косинусов, так как косинус не чувствителен к малым изменениям в районе нуля.

$sin\ x \sim x,$

$cos\ x \sim 1 - x^2$

в районе нуля. Синус вычисляется через длину векторного произведени\, которая также является уравнением третьего порядка.

Рассмотрим теперь способ работы с азимутом проекции нормали. Можно посчитать эту величину в явном виде: взять проекцию нормали, взять эталонный вектор в плоскости 0, относительно которого считаются азимуты, посчитать через скалярное произведение косинус между проекцией и эталонным вектором, после чего вычислить непосредственно сам азимут. Но такой способ работает медленно. Лучше использовать непосредственно проекции нормалей и ограничивать углы между ними. Для удобства необходимо, чтобы проекции были единичной длинны. Основной интерес представляет способ эффективного вычисления проекции единичной длинны. Проекцию произвольной длины записать уравнениями легко:

$$y_{proj\_a} = y_a - (y_{a0}y_a + y_{b0}y_b + y_{c0}y_c) * y_{a0}$$

$$y_{proj\_b} = y_b - (y_{a0}y_a + y_{b0}y_b + y_{c0}y_c) * y_{b0}$$

$$y_{proj\_c} = y_c - (y_{a0}y_a + y_{b0}y_b + y_{c0}y_c) * y_{c0}$$

Логичным способом после этого является вычислить длину вектора y_{proj} , добавить новые переменные для отнормированного вектора проекции и сделать нормировку. Но численное решение позволяют избежать добавления новых переменных. Приведенную систему нужно заменить на следующую:

$$norma * y_{proj\_a} = y_a - (y_{a0}y_a + y_{b0}y_b + y_{c0}y_c) * y_{a0}$$

$$norma * y_{proj\_b} = y_b - (y_{a0}y_a + y_{b0}y_b + y_{c0}y_c) * y_{b0}$$

$$norma * y_{proj\_c} = y_c - (y_{a0}y_a + y_{b0}y_b + y_{c0}y_c) * y_{c0}$$

$$y_{proj\_a} * y_{proj\_a} + y_{proj\_b} * y_{proj\_b} + y_{proj\_c} * y_{proj\_c} = 1.0$$

При этом саму переменную *norma* никак вычислять не надо.

Глава 3. Реализация комплекса программ для численного решения задачи вписывания выпуклого многогранника в невыпуклый многогранник

На языке программирования C++ реализован комплекс программных модулей для численного решения задачи вписывания выпуклого многогранника с заданными свойствами в невыпуклый многогранник. Результатом работы программы является либо решение, полученное на солвере Ipopt, установленном на компьютере, либо файл с описанием задачи на языке AMPL, который может быть запущен на удаленном солвере, поддерживающем работу с AMPL'ом. Общая длина кода около 8000 строк. Код разбит на отдельные модули, которые могут быть использованы по отдельности или вместе. Наиболее актуальная версия программы разрабатывалась для задачи раскроя алмазов, поэтому в названиях часто встречается терминология для многогранников из этой области:

- *diamond* – начальное расположение внутреннего многогранника
- *original_stone* – внешний многогранник
- *stone* – выпуклая оболочка внешнего многогранника
- *inclusions* – внутренние невыпуклости внешнего многогранника
- *result* – полученное решение задачи

3.1. Модули передачи и хранения данных

На вход программе необходимо дать начальное расположение внутреннего многогранника, внешний многогранник, и включения - внутренние невыпуклости внешнего многогранника. Для чтения из файлов и записи в файлы многогранников разработан модуль *CConvector*, отвечающий за операции экспорта и импорта. Внутри программы многогранники являются элементами класса *ClPolyhedron*, для

работы с которым разработан модуль *CPolyhedron.* В разделах 3.1.1 и 3.1.2. более подробно описано содержание этих двух модулей.

3.1.1. Модуль экспорта и импорта многогранников

Модуль *CConvector* содержит в себе единственный одноименный класс *CConvector*, отвечающий за работу с разными форматами. Он содержит в себе 6 основных открытых методов:

- *CPolyhedron readOFF(char* adress)*
- *CPolyhedron readMMEv1(char* adress)*
- *CPolyhedron readMMEv2(char* adress)*
- *void writeOFF(CPolyhedron polyhedron, char* adress)*
- *void writeMMEv1(CPolyhedron polyhedron, char* adress)*
- *void writeMMEv2(CPolyhedron polyhedron, char* adress)*

и 4 закрытых:

- *unsigned int intRead(ifstream& file)*
- *double doubleRead(ifstream& file)*
- *void intWrite(ofstream& file, int number)*
- *void doubleWrite(ofstream& file, double number)*

Первые 6 открытых методов отвечают за чтение из трех разных форматов в экземпляр класса *ClPolyhedron* и запись из экземпляра класса *ClPolyhedron* в эти форматы. Формат .off(Object File Format) является одним из стандартных форматов для описания 3D-объектов, в нем хранятся координаты вершин (*vertices*) и

комбинаторная структура многогранника в виде массивов индексов вершин, лежащих на гранях (*faces*). Формат .mme первой версии похож на .off, только имеет бинарное представление записи. Формат .mme второй версии аналогичен первой версии, только помимо передачи граней в виде массива *faces*, передает также и точные параметры граней в виде массива *planes*. Закрытые методы используются при работе с .mme форматами и отвечают за чтение и запись чисел, записанных в бинарном виде.

Кроме этих основных функций, отвечающих за экспорт и импорт, класс *CConvector* содержит методы для вывода данных (многогранников, отдельных вершин, отдельных граней) в формате .off в раскрашенном виде. Эти методы помогают отлаживаться при работе с геометрическими объектами за счет визуализации возникшей проблемы.

3.1.2. Модуль работы с геометрическими объектами

Модуль *CPolyhedron* предназначен для работы с многогранниками в трехмерном пространстве. Он содержит в себе класс *ClPolyhedron*, который является непосредственным представлением многогранника, а также вспомогательные классы, отвечающие за представление и работу с более простыми геометрическими объектами, такими как: точка, вектор, грань, ребро, плоскость.

Класс для объекта точка *ClPoint* состоит из:

- *double x* – координата по оси *X*
- *double y* – координата по оси *Y*
- *double z* – координата по оси *Z*

- *double distance(ClPoint p) const* – метод для нахождения расстояния между двумя точками

Класс для объекта вектор *ClVector* состоит из*:*

- *double x* – координата по оси *X*
- *double y* – координата по оси *Y*
- *double z* – координата по оси *Z*
- *double length* – длина вектора

- *void create(ClPoint p1, ClPoint p2)* – метод, создающий вектор по двум точкам
- *void lengthCalculate()* – метод, вычисляющий длину вектора
- *double lengthGet()* – метод, вычисляющий и возвращающий длину вектора
- *void norm()* – метод, нормирующий вектор
- *double scalarProduct(ClVector v)* – метод, возвращающий скалярное произведение двух векторов
- *ClVector vectorProduct(ClVector v)* – метод, возвращающий векторное произведение двух векторов
- *double mixedProduct(ClVector v1, ClVector v2)* – метод, возвращающий смешенное произведение трех векторов
- *double sin(ClVector v)* – метод, возвращающий синус угла между двух векторов
- *double cos(ClVector v)* – метод, возвращающий косинус угла между двух векторов

Плоскость в программе задается параметрами уравнения $ax + by + cz – d = 0$. Класс для объекта плоскость *ClPlane* состоит из:

- *double a* – параметр плоскости
- *double b* – параметр плоскости
- *double c* – параметр плоскости

- *double d* – параметр плоскости

- *void create(ClPoint p1, ClPoint p2, ClPoint p3)* – метод создающий плоскость по трем точкам
- *double distance(ClPoint p) const* – метод, возвращающий расстояния от точки до прямой

Некоторые переменные класса, отвечающего за ребро, привязаны к конкретному многограннику, они используются для представления комбинаторной структуры многогранника. В остальном ребро – это отрезок между двумя точками. Класс для объекта плоскость *ClEgde* состоит из:

- *int v0* – индекс нулевой точки ребра в многограннике
- *int v1* – индекс первой точки ребра в многограннике
- *int f0* – индекс одной грани, образующей ребро многогранника
- *int f1* – индекс другой грани, образующей ребро многогранника
- *ClPoint p0* – нулевая точка ребра
- *ClPoint p1* – первая точка ребра
- *ClPoint pmiddle* – середина ребра
- *double length* – длина ребра

- *void pmiddleCalculate()* – вычисление середины ребра
- *void lenghtCalculate()* – вычисление длины ребра
- *double distance(ClPoint point) const* – расстояние от отрезка до точки
- *double height_distance(ClPoint point) const* – расстояние от точки для прямой, содержащей отрезок
- *double distance(ClEdge edge) const* – расстояние от отрезка до отрезка

Класс для объекта многогранник *ClPolyhedron* состоит из:

- *int NVertices* – количество вершин многогранника
- *int NFaces* – количество граней многогранника
- *int NEdges* – количество ребер многогранника
- *ClPoint InteriorPoint* – центр масс вершин многогранника
- *vector<ClPoint> vertices* – массив вершин
- *vector<ClPlane>planes* – массив плоскостей граней
- *vector<ClEdge> edges* – массив ребер

- *vector<vector<int>> faces* – массив граней, представленных через вектора номеров вершин, принадлежащих грани
- *vector<vector<int>> edgesOfFaces* – массив граней, представленных через вектора номеров ребер, принадлежащих грани
- *vector<vector<int>> points* – массив вершин, представленных через вектора номеров граней, на которых они лежат

- *vector<int> nonconvex_vertices* – номера невыпуклых вершин
- *vector<int> nonconvex_faces* – номера невыпуклых граней
- *vector<int> nonconvex_edges* – номера невыпуклых ребер

- *void set_nonconvex_faces()* – метод, вычисляющий невыпуклые грани
- *ClPoint findInteriorPoint()* – метод, вычисляющий *InteriorPoint*
- *ClPoint faceCentrMass(int number)* – метод, вычисляющий центр масс грани
- *void createNorms()* – метод, создающий массив *planes* по массивам *vertices* и *faces*
- *void checkNorms()* – метод, поворачивающий нормали массива *planes* наружу многогранника

- *void gen_points()* – метод, создающий массив *points* по массиву *faces*
- *void EdgesOfFaces()* – метод, создающий массив *edgesOfFaces* по массиву *edges*
- *int findEdge(int i, int j) const* – метод, находящий номер ребра, образованного двумя гранями

Кроме классов, характеризующих геометрические объекты модуль *CPolyhedron* содержит классы *ClMatrix3x3* и *ClMatrix2x2*, которые отвечают за работу с матрицами 3 на 3 и 2 на 2 соответственно. Матрицы необходимы для работы с поворотами пространства.

1.3. Модуль обработки входных и выходных данных

За обработку входных и выходных данных отвечает модуль *IpoptStarter*. Основную роль в нем играет одноименная функция:

void ipoptstarter(const ClPolyhedron &diamondIn, const ClPolyhedron &stoneIn, ClPolyhedron &result, const ClPolyhedron &original_stoneIn, const vector<ClPolyhedron>& InclusionsIn, int &status, std::string dirStr, IpoptConfig& iConfig, OxygenData& oData,TimeCounter &tCounter)

Кроме необходимых многогранников *diamondIn, stoneIn, original_stoneIn, InclusionsIn* функция получает следующие параметры:

- *int &status* – сюда будет записан статус оптимизации, если требуется найти решение на локальном солвере Ipopt
- *ClPolyhedron &result* – сюда будет записан результат оптимизации, если требуется найти решение на локальном солвере Ipopt
- *std::string dirStr* – адрес заранее созданной директории в Temp, по этому адресу пишутся временные файлы, отладочная информации, и задача в

формате AMPL, если она необходима как результат работы программы. Вспомогательная функция удаляет все ненужные временные файлы из директории по окончанию работы

- *TimeCounter &tCounter* – класс для учета времени работы разных частей программы
- *IpoptConfig& iConfig*, *OxygenData& oData* – два класс для передачи дополнительных настроек запуска оптимизации. В первом передаются настройки, отличающие текущий запуск от других. Во втором - какие-либо дополнительные данные о многогранниках, поступающие извне.

Функция *ipoptstarter* проверяет корректность входных данных, после чего масштабирует и поворачивает многогранники. Для этого предназначены вспомогательные функции *zoom_in* и *translation*. Дело в том, что от поворота и масштаба многогранников решение задачи не меняется. При этом на некоторых задачах меняется время работы солвера, особенно от масштабирования, так как настройки солвера заданы в абсолютных величинах. Соответственно масштабирование может влиять на точность и время работы солвера, что особенно важно, если солвер удаленный и нет доступа к его настройкам. На следующем шаге *ipoptstarter*, заполняет данные о многогранниках (поля класс *ClPolyhedron*), которые необходимы для вычислений, но не содержатся в файлах передачи многогранников. После этого вызывается функция *generate_nlp()*. *generate_nlp()* составляет задачу нелинейного программирования, которую необходимо решить. Эта задача либо записывается в AMPL файл функцией *NLP_Problem::print_ampl_NLP(const char* address)*, либо запускается на расчет установленному солверу Ipopt, предварительно задав его настройки. Настройки солвера необходимо подбирать индивидуально под конкретную задачу, согласно требованиям к точности решения и времени работы. После того, как солвер закончил работать, в *status* записывает статус решения, решение преобразуется в *ClPolyhedron* и передается в *result*, который в дальнейшем может быть передан куда-либо или сохранен в файл.

3.2. Модуль взаимодействия с солвером

Как уже было сказано выше, был разработан интерфейс для работы с солвером Ipopt напрямую, без использования AMPL. Программная реализация этого интерфейсного модуля состоит из двух частей *CNLP* и *CNLPFunc*. *CNLP* содержит классы общего назначения, необходимые для взаимодействия с Ipopt. А в *CNLPFunc* хранится реализация тех выражений, которые можно использовать в задаче нелинейного программирования.

CNLP состоит из трех классов:

- *class NLP_Problem : public Ipopt::TNLP* – класс, в котором содержится вся решаемая задача нелинейного программирования. Именно через этот класс идет взаимодействие в Ipopt. НП-задача записана удобным для программирования способом, о котором библиотеке Ipopt ничего не известно, поэтому класс содержит также набор специальных стандартных инструкций для Ipopt о том, как он должен с ней работать. В теоретической части эти инструкции назывались функциями обратного вызова
- *class Constraint* – класс для представления ограничений *g(x)*. Ограничения состоят из выражения и ограничений на него.
- *class Expression* – класс, который является выражением для ограничений. Наследники класса *Expression* описываются в *CNLPFunc*

Класс *NLP_Problem* состоит из:

- *int n_var* – количество переменных в задаче
- *vector< pair<Number,Number> > bounds_x* – ограничения на переменные
- *Terms f_terms* – многочлен третьей степени, являющийся целевой функцией
- *vector<const Expression*> g_expr* – выражения ограничений *g(x)*
- *vector< pair<Number,Number> > bounds_g* – границы ограничений *g(x)*

• *vector<Number> start_x* – начальные значения переменных

• *vector<Number> result_x* – значение вектора переменных, являющееся решением задачи

• *int jac_nnz* – количество ненулевых членов якобиана

• *int hes_nnz* – количество ненулевых членов гессиана

• *Triangular_matrix jac_map* – структура компактного хранения якобиана

• *Triangular_matrix hes_map* – структура компактного хранения гессиана

• *void set_bounds_x(int ind, Number lower, Number upper)* – метод, устанавливающий границу на переменную с номером *ind*

• *void set_start_x (int ind, Number xx)* – метод, устанавливающий начальное значение переменной с номером *ind*

• *void push_f_term(Number c, int i1, int i2 = -1, int i3 = -1)* – метод для формирования целевой функции

• *void set_constraints(list<Constraint>& cnst)* – метод, устанавливающий ограничения *g(x)* на основании сформированного списка *cnst*

• *void print_NLP(const char* address)* – метод, печатающий НП-задачу в файл в удобном для чтения формате

• *void print_ampl_NLP(const char* address)* – метод, печатающий НП-задачу в файл в формате AMPL

Ниже описаны функции обратного вызова:

• *virtual bool get_nlp_info(Index& n, Index& m, Index& nnz_jac_g, Index& nnz_h_lag, IndexStyleEnum& index_style)*

• *virtual bool get_bounds_info(Index n, Number* x_l, Number* x_u, Index m, Number* g_l, Number* g_u)*

• *virtual bool get_starting_point(Index n, bool init_x, Number* x, bool init_z, Number* z_L, Number* z_U, Index m, bool init_lambda, Number* lambda)*

• *virtual bool eval_f(Index n, const Number* x, bool new_x, Number& obj_value)*

- *virtual bool eval_grad_f(Index n, const Number* x, bool new_x, Number* grad_f)*
- *virtual bool eval_g(Index n, const Number* x, bool new_x, Index m, Number* g)*
- *virtual bool eval_jac_g(Index n, const Number *x, bool new_x, Index m, Index nele_jac, Index *iRow, Index *jCol, Number *values)*
- *virtual bool eval_h(Index n, const Number* x, bool new_x, Number obj_factor, Index m, const Number* lambda, bool new_lambda, Index nele_hess, Index* iRow, Index* jCol, Number* values)*

Метод *get_nlp_info* предназначен для передачи количественных параметров решаемой НП-задачи, таких как: количество переменных, количество ограничений, количество ненулевых членов якобиана, количество ненулевых членов гессиана. Эти параметры необходимы Ipopt'у для корректной работы с массивами данных.

Метод *get_bounds_info* передает солверу информацию о границах ограничений.

Метод *get_starting_point* задает известные начальные значения переменных, участвующих в задаче.

Перечисленные три метода вызываются однократно в начале вычислений, поэтому их быстродействие не так важно по сравнению со скоростью работы остальных методов, которые вызываются многократно на каждом шаге работы солвера. Остальные методы вычисляют значение необходимых функций в точках.

Метод *eval_f* вычисляет значение целевой функции $f(x)$ в точке.

Метод *eval_grad_f* вычисляет значение градиента целевой функции $\nabla f(x)$ в точке.

Метод *eval_g* вычисляет вектор значений вектор-функции ограничений $g(x)$ в точке.

Метод *eval_jac_g* вычисляет значения якобиана вектор-функции ограничений $\nabla g(x)^{\mathrm{T}}$ в точке.

И метод *eval_h* вычисляет значение гессиана функции Лагранжа $\sigma_f \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x)$ в точке.

Класс *Constraint* необходим, чтобы создать список ограничений *g(x)*, которые все сразу добавляются в НП-задачу функцией *set_constraints*. Его использование повышает удобство создания ограничений по сравнению с прямой записью в НП-задачу. Кроме того, перед добавлением ограничений в задачу используется функция фильтра, которая удаляет из НП-задачи зарезервированные, но ни разу не использованные переменные, а также нулевые члены полиномов, использованных в ограничениях. Класс *Constraint* состоит из:

- *const Expression* expression* – выражений
- *pair<Number, Number> bounds_lu* – границы ограничения
- *Constraint(const Expression& e, Number lower = 0., Number upper = 0.)* – метод, создающий элемент класса
- *void set_bounds(Number lower, Number upper)* – метод, устанавливающий новые границы
- *const Expression* expr() const* – метод, возвращающий выражение
- *pair<Number, Number> bounds()* – метод, возвращающий границы

Класс *Expression* абстрактный, от него наследуются всевозможные выражения, которые поддерживает интерфейс взаимодействия с Ipopt. В нашем случае это полиномы третьей степени, косинус и синус. Класс *Expression* состоит только из методов, которые должны быть переопределены у его наследников. А именно вспомогательные методы:

- *virtual const Expression* clone() const = 0* – метод копирования элемента

- *virtual void print_formula(ofstream& output) const = 0* – метод показывающий как выражение печатается функцией *print_NLP*
- *virtual void print(ofstream& output) const = 0* – метод показывающий как выражение печатается функцией *print_ampl_NLP*
- *virtual bool check_index(int n_var) const = 0* – метод, проверяющий, что в выражении не использованы несуществующие переменные

и набор инструкций для вычисления значений и производных выражения, необходимых для функций обратного вызова:

- *virtual bool build_jac(map<int, int>& jac_map_i, int& jac_nnz) const = 0*
- *virtual bool build_hes(Triangular_matrix& hes_map, int& hes_nnz) const =0*
- *virtual Number eval(const Number* x) const = 0*
- *virtual bool eval_d(const Number* x, map<int, int>& jac_map_i, Number *values) const = 0*
- *virtual bool eval_dd(const Number* x, Triangular_matrix& hes_map, Number lambda_i, Number obj_factor, Number *values) const = 0*

Блок *CNLPFunc* состоит из наследников класса *Expression*, для которых нужно переопределить его методы.

3.3. Модуль составления задачи нелинейного программирования

Модуль составления НП-задачи называется *CGenNLP*. Это модуль содержит в себе один обширный класс *ClGenNLP*. В этом классе только два открытых метода:

- *virtual void Constructor(ClPolyhedron* diamond_poly, const ClPolyhedron* stone_poly, ClPolyhedron* original_stone_poly, const vector<ClPolyhedron>& Inclusions_, const IpoptConfig& iConfig, OxygenData& oData)*

- *virtual NLP_Problem* generate_nlp(TimeCounter &tCounter)*

Первый метод *Constructor* отвечает за заполнение пустого элемента класса *ClGenNLP* данными о задаче, приходящими на вход. Второй метод *generate_nlp* отвечает за формирование всей задачи, в нем вызываются закрытые методы, формирующие отдельные подзадачи.

Все переменные класса *ClGenNLP* можно разбить на три группы. Первая группа состоит их входных данных, которые записываются методом *Constructor* и используются при построении задачи. Вторая группа состоит из массивов данных в формате, необходимом классу *NLP_Problem*. Эти данные в конце метода *generate_nlp* записываются в НП-задачу, которая является результатом работы модуля. Третья группа переменных - это представление для переменных оптимизации. Дело в том, что Ipopt работает с одним списком переменных от 0 до (*n_vars* – 1), где *n_vars* – общее количество переменных оптимизации. Для программирования ограничений невозможно оперировать только этими номерами. Поэтому третья группа переменных – это массивы, в которых лежат номера переменных оптимизации, относящиеся к какому-либо типу объектов, названные соответствующим образом. Например, в двумерном массиве *vertices*, лежат номера переменных оптимизации, отвечающих за координаты вершин внутреннего многогранника. Для правильного заполнения этих массивов создан переопределенный метод *reserve_var(type ids)*, который при первом использовании массива переменных записывает в него уникальные номера переменных из оптимизации и увеличивает число *n_vars*, а при последующем использовании массива ничего в нем не меняет, чтобы за один геометрический объект во всей НП-задаче отвечал один набор переменных.

Все закрытые методы можно также разделить на три группы. К первой группе относятся методы для вписывания многогранника с фиксированной комбинаторной структурой без дополнительных геометрических ограничений:

• *virtual void create_variables()* – метод, инициализирующий переменные третьей группы класса *ClGenNLP* массивами нужного размера, состоящими из -1, это нужно для работы метода *reserve_var(type ids)*

• *virtual void gen_start_point()* – метод, задающий начальное значение для переменных оптимизации. Если начальное значение не известно, то ставится 0

• *virtual void gen_max_vertex_shift()*

• *virtual void gen_max_face_shift()* – Методы *gen_max_vertex_shift* и *gen_max_face_shift* задают ограничения на координаты вершин и граней внутреннего многогранника. Если заранее не известно на сколько начальная точка близка к желаемому результату, то ограничения и на переменные вершин, и на переменные граней - $(-\infty, +\infty)$. В противном случае ограничения берутся исходя из известной информации или требований не отходить далеко от начального положения.

• *virtual void gen_combinatorics()* – метод, задающий ограничения на комбинаторную структуру внутреннего многогранника

•*virtual void gen_diamond_convex()* – метод, задающий выпуклость внутреннего многогранника

• *virtual void gen_stone_restrictions()* – метод, задающий ограничения на вписанность результата в выпуклую оболочку внешнего многогранника.

• *virtual void gen_nonconvex(int incl)* – метод, отвечающий за то, чтобы результат не пересекал внешние невыпуклости внешнего многогранника

• *virtual void gen_inclusion(int incl)* – метод, отвечающий за то, чтобы результат не пересекал невыпуклое включение

• *virtual void gen_convex_inclusion(int incl)* – метод, отвечающий за то, чтобы результат не пересекал выпуклое включение

• *virtual void inclusion_filter(vector<int>& isConvex)* – если заданы ограничения на отклонение вершин от их начального положения, то этот

метод отфильтровывает те включения, которые решение даже теоретически не может достигнуть. Это позволяет не добавлять в задачу лишние ограничения

• *virtual void gen_objective()* – метод, задающий целевую функцию. По-умолчанию, это объем внутреннего многогранника.

Вторая группа закрытых методов – это шаблоны, позволяющие быстро записывать в НП-задачу универсальные геометрические ограничения, часть из которых описана в разделе 2.4. В модуле изначально представлены следующие шаблоны:

• *virtual void gen_dev(const vector<int>& value, int min, int max, double dev)* – ограничивает на *dev* разброс значений *value*

• *virtual void gen_avg(const vector<int>& value, int avg)* – считает среднее значение *value*

• *virtual void gen_sup(const vector<int>& value, int sup)* – считает супремум группы *value*

• *virtual void gen_inf(const vector<int>& value, int inf)* – считает инфинум группы *value*

• *virtual void gen_scalar_product (const vector<int>& v1, const vector<int>& v2, int product)* – считает скалярное произведение векторов *v1* и *v2*

• *virtual void gen_vector_product (const vector<int>& v1, const vector<int>& v2, int product)* – считает векторное произведение векторов *v1* и *v2*

• *virtual void gen_angles_dev(int table_face, const vector<int>& angles, int min, int max, int cos_min, int cos_max, double dev, bool positive)* – ограничивает на *dev* разброс углов между плоскостями *angles* и плоскостью *table_face*

• *virtual void gen_angles_cut(int table_face, const vector<int>& angle_planes, const vector<int>& angles, const vector<int> cos, double min_boadr, double*

max_board, bool positive) – ограничивает средний угол между плоскостями *angles* и плоскостью *table_face*

• *virtual void gen_projection(const vector<int>& point, const vector<int>& plane, const vector<int>& projs, double norma, int dimension, bool d_flag)* – считает проекцию точки *point* на плоскость *plane*

• *virtual void gen_2D_azimuth_vector(const vector<int>& point, const vector<int>& plane, const vector<int>& projs, int norma)* – создает азимутальный вектор для вектора

• *virtual void gen_2D_azimuth_vector(const vector<int>& point, const vector<int>& center, const vector<int>& plane, const vector<int>& projs, int norma)* – создает азимутальный вектор для вершины

• *virtual void gen_area(const vector<int>& points, const int area, bool sign)* – считает площадь поверхности грани

• *virtual void gen_norma2d(const vector<int>& first, int norma2d)* – вычисляет норму вектора в плоскости XY

• *virtual void gen_norma3d(const vector<int>& first, int norma3d)* – вычисляет норму вектора

• *virtual void gen_square_distance_val(vector<int> first, vector<int> second, int distance, int dimension)* – вычисляет расстояние от точек *first* до точек *second*

• *virtual void gen_square_distance_max(vector<int> first, vector<int> second, double dev, int dimension)* – ограничивает сверху расстояние от точек *first* до точек *second*

• *virtual void gen_square_distance_dev(vector<int> first, vector<int> second, int min, int max, double dev, int dimension)* – ограничивает разброс расстояний от точек *first* до точек *second*

• *virtual void gen_2points_axis_symmetry (vector <int>& edge_1, vector <int>& edge_2,vector <int>& edge_2_dis, vector <int>& axis_symm, double dev)* – ограничивает расстояние между точкой и отражением второй точки относительно грани симметрии

- *virtual void gen_azim_for_2planes2 (vector<int>& plane_table, vector<int>& planes_1, vector<int>& planes_2, vector<int>& axis_symm, int& norma_1, int& norma_2, vector<int>& planes_1_proj, vector<int>& planes_2_proj, vector<int>& planes_2_mirror, double dev)* – ограничивает угол между вектором и отражением второго вектора относительно грани симметрии

К третьей группе закрытых методов относятся методы для вычислений понятий и метрик, относящихся к конкретной области применения данного алгоритма. На базе представленных инструментов пользователь может задать любые необходимые ему ограничения.

Глава 4. Применение комплекса программ в ювелирной промышленности

Одной из стадий обработки алмаза является предпроизводственный анализ. Он выполняется с целью определения технологической направленности обработки алмазов. Здесь идет сортировка по форме будущих бриллиантов, определяются кристаллы на распиливание (однократное или многократное), раскалывание или на подшлифовку; определяются особенности каждого кристалла, выявляются напряженные и дефектные кристаллы, характер и расположение природных дефектов и т.п. По сути, на стадии предпроизводственного анализа делается прогноз веса готового бриллианта, основных геометрических параметров, оценочных характеристик и стоимости будущего бриллианта.

Сегодня на ограночных фабриках нового поколения используют современные технологии при анализе, оптимизации и планировании огранки алмазов. Оценить возможность алмаза и спланировать его обработку технологу (огранщику) помогают компьютерные системы по моделированию огранки бриллианта. Система производит высокоточное сканирование алмаза и на полученной модели вычисляет, каким образом из него можно получить оптимальный бриллиант. На основании предоставленного плана огранщик начинает обрабатывать бриллиант. Технологии в данной отрасли еще только на стадии своего зарождения и непрерывно развиваются: улучшается точность сканеров, повышается качество и скорость алгоритмов планирования, совершенствуются методы автоматической оценки полученных бриллиантов. При достижении достаточного уровня технологий начнется полная роботизации процесса обработки алмазов, что позволит убрать фактор человеческой ошибки при производстве. На текущем же этапе огранщик всё еще остается ключевым звеном в цепочке. И решения, полученные с помощью программы являются только планом, в области которого огранщик ищет оптимальный вариант конечного результата.

Задача поиска оптимального бриллианта в алмазе с математической точки зрения является задачей нахождения в невыпуклом многограннике произвольной формы объекта, обладающего заданными свойствами и имеющего объемом. Это сложная задача, и не существует алгоритмов, позволяющих ее точно решить за ограниченный промежуток времени. А так как речь идет о внедрении в производство с непрерывным рабочим процессом, то время работы алгоритма является очень критичным фактором и ограничивается секундами. Поэтому существующие алгоритмы либо находят слишком локальное решение вместо глобального, либо работают с упрощенными моделями, что понижает точность вычислений и получаемый объем. Их необходимо совершенствовать для получения более хороших результатов.

Рассмотрим критерии оценки результирующего бриллианта с точки зрения существующих алгоритмов планирования и их возможного совершенствования. Обычно, экспертная оценка полученного камня осуществляется по четырем параметрам (4 «С»):

- Первая «С» – carat weight (вес в каратах). На этом этапе идет точное определение веса камня путем взвешивания на весах или расчета по формулам, если бриллиант закреплен в изделии. Вес бриллианта выражается в каратах (1 ct = 0,2 грамма).
- Вторая «С» – color (цвет). Совершенно бесцветные алмазы встречаются довольно редко, и практически все камни имеют оттенки различных цветов и степень интенсивности окраски. В задачу эксперта входит точное определение интенсивности и цвета бриллианта при стандартном освещении с использованием эталонов цвета и присвоение оценки по цвету.
- Третья «С» - clarity (чистота). На этом этапе выявляются все внутренние несовершенства (дефекты) камня. По чистоте камню присваивается оценка.
- Четвертая «С» – cut (качество огранки). На этом этапе дается характеристика формы бриллианта, качества огранки и финальной обработки.

Главной задачей алгоритмов планирования является получить максимальный вес, сохранив необходимую чистоту и качество бриллианта. Вес является основой для целевых функций алгоритмов планирования. В процессе огранки очень важно минимизировать отходы производства. Кроме того, цена конечного бриллианта меняется не линейно от изменения веса, а делает скачки на некоторых круглых значениях веса. Поэтому даже маленькая прибавка веса в некоторых случаях дает большой прирост к цене бриллианта.

Цвет бриллианта очень слабо меняется от формы бриллианта, особенно если не менять тип огранки. Он по большей части обусловлен природным материалом, из которого состоит алмаз, поэтому не учитывается в алгоритмах планирования, и нет существенных перспектив в этом направлении.

Чистота бриллианта обуславливается чистотой попавших в него дефектов, и влияет на пропорциональную цену камня. В подавляющем большинстве случаев самых простых приблизительных алгоритмов достаточно, чтобы понять какой чистоты дефектам разрешать попадать в решение, а каким не разрешать, для достижения максимальной цены бриллианта. В пограничных же ситуациях, обычно, владелец алмаза сам решает, что для него важнее: чистота или вес, при примерно одинаковой итоговой цене бриллианта. Поэтому для сложных алгоритмов планирования в итоге остается просто список включений, которые не должно пересекать решение. Это накладывает на алгоритмы требование – уметь работать с невыпуклыми объектами, что усложняет задачу.

Существуют различные системы оценок качества огранки. Наиболее распространенной является система, разработанная Геммологическим Институтом Америки (GIA). Все подобные системы оценок заключаются в том, что они делят бриллианты на группы качества в зависимости от пропорций бриллианта и его отклонения от идеально симметричного. И если бриллианты находятся в одной группе качества, то у них одинаковая цена за единицу веса. Главным недостатком всех современных общеиспользуемых алгоритмов планирования является то, что

они умеют работать только с идеально-симметричными бриллиантами, а пропорции бриллианта могут меняться только дискретно с определенным шагом. Это сильно упрощает алгоритм и уменьшает время работы, но приводит к существенной потере в весе, которую можно избежать за счет допустимой асимметрии в пределах одной группы качества.

Реализованный программный комплекс для численного решения задачи вписывания многогранников позволяет улучшить результаты (увеличить объем бриллианта) за счет допустимой асимметрии бриллианта в регулируемых, заданных пользователем, границах. Кроме того, оптимизационный подход позволяет найти исключительные случаи, когда критерии качества формально соблюдены, но бриллиант является плохим, некрасивыми с точки зрения оптической или 3D-симметрии. Эти случаи позволяют формализовать новые критерии оценки качества бриллиантов и внедрять их в международные стандарты.

4.1. Существующие алгоритмы планирования и их недостатки

На данный момент есть несколько программных продуктов для огранщиков, которые ищут план огранки драгоценных камней. Но все они работают с конечным набором жестко симметричных форм, поэтому не способны найти оптимальную массу. В качестве стартовой точки для метода колебания вершин используются решения этих алгоритмов; искажается симметрия в допустимых международными и фабричными стандартами рамках, тем самым увеличивается масса и стоимость решений.

Самым распространенным алгоритмом для поиска наибольшей огранки является перебор ее возможных положений с отсечением случаев с заведомо маленьким объемом. Принцип работы алгоритма – есть жестко зафиксированная форма огранки, которая может подвергаться только повороту, трансляции и

гомотетии. Внутри драгоценного камня перебираются по сетке с заданным шагом возможные центры масс огранки и ее ориентация в пространстве. Для каждого центра масс и ориентации, огранка увеличивается до пересечения с камнем. Случай, когда получается наибольший коэффициент увеличения, является искомым максимальным решением. В такой упрощенной формулировке алгоритм работает слишком долго при маленьком шаге сетки и дает слабые результаты на большом шаге. Недостатки данного алгоритма исправляются следующими способами.

Во-первых, необходимо максимально ускорить процедуру нахождения коэффициента увеличения. Для этого минимизируют вычислительную сложность компонент процедуры. Повышают эффективность работы с памятью. Используют различные способы более быстрого нахождения пересечения двух многогранников. Например, с помощью суммы Минковского или с помощью представления границы в виде вокселей (многомерный пиксел).

Во-вторых, возможно улучшить обход по сетке центров масс и ориентаций. На основании ранее полученных результатов исключаются некоторые направления поиска. Оптимизируется последовательность обхода сетки. Кроме того, сетка может быть не стабильная, а меняться в зависимости от предыдущих результатов. В частности, может уменьшаться шаг сетки, когда найдена зона локального максимума.

Чаще всего есть не одна форма вписываемой огранки, а некоторый конечный набор форм, имеющих одинаковую комбинаторную структуру. Это обусловлено тем, что относительные параметры огранки могут отличаться, и необходимо оптимально использовать драгоценный камень. Более слабые алгоритмы для каждой из форм запускаются отдельно. В более современных и развитых алгоритмах перебор форм участвует наравне с перебором центра масс и ориентации, как еще одна размерность сетки перебора.

Наилучший алгоритм такого типа работает в среднем 10 секунд.

Описанные алгоритмы для нахождения начального приблизительного расположения огранки являются ненадежными по двум основным причинам:

1) Они, как и любая оптимизация, находят локальный максимум. Кроме того, так как они работают с частными геометрическими объектами, а не с системами уравнений, то не существует развитого математического аппарата для решения подобных задач. Поэтому эти алгоритмы работают на основе частных разработок, в которых из нескольких существующих локальных максимумов может находиться более плохой с точки зрения глобальности, чем в аналогичной системе уравнений, решенной стандартными математическими методами.

2) Эти алгоритмы работают с набором жестко зафиксированных форм, которые могут подвергаться только повороту, трансляции и гомотетии. В то время как метод колебания вершин может менять форму, сохраняя только комбинаторную структуру и определенные требования на форму. И получается, что даже, если отбросить проблемы локальности, бывают ситуации, когда лучшее решение с зафиксированной формой и лучшее решение метода колебания вершин находятся в совершенно разных точках, и заданного отклонения *dx* не достаточно, чтобы метод колебания вершин нашел свое лучшее решение. Из-за этого иногда приходится запускать метод колебания вершин на различных начальных точках, чтобы достичь более хорошего результата. Поэтому для повышения эффективности алгоритма необходимо найти альтернативный подход без использования начальной точки.

По этим причинам была предпринята попытка реализовать метод, описанный в разделе 2.2.3., но пока время его работы не удовлетворяет производственным требованиям.

4.2. Алгоритм планирования круглой огранки

Созданный комплекс программ был использован для создания алгоритма поиска лучшей круглой огранки в алмазе «SmartRecut». Круглая огранка выбрана первой для реализации, потому что именно эта огранка занимает 90% рынка, благодаря своим хорошим оптическим свойствам, обеспечивающим высокий уровень блеска бриллианта. Кроме того, только для круглой огранки существую мировые стандарты качества в формализованном математическом виде. Если рассматривать задачу нахождения оптимального бриллианта в терминах разработанного комплекса программ, то огранка – это внутренний многогранник, камень-алмаз – внешний многогранник, а дефекты алмаза – это включения. Общих разработанных методов для решения задачи вписывания многогранника достаточно, чтобы обеспечить нахождение огранки внутри камня, и без пересечения с дефектами. Для обеспечения качества искомой огранки был разработан набор параметров – геометрических ограничений. Параметры делятся на два типа:

- Cut – параметры, отвечающие за относительные размеры элементов огранки
- Symmetry – параметры, отвечающие за разброс значений в группах одинаковых элементов. Или, другим словами, параметры, не позволяющие сильно нарушать осевые симметрии огранки

Группа Cut состоит из 14 параметров (Рисунок 4.1). Группа Symmetry состоит из 38 параметров (Рисунок 4.2). Как можно видеть из рисунков, в интерфейсе в столбце Value пишется значение параметра для выбранного решения. В остальных столбцах задаются границы для оптимизации для разных групп качества и отмечается, как близко значение параметра находится к границе. При запуске оптимизации можно на Cut и Symmetry задавать разные группы качества, так как они по-разному влияют на конечную цену бриллианта.

	Value	[FR	[GD	[VG	[EX	EX]	VG]	GD]	FR]
Table	57.801	10	46,5	49,5	51,5	62,5	66,5	69,5	99
CrownAngle	36.007	10	21,75	26,25	31,25	36,75	38,75	40,25	90
PavilionAngle	40.726	10	38,7	39,7	40,5	41,9	42,5	43,1	90
StarLength	51.060	10	32,5	37,5	42,5	67,5	72,5	77,5	90
LowerGirdleLength	80.516	50	57,5	62,5	67,5	87,5	92,5	97,5	99
GirdleBezel	4.204	0	1,25	1,75	2,25	4,75	5,75	7,25	20
GirdleValley	2.263	0	0	0	0,75	2,94	4,14	6,14	20
CrownHeight	15.344	5	10,5	12	12,3	15,5	17,5	18,5	40
TotalHeight	62.517	10	54	57	58	62,5	64	66	90
Culet	0.333	0	0	0	0	1	1,5	2	20
CrownPainting	-0.353	-9	-6	-3	-2,5	2,5	5	7	20
PavilionPainting	0.532	-9	-5	-3	-2,5	2,5	4	6	20
SumPainting	0.179	-9	-6	-5	-3,5	5	8	10	20
GirdleAngleMax	2.000	0	0	0	0	2	4	6	20

Рисунок 4.1

	Value	EX]	VG]	GD]	FR]
Diameter	0.393	0,7	1,4	2,8	20
Table	0.209	0,9	1,7	3,4	20
CrownAngle	0.570	0,9	1,8	3,6	20
PavilionAngle	0.452	0,6	1,2	2,4	20
StarLength	1.574	3	12	24	48
LowerGirdleLength	1.807	2	8	16	32
GirdleBezel	0.478	0,9	1,8	3,6	20
GirdleBezelLocal	0.312	0,45	0,9	1,8	20
StarAngle	0.752	2,8	5,6	11,2	22,4
UpperGirdleAngle	1.159	2	8	16	32
LowerGirdleAngle	0.757	1,3	2,6	5,2	10,4
CrownHeight	0.582	0,9	1,8	3,6	20
PavilionDepth	0.677	0,9	1,8	3,6	20
GirdleValley	0.427	0,9	1,8	3,6	20
GirdleValleyLocal	0.128	0,45	0,9	1,8	20
GirdleBone	0.659	0,9	1,8	3,6	20
GirdleBoneLocal	0.185	0,45	0,9	1,8	20
2RRoundness22_5	0.333	0,4	0,8	1,6	20
2RRoundness45	0.388	0,7	1,4	2,8	20
2RRoundness90	0.388	0,9	1,8	3,6	20
TableOffset	0.155	0,4	0,8	1,6	20
CuletOffset	0.379	0,4	0,8	1,6	20
TableCuletOffset	0.518	0,6	1,2	2,4	20
TableEdge_TEV	0.992	1	2	4	20
BezelWidth	0.992	1	2	4	20
StarEdge	0.496	0,5	1	2	20
CrownPainting	0.748	1	2	4	20
PavilionPainting	0.748	1	2	4	20
TableAngle	1.000	1	2	4	20
OppositeAzimuth	0.750	1	4	6	20
FacetTwistMax	0.750	1	2	3	20
JunctionBezelTwistMax	0.735	1	2	3	20
OppositeSlopeSumHalf	0.275	0,5	1	1,5	20
StarFacetTwist	0.535	1	2	3	20
JunctionBoneTwistMax	0.980	1	2	3	20
MainCrownFacetsAzimuthSymm	0.750	2	4	6	20
MainPavilionFacetsAzimuthSymm	0.750	2	4	6	20
StarFacetsAzimuthSymm	0.810	2	4	6	20

Рисунок 4.2.

Не все из этих параметров реализованы точно, т.к. это не всегда возможно из-за ограничения нелинейного программирования, созданного комплекса программ или требований на время работы алгоритма. Для тех параметров, которые реализованы с использованием аппроксимации, были проведены теоретические и эмпирические (с помощью высокоразвитой тестовой системы) оценки, на сколько эти параметры могут выходить за границы. На основе этих оценок были сделаны затяжки, которые уменьшают границы на параметр в оптимизации так, чтобы он не выходил за границы, ожидаемые пользователем. Если есть затяжка, она чаще всего не превышает 3% интервала. Нельзя выставлять затяжку по самому худшему случаю, потому что это уменьшет массу всех решений, поэтому затяжки ставятся по среднему уровню вылета. В случае же выхода параметра за границы, запускается второй запуск, перед которым анализируется размер выхода за границы параметров. И для этого конкретного примера ставится индивидуальная затяжка для второго запуска. Количество примеров со вторым запуском составляет примерно 9%.

Алгоритм SmartRecut используется в фабричном производстве. Это накладывает на него ограничения временного и качественного характера, так как, пока работает алгоритм, производственный процесс по конкретному камню останавливается. Для любого драгоценного камня за ограниченное время, должно быть получено решение. Чаще всего время составляет минуту, если пользователи используют компьютер рекомендуемой мощности. Это время может быть увеличено в трех случаях:

- Обрабатывается камень близкий к 5 каратам и больше. Это категория особенно дорогих камней. Поэтому для них всё намного тщательней просчитывается, чтобы найти наилучший результат. И время работы алгоритма перестает быть критичным требованием.
- Расчет ведется для камня на ранней стадии обработки, у которого очень много невыпуклых граней. На большом количестве невыпуклых граней размер НП-задачи резко увеличивается, поэтому алгоритму чисто технически нужно

больше времени. Невыпуклый случай на порядок сложнее выпуклого, поэтому все алгоритмы планирования работают дольше в этом случае.

- У пользователя стоит компьютер с низкой частотой процессора. Чтобы на такой машине успевали решаться задачи, которые успевают решаться на рекомендуемой мощности, время работы увеличивается пропорционально частоте. Пользователь может отключить эту опцию, чтобы сэкономить время, но чаще будут получаться не все решения.

На практике получается, что какие бы фиксированные настройки не были выбраны, найдутся камни, для которых Ipopt работает дольше отведенного времени. Поэтому сделаны распараллеленные запуски SmartRecut. Делается 8 одновременных запусков (если рассматривать стандартный современный четырехъядерный процессор с multithreading) для одного и того же драгоценного камня на разных настройках параметров. Распараллеливание сделано в отдельных процессах. Каждый процесс имеет заданное время завершения по правилам описанным выше. Если Ipopt не успел решить задачу за это время, то процесс закрывается и возвращает начальное решение. Такое распараллеливание гарантирует получение результата за отведенное время, а также позволяет получать камни разного качества. Камни разного качества нужны, потому что в ювелирной промышленности стоимость ограненного камня зависит не только от массы. Иногда выгоднее сделать более большой бриллиант худшего качества, а иногда бриллиант меньшего размера, но с идеальным качеством. Для настройки этих 8 запусков сделаны пресеты. Интерфейс для редактирования открывается слева от списка параметров. На Рисунке 5.1 изображен интерфейс для настройки пресетов группы параметров Cut, а на Рисунке 5.2 для Symmetry. На эти числа умножаются заданные интервалы ограничений на параметры, тем самым делая настройки запуска каждого пресета уникальными. Помимо названий, которые тут не приведены, все пресеты маркированы сверху своим цветом. Получаемые решения маркируются аналогичными цветами, чтобы пользователь понимал, на каком пресете получено интересующее решение. Цифра 100 в пресете означает, что

параметр либо не ограничивается, либо ограничивается значениям GIA, если они есть. Все решения должны быть не хуже GIA. На тестовой выборке, состоящей из 245 камней на разных стадиях обработки, в 98.8% случаев решение успевает посчитаться и имеет заданную группу качества.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	100
1	1	1	1	1	1	1	1	1	1	0	100	0	100	0	100
1	1	1	1	1	1	1	1	1	1	0	100	0	100	0	100
-	0,5	-	1	-	1	-	1	-	1	-	1	-	1	-	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	100	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	100	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	100	100
-	1	-	1	-	1	-	1	-	1	-	1	-	1	-	1

Рисунок 5.1.

0,7	1	1	1	1	1	1	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,15	1,15	100
0,5	1	1	1	1	1,5	1,5	100
0,5	1	1	1	1	1,5	1,5	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,5	1,5	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,1	1,1	100
0,5	1	1	1	1	1,1	1,1	100
1	1	1	1	1	1	1	1,5
1	1	1	1	1	1	1	1,5
1	1	1	1	1	1	1	1,5
0,5	1	1	1	1	1,25	1,25	100
0,5	1	1	1	1	1,25	1,25	100
0,5	1	1	1	1	1,15	1,15	100
0,5	1	1	1	1	1	1	100
0,5	1	1	1	1	1	1	100
0,5	1	1	1	1	1	1	100
0,1	0,2	0,35	0,5	0,75	1	2	8
0,1	0,2	0,35	0,5	0,75	1	2	8
0,5	1	1	1	1	1	1	100
0,1	0,2	0,35	0,5	0,75	1	2	100
0,1	0,2	0,35	0,5	0,75	1	2	100
0,1	0,2	0,35	0,5	0,75	1	2	100
0,1	0,2	0,35	0,5	0,75	1	2	100
0,5	1	1	1	1	1	2	100
0,5	1	1	1	1	1	2	100
0,5	1	1	1	1	100	100	100
0,1	0,2	0,35	0,5	0,75	100	100	100
0,5	1	1	1	1	100	100	100

Рисунок 5.2.

В международном стандарте качества GIA присутствуют всего 23 параметра. Все они есть среди этих 52 параметров. Оказалось, что международные стандарты качества не приспособлены к полноценной работе с несимметричными бриллиантами. У бриллианта кроме трехмерной симметрии есть еще оптическая симметрия - насколько он хорошо и симметрично блестит.

Приведенные Рисунке 6. отображают ход световых лучей в бриллианте. Чем симметричней рисунок, тем лучше оптическая симметрия огранки. На Рисунке 6. слева приведена оптическая картинка для идеально ровной огранки массой 1.0149 карат, которая была взята в качестве начальной точки алгоритма. Масса неогранённого камня в этом примере 1,2904. Справа же приведена оптическая картинка для результата работы SmartRecut, когда включены только ограничения, соответствующие международному стандарту оценки качества бриллиантов. Масса этого решения 1.0601 и формально оно считается высшего качества. Но как можно видеть, оптическая симметрия очень далека от идеальной, а это значит, что камень будет очень хаотично блестеть и его сложно будет продать.

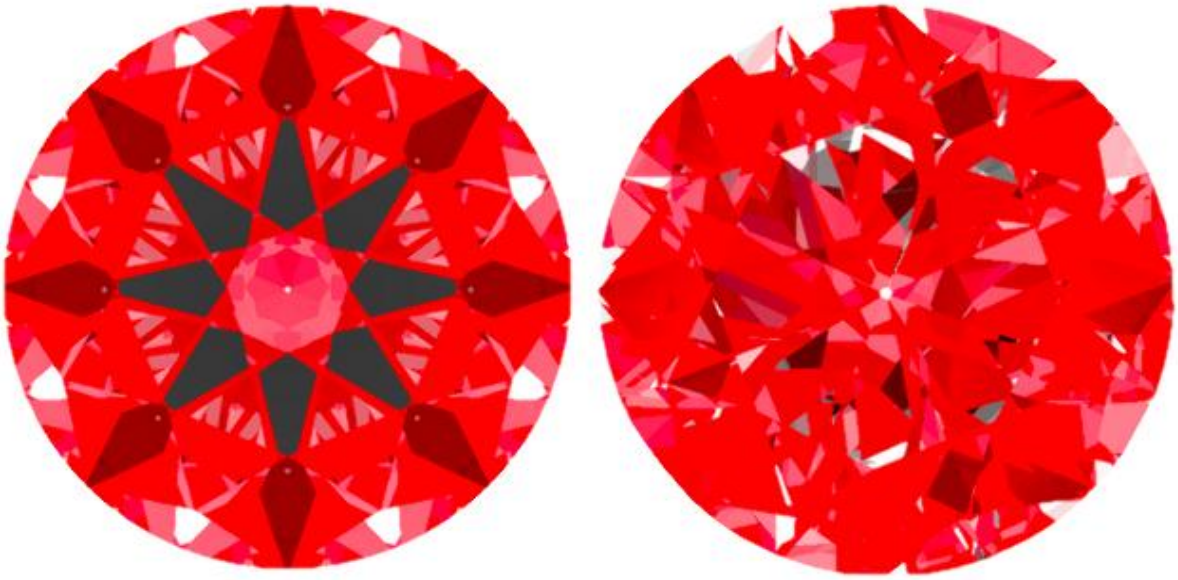

Рисунок 6.

Поэтому были изобретены дополнительно параметры, которые контролируют оптическую симметрию бриллианта. И восемь одновременных запусков настроены

так, чтобы получался спектр решений от идеальной оптической симметрии до максимальной массы с необходимым качеством. Для рассмотренного примера получается следующий спектр масс: 1.0601, 1.0428, 1.0406, 1.0368, 1.0356, 1.0316, 1.0315, 1.0268. Все решения, начиная с массы 1.0428, имеют достаточно красивую оптическую симметрию, а решение 1.0268 практически не отличается от идеальной оптической симметрии.

Аналогично оптической симметрии существуют и случаи, когда формально качественное решение имеет недопустимое искажение 3D-симметрии. Выявление и анализ таких случаев позволяет формализовать новые критерии оценки качества бриллиантов и внедрять их в международные стандарты. Например, параметры TableEdge_TEV, Junction Bezel Twist и Junction Bone Twist появились в SmartRecut раньше, чем в GIA стандарте. И появление моделей с искривлениями по этим параметрам ускорило введение их параметров в стандарт, хотя это и так планировалось, не зависимо от SmartRecut. Для огранок кроме круглой вообще нет общепринятых стандартов, поэтому приходится разрабатывать свои правила, чтобы решения удовлетворяли ювелиров. Со временем эти правила могут быть стандартизированы.

4.2.1. Алгоритм для работы с камнями со сколами

Разработанные в диссертации методы, примененные при создании алгоритма SmartRecut, оказались эффективными для решения геометрических оптимизационных задач. Благодаря этому, на базе SmartRecut для круглой огранки, удалось создать модификации алгоритма, которые полезны огранщикам, и не имеют других промышленных аналогов.

В английском языке есть значение слова knife как небольшая грань бриллианта, касающаяся рундиста и имеющая неправильный для рундиста угол наклона. Прямого русского аналога этого термина не существует, поэтому будем

использовать английское слово. Эти грани образуются из-за того, что в процессе огранки от бриллианта может отколоться кусок. Существование таких граней в конечном бриллианте не желательно, так как они портят идеальную форму бриллианта и оптические свойства. Но, если knife достаточно маленький и расположен со стороны павильона, то из-за коэффициента преломления этот knife не виден, если смотреть на бриллиант, закрепленный в оправу. Размеры и количество knife'ов накладывают штрафы на цену бриллианта. Но, если скол бриллианта произошел на достаточно позднем этапе обработки, и запланированное огранщиком решение больше недоступно, то ему приходится оставлять knife, иначе придется сточить существенную часть бриллианта. Огранщику необходимо изменить свой план так, чтобы knife'ов стало допустимое количество, и они имели допустимый размер. Раньше не существовало автоматических алгоритмов, которые могут планировать решения с knife'ами. Огранщикам приходилось в ручную моделировать решение в 3D-редакторах.

На базе SmartRecut создан алгоритм, умеющий работать с knife'ами. Сначала алгоритмически анализируется модель камня со сколом и начальное расположение огранки. Исходя из анализа, вычисляется, где у решения наиболее перспективно оставить заданное количество knife'ов. С учетом этого перестраивается комбинаторная структура огранки и алгоритм запускается с такой комбинаторной структурой. Помимо изменения комбинаторной структуры, добавляются необходимые ограничения на размеры knife'ов, а также меняется вычисление некоторых стандартных параметров, на которые влияет наличие knife'ов.

4.2.2. Алгоритм, фиксирующий часть камня

Запуск алгоритмов планирования происходит многократно на протяжении огранки бриллианта. Огранщик проводит какую-то часть работ, сканирует заново камень и заново создает план огранки. Это необходимо, чтобы подтвердить, что

старый план огранки не поврежден и всё еще доступен. Иногда выявляются дефекты, которые были не видны ранее. На поздней стадии, когда огранщик приступает к окончательной полировке частей бриллианта (рундиста, короны и павильона), огранщику необходимо, чтобы уже завершенная часть бриллианта не менялась при новом запуске алгоритма планирования. Но, по умолчанию это невозможно, потому что огранщик не может физически выпилить идеальное схождение граней в точку, на его бриллианте всегда появляется микроскопическое расщепление вершин. Кроме того, у планируемых бриллиантов обычно 64 грани рундиста, а настоящий рундист точится на станке, и он состоит из очень мелких граней, образующих окружность. В итоге комбинаторная структура частей, которые нужно зафиксировать, не совпадает с комбинаторной структурой планируемого бриллианта, и новый план может только примерно соответствовать уже готовым частям. Это плохо, потому что существуют сквозные параметры, которые считаются по элементам разных частей бриллианта, например, расстояние между вершинами на короне и вершинами на павильоне. В итоге, огранщик не знает настоящего значения таких параметров, т.к. для расчета используются части нового плана, а не его ограненная часть, которая больше не будет меняться.

На базе SmartRecut создан алгоритм, который умеет фиксировать части камня (рундист, павильон, корону). Известных аналогов такого алгоритма не существует. При фиксации какой-то из частей, модель камня анализируется, разбивается на соответствующие части, запоминаются грани, которые необходимо оставить. В оптимизации отключаются все параметры, которые считаются только по зафиксированной части. Для всех сквозных параметров переменные величины, отвечающие за этот параметр в фиксированной части, заменяются на константы, предрасчитанные по зафиксированной части. Делается запуск оптимизации, после чего берутся грани от полученного решения, которые не были зафиксированы, и грани от камня, которые были зафиксированы, и по ним строится искомый многогранник через выпуклую оболочку в двойственном пространстве. В итоге огранщик имеет искомый новый план с правильными сквозными параметрами.

Стоит отметить, что при таком способе построения в план может попасть несуществующий кусок зафиксированного рундиста, т.е. план выйдет за границы камня. Это дополнительно ограничивается тем, что грани короны и павильона нового решения должны пересекать зафиксированный рундист между его верхней и нижней кромками, составленными из точек.

4.3. Алгоритм планирования овальной огранки

Реализован алгоритм нахождения классической овальной огранки аналогичный алгоритму нахождения круглой огранки. Есть несколько принципиальных отличий реализации алгоритма овальной огранки.

Не существует формализованных математических критериев качества овальной неидеальной огранки. Все огранoчные предприятия гранят овальные огранки по своим стандартам. Есть только наиболее распространенная классическая овальная огранка и небольшой набор устоявшихся для нее правил, исполнение которых приводит к более красивому овальному бриллианту. Поэтому полный перечень параметров разрабатывался самостоятельно. На разных стадиях разработки решения отсылались экспертам, которые говорили, что является некрасивым в решениях, и это исправлялось добавлением новых параметров. Основной перечень параметров сделан по аналогии с круглой огранкой, так как овальная огранка, по сути, является растянутой круглой, а также ювелиры привыкли иметь дело с этими параметрами. В итоговом продукте огранщики могут сами настроить границы предложенных параметров, чтобы их удовлетворяло качество и объем решений.

Второе принципиальное отличие в том, что у овала в отличие от круга, только две оси симметрии. И если все элементы круглой огранки делятся на группы размера кратного 8, и все элементы группы должны быть похожими, что

ограничивается разбросом значений какой-то метрики для элемента. То для овала приходится в явном виде работать с осями симметрии и с отражениями объектов относительно них.

Отдельную сложность представляют ограничения на овальный профиль рундиста. Они заменяют ограничения на диаметр и радиус в случае круглой огранки. Чтобы огранка считалась овальной должны выполняться следующие требования:

- Отношение максимального и минимального диаметров в заданном интервале
- Достаточно гладкий, идеально-симметричный профиль рундиста
- Отношение площади профиля рундиста к площади минимального описанного вокруг него прямоугольника в заданном интервале

Все эти необходимые ограничения учтены в SmartRecut для овальной огранки.

У овальной огранки нет универсальной комбинаторной структуры. Потому что в отличие от круглой огранки, блеск у классической овальной огранки не сильно превосходит блеск овальных огранок с немного измененной комбинаторной структурой. Поэтому многие ограночные фабрики изобретают свои комбинаторные структуры, что позволяет на некоторых камнях получать лучшую массу. Такие огранки задаются в стандартном .asc формате. Некоторые существующие алгоритмы планирования умеют работать с такими огранками, но работают дольше и получают решения значительно худшей массы, чем для стандартных настроенных огранок. Для SmartRecut разработан модуль распознавания вершин и граней произвольной овальной огранки в формате .asc, который позволит с ними работать. Так как SmartRecut не привязан к конкретной огранке, в отличие от других алгоритмов, то качество его работы не ухудшается при работе с произвольной овальной огранкой, что гарантирует еще больший прирост массы по отношению к другим алгоритмам.

4.4. Распределенная система для проведения вычислительных экспериментов

Развитие алгоритма требует постоянного тестирования на большом количестве примеров и с различными настройками. Чтобы принять решение о введении новой идеи алгоритма, необходимо запустить многочисленные тесты, чтобы выявить слабые места (то есть ситуации, когда новая идея приводит к замедлению алгоритма или ухудшению качества результата). Для решения этой задачи была создана система удалённого тестирования задачи.

Для выполнения теста создается его описание в формате json, которое содержит полный набор тестовых данных, на которых будет запускаться пробный алгоритм. Основные поля файла:

- name – название теста
- description – описание теста. Все тесты хранятся длительное время. Это поле помогает вспомнить, что именно проверял этот тест
- recipients – в это поле пишутся электронные адреса, заинтересованных в тесте людей, которым по его окончании будет выслано письмо с кратким отчетом или ошибками
- host – система тестирования распределенная, к ней может быть подключено несколько серверов. В это поле можно записать уникальный идентификатор сервера, если нужно, чтобы тест обработал именно он.
- dll – адрес, откуда брать тестируемый программный модуль
- stone_storage – примеры, на которых запускается тест
- presets_storage – пресеты, на которых запускается тест
- quality-group – получаемое качество бриллиантов. Может быть задано отдельно для Cut и Symmtry

- report_groups – разбивка примеров на группы. Если это поле не пустое, то данные отчета будут агрегированы не только по всем примерам, но и на заданных подгруппах примеров.
- csv_fields – тесты собирают максимальное количество данных о полученных решениях в архив, но в отчет будут добавлены только те данные, которые указаны в этом поле
- Перечислены только основные возможные поля. Есть также поля для более тонкой настройки теста

После создание тестового задания оно заносится в очередь. Транспортным каналом передачи данных на сервер является система Dropbox, что оказывается очень удобно, так как Dropbox позволяет следить за ходом работы с широкого спектра мобильных устройств, например, с iPad.

Далее на тестовом сервере, или нескольких серверах, по расписанию запускается скрипт, проверяющий очередь (queue_starter.py), и, в случае, если очередь не пуста и на данный момент сервер свободен, запускает первое задание из очереди.

Запуск осуществляется отдельным скриптом (big_button.py). Этот скрипт по полученному тестовому заданию подготавливает все необходимые данные для запуска. После чего параллельно запускает множество процессов с тестируемым приложением на разных данных. Одновременно выполняются по одному процессу на каждом ядре сервера (или по два, если ядра имеют технологию Hyper threading). В случае падения приложения, скрипт анализирует уже пройденные тесты и перезапускает приложение заново для продолжения задачи. Если падение происходит регулярно, тестовая система закрывает тест и отсылает письмо о некорректном тестовом модуле. По окончании работы, собирается архив с результатами в формате json, в который складываются все данные о решениях, к которым есть доступ у тестируемого модуля.

Далее на основе этого архива формируется удобная для визуального представления результатов таблица в формате Excel. Помимо основного способа формирования отчетов, есть скрипты, которые создают сравнительные таблицы на основании нескольких отчетов. Общая система генерации этих отчетов report.py содержит мини-язык генерации сводных тестируемых параметров. При необходимости, можно легко ввести новый параметр, оценивающий работу алгоритма, и вывести его для всех тестов, включая уже пройденные. Например, показать минимальное, максимальное, среднее значение, стандартное отклонение и т.п.

В Excel-отчете есть несколько закладок:

- Агрегация данных по всем запускам
- Агрегация данных по группам, указанным в report_groups
- Агрегация данных по отдельным примерам
- Агрегация данных по отдельным пресетам
- Полная таблица данных с полями, перечисленными в csv_fields
- Страница, показывающая как часто какой параметр выходит за границы

Основными тестируемыми параметрами являются:

- Средний прирост массы тестируемого алгоритма
- Средний прирост массы тестируемого алгоритма на пресетах с нормальной оптической симметрией
- Среднее время работы алгоритма
- Процент корректно сработавших запусков

Тестовая система постоянно совершенствуется. Созданная система достаточно стабильна и устойчива к падениям тестируемого модуля или перезагрузке сервера. Она позволяет глубоко исследовать результаты тестов и

находить неочевидные зависимости, а также легко получить любые необходимые данные о любом примере.

Надо заметить, что язык Python оказывался очень удобен для написания подобных тестовых систем. По сравнению с языком R, который для таких задач считается более подходящим, Python оказался предпочтительней в связи с гибкими возможностями программирования и решением смежных задач – генерацией отчетов.

4.5. Результаты работы алгоритмов планирования

Алгоритм SmartRecut и его описанные модификации внедрены под названием «Automatic Asymmetrical Smart Recut» в коммерческий продукт «HPOxygen» кампании «OctoNus Software Ltd» и уже используется на ограночных фабриках в Индии и России. Базовый алгоритм планирования круглой огранки является завершенным продуктом. Существенных изменений в нем не было с сентября 2016 года. Дополнительные модули непрерывно развиваются. Основное направление развития – работа с огранками разной формы, заданными в .asc формате.

Основные критерии оценки работы алгоритма приведены в Таблице 4. Приведенные данные получены на компьютере с процессором Intel(R) Core(TM) i7-2600 CPU @ 3.40 GHz. Также были проведены тесты с заниженной частотой процессора для имитации более слабых компьютеров. На максимальных частотах 0.9, 1.4, 1.9, 2.4, 2.9 GHz среднее время работы SR Round Convex составляет 90, 57, 42, 34, 28 секунд соответственно.

Алгоритм	SR Round convex	SR Round nonconvex	SR Oval convex	SR Oval nonconvex

Количество переменных НП-задачи	~3000	~5000	~2500	~7500
Количество ограничений НП-задачи	~11000	~20000	~10000	~45000
Средний прирост массы	2.95%	4.63%	5.95%	5.35%
Средний прирост массы с норм. опт. симметрией	2.55%	3.60%		
Среднее время работы	18.44 с.	28.73 с.	33.71 с.	110.57 с.
Процент вторых запусков	7.67%	11.24%	5.73%	6.25%
Процент успешных запусков	98.8%	98.9%	100%	100%

Таблица 4.

Заключение

Разработанная модель процесса вписывания выпуклого многогранника с заданными свойствами в невыпуклый многогранник показала себя эффективной для решения задач вписывания высокой сложности.

Опыт работы с созданным комплексом программ для численного решения задачи вписывания выпуклого многогранника с заданными свойствами позволяет сказать, что применение данного комплекса программ в научно-исследовательских или прикладных проектах, связанных с вписыванием многогранников, заметно сокращает трудоемкость программной разработки. Комплекс позволяет быстро создать основу задачи вписывания и сконцентрировать внимание на составлении геометрических ограничений, отличающих конкретную задачу от подобных ей.

Перечислим основные результаты работы:

- Дано расширенное определение для понятия вписывания многогранника в многогранник
- Предложена математическая модель процесса вписывания выпуклого трехмерного многогранника с заданными свойствами в невыпуклый многогранник.
- Проведено исследование различных алгоритмов и выбран алгоритм на базе метода внутренней точки, являющийся наиболее эффективным для решения задачи вписывания выпуклого многогранника в невыпуклый многогранник.
- На основе предложенной модели разработан и реализован комплекс программ, предназначенный для численного решения прикладных задач вписывания выпуклого многогранника с заданными свойствами в невыпуклый многогранник.
- На основе разработанного программного комплекса создан алгоритм нахождения оптимального круглого бриллианта в алмазе и его модификации, необходимые на разных стадиях обработки камня. Алгоритм внедрен в

коммерческий продукт и уже используется на ограночных фабриках в Индии, России и Бельгии.

Список литературы

1. *Moritz Firsching*, Computing maximal copies of polyhedra contained in a polyhedral // *Experimental Mathematics*. – 24:1(2015). – pp. 98–105.
2. *А.А.Болибрух*, Проблемы Гильберта (100 лет спустя) // Издательство Московского центра непрерывного математического образования. – Москва 1999.
3. *П.С. Александров*а, Проблемы Гильберта // издательство «Наука». – Москва 1969.
4. *В. А. Чуркин*, Ослабленная теорема Бибербаха для кристаллографических групп в псевдоевклидовых пространствах // *Сибирский математический журнал*. – 51/3(2010). – стр. 700–714.
5. *Bieberbach L.*, Über die Bewegungsgruppen der Euklidischen Räume. I // *Mathematische Annalen*. – 70 (1911). – pp. 297–336.
6. *Bieberbach L.*, Über die Bewegungsgruppen der Euklidischen Räume. II // *Mathematische Annalen*. – 72 (1912). – pp. 400–412.
7. *И. Кеплер.* О шестиугольных снежинках / Пер. Ю. А. Данилова. // М.: Наука. – 1982.
8. *Гильберт Д., Кон-Фоссен С.* Наглядная геометрия. – изд. 3. – М.: Наука. – 1981.
9. *T. Hales*, The status of the Kepler conjecture // *Mathematical Intelligencer*. – 16(1994). – pp. 47-58.
10. *Дж. Конвей, Н. Слоэн.* Упаковки шаров, решётки и группы. – М.: Мир, 1990. – Т. 1.
11. *И.М.Яглом*, Проблема тринадцати шаров. – издательство «Вища школа». – Киев. – 1975.
12. *О.Р.Мусин*, Проблема двадцати пяти сфер // *Успехи математических наук*. – 2003. – Т.58. – №4(352). – стр. 153-154.

13. *Shannon C. E.* A Mathematical Theory of Communication // *Bell System Technical Journal*. – 1948. – T. 27. – pp. 379–423, 623–656.
14. *E.Mooers*, Tammes's Problem. – URL: *http://www.uvm.edu/~pdodds/files/papers/others/1994/mooers1994a.pdf*
15. *A.Tarasov*, *O.Musin*, The strong thirteen spheres problem // *Topology, Geometry, and Dynamics: Rokhlin Memorial January 11-16*. – 2010. – Saint Petersburg, Russia.
16. *O.R.Musin, A.S.Tarasov*, The strong thirteen spheres problem // *Discrete and Computational Geometry*. – 48 (2012). – pp. 128-141.
17. *O.R.Musin, A.S.Tarasov*, The Tammes problem for N=14 // *Experimental Mathematics*. – 24:4(2015). – pp. 460–468.
18. *Ю. И. Валиахметова, А. С. Филиппова,* Теория оптимального использования ресурсов Л. В. Канторовича в задачах раскроя-упаковки: обзор и история развития методов решения // *Вестник УГАТУ*. – Том 18. – №1(62).
19. *Hallard T. Croft, Kenneth J. Falconer, and Richard K. Guy*. Unsolved Problems in Geometry. – Springer. – 1991.
20. *Johannes Kepler*. Harmonices Mundi. – 1619.
21. *Hallard T. Croft*. On Maximal Regular Polyhedra Inscribed in a Regular Polyhedron // *Proceedings of the London Mathematical Society 3*. – 1980. – pp. 279–296.
22. *Bernard Chazelle* The Polygon Containment Problem. // *In Advances in Computing Research I*, edited by Franco P. Preparata. – pp. 1–33. – JAI Press. – 1983.
23. *Pankaj K. Agarwal, Nina Amenta, and Micha Sharir*. Largest Placement of One Convex Polygon Inside Another. // *Discrete and Computational Geometry 19*. – 1998. – pp. 95– 104.
24. *Stephen J. Dilworth and Sateesh R. Mane.* On a Problem of Croft on Optimally Nested Regular Polygons // *Journal of Geometry 99*. – (2010). – pp. 43–66.
25. *Matthew Hudelson, Victor Klee, and David Larman.* Largest j-Simplices in d-Cubes: Some Relatives of the Hadamard Maximum Determinant Problem // *In*

Linear Algebra and Its Applications. – pp. 519–598. – Proceedings of the Fourth Conference of the International Linear Algebra Society. – 1996.

26. *Hiroshi Maehara, Imre Z. Ruzsa, and Norihide Tokushige*. Large Regular Simplices Contained in a Hypercube // *Periodica Mathematica Hungarica 58*. – 2009. – pp. 121– 126.
27. *В.А. Емеличев, М.М.Ковалев, М.К. Кравцов*, Многогранники, графы, оптимизация. – Москва "Наука". – 1981.
28. *R. Fourer, D. M. Gay, and B. W. Kernighan*, AMPL: A Modeling Language For Mathematical Programming. – Duxbury Press/Brooks/Cole Publishing Company. – 2003.
29. *Y. Kawajir, C. Laird, A. Wächter*, Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt. – 2010. – URL: http://www.coin-or.org/Ipopt/documentation/
30. *И.И. Дикин* Метод внутренних точек в линейном и нелинейном программировании. – Красанд. – 2010.
31. *А.П.Афанасьев, В.В.Волошинов, А.А.Лисов, А.К.Наумцева*, Организация распределенной обработки данных с помощью RESTful-веб-сервисов // Современные проблемы науки и образования. – 2012. – №6. – (приложение "Технические науки"). – С. 31
32. *В.В. Волошинов, С.А. Смирнов.* Принципы интеграции программных ресурсов в научных вычислениях // *Журнал "Информационные технологии и вычислительные системы"*. – №3. – 2012. – с. 66-71

Публикации автора по теме диссертации

К1. *Кокорев Д.С.* Разработка алгоритма нахождения наибольшей сферической области в многограннике, заданном набором вершин // *Труды 55-й научной конференции МФТИ «Проблемы фундаментальных и прикладных естественных и технических наук в современном информационном обществе»*. – М.: МФТИ. – 2012. – стр. 35-36.

К2. *Кокорев Д.С.* Алгоритм поиска выпуклого многогранника максимального объема, вписанного в другой многогранник // *Информационные технологии и вычислительные системы*. – выпуск №3. – 2013. – стр. 27-31.

К3. *Кокорев Д.С.* Алгоритм поиска выпуклого многогранника максимального объема с заданной комбинаторной структурой, вписанного в другой многогранник // *Труды 56-й научной конференции МФТИ «Актуальные проблемы фундаментальных и прикладных наук в современном информационном обществе»*. – М.: МФТИ. – 2013. – стр. 88-90.

К4. *Кокорев Д.С.* Алгоритмы вписывания выпуклых многогранников, использующие численные методы оптимизации // *Сборник трудов 38-ой конференции-школы ИППИ РАН «Информационные технологии и системы»*. – 2014. – Нижний Новгород, Россия. – стр. 297-301.

К5. *Кокорев Д. С., Тарасов А. С.* Тестирование алгоритма, максимизирующего объем вписанного многогранника, в среде высокопроизводительных вычислений // *Сборник тезисов докладов НСКФ'2014*. – URL: http://2014.nscf.ru/TesisAll/7_Reshenie_zadach_optimizatsii/09_161_KokorevDS.pdf

К6. *Кокорев Д. С.* Тестирование алгоритма, максимизирующего объем вписанного многогранника, в среде высокопроизводительных вычислений //

сборник тезисов докладов НСКФ'2015. – URL: http://2015.nscf.ru/TesisAll/4_Reshenie_zadach_optimizatsii/08_467_KokorevDS.pdf

К7. *Кокорев Д.С.* Оптимизационный алгоритм поиска вписанного многогранника максимального объема // *Программные продукты и системы*. – выпуск №1. – 2016. – стр. 90-95.

К8. *Denis Kokorev, Nikolay Piskovatskii* Applied problems of polyhedron inscribing task solving // *Procedia Computer Science*. – Volume 101. – 2016. – pp. 227-232.

К9. *Denis Kokorev*, Прикладные проблемы решения задачи вписывания многогранников // *Selected Papers of the 7th International Conference Distributed Computing and Grid-technologies in Science and Education (GRID 2016)*. – Dubna, Russia. – 2016. – стр. 295-301. – URL: http://ceur-ws.org/Vol-1787/295-301-paper-50.pdf