МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ

Факультет Механико-математический
Кафедра Программирования

Направление подготовки Математика и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Маляренко Артём Андреевич

(Фамилия, Имя, Отчество автора)

Тема работы: Алгоритмы геометрической декомпозиции и методы алгебраического моделирования в задаче удовлетворения геометрических ограничений.

«К защите допущен»

Вр.и.о. зав. кафедрой,
к.ф.-м.н.

Емельянов П.Г /_____
(фамилия , И., О.) (подпись, МП)

«….»……2026 г.

Научный руководитель

к.ф.-м.н.,
доцент кафедры применения
мат. методов в экономике

Рыков И.А. /_____
(фамилия , И., О.) (подпись, МП)

«….»……2026 г.

Дата защиты: «….» ……2026 г.

Новосибирск, 2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **4**

1. Постановка задачи и алгоритмическая схема её решения **8**

1.1. Задача удовлетворения геометрических ограничений 8

1.2. Архитектура решателей 10

1.3. Геометрическая декомпозиция 11

1.3.1. Представление системы геометрических ограничений 12

1.3.2. Методы структурной декомпозиции 14

1.4. Алгебраическое моделирование 22

1.4.1. Построение системы нелинейных уравнений 22

1.5. Формальная постановка задачи 25

2. Реализованный подход **25**

2.1. Реализация основных классов 25

2.1.1. Класс сущности 26

2.1.2. Архитектура геометрических объектов 26

2.1.3. Архитектура ограничений 27

2.1.3.1. Логические ограничения 27

2.1.3.2. Размерные ограничения 28

2.2. Реализация декомпозиции 29

2.2.1. Упрощение 30

2.2.1.1. Шаблоны, основанные на классах эквивалентности 30

2.2.1.2. Удаление дубликатов 32
2.2.1.3. Удаление ограничений между фиксированными объектами 33
2.2.1.4. Удаление симметричных ограничений . . . 34
2.2.1.5. Слияние объектов 36
2.2.1.6. Другие шаблоны 37
2.2.2. Отделяющая декомпозиция 38
2.2.2.1. Отделение по единственному ограничению 39
2.2.2.2. Прямая, связанная ограничением расстояния с прямой, инцидентной той же плоскости 40
2.2.3. Введение жёсткости 40
2.2.4. Двусвязная декомпозиция 42
2.3. Моделирование 45
2.3.1. Моделирование геометрических объектов 45
2.3.2. Моделирование ограничений 46

3. Результаты **47**

4. Заключение **51**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **52**

ВВЕДЕНИЕ

Системы автоматизированного проектирования или САПР (англ. Computer-Aided Design, CAD) по праву являются важнейшей частью инженерного программного обеспечения. Использование подобных систем повышает точность, скорость и эффективность проектирования. САПР широко используется в множестве отраслей: архитектура и строительство, автомобилестроение, энергетика, медицина и биоинженерия.

Развитие систем проектирования выявило потребность в параметризации геометрических моделей, что помогло значительно ускорить производство и уменьшить его издержки.

Наиболее важной и сложной задачей при реализации САПР является решение задачи удовлетворения геометрических ограничений (ЗУГО). Исходными данными задачи являются геометрические объекты и ограничения, задающие отношения между этими объектами. Такие задачи сводятся к решению системы нелинейных уравнений, в общем случае не имеющей алгоритма точного решения за полиномиальное время. Программные компоненты, решающие эти задачи, называются решателями геометрических ограничений. Решатели геометрических ограничений состоят из нескольких компонент. Ключевыми среди них являются компоненты алгебраического моделирования и геометрической декомпозиции исходной задачи.

Исходная задача представляется следующим образом: геометрические объекты представляются набором параметров (точка — её координатами в пространстве, плоскость — вектором нормали и сдвигом относительно начала отсчёта), а ограничения, в свою очередь, задаются уравнениями на

эти параметры. Например, перпендикулярность двух плоскостей может быть задана уравнением $\mathbf{n}_1 \cdot \mathbf{n}_2 = 0$, где $\mathbf{n}_1$ и $\mathbf{n}_2$ — векторы нормалей этих плоскостей. Далее задача сводится к поиску решения системы уравнений, задающих ограничения в терминах параметров геометрических объектов.

Для нахождения искомых параметров применяются методы алгебраического моделирования ЗУГО, при которых заданные ограничения приводят к системе, в общем случае, нелинейных уравнений. Полученная система нелинейных алгебраических уравнений (СНАУ) решается с использованием итерационных численных методов, в частности метода Ньютона-Рафсона. Метод Ньютона-Рафсона обладает квадратичной скоростью сходимости, то есть норма невязки на каждой итерации убывает пропорционально квадрату невязки на предыдущей итерации.

Трудоёмкость одной итерации метода Ньютона при решении системы линейных уравнений с общей плотной матрицей Якоби размера $n \times n$ оценивается как $O(n^3)$ [2]. Это определяет асимптотику трудоёмкости всего алгоритма, поскольку остальные этапы имеют линейную или квадратичную сложность. В ЗУГО матрицы Якоби, как правило, являются разреженными: в каждой строке присутствует ограниченное число ненулевых элементов. Использование специализированных методов для разреженных матриц позволяет существенно снизить вычислительную сложность одной итерации. Тем не менее, при типичных размерах практических задач, включающих тысячи геометрических объектов и ограничений, вычислительные затраты остаются критичными с точки зрения интерактивности работы пользователя.

Важными метриками оценки работы геометрического решателя явля-

ются процент успешно решённых задач (то есть доля задач, решённых на заданной базе), а также производительность (то есть скорость решения типичных индустриальных задач с большим количеством ограничений). Значимость этих метрик определяется областью применения геометрических решателей: пользователь САПР должен иметь возможность максимально точно проектировать модель, не замечая процесса поиска её корректного положения и связанных с этим задержек.

Существенное влияние на указанные характеристики оказывает модуль геометрической декомпозиции — совокупность методов, выполняющих разбиение исходной задачи на более простые подзадачи ещё на этапе обработки входных данных. Такое разбиение должно обеспечивать, что решение всех полученных подзадач приводит к решению исходной задачи в целом. Эффективность работы данного модуля во многом определяет качество решателя: применение метода Ньютона–Рафсона непосредственно к полной системе ограничений может приводить к значительным задержкам, заметным пользователю, а также к снижению надёжности получения решения.

Модуль геометрической декомпозиции, пожалуй, является одной из важнейших частей всего решателя геометрических ограничений. Как было сказано ранее, именно этот модуль обеспечивает высокую производительность, а также повышает успешность решения больших задач. Для декомпозиции исходной системы геометрических ограничений часто применяются различные алгоритмы. В свою очередь, каждый из алгоритмов реализует множество техник, с помощью которых изначальная задача упрощается и разбивается на некоторое количество мелких подзадач.

Известные методы геометрической декомпозиции могут быть отнесены

к одному из четырёх классов: упрощение (англ. simplification), введение жесткости (англ. rigidification), отделяющая декомпозиция (англ. cutting decomposition), разделение на двусвязные компоненты (англ. bicomponent decomposition).

Алгоритм упрощения удаляет избыточные ограничения, что упрощает исходную систему.

Метод введения жесткости, в свою очередь, помогает объединить несколько так называемых твёрдых тел (определение будет дано позже) в одно, благодаря чему уменьшается количество независимых геометрических объектов.

Главная цель модуля отделяющей декомпозиции — поиск таких объектов, отделение которых не влияет на решение полной задачи. После отделения и решения редуцированной системы, отрезанные объекты могут быть корректно восстановлены, а значит, решение полной системы может быть легко найдено.

С помощью алгоритма разделения на двусвязные компоненты исходная система геометрических ограничений при выполнении некоторых условий может быть разбита на подсистемы, каждая из которых решается независимо. Последующее совмещение решений подсистем позволяет получить решение исходной задачи, что способствует повышению производительности решателя.

Цель данной работы: разработка модулей алгебраического моделирования и геометрической декомпозиции для решателя геометрических ограничений, а также исследование влияния различных техник декомпозиции на его производительность.

1. Постановка задачи и алгоритмическая схема её решения

1.1. Задача удовлетворения геометрических ограничений

Одной из важнейших предпосылок для формирования задачи удовлетворения геометрических ограничений стало появление параметрического проектирования в САПР. Параметрическое проектирование строится на основе параметрических операций, которые задаёт пользователь. Эти операции представляются в виде последовательности шагов, которой может управлять разработчик модели. Такая последовательность называется историей построения модели. Каждая параметрическая операция принимает на вход некоторый параметр (например, радиус скругления угла) и возвращает новую геометрию, в соответствии с этим параметром. Следовательно, изменение параметра ведёт к пересчёту всей истории построения и позволяет автоматически обновлять геометрическую модель.

Главной особенностью таких операций является фиксированная направленная зависимость между входными и выходными объектами. Такая направленная зависимость позволяет строить сложные модели, однако требует, чтобы пользователь заранее планировал, как управлять параметрами модели. В противном случае, могут возникать циклические зависимости, которые придётся разрывать вручную.

В задаче удовлетворения геометрических ограничений, напротив, не существует фиксированной направленной зависимости между объектами. Пользователь сначала задаёт набор геометрических объектов, с которыми

планирует работать, а затем накладывает необходимые ограничения. Далее в работу вступает решатель геометрических ограничений, который находит положение геометрических объектов, удовлетворяющее всем заданным ограничениям.

Таким образом, параметрическое проектирование можно рассматривать как частный случай ЗУГО, в которой существует жёсткая направленность между объектами. Именно это стало предпосылкой формирования задачи удовлетворения геометрических ограничений.

Определение 1.1. *Пусть существует язык с заданной семантикой. Тогда системой геометрических ограничений будем называть упорядоченную тройку* (G, C, P)*, где* G *— конечное множество переменных (геометрических объектов),* C *— конечное множество геометрических ограничений (термов языка), а* P *— множество параметров (констант).*

Именно геометрические объекты и ограничения являются главными сущностями в задаче удовлетворения геометрических ограничений [6].

$$\begin{cases} c1 : \text{fix}(A) \\ c2 : \text{fix}(B) \\ c3 : \text{dis}(A, B) = 3 \\ c4 : \text{dis}(C, D) = 4 \\ c5 : \text{dis}(D, C) = 4 \\ c6 : \text{par}(\delta_1, \delta_3) \end{cases} \qquad \begin{array}{r} c7 : \text{par}(\delta_1, \delta_2) \\ c8 : \text{par}(\delta_2, \delta_3) \\ c9 : \text{inc}(\gamma_1, \gamma_2) \\ c10 : \text{dis}(\gamma_2, \gamma_3) = \alpha \\ c11 : \text{ang}(\gamma_3, \delta_3) = d \end{array}$$

Рис. 1.1: Пример системы геометрических ограничений

Определение 1.2. *Пусть дана система геометрических ограничений* $CS = (G, C, P)$*, где* D *— множество всех допустимых значений параметров геометрических объектов. Говорят, что система геометрических ограничений имеет решение, если существует такое означивание* $\sigma : G \rightarrow$

$D : \forall c \in C : c$ *— истинен на этом означивании. При этом означивание* σ *называется решением системы геометрических ограничений* CS.

Определение 1.3. *Задачей удовлетворения геометрических ограничений (ЗУГО) будем называть нахождение решения означивания* σ *для заданной системы геометрических ограничений* $CS = (G, C, P)$. *Если означивания не существует, то задача считается неразрешимой.*

1.2. Архитектура решателей

Современные решатели геометрических ограничений, как правило, имеют модульную архитектуру. Основными модулями решателя являются:

- Модуль предварительной обработки ограничений и геометрических объектов, позволяющий упростить сложные ограничения, которые были переданы на вход решателю.
- Модуль геометрической декомпозиции, выполняющий разбиение исходной системы ограничений на подзадачи, решение которых позволяет получить решение исходной задачи.
- Модуль алгебраического моделирования, позволяющий перевести геометрические ограничения в систему нелинейных уравнений.
- Модуль численного решения системы нелинейных уравнений, позволяющий найти решение системы с помощью численных методов.
- Модуль валидации найденного решения, позволяющий проверить корректность найденного решения.

Такая модульность обусловлена сложностью решения исходной задачи, а также необходимостью обеспечить высокую производительность

решателя при работе с большими индустриальными задачами. Модуль геометрической декомпозиции работает с графовым представлением системы геометрических ограничений и позволяет упростить исходную задачу.

Модуль алгебраического моделирования отвечает за перевод геометрических ограничений в систему нелинейных уравнений. Далее система решается с помощью численных методов, таких как метод Ньютона–Рафсона.

Модули валидации позволяют избежать лишних вычислений в случае, если система изначально неразрешима, а также проверить корректность найденного решения.

1.3. Геометрическая декомпозиция

Для ЗУГО характерен экспоненциальный рост числа возможных конфигураций по мере увеличения количества объектов и ограничений, что делает решение крупных задач вычислительно затратным.

Именно поэтому в современных решателях геометрических ограничений широко применяются численные методы поиска решения, например метод Ньютона–Рафсона. Однако в общем случае такие методы не гарантируют нахождение решения, поскольку их сходимость зависит от свойств задачи и выбора начального приближения. Кроме того, один шаг подобных алгоритмов, как правило, имеет кубическую трудоёмкость от количества переменных, поэтому уменьшение размера системы позволяет существенно сократить время решения.

Всё это приводит к необходимости применения различных техник гео-

метрической декомпозиции, позволяющих упростить исходную задачу. Именно геометрическая декомпозиция является одним из основных способов повышения производительности решателя [8], а также увеличения процента успешно решаемых задач.

1.3.1. Представление системы геометрических ограничений

Перед началом обзора техник геометрической декомпозиции, необходимо рассмотреть то, каким образом может быть представлена исходная система ограничений.

Существует шесть основных представлений системы геометрических ограничений [4]: графовый, алгебраический, логический, численный, символьный и метод, основанный на доказательстве теорем. В геометрических решателях часто совмещаются несколько представлений, однако наиболее популярными являются графовый, алгебраический и численный подходы. Именно представление исходной системы в виде графа активно применяется при декомпозиции задачи.

Определение 1.4. *Графом системы геометрических ограничений* $CS = (G, C, P)$ *называется граф* $H = (V, E)$*, где* V *- множество, соответствующее геометрическим объектам* G*, а* E *- множество рёбер, где каждое ребро* $e = (v_i, v_j) \in E \iff c(v_i, v_j) \in C$.

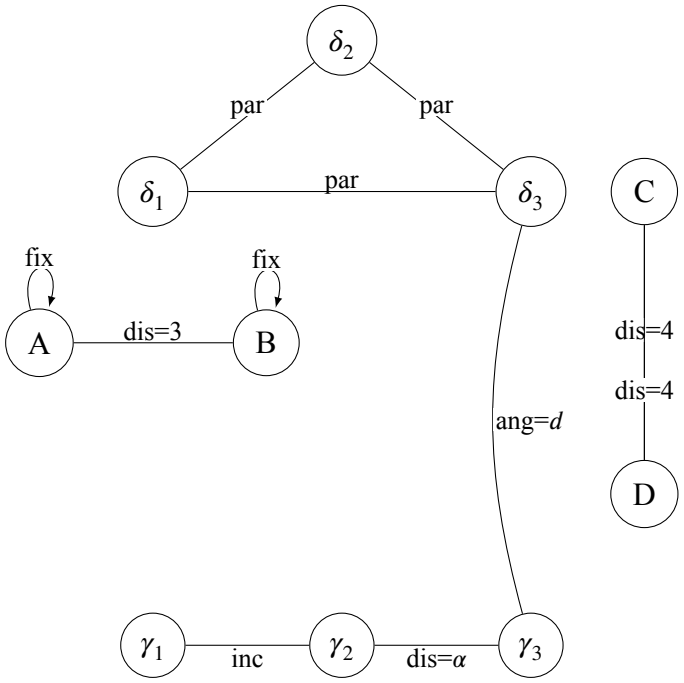

Рис. 1.2: Граф системы геометрических ограничений для системы 1.1

Благодаря графовому представлению системы геометрических ограничений, возможно применение структурного анализа графов для упрощения исходной задачи и разбиения её на подзадачи, а также анализа степеней свободы геометрических объектов [1].

Структурный подход заключается в том, что исходный граф системы анализируется на наличие в нём различных подструктур, которые могут быть упрощены, удалены или перекомбинированы без потери корректности задачи. Такие подструктуры могут быть найдены с помощью различных алгоритмов, таких как поиск двусвязных компонент, нахождения классов эквивалентности и т. д. Получившийся граф в итоге разбивается на несколько подграфов, которые соответствуют независимым подсистемам. К каждой из подсистем далее может быть применён алгебраический или численный подход. Далее будут более подробно рассмотрены различные методы декомпозиции.

1.3.2. Методы структурной декомпозиции

В данной секции будут рассмотрены основные методы геометрической декомпозиции, которые были в основном описаны в [1; 4; 5].

Часто кроме разделения методов декомпозиции на структурные и анализирующие количества степеней свободы, выделяют также разделяющие и объединяющие систему ограничений методы. [5].

Разделяющие методы направлены на удаление избыточных ограничений, а также на разбиение исходной задачи на несколько независимых подзадач.

Корректные разделяющие методы геометрической декомпозиции разбивают систему на подсистемы так, что каждую из них можно решать отдельно и затем из решений подсистем получить решение общей системы.

Объединяющие, напротив, направлены на слияние геометрических объектов и ограничений, что позволяет уменьшить общее количество независимых геометрических объектов и, соответственно, упростить исходную задачу.

Метод упрощения

Упрощение системы ограничений является одним из основных методов геометрической декомпозиции. Как правило, упрощение применяется на начальном этапе работы решателя. Её главная цель — оптимизировать исходную систему ограничений путём удаления из неё избыточных ограничений. В работах А.Г.Ершова [1] и C. Jermann [5] упоминается возможность такого анализа исходной системы, однако конкретные алгоритмы не приво-

дятся.

Наиболее популярным способом нахождения избыточных ограничений является выделение классов эквивалентности на геометрических объектах относительно определённых свойств, задаваемых ограничениями. Основные классы эквивалентности, которые могут быть выделены в системе геометрических ограничений:

- Классы инцидентности, представляющие собой множества геометрических объектов, на которые наложено ограничение совпадения (инцидентности). Исходя из графа системы 1.2, можно выделить один класс инцидентности γ_1 и γ_2 (при условии, что они принадлежат одному и тому же геометрическому типу).
- Классы параллельности, представляющие собой множества геометрических объектов, на которые наложено ограничение параллельности. Исходя из графа системы 1.2, можно выделить один класс параллельности δ_1, δ_2 и δ_3. Благодаря выделению подобного класса легко видеть избыточные ограничения: в данном случае, любое из трёх ограничений будет избыточным, так как параллельность любых двух геометрических объектов из класса влечёт параллельность третьей.
- Классы соосности, классы равного радиуса могут быть выделены аналогичным образом.

Несмотря на то, что по своей сути упрощение является разделяющим методом декомпозиции, в рамках данного метода могут использоваться техники, добавляющие ограничения в систему.

Метод двусвязной декомпозиции

Метод двусвязной декомпозиции основывается на поиске так называемых точек сочленения в графе системы геометрических ограничений. [10]

Определение 1.5. *Точкой сочленения в неориентированном графе* $G = (V, E)$ *называется такая вершина* $v \in V$*, что удаление этой вершины вместе со всеми инцидентными ей рёбрами увеличивает количество связных компонент графа.*

Удаление точек сочленения позволяет разбить исходную задачу на несколько подзадач, которые могут решаться полностью независимо. Другими словами удаление подобных точек разбивает граф системы на двусвязные компоненты, которые, очевидно, не являются зависимыми друг от друга.

Поиск точек сочленения и двусвязных компонент может быть выполнен за линейное время $O(|V| + |E|)$ с использованием обхода графа в глубину [9]. Сам по себе данный метод позволяет выделять независимые подзадачи, однако его эффективность существенно возрастает при совместном использовании с другими техниками декомпозиции. В частности, отделяющая декомпозиция может разрывать циклы в графе ограничений и тем самым создавать новые точки сочленения, а введение жёсткости объединяет группы вершин, увеличивая вероятность появления точек сочленения. В результате комбинация этих методов позволяет выполнять более глубокую декомпозицию системы геометрических ограничений.

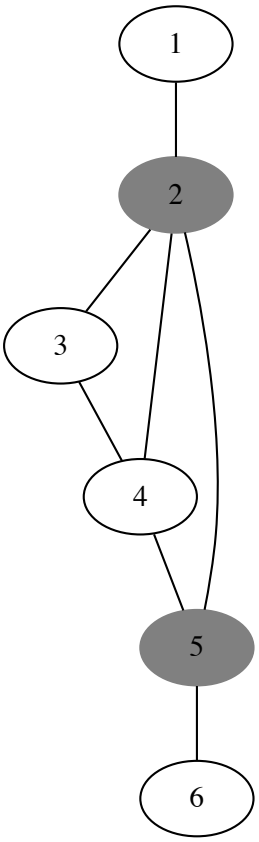

Рис. 1.3: Пример графа с двумя точками сочленения (выделены серым)

Отделяющая декомпозиция

Не менее важным методом является геометрическая декомпозиция. Именно этот метод применяется в тех случаях, когда декомпозиция по вершинам сочленения 1.3.2 не может быть применена. [1]. Суть метода заключается в поиске и циклов ограничений, разрыв которых не повлияет на корректность решения задачи.

Для нахождения таких ограничений применяется локальный анализ графа системы. т. е. анализируется соответствие геометрического объекта и наложенных на него ограничений некоторому шаблону. Если шаблон найден, то соответствующие ограничения и объекты удаляются из системы, а затем, после решения редуцированной задачи, корректно восстанавлива-

ются.

Популярным примером такого шаблона является прямая, на которую наложено два ограничения инцидентности двум различным точкам. В этом случае мы можем отделить из системы прямую и наложенные ограничения совпадения, решить редуцированную задачу, а затем корректно восстановить положение прямой исходя из положений двух точек. Эта декомпозиция корректна, ведь через любые две точки можно однозначным образом провести прямую, а соответственно и удовлетворить два удалённых ограничения инцидентности.

Пример с прямой и двумя точками не является единственным шаблоном, который может быть найден в системе. В [1] приводится несколько примеров различных шаблонов:

- Прямая в двумерном пространстве с двумя наложенными ограничениями: параллельность другой прямой и расстояние до точки или окружности.
- Прямая в двумерном пространстве с двумя наложенными ограничениями: угол с другой прямой и инцидентность точке или касание окружности.
- Другие комбинации этих ограничений с прямой.
- Окружность в двумерном пространстве с двумя наложенными ограничениями: инцидентность с точкой и касание прямой или другой окружности.

Часто удаление таких шаблонов приводит к появлению новых шаблонов, которые также могут быть удалены. Поэтому отделяющая декомпозиция

реализуется, как итеративный процесс нахождения шаблонов.

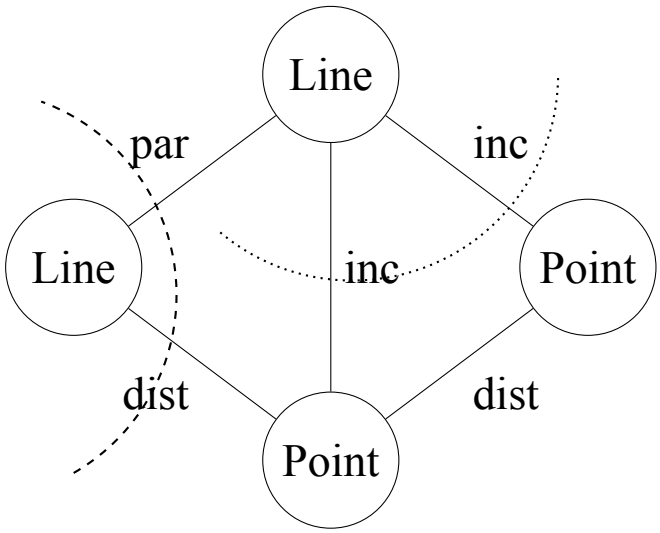

Рис. 1.4: Пример последовательного применения отделяющих шаблонов к системе геометрических ограничений

Кроме того, отделяющая декомпозиция помогает эффективно разбивать циклы ограничений в графе системы, что является важнейшим преимуществом данного метода, ведь наиболее сложными задачами являются ЗУГО, в которых присутствуют циклы ограничений [3].

Несмотря на то, что список является далеко не исчерпывающим, применение метода отделяющей декомпозиции в трёхмерном пространстве не является тривиальной задачей и требует дальнейших исследований, ведь на данный момент не даёт столь же хороших результатов, как в двумерном пространстве.

Метод введения жесткости

Введение жёсткости является одним из немногих объединяющих методов геометрической декомпозиции. Для понимания данного метода, необходимо ввести понятие жёсткого множества.

Определение 1.6. *Жёстким множеством (англ. rigid set) будем называть такой набор геометрических объектов $R \subseteq G$ в системе геометрических ограничений $CS = (G, C, P)$, что при любом решении системы относительное положение объектов внутри R остаётся неизменным.*

Суть введения жёсткости заключается в слиянии нескольких жёстких множеств, которые связаны между собой некоторыми ограничениями, в одно. Такое слияние обнаруживается путём анализа графа системы геометрических ограничений. Находятся пары жёстких множеств, взаимное положение которых может быть однозначно определено на основе ограничений между ними. Далее оба жёстких множества объединяются в одно, с помощью применения трансформации. После этого процесс повторяется итеративно.

На первом этапе осуществляется поиск ограничений между геометрическими объектами, принадлежащими различным жёстким множествам. Рассматриваются только бинарные ограничения.

Каждое такое ограничение интерпретируется как локальное ограничение относительной степени свободы между соответствующими жёсткими множествами. В процессе анализа ограничения последовательно уточняют допустимые типы относительного движения между множествами (например, от свободного движения к вращательному, далее к поступательному и, при достаточном числе ограничений, к жёсткому соединению).

Если совокупность ограничений между двумя жёсткими множествами полностью устраняет относительную степень свободы, соответствующие множества объединяются. Более детальное описание алгоритма анализа ограничений и его реализации выходит за рамки данной работы и рассмат-

ривается как направление дальнейших исследований.

1.4. Алгебраическое моделирование

Несмотря на рассмотренные выше методы геометрической декомпозиции, решение ЗУГО в общем случае [1] требует применения численных методов и сводится к решению системы нелинейных уравнений (СНАУ).

1.4.1. Построение системы нелинейных уравнений

Популярным методом алгебраического моделирования является декартово моделирование [1], при котором координаты объектов рассматриваются как переменные, а ограничения представляются в виде уравнений, связывающих эти переменные. Например, ограничение расстояния между двумя точками $dis(A, B) = d$ может быть представлено в виде уравнения:

$$(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2 - d^2 = 0$$

Таким образом, каждое ограничение из системы геометрических ограничений может быть представлено в виде одного или нескольких уравнений, а значит, вся система может быть представлена в виде системы нелинейных уравнений.

При генерации системы уравнений важно также учитывать класс, к которому принадлежит геометрическое ограничение.

Определение 1.7. *Пусть есть система геометрических ограничений* $CS = (G, C, P)$. *Размерным ограничением будем называть такие ограничения* $c \in C$, *которые содержат параметры из множества* P.

[1] Иногда благодаря применению декомпозиций возможно разбить задачу на достаточно простую подзадачу, решение которой можно найти аналитически.

Примером размерных ограничений являются, например, ограничения расстояния, радиуса, угла и т. д.

Определение 1.8. *Пусть есть система геометрических ограничений* $CS = (\boldsymbol{G}, \boldsymbol{C}, \boldsymbol{P})$. *Логическим ограничением будем называть такие ограничения* $c \in \boldsymbol{C}$, *которые задают относительное положение геометрических объектов без использования параметров из множества* $\boldsymbol{P}$.

Примерами таких ограничений являются ограничения параллельности, перпендикулярности, инцидентности и т. д.

Определение 1.9. *Пусть есть система геометрических ограничений* $CS = (\boldsymbol{G}, \boldsymbol{C}, \boldsymbol{P})$. *Инженерным ограничением будем называть такие ограничения* $c \in \boldsymbol{C}$, *которые используются для сопоставления размеров проектируемой модели.*

Примерами инженерных ограничений являются ограничения симметрии, равенства расстояний и т. д.

Также у ограничений выделяют логические атрибуты. Атрибутами геометрических ограничений являются:

- Ориентация (используется только в размерных ограничениях), определяющая положение одного объекта относительно нормали другого объекта.
- Выравнивание, которое определяет со- или противонаправленность направлений двух геометрических объектов.
- Вспомогательные точки (параметры) — специальные точки (параметры), которые задаются пользователем для уточнения геометрического положения, в котором должно быть удовлетворено ограничение.

Для моделирования геометрических объектов используются наборы независимых параметров, которые однозначно определяют положение геометрического объекта. Например, для точки в трёхмерном пространстве таких параметров три: её декартовы координаты (x, y, z). Прямая моделируется пятью независимыми параметрами: координатами начальной точки и направляющим вектором, который может быть представлен с помощью двух углов.

Тем самым при генерации системы нелинейных уравнений необходимо учитывать тип ограничения, класс, к которому оно принадлежит (размерное, логическое или инженерное), набор геометрических объектов, на которые наложено данное ограничение, а также его атрибуты (ориентация, выравнивание, вспомогательные точки). Именно все эти характеристики влияют на то, каким образом ограничение будет представлено в виде уравнения или набора уравнений.

1.5. Формальная постановка задачи

В рамках данной работы планируется реализация модуля геометрической декомпозиции, включающего в себя описанные выше методы, исследование применимости различных техник и шаблонов для метода упрощения и отделяющей декомпозиции, а также реализация модуля алгебраического моделирования для следующих геометрических ограничений и объектов:

Ограничения	**Геометрические объекты**
Параллельность	Прямая
Перпендикулярность	Точка
Угол	Цилиндр
Расстояние	Плоскость
Совпадение	Окружность
Соосность	Тор
Фиксация	Эллипс
Радиус	Конус
Касание	Сфера
Концентричность	-

Таблица 1.1

2. Реализованный подход

2.1. Реализация основных классов

Кроме геометрической декомпозиции и алгебраического моделирования, в рамках данной работы были реализованы базовые классы, которые используются повсюду в реализации.

2.1.1. Класс сущности

Для обобщенной работы с ограничениями и геометрическими объектами был создан абстрактный класс `Entity`. Этот класс предоставляет следующий интерфейс:

```
abstract class Entity:
    abstract def getEntityType() -> EntityType;
    abstract def isConstraint() -> bool;
    abstract def isGeometry() -> bool;
    abstract def accept(EntityVisitor visitor);
```

Рис. 2.1: Псевдокод интерфейса класса Entity

Классы геометрических объектов и ограничений наследуются от класса сущности и реализуют соответствующие методы.

2.1.2. Архитектура геометрических объектов

Класс `Geometry`, являющийся наследником Entity, является базовым классом для всех геометрических объектов. Данный класс реализует методы isGeometry и isConstraint. Методы accept и getEntityType реализуются лишь в листьях дерева наследования т. е. в конкретных геометрических объектах.

Геометрические объекты можно разбить на два вида: канонические и параметрические. Каноническими геометрическими объектами являются прямые наследники класса `Geometry`. К данному виду относится точка, прямая, плоскость и сфера. Параметрические классы реализуют интерфейс абстрактных классов `Curve` и `Surface` наследников класса `Geometry`. Наследниками `Curve` являются классы `Circle` (Окружность), `Ellipse` (Эллипс). В свою очередь наследниками класса `Surface` являются `Cylinder` (Цилиндр), `Cone`

(Конус) и `Torus` (Тор). Кроме того, существует геометрический объект `RigidSet`, который не относится ни к каноническим, ни к параметрическим геометрическим объектам. Он представляет собой множество геометрических объектов, положения которых зафиксированы относительно друг друга.

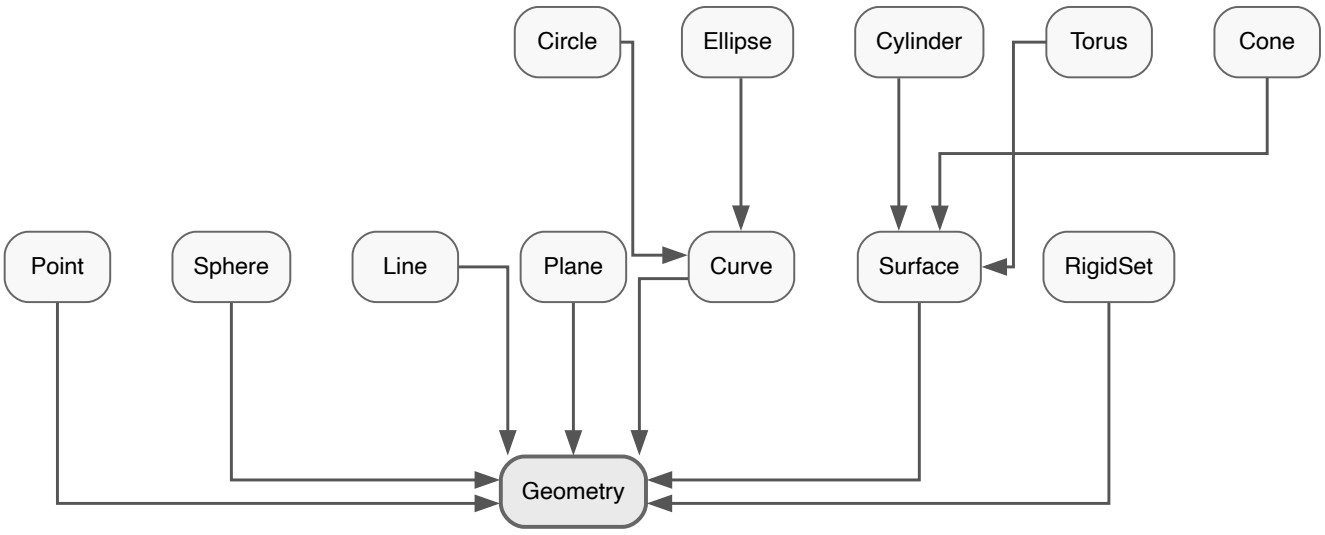

Рис. 2.2: Граф наследования классов геометрических объектов

2.1.3. Архитектура ограничений

Аналогично классу `Geometry`, класс `Constraint` является базовым классом для всех геометрических ограничений. Указанный класс реализует методы isGeometry и isConstraint. Методы accept и getEntityType реализуются лишь в листьях дерева наследования т. е. в конкретных ограничениях.

2.1.3.1 Логические ограничения

Класс логических ограничений `LogicalConstraint` является наследником класса `Constraint`. Кроме реализации методов базового класса, он

также имеет атрибут выравнивания и набор вспомогательных точек и параметров, которые определяют конкретное положение, в котором ограничение должно быть удовлетворено.

Атрибут выравнивания определяет со- или противонаправленность векторов, используемых при моделировании ограничения. Выравнивание может быть положительным (англ. Positive), отрицательным (англ. Negative), текущим (англ. Current) или неопределённым (англ. Undefined). Текущее выравнивание автоматически преобразуется в положительное или отрицательное в зависимости от изначального положения геометрических объектов, на которые наложено логическое ограничение.

```
enum Alignment:
    POSITIVE
    NEGATIVE
    CURRENT
    UNDEFINED
```

Рис. 2.3: Псевдокод перечисления Alignment

Вспомогательные точки и параметры либо задаются пользователем при определении системы ограничений, либо генерируются автоматически на основе текущих положений геометрических объектов.

2.1.3.2 Размерные ограничения

Размерные ограничения, в свою очередь, являются наследниками класса `DimensionalConstraint`, который, в свою очередь, является наследником класса `LogicalConstraint`. Важнейшим отличием размерных ограничений от логических является наличие параметра, который задаётся пользовате-

лем. Например, ограничения расстояния, угла, радиуса являются размерными.

Кроме наличия параметра, размерные ограничения также имеют атрибут ориентации, который определяет конкретное положение, в котором должно быть удовлетворено ограничение. Ориентация может быть положительной, отрицательной или неопределённой. Например, для ограничения расстояния между точкой и плоскостью, положительная ориентация будет означать, что точка должна находиться с той же стороны от плоскости, что и её нормаль, отрицательная ориентация будет означать, что точка должна находиться с противоположной относительно нормали стороны. Если же ориентация является неопределенной, то ограничение может быть удовлетворено в любом положении относительно нормали плоскости.

2.2. Реализация декомпозиции

Применение декомпозиций к системе геометрических ограничений реализовано с помощью специального класса `DecompositionManager`, который определяет, как декомпозиции будут применены, а также их порядок применения. `DecompositionManager` инициализируется очередью типов стадий декомпозиций. При этом каждая декомпозиция реализована в виде отдельного класса, наследующего абстрактный класс `Decomposer`. Каждая декомпозиция инициализируется системой ограничений, реализует метод `execute` и метод `update`. Функция `execute` применяет соответствующую декомпозицию к системе и возвращает список систем, полученных после данной декомпозиции. Для каждой системы из списка инициализируется

новая стадия, после чего для неё также вызывается метод `execute`. Метод `update` в свою очередь вызывается на обратном ходе при обходе дерева декомпозиций. Он позволяет корректно обновить исходную систему ограничений с помощью решения подсистемы, полученной после применения декомпозиции.

2.2.1. Упрощение

Упрощение системы ограничений реализовано в виде класса `SimplifierDecomposer`, который наследуется от абстрактного класса `Decomposer`. В рамках данной стадии реализованы 11 различных шаблонов упрощения.

2.2.1.1 Шаблоны, основанные на классах эквивалентности

В самом начале применяются шаблоны, основанные на выделении классов эквивалентности. На данном этапе выделяются классы геометрических объектов, между которыми наложены ограничения параллельности, соосности, концентричности. Выделение классов эквивалентности осуществляется с помощью структуры данных `DisjointSet` (система непересекающихся множеств) [7]. Данная структура поддерживает операции объединения множеств и поиска представителя множества и обеспечивает амортизированную сложность операций, близкую к константной, точнее $O(\alpha(n))$, где $\alpha(n)$ — обратная функция Аккермана. После выделения классов, избыточные ограничения между объектами одного класса могут быть корректно удалены, а затем восстановлены после решения редуцированной системы. Ниже приведён псевдокод шаблона упрощения, основанного на выделении

классов эквивалентности:

```
def simplifyByEquivalenceClasses(CS, type):
    ds = DisjointSet()

    for constraint in CS.constraints.filterByType(type):
        if ds.connected(constraints.geometries):
            CS.delete(constraint);
        else:
            ds.union(constraint.geometries);
    return CS
```

Рис. 2.4: Псевдокод шаблона упрощения, основанного на выделении классов эквивалентности

Простейшим примером системы, которая будет разбита с помощью данной техники, является цикл из ограничений параллельности между прямыми.

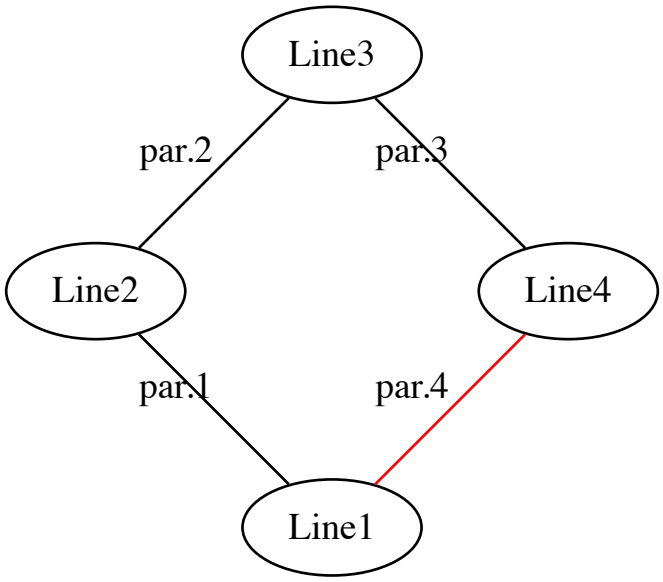

Рис. 2.5: Цикл ограничений параллельности. Красное ребро соответствует избыточному ограничению.

Классы эквивалентности также выделяются для геометрических объектов, между которыми наложены ограничения совпадения. Однако схема

работы с этими классами отличается от остальных, поскольку в данном случае классы используются не для удаления избыточных ограничений, а для слияния геометрических объектов. Техника слияния инцидентных геометрических объектов будет описана ниже.

2.2.1.2 Удаление дубликатов

Следующий шаблон упрощения основан на поиске и удалении дубликатов геометрических ограничений. Дубликатами ограничений будем называть такие ограничения, которые наложены на одни и те же геометрические объекты и имеют одинаковый тип и одинаковые атрибуты. Например, если в системе есть два ограничения параллельности между одними и теми же прямыми, то одно из них будет дубликатом другого. Однако, стоит отметить, что не все ограничения являются симметричными относительно своих аргументов. Поэтому, в некоторых случаях, даже если ограничения наложены на одни и те же геометрические объекты и их атрибуты совпадают, они не будут дубликатами друг друга. Например, если в системе заданы два ограничения расстояния между одними и теми же плоскостями, но с разным порядком аргументов, то эти ограничения не будут дубликатами друг друга, так как расстояние имеет атрибут ориентации, который зависит от порядка аргументов.

Наличие дубликатов в системе ограничений нередко возникает в результате её предобработки, в ходе которой сложные ограничения могут разбиваться на несколько более простых, уже ранее заданных пользователем.

Нахождение подобных дубликатов реализовано с помощью простого

алгоритма, который проходит по всем ограничениям системы и добавляет их в хэш-множество. Если при добавлении ограничения в множество, уже такое ограничение присутствует, то оно является дубликатом и может быть удалено из системы. Хэш каждого ограничения строится на основе его типа, атрибутов и геометрических объектов, на которые оно наложено.

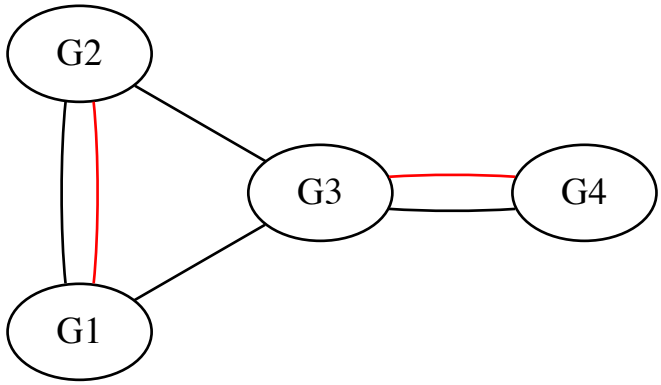

Рис. 2.6: Пример системы ограничений с дубликатами. Красные кратные рёбра соответствуют дубликатам ограничений.

2.2.1.3 Удаление ограничений между фиксированными объектами

Рассматриваемый шаблон основан на поиске геометрических объектов, положение которых полностью зафиксировано либо соответствующими ограничениями, либо их добавлением в жёсткое множество. Поскольку относительное положение таких объектов не может быть изменено, любые ограничения между ними являются избыточными и могут быть удалены из системы. Часто данный шаблон помогает убрать несовместность в системе ограничений.

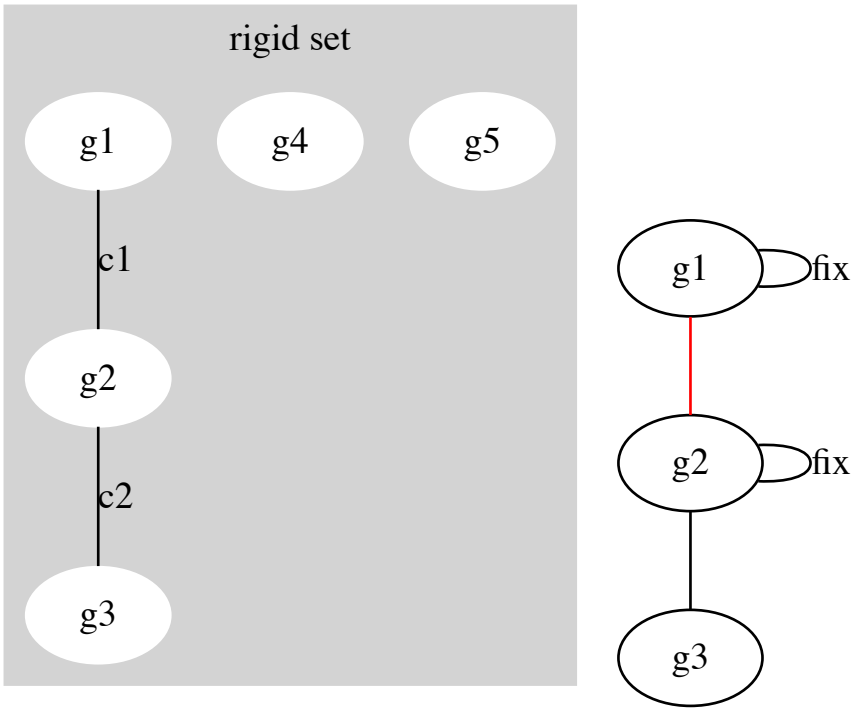

Рис. 2.7: Пример систем с ограничениями между фиксированными объектами. На рисунке (a) ограничения `c1` и `c2`. На рисунке (b) красным цветом выделены ограничения между фиксированными объектами, которые могут быть удалены из системы.

```
def deleteConstraintsBetweenFixed(CS):

    for constraint in CS.constraints.filterByType:
        if allGeometriesFixed(constraint):
            CS.delete(constraint)

    return CS
```

Рис. 2.8: Псевдокод шаблона упрощения, основанного на удалении ограничений между фиксированными объектами.

2.2.1.4 Удаление симметричных ограничений

Следующий шаблон основан на поиске пар ограничений, аргументы которых являются симметричными относительно одной и той же геомет-

рической сущности. Например, два ограничения расстояния считаются симметричными друг другу, если точки одного ограничения симметричны соответствующим точкам другого относительно одной и той же плоскости. В таком случае одно из ограничений может быть удалено из системы.

Для нахождения таких пар применяется следующий алгоритм. Сначала для системы ограничений вычисляется вспомогательная информация: для каждого геометрического объекта сохраняются симметричные ему объекты, оси или плоскости симметрии, а также ограничения, заданные между парами объектов.

После этого алгоритм последовательно рассматривает все ограничения системы. Для каждого ограничения проверяется наличие симметричных объектов у его аргументов и наличие общей оси или плоскости симметрии. Затем среди симметричных объектов выполняется поиск ограничения того же типа. Если такое ограничение существует, текущее ограничение считается избыточным и удаляется из системы.

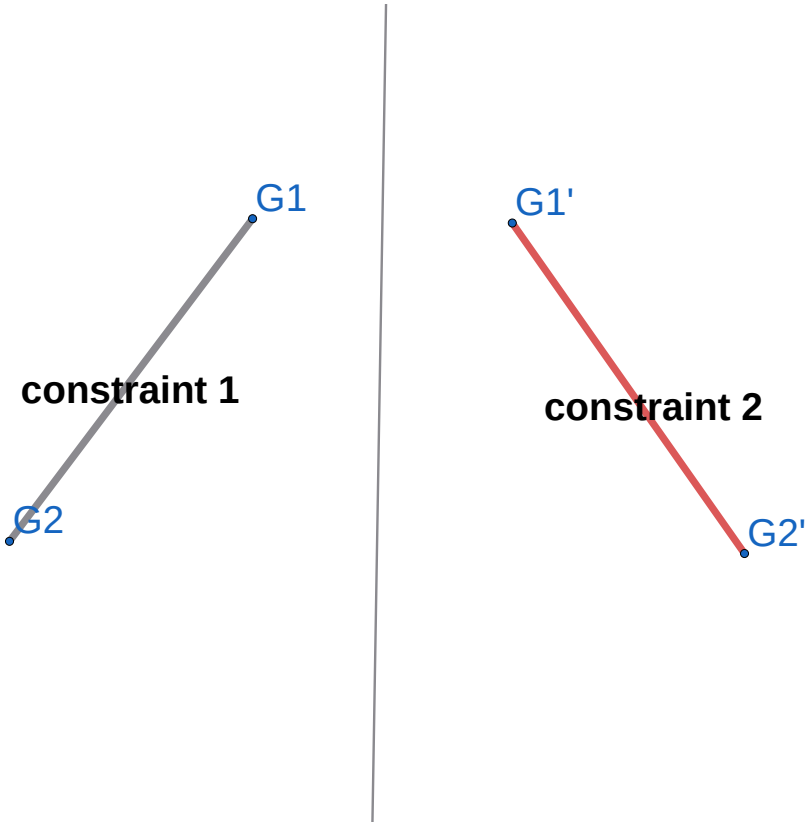

Рис. 2.9: Пример системы с симметричными ограничениями. Красное ребро соответствует симметричному ограничению, которое может быть удалено из системы.

2.2.1.5 Слияние объектов

Шаблон слияния объектов основан на поиске геометрических объектов, между которыми наложены ограничения совпадения, и их последующем слиянии в один объект.

Для начала осуществляется проход по всем ограничениям инцидентности в системе. Среди них выделяются те ограничения, которые наложены между геометрическими объектами одного типа. Данные ограничения используются для построения классов эквивалентности, в которых объекты являются эквивалентными, если между ними наложено ограничение совпадения. После этого осуществляется проход по всем ограничениям системы

(не только по ограничениям инцидентности) и аргумент каждого ограничения заменяется представителем класса эквивалентности, которому данный аргумент принадлежит. Все геометрические объекты, не являющиеся представителем своего класса эквивалентности могут быть удалены из системы.

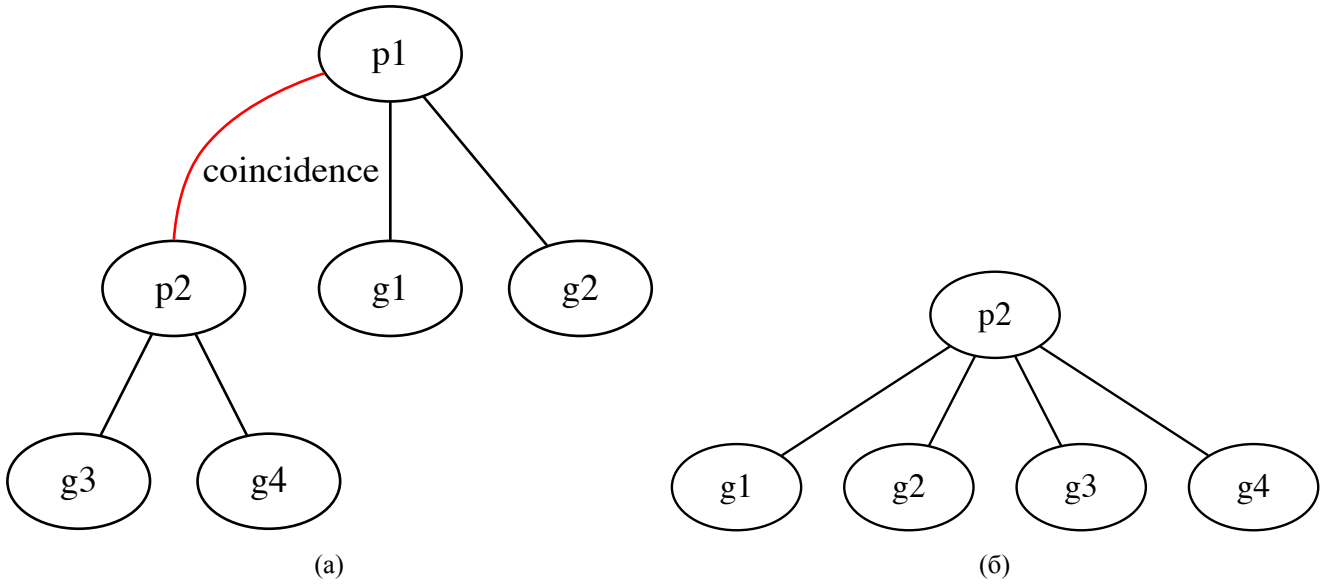

Рис. 2.10: Пример слияния двух точек, между которыми наложено ограничение совпадения. Ограничение совпадения выделено красным цветом.

2.2.1.6 Другие шаблоны

Описанные выше шаблоны являются лишь частью всех шаблонов, которые были реализованы в рамках данной работы. Кроме них были реализованы шаблоны, основанные на свойствах более специфических ограничений, таких как линейные и угловые сдвиги. Также было реализовано удаление геометрических объектов, которые не участвуют ни в одном ограничении, удаление избыточных инцидентностей (например, при совпадении точки двум линиям, которые также совпадают между собой, одна из инцидентностей будет избыточной) и т. д.

2.2.2. Отделяющая декомпозиция

Метод отделяющей декомпозиции реализован в виде класса `CuttingDecomposer`, наследующего абстрактный класс `Decomposer`. Исходная система ограничений представляется в виде графа. На этапе применения отделяющей декомпозиции выполняется последовательный анализ объектов системы. Для каждого объекта производится проверка возможности применения одного из шаблонов отделяющей декомпозиции.

Если для объекта найден подходящий шаблон, то для него формируется локальный решатель. После этого объект вместе со связанными ограничениями временно удаляется из системы, а информация, необходимая для его последующего восстановления, сохраняется. Удаление объекта приводит к изменению структуры графа ограничений, поэтому соседние геометрические объекты повторно проверяются на возможность отделения. После решения редуцированной системы выполняется этап восстановления, в рамках которого удалённые объекты восстанавливаются в порядке, обратном их исключению, с использованием сохранённых локальных решателей. В рамках данной работы реализованы несколько шаблонов отделяющей декомпозиции. Ниже приведены два примера, демонстрирующие различную глубину анализа графа ограничений относительно рассматриваемого объекта: «безусловные» шаблоны (как в примере в 2.2.2.1) учитывают только типы инцидентных ограничений и смежных объектов, тогда как более сложные шаблоны дополнительно анализируют ограничения, инцидентные соседним (смежным) геометрическим объектам, что позволяет учитывать более глубокие структурные зависимости в графе.

2.2.2.1 Отделение по единственному ограничению

Шаблон `CuttingPattern1C` реализует отделение геометрического объекта, связанного только с одним ограничением и не принадлежащего жёсткому множеству геометрических объектов. При применении шаблона анализируются все ограничения, инцидентные рассматриваемому объекту. Если объект имеет единственное ограничение, то для него может быть сформирован локальный численный решатель. Этот шаблон позволяет эффективно исключать из системы геометрические объекты, положение которых может быть восстановлено на основе единственного ограничения и параметров соседних объектов.

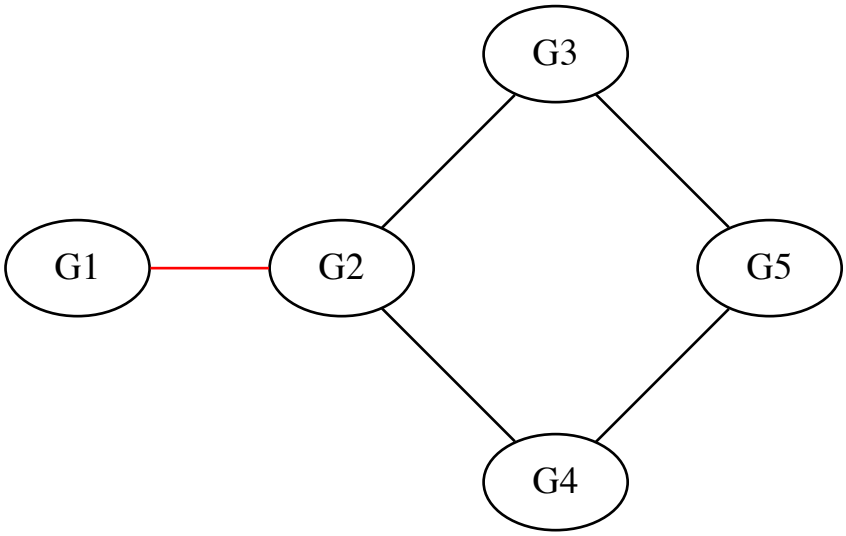

Рис. 2.11: Пример системы, к которой может быть применён шаблон `CuttingPattern1C`. Красное ребро соответствует единственному ограничению, которое связывает `G1` с остальной системой.

2.2.2.2 Прямая, связанная ограничением расстояния с прямой, инцидентной той же плоскости

Шаблон `LineDistanceToLineIncidentToTheSamePlane` применяется к прямым и предназначен для обработки случаев, в которых две прямые принадлежат одной плоскости и связаны ограничением расстояния. При проверке шаблона анализируются ограничения, инцидентные рассматриваемой прямой. Если обнаруживается конфигурация, в которой обе прямые инцидентны плоскости, а между прямыми задано ограничение расстояния, то для рассматриваемой прямой формируется локальный решатель. После применения шаблона прямая временно удаляется из системы вместе со связанными ограничениями, а информация, необходимая для её последующего восстановления, сохраняется. После решения редуцированной подсистемы положение удалённой прямой восстанавливается на основе параметров второй прямой, параметров плоскости и ограничения расстояния между прямыми.

```
def solve(plane, line1, line2, distance):
    vector = cross(plane.normal, line1.direction) * distance
    line2.direction = line1.direction
    line2.origin = line1.origin + vector
```

Рис. 2.12: Псевдокод локального решателя для шаблона `LineDistanceToLineIncidentToTheSamePlane`

2.2.3. Введение жёсткости

Метод введения жёсткости реализован в виде класса `RigidDecomposer`, наследующего абстрактный класс `Decomposer`. Целью данного этапа деком-

позиции является уменьшение размерности системы ограничений за счёт объединения групп геометрических объектов в жёсткие множества (англ. rigid sets), если их взаимное расположение определяется однозначно.

Основная идея метода заключается в поиске пар жёстких множеств, между которыми существует достаточное количество ограничений для полного определения их взаимного расположения. Для этого анализируются ограничения системы, связывающие геометрические объекты из разных жёстких множеств. Каждая такая пара представляется объектом `RigidSetPairInfo`, содержащим список ограничений между двумя жёсткими множествами.

На первом этапе рассматриваются ограничения, связывающие геометрические объекты из различных жёстких множеств. Каждое ограничение анализируется независимо и используется для уточнения допустимого относительного движения между соответствующими жёсткими множествами. Исходно предполагается, что множества обладают полной относительной свободой перемещения. По мере обработки ограничений множество допустимых относительных перемещений последовательно сужается в соответствии с типом ограничения и геометрическими характеристиками связанных объектов. Если в результате анализа относительная степень свободы между двумя жёсткими множествами сводится к нулю, они считаются жёстко связанными и объединяются в одно множество. Конкретные правила сопоставления ограничений изменениям относительных степеней свободы, а также формализация соответствующих шаблонов введения жёсткости не рассматриваются в рамках данной работы и представляют собой направление дальнейших исследований.

После формирования пар жёстких множеств выполняется попытка по-

строения соединения (англ. joint) между ними. Для этого используется класс `JointBuilder`, реализующий как численный, так и аналитический способы построения соединений. В случае успешного построения определяется преобразование, позволяющее совместить одно жёсткое множество с другим.

Для аналитически найденных жёстких соединений дополнительно выполняется проверка корректности. Создаётся временная подсистема ограничений, содержащая только рассматриваемые жёсткие множества и ограничения между ними. Одно из множеств фиксируется ограничением типа `Fixation`, после чего проверяется совместность системы ограничений. Если подсистема совместна, то взаимное положение множеств считается определённым однозначно. Иначе объединение жёстких множеств не выполняется и ограничения между ними остаются в системе.

После успешной проверки жёсткие множества объединяются в новое множество, содержащее геометрические объекты обоих исходных множеств.

Процесс объединения выполняется итеративно. После каждого шага пересчитываются пары жёстких множеств и повторно выполняется поиск новых объединений. Алгоритм завершается, когда на очередной итерации не удаётся выполнить ни одного нового объединения.

2.2.4. Двусвязная декомпозиция

Метод двусвязной декомпозиции реализован в виде класса `BiComponentDecomposer`, наследующего абстрактный класс `Decomposer`. Исходная система ограничений представляется в виде графа системы

ограничений.

Во время применения двусвязной декомпозиции выполняется построение расширенного графа системы, включающего дополнительные виртуальные рёбра, учитывающие зависимости геометрических объектов и принадлежность к жёстким множествам (англ. rigid sets).

Перед поиском двусвязных компонент выполняется обход графа в глубину (англ. DFS), который автоматически покрывает все компоненты связности графа. Для каждой создаётся независимая подсистема ограничений, которая может быть решена отдельно от остальных. При этом для каждой компоненты графа выполняется поиск точек сочленения с использованием алгоритма Тарьяна [9]. Данный алгоритм основан на обходе в глубину (DFS) и работает за линейное время $O(V + E)$, где V — число вершин, а E — число рёбер. После выявления точек сочленения выполняется разбиение графа на двусвязные компоненты.

Разбиение на двусвязные компоненты позволяет выделить подграфы, устойчивые к удалению точек сочленения. Каждая такая компонента рассматривается как отдельная подсистема ограничений. При этом вершины, являющиеся точками сочленения, могут входить в несколько компонент, что требует специальной обработки. Если геометрический объект принадлежит нескольким компонентам, он клонируется для обеспечения независимости подсистем. Все ограничения, затрагивающие такие объекты, также клонируются с заменой исходных геометрических объектов на соответствующие копии. Для фиксированных геометрических объектов и жёстких множеств применяется специальная логика переноса без нарушения глобальных связей.

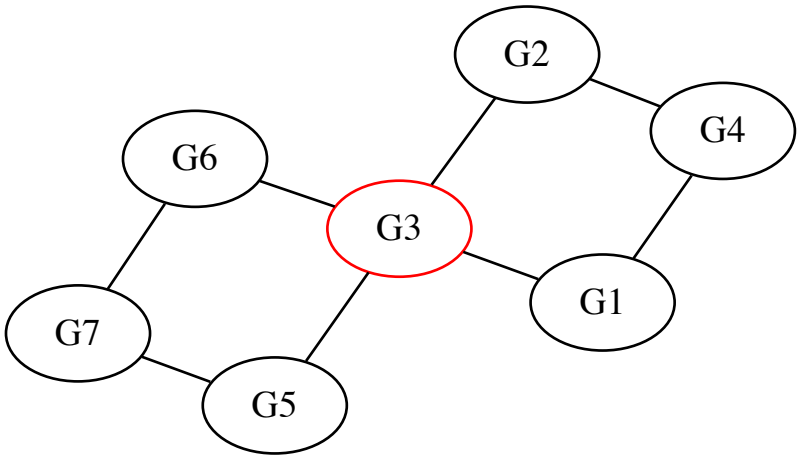

Рис. 2.13: Пример системы ограничений, которая будет разбита на две двусвязные компоненты. Красной вершине соответствует точка сочленения графа системы.

После формирования подсистем каждая компонента преобразуется в отдельную систему ограничений, которая может быть решена независимо. Соответствие между оригинальными и клонированными объектами сохраняется для последующего восстановления результатов и синхронизации решений.

Далее происходит синхронизация решений подсистем для получения решения исходной системы. На первом этапе для каждой подсистемы вычисляется преобразование, связывающее текущее положение клонированного геометрического объекта с его исходным положением, после чего данное преобразование применяется к соответствующей подсистеме. После этого выполняется перенос вспомогательных параметров и вспомогательных точек логических ограничений с клонированных объектов на оригинальные, что обеспечивает согласованность данных между копиями.

2.3. Моделирование

Для генерации системы нелинейных уравнений необходимо выбрать способ моделирования геометрических объектов и ограничений, т. е., определить, какие переменные будут использоваться для представления геометрических объектов и каким образом ограничения будут представлены в виде уравнений.

2.3.1. Моделирование геометрических объектов

В рамках данной работы был выбран метод декартового моделирования, который является одним из наиболее распространённых методов алгебраического моделирования в задачах удовлетворения геометрических ограничений [1]. Для моделирования координат точек используются их декартовы координаты (x, y, z), а для моделирования направления используются азимутальный и полярный углы (θ, ϕ).

Для каждого из поддерживаемых геометрических объектов был выбран следующий способ моделирования:

- Точка — декартовы координаты (x, y, z).
- Прямая — координаты начальной точки (x, y, z) и направление, представленное азимутальным и полярным углами (θ, ϕ).
- Плоскость — нормаль, представленная азимутальным и полярным углами (θ, ϕ), и расстояние от начала координат d.
- Сфера — координаты центра (x, y, z) и радиус r.
- Цилиндр — координаты центра (x, y, z), направление оси, представ-

ленное азимутальным и полярным углами (θ, ϕ), и радиус r.

- Конус — координаты вершины (x, y, z), направление оси, представленное азимутальным и полярным углами (θ, ϕ), угол раскрытия α и радиус сечения r в точке (x, y, z).
- Тор — координаты центра (x, y, z), направление оси, представленное азимутальным и полярным углами (θ, ϕ), большой радиус R и малый радиус r.
- Окружность — координаты центра (x, y, z), направление нормали плоскости, в которой лежит окружность, представленное азимутальным и полярным углами (θ, ϕ), и радиус r.
- Эллипс — координаты центра (x, y, z), направление нормали плоскости, в которой лежит эллипс, представленное азимутальным и полярным углами (θ, ϕ), большая полуось a, малая полуось b, а также направление большой полуоси, представленное азимутальным и полярным углами (θ_a, ϕ_a).
- Жёсткое множество — координаты начала отсчёта (x, y, z), а также два перпендикулярных направления, представленные азимутальным и полярным углами (θ_1, ϕ_1) и (θ_2, ϕ_2), определяющие положение множества в пространстве.

2.3.2. Моделирование ограничений

Для моделирования ограничений используется класс `ConstraintDispatcher`, который реализует шаблон посетитель. Реализация трансляции конкретного типа ограничений с определенными типами аргументов осуществляется в

соответствующем методе visit. Например, для ограничения расстояния реализован класс `DistanceTranslator`, наследующий `ConstraintDispatcher`. Для моделирования ограничения расстояния между точкой и плоскостью, реализован метод visit(Point, Plane) в классе `DistanceTranslator`, который добавляет уравнение в систему, выражающее расстояние между точкой и плоскостью в терминах параметров точки и плоскости. Однако, не все методы visit для всех типов ограничений и аргументов являются допустимыми. Например, для ограничения параллельности не имеет смысла реализовывать метод visit(Point, Plane), так как ограничение параллельности не может быть наложено между точкой и плоскостью.

В рамках данной работы были реализованы трансляторы ограничений в уравнения для всех типов ограничений из таблицы 1.1 и всех возможных типов аргументов для этих ограничений.

3. Результаты

В работе были реализованы модули геометрической декомпозиции и алгебраического моделирования, а также базовые классы для работы с геометрическими объектами и ограничениями. Модуль алгебраического моделирования поддерживает генерацию системы нелинейных уравнений для всех типов ограничений и геометрических объектов из таблицы 1.1. В рамках модуля геометрической декомпозиции были реализованы 11 шаблонов упрощения, 3 шаблона отделяющей декомпозиции, а также методы введения жёсткости и двусвязной декомпозиции. Данные модули были интегрированы в существующий решатель геометрических ограничений.

Предложенные модули были реализованы на языке C++, с поддержкой кроссплатформенной компиляции (MSVC для Windows и GCC для Linux). Система тестирования была реализована на языке Python.

Исследование влияния методов декомпозиции проводилось на тестовом наборе из 4014 промышленных тестовых задач. Применение всех модулей в совокупности уменьшило среднее время решения задач в 1.813 раза. На отдельных тестах время решения уменьшилось в 917 раз. Среднее ускорение среди двадцати тестов с наибольшим ускорением составило 746 раз.

Методы декомпозиции подключались последовательно: каждая следующая конфигурация включала соответствующий метод и все предыдущие этапы. Например, конфигурация «отделяющая декомпозиция» включает как саму отделяющую декомпозицию, так и предварительное упрощение системы ограничений.

В таблице 3.1 приведены результаты влияния каждого метода на производительность решателя геометрических ограничений.

Конфигурация	**Ср. время, с**	**Общее время, с**	**Ср. ускорение, раз**
Без декомпозиций	1.166	4682.73	1.000
Упрощение	1.155	4637.53	1.009
Отделяющая деком.	1.161	4661.97	1.004
Введение жёсткости	1.055	4234.80	1.105
Все стадии	0.643	2582.16	1.813

Таблица 3.1: Производительность решателя при последовательном применении различных стадий декомпозиции.

При этом наблюдаемая динамика времени решения не является монотонной: в ряде случаев добавление метода не даёт ожидаемого ускорения, а может приводить к незначительному ухудшению (например, при переходе от одного лишь упрощения к комбинации упрощения и отделяющей деком-

позиции). Это объясняется особенностями тестового набора. Эксперименты проводились преимущественно на задачах 3D-сборки, в которых геометрии организованы в жёсткие множества, каждое из которых содержит несколько объектов, связанных ограничениями с объектами других множеств. В таких условиях упрощение системы ограничений и отделяющая декомпозиция дают ограниченный эффект, в отличие от 2D-задач, где их вклад существенно выше. Напротив, методы, связанные с выявлением структуры графа — в частности, декомпозиция на компоненты двусвязности и введение жёсткости — оказываются здесь значительно более эффективными. Наиболее выраженный вклад действительно даёт двусвязная декомпозиция: после её применения наблюдается существенное снижение времени решения. Это связано с тем, что метод введение жёсткости фактически сжимает граф зависимостей, склеивая некоторые вершины и преобразуя часть циклических структур в точки сочленения, что резко упрощает последующую декомпозицию. В результате именно эти методы оказываются ключевыми для данного класса задач и во многом определяют общий выигрыш по времени.

Для оценки устойчивости эффекта методов декомпозиции на производительность решателя, было составлено распределение тестовых задач по изменению времени решения при применении всех стадий декомпозиции. Результаты представлены в таблице 3.2.

Статистика	**Значение**
Ускорено	16.19%
Без существенных изменений	70.05%
Замедлено	13.75%

Таблица 3.2: Распределение тестовых задач по изменению времени решения.

Кроме того, было составлено распределение тестовых задач по величине ускорения при применении всех стадий декомпозиции. Результаты представлены в таблице 3.3

Диапазон ускорения	**Количество тестов**
Более 10x	123
От 5x до 10x	81
От 2x до 5x	64
От 1x до 2x	1731
От 0.95x до 1x	1463
От 0.8x до 0.95x	502
От 0.2x до 0.8x	46
От 0.1x до 0.2x	2
Менее 0.1x	2

Таблица 3.3: Распределение тестовых задач по величине ускорения.

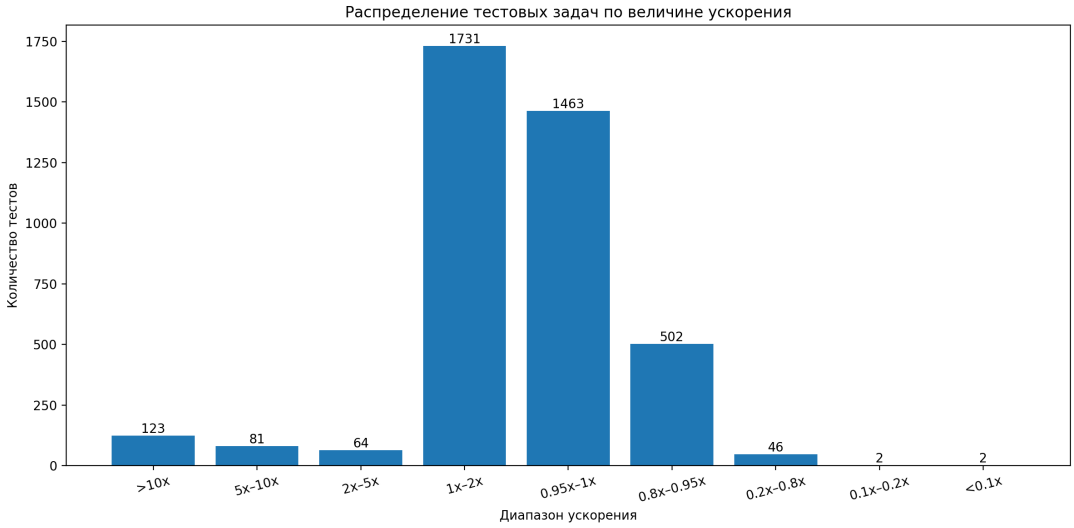

Рис. 3.1: Гистограмма построенная по таблице 3.3.

Из таблицы 3.3 и графика 3.1 можно видеть, что решение ускорилось для 1999 тестовых задач. Для 1965 задач увеличение времени решения оказалось несущественным. Лишь для 1% задач наблюдается увеличение времени решения более чем на 20%.

4. Заключение

В рамках данной работы были реализованы методы геометрической декомпозиции и алгебраического моделирования для задач удовлетворения геометрических ограничений. Реализованные методы были интегрированы в существующий решатель геометрических ограничений и протестированы на обширной базе индустриальных тестов. Полученные результаты показали значительное улучшение производительности решателя при применении предложенных методов декомпозиции.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ершов А. Г. Алгоритмы и программные системы для геометрических задач параметрического проектирования : автореф. дис. ... канд. физ.-мат. наук. — Новосибирск, 2007. — 19 с.
2. *Поляк Б. Т.* Метод Ньютона и его роль в оптимизации и вычислительной математике // Труды ИСА РАН. — 2006. — С. 48—63.
3. *Ушаков Д. М.* Технологии вариационного проектирования для разработки типичных приложений САПР. — 2021. — URL: http://isicad.ru/ru/articles.php?article_num=11571 (дата обр. 15.02.2026).
4. *Bettig B. P.*, *Hoffmann C. M.* Geometric Constraint Solving in Parametric CAD. — 2011. — URL: https://api.semanticscholar.org/CorpusID:14534815 (visited on 02/17/2026).
5. Decomposition of geometric constraint systems: A survey / C. Jermann [et al.] // International Journal of Computational Geometry and Applications. — 2006. — Vol. 16, no. 5/6. — P. 379–414. — DOI: 10.1142/S0218195906002105.
6. *Hoffmann C. M.*, *Joan-Arinyo R.* A brief on constraint solving // Computer-Aided Design and Applications. — 2005. — Vol. 2, no. 5. — P. 655–663.
7. Introduction to Algorithms / T. H. Cormen [et al.]. — 3rd ed. — MIT Press, 2009.

8. *Mathis P.*, *Thierry S. E. B.* A formalization of geometric constraint systems and their decomposition // Formal Aspects of Computing. — 2010. — Vol. 22, no. 2. — P. 129–151.

9. *Tarjan R.* Depth-First Search and Linear Graph Algorithms // SIAM Journal on Computing. — 1972. — Vol. 1, no. 2. — P. 146–160. — DOI: 10.1137/0201010.

10. *West D. B.* Introduction to Graph Theory. — 2nd ed. — Prentice Hall, 2001.