****

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»**

Институт (факультет) информатики

Кафедра программных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(магистрант)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Эвристический алгоритм поиска глобально-оптимальной

конформации атомного кластера Морса

Выпускник Шашов Кирилл Владимирович

Руководитель (Коварцев А.Н.)

Рецензент

Самара 2017

  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»

Кафедра программных систем

УТВЕРЖДАЮ

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Коварцев А.Н./

\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

**ЗАДАНИЕ  
на выпускную квалификационную работу магистра  
(магистерскую диссертацию)**

Студенту *Шашову Кириллу Владимировичу*  группы *6224 М 405*

Тема работы: *Эвристический алгоритм поиска глобально-оптимальной конформации атомного кластера Морса*

утверждена приказом по университету от 04 апреля 2017 г. № 387

Перечень вопросов, подлежащих разработке:

1. Провести анализ предметной области, сформулировать основные цели исследования
2. Проанализировать особенности математических моделей атомных кластеров
3. Разработать эвристический алгоритм поиска оптимальных конформаций атомного кластера Морса на основе использования икосаэдрических сеток
4. Разработать и отладить программную реализацию алгоритма
5. Провести вычислительный эксперимент поиска оптимальных конформаций

Срок представления на кафедру законченной работы: 20 мая 2017 г.

Дата выдачи задания: 20 февраля 2017 г.

*Руководитель работы Коварцев А. Н., зав.каф.ПС*

(фамилия, и., о., должность, подпись)

*Задание принял к исполнению*

(дата)

(подпись студента)

**ПРИЛОЖЕНИЕ**

**к заданию на выпускную квалификационную работу магистра  
(магистерскую диссертацию)**

студенту группы №6224 М 405 К.В. Шашову

Тема ВКРМ: **«Эвристический алгоритм поиска глобально-оптимальной конформации атомного кластера Морса»**

**Исходные данные к работе:**

1. **Характеристики объекта исследования:**
2. объект исследования – атомный кластер Морса;
3. виды автоматизируемой деятельности:

* процесс изменения параметров для расчетов;
* процесс построения 3D-графиков конформаций;

1. минимальное количество атомов в кластере – 1;
2. минимальное количество вводимых величин – 1;
3. максимальное количество вводимых величин – 17;
4. система единиц измерения физических величин – СИ.
5. **Требования к информационному обеспечению:**
6. информационное обеспечение разрабатывается на основании следующего документа:

The Cambridge Energy Landscape Database [Электронный ресурс] // Wales group home page: [сайт]. URL:   
http://www-wales.ch.cam.ac.uk/CCD.html (дата обращения: 01.05.2017);

и следующих сведений:

* сведений о постоянных величинах;
* сведений об изменяемых величинах;

1. входные параметры хранятся в текстовом документе формата \*.json;
2. результаты вычислений хранятся в текстовом документе формата \*.txt.
3. **Требования к техническому обеспечению:**

* тип ЭВМ – IBM PC совместимый;
* монитор с разрешающей способностью не ниже 1280 х 786;
* тактовая частота процессора – не менее 1 ГГц;
* объем оперативной памяти – не менее 256 МБ;
* объём свободного дискового пространства – не менее 2 ГБ;
* клавиатура;
* манипулятор – мышь.

1. **Требования к программному обеспечению:**

* операционная система – любая при наличии JDK версии 8.0 и выше;
* среда программирования – Intellij IDEA 14;
* язык программирования – Java.

1. **Общие требования к проектируемой системе.**

**5.1 Функции, реализуемые системой:**

* считывание входных параметров;
* расчет оптимальной конформации;
* визуализация результатов работы;
* сохранение результатов работы в файл;
* выдача справочной информации о системе.

**5.2 Технические требования к системе:**

1. режим работы – диалоговый;
2. температура окружающего воздуха – 15-25°С;
3. влажность окружающего воздуха – 45-75%;
4. система должна удовлетворять санитарным правилам и нормам СанПин 2.2.2/2.4.2198-07;
5. условия работы средств вычислительной техники должны соответствовать ГОСТ 12.1.005, 12.1.007.

Руководитель ВКРМ,   
д.т.н., профессор А.Н. Коварцев

# Реферат

Пояснительная записка 79 с., 31 рисунок, 44 источника, 1 приложение.

Графическая документация: 20 слайдов презентации Power Point.

АТОМНЫЙ КЛАСТЕР, ГЛОБАЛЬНАЯ ОПТИМИЗАЦИЯ, ЛОКАЛЬНАЯ ОПТИМИЗАЦИЯ, ПОТЕНЦИАЛЬНАЯ ЭНЕРГИЯ, ПОТЕНЦИАЛ МОРСА, КОНФОРМАЦИЯ, ИКОСАЭДРИЧЕСКАЯ СЕТКА, JAVA.

В работе приводится исследование задачи оптимизации атомных кластеров Морса. Разработана методика глобальной оптимизации атомных структур больших размерностей, основанная на переходе к оптимизации функции одной переменной. Написана программная реализация предложенного алгоритма на языке программирования Java, с помощью которой были проведены вычислительные эксперименты.

Результаты расчетов и экспериментальных исследований доказали пригодность разработанного алгоритма для решения поставленной задачи оптимизации.

# Содержание

[Введение 7](#_Toc482961936)

[1 Глобальная оптимизация атомных кластеров 10](#_Toc482961937)

[1.1 Задача глобальной оптимизации атомных кластеров 11](#_Toc482961938)

[1.2 Обзор существующих методов оптимизации 17](#_Toc482961939)

[1.2.1 Эволюционные методы 19](#_Toc482961940)

[1.2.2 Семейство методов BH 21](#_Toc482961941)

[1.2.3 Методы с конструированием ядра 28](#_Toc482961942)

[2 Методика решения поставленной задачи 32](#_Toc482961943)

[2.1 Формирование плотной упаковки атомов 33](#_Toc482961944)

[2.2 Оптимизация функции одной переменной 35](#_Toc482961945)

[3 Используемые методы и алгоритмы 38](#_Toc482961946)

[3.1 Метод Стронгина 38](#_Toc482961947)

[3.2 Поиск двоичных чисел с заданным количеством единиц 40](#_Toc482961948)

[3.3 Алгоритм развития 42](#_Toc482961949)

[3.4 Локальная оптимизация 44](#_Toc482961950)

[4 Вычислительные эксперименты 46](#_Toc482961951)

[4.1 Программная реализация 46](#_Toc482961952)

[4.1.1 Последовательность действий работы с приложением 46](#_Toc482961953)

[4.2 Результаты поиска оптимальных конформаций 51](#_Toc482961954)

[4.2.1 Экспериментальное исследование параметров алгоритма 57](#_Toc482961955)

[Заключение 58](#_Toc482961956)

[Список использованных источников 60](#_Toc482961957)

[Приложение А Листинг исходного кода Java 65](#_Toc482961958)

# Введение

Множество усилий современных исследований направлено на такие задачи как предсказание структуры биологических молекул, сворачивание и стыковку белков. Они в известной мере подтолкнули и исследования в направлении предсказания структуры атомных кластеров в качестве испытательной площадки для методов предсказания структуры биомолекул.

Задача предсказания по линейной аминокислотной последовательности белка пространственной структуры представляет фундаментальную важность для целого ряда дисциплин. Совместные усилия ученых, направленные на решение этих задач, имеют своей целью открыть новые возможности интерпретации данных, моделирования новых биологически активных соединений, в том числе для целей создания лекарственных препаратов. Известно, что в последние годы геномные исследования заметно увеличили число установленных аминокислотных последовательностей белков, опережая тем самым возможности экспериментальных методов определения пространственных структур белковых молекул. Поэтому возникший дефицит потенциально возможных пространственных структур белков стимулирует современные методы компьютерного моделирования.

Изучение механизмов структурирования наночастиц связано с тем, что образование функциональных структур из нанокристаллических кластеров позволяет придавать материалам новые, очень неожиданные свойства. Знание деталей формирования кластерной структуры является определяющим для повышения эффективности различных способов производства нанокластеров с фиксированными физическими свойствами, что открывает широкие перспективы для их практического применения при создании новых веществ с заданными механическими, электрическими, магнитными и оптическими свойствами. Под кластерами в химии и материаловедении чаще всего понимают одно из промежуточных состояний в организации вещества между одиночным атомом (молекулой, ионом) и твердым телом (наночастицей). То есть кластер представляет собой группу взаимодействующих атомов, ионов или молекул.

Поскольку сложно идентифицировать структуру свободных кластеров в молекулярных пучках, обычно оценивают параметры, зависящие от структуры, и используют модели с прогнозируемыми предпочтительными геометрическими конфигурациями. Одной из основных задач при формировании кластерной структуры является минимизация энергии взаимодействия входящих в нее атомов.

Даже в случае упрощенного допущения об энергии взаимодействия, задача нахождения минимума суммарной потенциальной энергии является крайне сложной, т.к. она представляет собой случай невыпуклой оптимизации с большим числом локальных минимумов. Было доказано, что сложность задачи определения минимума потенциальной энергии кластера, в котором между атомами учитывается парное взаимодействие, принадлежит к классу NP-полных задач, и до сих пор не известен не экспоненциальный алгоритм ее решения. На практике наивысших результатов добились генетические и эвристические алгоритмы, учитывающие известные закономерности формирования кластерных структур [1].

Целью настоящей работы является разработка эвристического алгоритма поиска атомных структур, обладающих минимальными значениями потенциала Морса.

Для достижения указанной цели были поставлены следующие задачи:

1. Рассмотреть особенности постановки задачи глобальной оптимизации при поиске атомных структур.
2. Проанализировать существующие методы поиска оптимальных атомных структур.
3. Исследовать особенности формирования конформаций кластеров Морса, выявить закономерности для их дальнейшего использования в алгоритме при построении начальных конформаций.
4. Разработать эвристический алгоритм глобальной оптимизации атомных кластеров Морса.
5. Разработать программную реализацию алгоритма, провести тестирование и отладку.
6. Провести вычислительные эксперименты.

Структура работы обусловлена предметом, целью и задачами исследования. Работа состоит из введения, четырех глав и заключения.

Введение раскрывает актуальность, определяет степень научной разработки темы, цель исследования, раскрывает теоретическую и практическую значимость работы.

В первой главе работы рассмотрена задача глобальной оптимизации, в частности, особенности ее постановки и решения при работе с атомными структурами. Приведен обзор основных существующих методов поиска конформаций атомных кластеров. Рассмотрены генетические методы, методы семейства BH (basin hopping), методы поиска на динамической сетке с конструированием ядра.

Во второй главе описана методика решения поставленной задачи, которая лежит в основе разработанного алгоритма оптимизации.

В третьей главе описаны теоретические и вычислительные методы, адаптации известных алгоритмов оптимизации и прочие эвристики, использованные в работе для определения оптимальных структур кластеров.

В четвертой главе приведены экспериментальное исследование параметров разработанного алгоритма, результаты вычислительных экспериментов по поиску оптимальных конформаций кластеров Морса, а также описание программной реализации алгоритма, с помощью которой проводились данные эксперименты.

В заключении подводятся итоги исследования, формируются окончательные выводы по рассматриваемой теме.

# Глобальная оптимизация атомных кластеров

Задачи глобальной оптимизации доказали свою актуальность во многих прикладных областях математики, информатики, физики, биологии, химии и смежных с ними отраслями знаний. Широко обсуждаются разновидности важной для нефтехимической отрасли задачи о транспортных сетях с пулами, различные формулировки задачи об оптимальном распределении ресурсов, планировании ресурсов, составлении расписаний, задачи об оптимальном расположении фигур на плоскости, задачи, связанные с расчетом электронной плотности внутри элементарной ячейки белкового кристалла, геометрические задачи восстановления трехмерных структур сцены по нескольким изображениям, многие задачи комбинаторной оптимизации, например, задачи о максимальной клике или максимальном разрезе в графе. Кроме того, задачи глобальной оптимизации так же рассматривают в пределах интересов конформационного анализа, занимающегося поиском оптимальных конформаций (структур) молекул, исследованием их химических и физических свойств и т.д.

Задача предсказания по линейной аминокислотной последовательности белка пространственной структуры белка представляет фундаментальную важность для целого ряда дисциплин. Совместные усилия ученых, направленные на решение этих задач, имеют своей целью открыть новые возможности интерпретации данных, полученных на моделирования новых биологически активных соединений, в том числе для целей создания лекарственных препаратов.

В последние годы геномные исследования заметно увеличили число установленных аминокислотных последовательностей белков, опережая тем самым возможности экспериментальных методов определения пространственных структур белковых молекул. Поэтому возникший дефицит потенциально возможных пространственных структур белков стимулирует современные методы компьютерного моделирования.

Известен парадокс Левинталя [2], согласно которому, если бы молекула небольшого белка (около 100 аминокислотных остатков) для достижения нативной формы перебирала все возможные конформации, этот процесс потребовал бы времени, превышающего время существования Вселенной. В то же время из эмпирических данных следует, что обычное время сворачивания белка составляет миллисекунды. Это объясняется тем, что молекуле белка не за чем принимать все возможные конформации – она находит кратчайший путь на гиперплоскости потенциальной энергии к точке, соответствующей нативной конформации белка.

Современные исследования в области ГО, занимающиеся анализом конформаций белка [3], [4], используют модели эмпирических силовых полей, которые были разработаны с привлечением точных квантовых расчетов. Задача докинга или стыковки молекул представляет еще большую сложность: предметом анализа в этом случае выступают две и более взаимодействующих молекул, которые в итоге рассматривают как единый комплекс. При этом во время стыковки пространственная структура молекул претерпевает локальные изменения. Описание задачи докинга в терминах методов глобальной оптимизации также можно найти во многих исследованиях [5], [6]. При наличии соответствующих моделей силовых полей задачу докинга белков можно решать с помощью подходов, аналогичным тем, что используются в задачах сворачивания белков – общим будет наличие большого числа переменных.

Более простым и частным случаем конформационного анализа является исследование моделей *атомных кластеров*, длительное время, служившее источником проверочных испытаний для новых методов глобальной оптимизации.

## Задача глобальной оптимизации атомных кластеров

Глобальная оптимизация как область математического программирования занимается разработкой методов, решающих задачи поиска точек и значения абсолютного (глобального) минимума многоэкстремальной многомерной функции при наличии ограничений, которые в свою очередь могут быть многоэкстремальными функциями.

В общем виде задача глобальной оптимизации может быть сформулирована следующим образом [7]:

, (1.1)

где – размерность задачи, – целевая функция, – ограничения задачи, – допустимое множество, – глобальное решение.

Область определяется как:

, (1.2)

, (1.3)

где – область поиска.

Среди основных задач детерминированной глобальной оптимизации можно выделить:

1. определение глобального минимума целевой функции, удовлетворяющий множеству ограничений;

2. определение верхних и нижних границ глобального минимума целевой функции, справедливых для всей рассматриваемой области;

3. определение множества локальных решений в окрестностях глобального оптимума;

4. доказательство того, что задача ограниченной нелинейной оптимизации имеет решение или не имеет решения.

Разнообразие задач глобальной оптимизации влечет за собой разнообразие подходов к их решению. С одной стороны, узкоспециализированные методы могут оказаться неприемлемыми для решения задач из более широкого класса; с другой стороны, метод, разработанный для решения задач слишком общего класса, может быть малоэффективным при решении конкретной прикладной задачи.

Задача поиска оптимальных конформаций атомных кластеров может быть сформулирована как следующая задача глобальной оптимизации:

, (1.4)

где – число атомов кластера, – координаты центра - ого атома, – евклидова норма, – парный потенциал взаимодействия.

Наиболее распространенные в исследованиях потенциальные функции – потенциал Леннарда-Джонса:

(1.5)

и потенциал Морса:

, (1.6)

где – параметр, характеризующий физические свойства кластера.

Сложность задачи определения минимума потенциальной энергии кластера, в котором между атомами учитывается парное взаимодействие, принадлежит к классу NP-полных задач.

Обе приближенные модели потенциала часто используются для описания кластеров с определенным типом атомов, например, некоторых газов. На рисунке 1 изображены графики потенциалов кластеров Морса и Леннарда-Джонса.

Функции имеют единственную точку минимума, резко растут при стремящемуся к нулю расстоянию между атомами, и асимптотически стремятся к нулю при бесконечно большом расстоянии между парами атомов. Таким образом, функции зависят от расстояния между взаимодействующими атомомами и не являются выпуклыми.

Функция, соответствующая потенциальной энергии кластера и полученная суммированием потенциалов парных взаимодействий, также представляет собой невыпуклую мультимодальную функцию пространственных координат атомов.

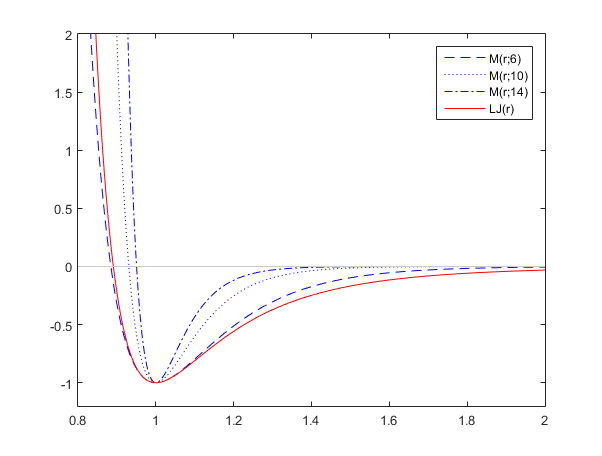


Рисунок 1 – Потенциалы кластеров Морса и Леннарда-Джонса ()

Задача оптимизации кластеров Леннарда-Джонса долгое время считалась классическим тестовым испытанием алгоритмов глобальной оптимизации, специализирующихся на поиске оптимальных молекулярных структур – не важно атомных кластеров или биомолекул.

Функция потенциала Леннарда-Джонса достаточно проста – гипотетические глобальные минимумы были успешно зарегистрированы для кластеров размерами более 300 атомов. Поверхность потенциальной энергии обладает ярко выраженными *вороночными* свойствами, таким образом, что поиск направляется в сторону глобального минимума. Например, для кластера Леннарда-Джонса из 55 атомов с более чем известными локальными минимумами, алгоритм BH, представленный в работах D.J.Wales и J.P.K.Doye, сумел отыскать глобальный минимум из случайной стартовой структуры после перебора всего 150 минимумов [8]. Кластеры Леннарда-Джонса выступили в качестве модели для исследования широкого набора свойств кластеров: структурных, термодинамических, динамических, энергетических.

Тем не менее, задача оптимизации кластеров Леннарда-Джонса существенные недостатки и потому не может считаться идеальной проверочной системой. Так, кластерам Леннарда-Джонса свойственно структурное однообразие – практически все глобальные минимумы в соответствующем диапазоне числа атомов основаны на икосаэдре Маккея – структуры из 20 граней, 12 вершин, 30 ребер.

Самыми сложными для оптимизации оказались кластеры Леннарда-Джонса лишь нескольких размерностей, которым соответствуют оптимальные конформации, основанные не на икосаэдре Маккея. В этих случаях поверхность потенциальной энергии имеет две ярко выраженные воронки – одну доминирующую, более широкую, ведущую к икосаэдрическим структурам, и одну значительно более узкую, ведущую к глобальному минимуму. Как правило, примеры таких задач оказываются на несколько порядков сложнее.

По причине указанных недостатков задача оптимизации кластеров Морса была предложена в качестве альтернативной и более удачной проверочной системы [9].

Потенциал Морса имеет единственный параметр , который определяет ширину потенциальной ямы и позволяет рассматривать взаимодействия самых различных материалов. Например, график потенциала Леннарда-Джонса имеет такую же кривизну, как и график потенциала Морса при , поэтому можно допустить, что при таком значении параметра системы ведут себя идентично.

Кластеры Морса демонстрируют широкое структурное разнообразие (рисунок 2), зависящее от параметра .

\begin{figure}
\epsfig {figure=figures/2.phased.eps,width=14.0cm}

\vspace{3mm}\end{figure}

Рисунок 2 – Зависимость структуры глобальных оптимумов кластеров Морса от размера и параметра .

Было доказано, что по мере увеличения значения структура глобальных минимумов меняется от икосаэдрической к декаэдрической и гранецентрированной кубической [10].

Параметр также влияет на характер поверхности потенциальной энергии. По мере увеличения значения число локальных минимумов резко возрастает, меняя ландшафт потенциала от более гладкого к более шероховатому.

Важно, что при больших значениях параметра поверхность потенциальной энергии имеет несколько воронок, усложняя задачу оптимизации. Это приводит к большому числа конформаций с конкурентно малым значением энергии (в основном с декаэдрической и плотно упакованной структурой) и сильно различающихся геометрически. Именно эта особенность сближает задачу оптимизации кластеров Морса с задачей оптимизации биомолекул, потому как в случае предсказания структуры белка одну из основных сложностей и при проведении экспериментов, и при компьютерном моделировании процесса фолдинга, представляют собой воронки-ловушки, которые препятствуют сворачиванию [11], [12]. Наличие таких ловушек связывают с дополнительными воронками на поверхности потенциальной энергии [13], [14].

Сочетание вышеперечисленных особенностей делает задачу глобальной оптимизации кластеров Морса на несколько порядков сложнее оптимизации кластеров Леннарда-Джонса. Предполагаемые глобальные минимумы зарегистрированы в Кэмбриджской базе данных (The Cambridge Energy Landscape Database) [15]. Опубликованные там результаты получены благодаря не только применению адаптированных алгоритмов глобальной оптимизации, но и генерации исходных конформаций-кандидатов на основании имеющихся представлений о структурных свойствах кластеров.

По вышеуказанным причинам задача глобальной оптимизации кластеров Морса более предпочтительна в роли основной проверочной системы для алгоритмов глобальной оптимизации. Однако до сих пор ни один алгоритм не сумел обнаружить все глобальные минимумы потенциала Морса без привлечения дополнительных знаний о предметной области – о структурных, термодинамических закономерностях и т.д.

## Обзор существующих методов оптимизации

Как было отмечено ранее, проблема поиска оптимальных конформаций атомных кластеров сводится к задаче глобальной оптимизации потенциальной функции. Однако, несмотря на существование большого числа прямых методов глобальной оптимизации, позволяющих найти точное решение, задача все еще остается актуальной. Связано это с высоким ростом сложности при увеличении размера задачи. На текущий момент прямые методы способны находить решение лишь для функций с десятками переменных, однако в задачах оптимизации атомных кластеров участвует более 900 оптимизируемых переменных. Таким образом, основной трудностью при поиске глобального минимума является сильное усложнение поверхности потенциальной энергии при увеличении размеров кластера. Поэтому наиболее эффективными становятся алгоритмы, использующие разнообразную дополнительную информацию об объекте исследования [16].

Наиболее популярные методы оптимизации можно разделить на три группы. Первая группа включает эволюционные и генетические алгоритмы, методы *быстрого отжига* (*fast annealing evolutionary algorithm*, FAEA), метод *роя частиц* (*particle swarm optimization*, PSO), алгоритм *адаптивной свободной оптимизации* (*adaptive immune optimization algorithm*, AIOA), *популяционный* *алгоритм* (*populationbased algorithm*). Алгоритмы данной группы имеют схожую структуру, включающую создание исходных конфигураций, индивидуальный отбор и иммунную операцию. Методы были успешно использованы в других областях, к примеру, для производства ингибиторов бутирилхолинэстеразы.

Вторая группа алгоритмов, основанная на идее трансформирования энергетической поверхности, включает метод *случайного спуска* (*basin-hopping*, BH) и его модификации, алгоритм *скачкообразных минимумов* (*minima hopping*), алгоритм *перемещения по воронкам* (*funnel hopping method*).

Алгоритмы третьего класса осуществляют поиск на так называемой *динамической сетке* (*dynamic lattice*, DL), перемещая атомы с наивысшей энергией на вакантные узлы решетки с наименьшей энергией, тем самым позволяя отыскивать возможные местоположения исходного минимума с помощью построенных сеток. Семейство методов поиска на динамической сетке (*dynamic lattice searching*, DLS) включает метод *поиска с конструированием ядра* (*DLS with constructed core*, DLSc), метод *с операцией поворота* (DLS with rotation operation, DLS-RO), метод *с двухэтапной локальной оптимизацией* (*DLS with two-phase local minimization method*).

Существуют и другие эффективные алгоритмы, такие как методы *большого взрыва* (*big-bang method*) и *непрерывной экстремальной оптимизации* (*continuous extremal optimization*, CEO).

### Эволюционные методы

Название алгоритма объясняется тем, что его основные механизмы – наследование, мутации, селекция, и т. д. аналогичны механизмам биологической эволюции [17]. То же можно сказать и об используемой терминологии.

В последнее десятилетие было разработано множество эволюционных алгоритмов для глобальной минимизации энергетических ландшафтов, особенно для кластеров Леннарда-Джонса, а иногда и для кластеров Морса [18-20].

Большинство эволюционных подходов основаны на совместном использовании эволюционного поиска и процедуры локальной оптимизации, используемой для уточнения результатов, обеспечиваемых эволюционным алгоритмом.

Общая схема функционирования эволюционных алгоритмов представлена на рисунке 3.

Рассмотрим основные функциональные блоки алгоритма:

1. Инициализация входных данных: атомного состава, оптимизируемого параметра (в нашем случае это полная энергия системы), граничных условий.
2. Формирование первого поколения – набора точек в пространстве поиска, удовлетворяющего граничным условиям. Каждый представитель первого поколения подвергается локальной оптимизации (релаксация), для него вычисляется значение целевой функции. Из всех построенных структур первого поколения выбирается определенная доля наилучших структур, которые могут участвовать в производстве следующего поколения (селекция). По отношению к структурам следующего поколения они являются родительскими.

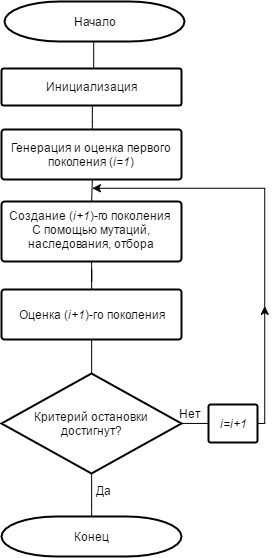


Рисунок 3 – Общая схема эволюционных алгоритмов

1. Формирование нового поколения из родительских структур путем применения эволюционных операторов. Существует несколько базовых эволюционных операторов, которые можно разделить на две категории: мутации, создающие новую структуру из одной родительской структуры и наследование (кроссинговер), использующее две и более родительских структуры. Также для сходимости предполагается оставлять определенное количество наилучших структур (элиту) без изменения.
2. Потомство используется как родительское поколение для создания следующего поколения согласно пунктам 2 и 3.
3. Последовательность пунктов 2 – 4 повторяется до выполнения критерия остановки.

Одной из основных проблем применения эволюционных алгоритмов для оптимизации кластера является их вычислительная сложность. Для снижения сложности используются эволюционные алгоритмы с быстрой сходимостью. Метод *дифференциальной эволюции* (*differential evolution*, DE) [21] - пример простой и быстрой реализации. Его основная идея - создание новых испытуемых из элементов текущей популяции путем случайного выбора трех так называемых родителей , и и объединения их   
. Первая адаптация этого алгоритма к задачам оптимизации кластера представлена в [22] (DELP). Особенности этого подхода: в каждом поколении кандидат уточняется путем применения процедуры локального поиска; это итерационный подход в том смысле, что популяция из частиц, генерируется из локальных минимумов, полученных для случая частиц, заполненных случайно генерируемыми компонентами.

### Семейство методов BH

Значительный прорыв в оптимизации кластеров начался в конце 90-х, когда в работах D.J.Wales и J.P.K.Doye был представлен метод BH, послуживший началом активного исследования возможностей подходов, трансформирующих ландшафт поверхности потенциальной энергии применением методов локальной оптимизации.

Основную идею метода составляет представление о том, что для потенциальной функции локальные экстремумы группируются в ограниченное число, так называемых, «бассейнов», в каждом из которых «воронки» локальных минимумов расположены настолько близко друг к другу, что за счет случайных возмущений координат атомов кластера возможен постепенный переход от локальных экстремумов к глобальному. При этом для определения локального экстремума в каждой из «воронок» использовались локальные методы непрерывной оптимизации.

До недавнего времени только в редких работах отмечалось успешное использование методов локальной оптимизации для задачи анализа конформаций атомных кластеров; сегодня же, напротив, ни одно исследование не обходится без упоминания методов локальной оптимизации – среди часто используемых, например, можно упомянуть метод квазиньютоновский метод с ограниченной памятью (L-BFGS) [23].

Именно с работ [24], [25] началось использование наиболее удачных техник, связанных с BH и его популяционными вариантами. Основополагающими для всех методов, которые позже развивали подход D.J.Wales, J.P.K.Doye и H.A.Scheraga, явились понятия и принципы, обобщенные в обзоре М. Locatelli [26], [27].

Так, для теории методов BH, как и для многих других методов, использующих метаэвристические подходы, важно понятие окрестности. Введенное понятие *структуры окрестности* (*neighborhood structure*) необходимо для решения задачи ГО в случае большого числа локальных оптимумов и относительно дешевой процедуры локального поиска.

Рассмотрим некоторые определения из [27].

Если обозначить за множество локальных оптимумов функции над и предположить, что множество конечно, то определить стратегию поиска окрестности можно как функцию:

, (1.7)

которая ставит в соответствие каждому локальному оптимуму подмножество . Тогда можно определить направленный граф со множеством вершин , таким образом, что любые две вершины будут считаться смежными, если:

(1.8)

Можно сформулировать задачу глобальной оптимизации как задачу комбинаторной оптимизации над графом , где каждый локальный оптимум , такой что:

, (1.9)

является локальным оптимальным решением для данной задачи комбинаторной оптимизации. Если обозначить за множество таких локальных оптимумов, очевидно, что , и, при условии определения стратегии поиска окрестности соответствующим образом, мощность множества будет значительно меньше мощности множества .

В [28] множество вида было названо множеством локальных оптимумов *уровня 1*, в то время как множество вида – множеством *уровня 0*. Таким образом, каждый можно связать со множеством:

, (1.10)

состоящим из всех , достижимых из в направленном графе ; причем путь будет определяться видом стратегии поиска окрестности и целевой функции . Указанное множество можно рассматривать как регион притяжения локальных оптимумов уровня 1. При этом локальный оптимум может принадлежать одновременно нескольким регионам притяжения оптимумов одного уровня. Такое множество принято называть *воронкой*.

Наиболее простой и в то же время эффективной стратегией поиска окрестности, часто используемой для решения задач глобальной оптимизации атомарных кластеров, заключается в композиции двух сдвиговслучайной генерации точки внутри шара с центром в текущем локальном оптимуме и запуске алгоритма локальной оптимизации из этой точки. В этом случае

, (1.11)

где , – функция вычисления локального оптимума по заданной точке .

В некоторых простых случаях, окрестность можно подобрать таким образом, что вычислительные затраты алгоритма будут невелики, а множество будет состоять из глобальных оптимумов. Если же множество , меньшей мощности, чем множество , будет состоять только из локальных оптимумов, уместно использовать мультистартовый подход, в котором производятся несколько запусков алгоритма из различных начальных точек. Мультистартовая версия данного алгоритма в литературе обычно известна как *monotonic* BH (MBH).

Содержательно методы BH и MBH используют структурную особенность оптимизируемой функции, заключающуюся в том, что вероятность обнаружения локального минимума с более низким потенциалом значительно выше из точек начальных приближений, расположенных в окрестностях локальных минимумов с низкими значениями потенциальной функции. В результате последовательно, переходя из одних локальных минимумов в локальные минимумы с меньшими значениями потенциальной функции, можно попасть в глобальный минимум. Если бы все локальные минимумы, включая глобальный, независимо от значений потенциальной функции были бы равномерно распределены в пространстве оптимизируемых переменных, то поиск минимума был бы невозможен.

Основной характеристикой методов BH принято считать введение стратегии поиска окрестности, которая призвана решить задачу умного исследования графа, построенного на множестве локальных оптимумов.

Недостатком мультистартового подхода является необходимость большого количества запусков алгоритма. Альтернативой может выступить популяционная модификация, представленная M. Locatelli и F.Schoen [27].

Пусть на каждой итерации доступна популяция . Генерируется множество точек-кандидатов , где . После этого точка сравнивается с представителем популяции , наиболее отличных от . В зависимости от результатов сравнения определяется новое значение .

Мера *отличности* точек *(diversity)* определяется некоторой функцией . В идеальном случае значение функции должно быть близким к нулю для любых двух оптимумов, имеющих высокую вероятность попадания в регион притяжения одного локального оптимума уровня 1. Тогда описанная функция выступает инструментом *диверсификации* при исследовании областей притяжения оптимумов и позволяет избежать их повторного исследования.

В то время как, описанные выше подходы определения стратегии поиска окрестности оказались удачными, для некоторых задач оптимизации кластеров Леннарда-Джонса, по мере роста размерности, становилась очевидной потребность в новых методах, менее требовательных к вычислительным ресурсам.

Первая такая модификация метода BH была предложена в [31] и позже доработана в многочисленных исследованиях ([27], [32], [33]). Основная идея подхода заключалась в использовании предположения, о том что оптимальным конформациям кластеров соответствует плотная упаковка атомов (со многими парами атомов), расположенными на расстоянии, близкими к . Для того чтобы генерация таких плотных конформаций производилась быстрее, предлагается заменить стандартную процедуру локального поиска двухфазной процедурой.

Первая фаза процедуры локальной оптимизации ориентирована на поиск «плотных» конформаций и использует модифицированную целевую функцию с добавлением штрафа:

, (1.12)

где первое слагаемое – стандартная функция потенциала, второе – линейная составляющая штрафа, третье – штраф, учитывающий только те пары атомов, между которыми взвешенное расстояние вида больше заданного значения порога .

Важно отметить, что вид штрафов сильно зависит от функции потенциала и от желаемого эффекта при генерации плотных конформаций. Когда требование плотности структуры не выполняется – например, как в случаях с кластерами воды, необходимо принимать во внимание иные геометрические особенности.

Другим важным направлением модификации метода BH является использование техники *прямых сдвигов* (*direct moves*) – сдвигов, которые учитывают специфику конкретной задачи для прямого манипулирования неудачно расположенными атомами. Полезность применения техники в совокупности с BH доказана тем, что она помогает ускорить спуск в воронку и быстрее достигать конформаций с меньшей энергией для кластеров с большим числом атомов.

Одним из первых указал на такую возможность Hartke [34]. Он использовал понятие *прямой мутации (direct mutaion)* для случайного перемещения атомов с большим вкладом энергии.

В данной области известны и другие популяционные методы, например, использующие метод роя частиц [35], но до сих пор применялись с ограниченным успехом. В работе [36] идея подверглась значительной модификации и сформировала основу метода, который авторы назвали *перемещение по воронкам* (*funnel hopping)*. Общая схема метода состоит из следующих этапов:

1. Выбор *поверхностных атомов*. Простой подсчет энергетического вклада каждого атома. При этом под поверхностными понимают атомы с наибольшим вкладом энергии.
2. Определение графа окрестностей. Граф строится на множестве вершин, соответствующих атомам, и множестве ребер, соединяющих пары атомов, расстояние межу которыми приближается к .
3. Конструирование треугольников. Выделение в графе треугольников.
4. Определение сетки. Для каждого треугольника осуществляется геометрическая процедура для нахождения двух вершин правильного тетраэдра, которые включаются в структуру, именуемую *динамической сеткой*. Сетка называется динамической в противопоставление *статической* сетке геометрически-обоснованных методов, использующих априорные представления о геометрии кластера.
5. Стягивание сетки. Для каждой точки запускается алгоритм локальной минимизации энергии кластера с фиксированными атомами и одним переменным положением атома, определяемое размещением точек динамической сетки.

Таким образом, описанная процедура позволяет сгенерировать выгодные свободные положения атомов. После конструирования сетки применяется стратегия комбинаторного поиска для нахождения удачной для перемещения комбинации атомов. Фактически реализуется перемещение атомов из позиций, где они вносят большой вклад в потенциальную энергию, в позиции атомов динамической решетки, если это уменьшает значение общего потенциала.

В сущности, указанный подход является разновидностью метода BH, когда стратегия поиска окрестности определяется *k* операциями обмена между свободными положениями и положениями атомов с большим энергетическим вкладом.

Использование прямых операций над неудачно расположенными атомами оказалось весьма продуктивным. В [36] докладывались успешные результаты – подтверждение всех известных конформаций для кластеров Леннарда-Джонса и Морса, полученные со сравнительно небольшими вычислительными затратами.

### Методы с конструированием ядра

Вышеперечисленные методы, основанные на теории BH, доказали свою эффективность в задачах оптимизации кластеров средних размеров. Однако для кластеров с числом атомов свыше 561 они требуют слишком много накладных расходов, а иногда и вовсе оказываются неприменимыми. Для того чтобы ускорить процесс оптимизации были предложены стратегии оптимизации, основанные на конструировании динамической сетки. И хотя с их помощью удалось сократить время поиска, основным их недостатком осталась замкнутость конечного решения внутри конфигурации сетки.

Не так давно была предложена модификация метода поиска динамической сетки с конструированием, где в качестве ядра начальных конформаций предлагались икосаэдрические, декаэдрические или гранецентрированные кубические структуры. Данный подход оказался эффективным – он позволил обнаружить геометрически несмещенные решения для кластеров Леннарда-Джонса из 100-200 атомов и кластеров серебра из 121-310 атомов с потенциалом Гупта. Тем не менее, в случае кластеров больших размеров метод DLSc дает лишь смещенные решения, что можно объяснить снижением эффективности техники перемещения атомов по мере увеличения числа числа атомов в кластере.

В некоторых алгоритмах для улучшения производительности методов предлагалось использовать различные эвристические операторы. Среди таких операторов оператор поворота доказал особую эффективность для исследования максимально различных решений. В ходе применения данного оператора часть кластера поворачивается на случайный угол вокруг заданной или случайной оси, что позволяет добиться достаточной структурной трансформации кластера для выхода из локального минимума.

Было показано, что икосаэдрические и декаэдрические структуры – важнейшие среди некристаллических структур (например, для кластеров Леннарда-Джонса [37], серебра [38]). Кроме того, было установлено, что декаэдрические структуры могут быть преобразованы в икосаэдрические структуры путем простого применения упомянутого оператора поворота – rotation operation (RO). При этом оператор поворота должен производиться над половиной декаэдрического кластера вокруг пятикратной оси симметрии, и угол поворота выбираться из соображений структурных различий между декаэдром и икосаэдром, а не случайно, как утверждается во многих исследованиях [39]. Вышесказанное означает, что добавление процедуры поворота к методу DLSc с декаэдрическим ядром может позволить добиться решений обоих типов структур.

Так родилась идея комбинированного метода DLSc-RO, представляющего собой объединение принципов DLSc и техники поворота. Данный метод был применен для оптимизации кластеров Леннарда-Джонса размером свыше 500 атомов, кластеров Морса и кластеров некоторых металлов: серебра и алюминия. В силу успешности достигнутых результатов, авторы метода убеждены в его универсальности.

Как уже было замечено, путем перемещения атомов возможно преобразование декаэдрических структур в икосаэдрические. Однако с помощью только лишь процедуры поворота из кластеров декаэдрической структуры со множеством оболочек (*multi-shell*) нельзя получить кластер икосаэдрической структуры. Необходима еще процедура локальной оптимизации. В основу такой процедуры X. Wu, W. Cai, X. Shao предлагают положить метод L-BFGS [38].

Обычно запуск процедуры DLS включает генерацию начальной структуры и последовательное выполнение процедур конструирования сетки и поиска на сетке. Стартовая конформация строится путем применения локальной минимизации к случайно сгенерированной структуре. После этого формируются и исследуются позиции динамической сетки для отыскания решений с меньшим значением энергии. Процедуры конструирования сетки и поиска на сетке повторяются до тех пор, пока не будет найден новый минимум с лучшим значением энергии.

Из анализа работы процедуры DLS очевидно, что в ходе оптимизации больший акцент делается на внешних атомах. Так, если атомы ядра в стартовой конформации генерируются неудачно, задача поиска глобального минимума становится крайне сложной и трудоемкой. Именно по этой причине был предложен метод DLSc [40], в котором использование ядра декаэдрической или икосаэдрической структуры существенно ускоряет вычисления, хотя и смещает решения для кластеров больших размеров.

Сам алгоритм DLSc-RO, который был призван добиться еще большей эффективности, чем его предшественник DLSc, можно представить последовательностью из следующих шагов:

1. Генерация стартовой структуры с ядром декаэдра. При этом размер ядра определяется размером кластера. Как и в DLSc, обычно используют ядро с оболочками.
2. Применение к стартовой структуре процедуры DLS для получения структуры с фрагментами декаэдра.
3. Применение оператора поворота для получения икосаэдро-подобной структуры.
4. Дальнейшее применение процедуры DLS для улучшения икосаэдрической структуры.

Очевидно, что при такой стратегии поиска глобальный минимум может быть обнаружен либо среди декаэдрических структур, полученных в результате первого применения процедуры DLS, либо среди икосаэдрических структур, полученных при последующем применении процедуры DLS. Более того, так как DLSc-RO не гарантирует нахождение глобального минимума, необходимо повторение итераций.

Кроме кластеров Леннарда-Джонса с помощью метода DLSc-RO были исследованы кластеры Морса и кластеры серебра и алюминия. Так, например, для кластеров Морса размером 309 и 561 атомов были найдены минимумы со структурой полного икосаэдра.

Несмотря на успешность представленных результатов и типичность икосаэдрических и декаэдрических структур, предложенный метод DLSc-RO только еще предстоит исследовать в отношении кластеров с принципиально другой геометрией. Кроме того, изучение кластеров с числом атомов 923 и более до сих пор представляет серьезные трудности из-за связанных с этим вычислительных затрат.

# Методика решения поставленной задачи

Описываемый в работе эвристический алгоритм глобальной оптимизации атомных кластеров Морса разработан заведующим кафедры программных система Самарского Университета, профессором А.Н. Коварцевым. Суть алгоритма в формировании некоторой начальной конфигурации, которая после локальной оптимизации дала бы решение задачи глобальной оптимизации. Цель достигается благодаря эффективному сочетанию нескольких алгоритмов и эвристик, адаптированных под специфику задачи: формирования плотной упаковки атомов, метода Стронгина, локальных оптимизаций и алгоритма роста.

Общая структура алгоритма представлена на рисунке 4.

Подробное описание разработанных методов приведено в соответствующих параграфах.

D:\Google Disk Files\ДИССЕРТАЦИЯ\глава 3\материалы\baseAlg1 (2).png

Рисунок 4 – Общая структура алгоритма

## Формирование плотной упаковки атомов

Во избежание экспоненциального роста сложности в предложенном алгоритме осуществляется переход от задачи пространственной оптимизации переменных к задаче оптимизации одной переменной. Такой переход становится возможным благодаря использованию заранее выгодной геометрической структуры.

При формировании атомной решетки для кластера важно достичь как можно более плотной упаковки. Для жестких шаров наибольшая плотность достигается для гранецентрированной кубической решётки. Однако для атомов кластера Морса, когда атомы могут частично «вминаться» друг в друга, более плотные упаковки формируются для икосаэдрических и додекаэдрических сеток [10, 36].

Практикой доказано, что икосаэдрические, додекаэдрические и гранецентрированные решетки являются хорошим начальным приближением для поиска глобального минимума потенциальной энергии. Благодаря этой закономерности можно заранее построить плотную упаковку атомов, и уже в ней осуществлять поиск наиболее выгодной конформации.

Как упоминалось ранее, кластеры Морса демонстрируют структурное многообразие. Выбор геометрической фигуры для построения решетки атомов должен зависеть от параметра . Например, при наиболее выгодна икосаэдрическая или ей подобную структура. В то же время при ρ = 14 среди глобально оптимальных конформаций часто встречаются додекаэдрические и гранецентрированные решетки. На рисунке 5 изображены икосаэдрическая и додекаэдрическая структуры.

Анализ атомных структур в базе данных показал, что найденные конформации можно разложить на слои. Этот факт позволяет упростить построение решетки атомов, поскольку вместо трехмерной структуры формируется лишь несколько двумерных слоев.

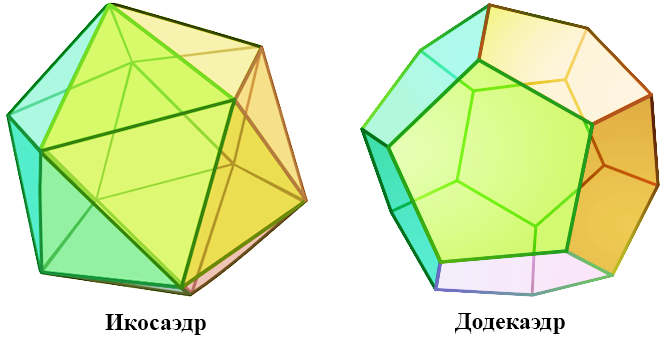


Рисунок 5 – Икосаэдрическая и додекаэдрическая структуры

Пример использования слоев представлен на рисунке 6.

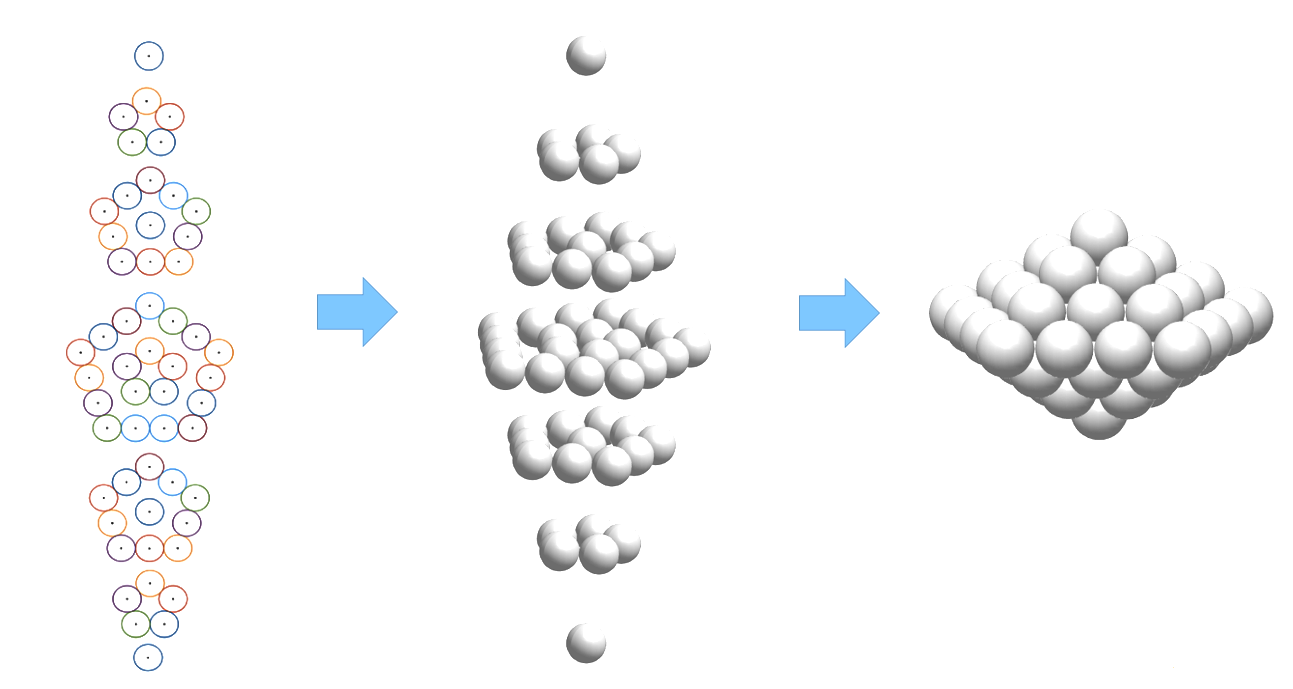


Рисунок 6 – Послойное формирование конформации

В предложенным алгоритме используется геометрически обоснованный метод [41]. Основу данного метода составляет алгоритм формирования конфигураций для размещения центров атомов, с помощью которых строятся пространственные слои плотной упаковки шаров.

Использование заранее сформированной решетки атомов позволит существенно упростить поставленную задачу поиска координат атомов, сведя ее к более простой задаче структурной оптимизации, а именно к выбору наиболее выгодных атомов из доступных.

Многочисленные эксперименты над кластерными структурами показали, что кластеры, имеющие особые размеры, обладают повышенной стабильностью. Такие кластеры называют «магическими», а их размеры – «магическими числами». Близкие к ним по размерам оптимальные кластеры часто образуются путем добавления или удаления небольшого числа атомов. Однако в общем случае каждая последующий кластер не всегда повторяет своего предшественника, добавляя к нему очередной атом, зачастую меняется и количество слоев, и их структура. Тем не менее, обладая информацией об оптимальных структурах предыдущих размеров, в большинстве на некотором интервале размерности можно заранее выделить устойчивую группу атомов, переходящую без кардинальных изменений от одной конформации к другой.

Для еще большего упрощения дальнейшей оптимизации, на данной этапе работы алгоритма предполагается зафиксировать несколько атомов, задав начальную конформацию и тем самым уменьшив интервал поиска.

## Оптимизация функции одной переменной

Полученную задачу структурной оптимизации можно привести к классической задаче глобальной оптимизации функции одной переменной:

, (2.1)

где – численное представление конформации, – потенциальная энергия кластера.

Так, имея решетку атомов, конфигурацию атомов можно представить в виде двоичного числа длиной , имеющее ровно единиц. Например, на рисунке 6 изображено представление конформации 3 атомов в сетке из 6 атомов.

Такой переход к новой постановке задачи позволил бы использовать существующие методы глобальной оптимизации, однако целевая функция не отвечает требованиям таких методов, например, непрерывности, поэтому требуется разработка собственного метода оптимизации.

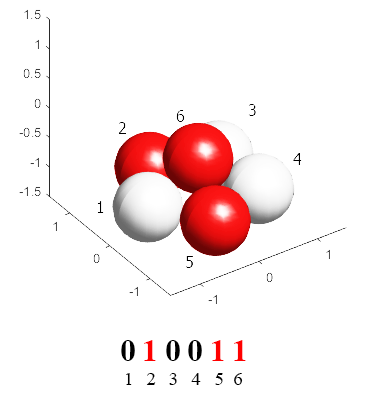


Рисунок 6 – Представление конформации в форме двоичного числа

Хотя методы глобальной оптимизации неприменимы или неэффективны для поставленной задачи, тем не менее можно использовать их адаптированные версии. В предложенном алгоритме ключевое место занимает модифицированный *метод Стронгина* [42]. Данный статистический метод старается размещать точки испытаний либо в окрестностях локальных минимумов оптимизируемой функции, либо в интервалах неопределенности, размеры которых велики по сравнению с другими участками функции. Метод Стронгина определяет некоторое количество атомов, из которых составляется конфигурация, в дальнейшем достраиваемая алгоритмом развития до необходимого размера .

Используемый алгоритм развития основан на идее восходящей последовательности, то есть конформация атомов образуется из найденной заранее конформации атомов присоединением наиболее выгодного атома. Данное допущение компенсируется рядом эвристик, позволяющих отыскать оптимальное решение.

В процессе работы метода Стронгина сохраняется несколько конфигураций, которым соответствовали наименьшие значения потенциальной энергии. Причем, для каждой малой окрестности алгоритм хранит не более одной конфигурацию, тем самым обеспечивая разнообразие найденных структур. Все найденные структуры подвергаются процедуре локальной оптимизации. Структуры с наименьшей потенциальной энергией считаются искомым результатом.

Таким образом, искомая конформация атомов образуется поэтапно, благодаря нескольким алгоритмам и эвристикам, а значение можно представить в виде суммы:

, (2.2)

где – количество атомов начальной конфигурации, – количество атомов, определяемых методом Стронгина, – количество «достраиваемых» атомов алгоритмом развития.

Подобное использование алгоритмов напоминает по своей структуре генетические методы. В данном случае родительскими структурами являются начальная структура, метод Стронгина, и алгоритм развития.

# Используемые методы и алгоритмы

Как упоминалось ранее, несмотря на существенное упрощение задачи, решить ее прямыми методами не представляется возможным. Поэтому в рамках предлагаемой методики были разработаны алгоритмы, учитывающие специфику работы с атомными кластерами.

## Метод Стронгина

Метод Стронгина позволяет находить абсолютный минимум функции на отрезке. Проводится ряд испытаний, в результате которых начальный интервал поиска делится на несколько меньших интервалов. Очередная точка испытаний выбирается внутри интервала, на котором наиболее вероятен абсолютный минимум, добавляется в список известных значений, после чего происходит переход к следующей итерации. Положение точки в интервале соответствует математическому ожиданию положения минимума. Алгоритм останавливается, когда расстояние между точками очередного делимого отрезка становится меньше заданного критерия.

Целевая функция должна удовлетворять обобщенному условию Липшица на всем интервале поиска:

, (3.1)

где , – любые числа из интервала поиска, – константа, – метрика.

Рассмотрим функцию на отрезке вещественной оси.

Пусть имеется некоторая последовательность испытаний:

. (3.2)

Для удобства обозначим – значения целевой функции в точках отрезка .

Процедура работы алгоритма состоит из нескольких этапов.

1. Оценить максимальное абсолютное значение относительной первой разности:

. (3.3)

1. Вычислить

, (3.4)

где – заданный параметр алгоритма.

1. Для каждого интервала вычислить характеристику, определяющую вероятность нахождения глобального минимума на этом интервале.

(3.5)

Величина характеристики пропорциональна вероятности.

1. Определить интервал , которому соответствует максимальная характеристика:

. (3.6)

Если максимальная характеристика соответствует нескольким интервалам, то в качестве берется минимальное число.

1. Вычислить

. (3.7)

1. Перенумеровать точки в порядке возрастания значений и повторить пункты 1-5.
2. Алгоритм завершает работу, когда , где – заданное число.

Для использования метода Стронгина в предложенной методике была использована целевая функция, использующая специально разработанный алгоритм развития.

В качестве начального интервала используется отрезок , составленный в зависимости от размеров искомой конформации и решетки атомов по принципу, изображенному на рисунке 7.

D:\Google Disk Files\ДИССЕРТАЦИЯ\глава 3\материалы\abStrongin.png

Рисунок 7 – Определение начального интервала поиска

Целевая функция ищет в текущем интервале ближайшие к числа и , содержащие единиц в двоичной форме, добавляет исходя из найденных значений определенное количество атомов в текущую конфигурацию и запускает алгоритм развития для формирования структур атомов. Потенциалы Морса полученных конформаций определяют итоговое значение целевой функции. В случае, когда числа с заданным количеством единиц отсутствуют, рассматриваемый интервал в дальнейшем игнорируется.

Для предотвращения возможных задержек алгоритма в одной и той же окрестности локального минимума, интервал удаляется из рассмотрения при достижении определенной длины.

Поскольку в нашем случае метод Стронгина из-за низкой скорости сходимости используется для определения лишь первых атомов, и полная его работа не требуется, критерий остановки был заменен на новый, зависящий от размера начального интервала и параметра .

## Поиск двоичных чисел с заданным количеством единиц

Пусть имеется число c единицами в двоичной системе счисления. Требуется найти ближайшие к числа , содержащие ровно единиц в двоичной системе счисления.

Данная задача решается на каждой итерации метода Стронгина для подсчета целевой функции, поэтому использовать перебор для поиска требуемых чисел неэффективно. Рассмотрим более выгодный вариант решения.

Алгоритм решения зависит от соотношения заданных чисел и , поэтому возможны 3 случая. Если , искомые числа .   
В таблицах 1-2 рассмотрены случаи, когда и соответственно.

Таблица 1 – Случай .

|  |  |
| --- | --- |
| Число | Алгоритм |
|  | Зануляем единиц, двигаясь по справа налево. |
|  | Пусть .   1. Ищется такой разряд с нулем, что справа от него есть хотя бы разрядов с единицами. Поиск ведется по числу справа налево. 2. Если разряд не найден, задача решается добавлением нуля в старший разряд и повторением шага 1. 3. В найденном разряде ноль заменяется единицей, а ближайшие единиц справа зануляются. 4. Оставшиеся единицы справа сдвигаются в младшие разряды. |

Таблица 2 – Случай .

|  |  |
| --- | --- |
| Число | Алгоритм |
|  | Меняем нулей на единицы, двигаясь по справа налево. Если в числе нет нулей, задача решается увеличением числа разрядов. |
|  | Пусть .   1. Ищется такой разряд с единицей, что справа от него есть хотя бы разрядов с нулями. Поиск ведется по числу справа налево. 2. Если разряд не найден, задача нерешаема. 3. В найденном разряде зануляется единица, а ближайшие нулей справа меняются на единицы. 4. Оставшиеся нули справа сдвигаются в младшие разряды. |

## Алгоритм развития

Идея алгоритма имеет много общего с подходом, лежащим у истоков детерминированной глобальной оптимизации атомных кластеров – подход *схемы развития* (*growth scheme*) [43]. Согласно этой схеме оптимальные кластеры Леннарда-Джонса искались путем присоединения дополнительного атома к кластерам с плотной структурой (выгодной конфигурации с точки зрения вкладов атомов в общую сумму потенциальной энергии). Все полученные локальные минимумы регистрировались для отбора наилучших.

Имеющуюся решетку атомов можно описать в форме графа , где   
 – множество вершин (атомов), – множество переходов, причем будем считать атомы смежными, если напряжение между и атомами меньше критерия ближайшего соседа: .

Тогда задача алгоритма заключается в поиске графа   
:

(3.8)

где – потенциальная энергия кластера, соответствующего заданному графу.

Графическая интерпретация алгоритма представлена на рисунке 8.

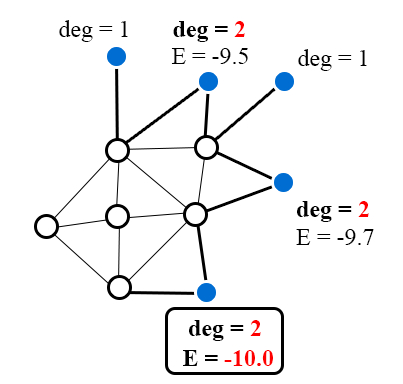


Рисунок 8 – Принцип работы алгоритма развития

Блок-схема алгоритма представлена на рисунке 9.

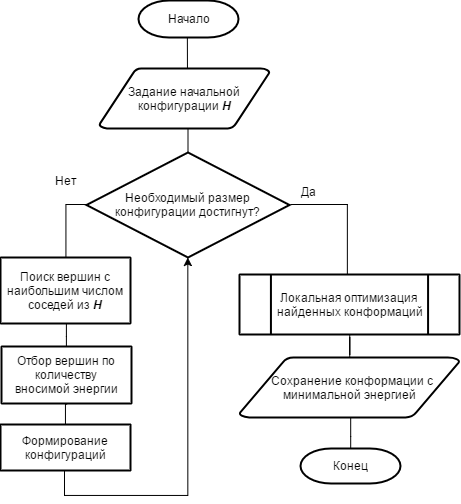


Рисунок 9 – Блок-схема алгоритма развития

Рассмотрим каждый этап подробнее.

Перед работой алгоритма задается начальная конформация , состоящая из одного или более атомов.

1. На текущем этапе имеется подграф . Среди вершин , ищутся те, что имеют максимальное количество смежных вершин из графа .
2. Пусть – множество вершин, полученное на предыдущем этапе. Если , алгоритм переходит к пункту 3. В противном случае для каждой вершины определяется значение энергии для графа   
   . В множестве остаются только те вершины, которым соответствует минимальное значение энергии или близкое к нему, тем самым проводится «фильтрация» .
3. Для каждой из вершин множества строится новый граф   
   .
4. Для каждого из найденных графов повторяются шаги 1-3 до тех пор, пока не будут сформированы структуры заданного размера .
5. Из всех построенных графов выбираются конформации с наилучшим значением потенциальной энергии, и для них решается задача локальной оптимизации (3.10). Решением считается кластер, которому соответствует минимальное значение энергии после локальной оптимизации.

Процедура отбора необходимы для получения эффективного подмножества локальных оптимумов с числом атомов , которые бы служили ядром для оптимумов следующего уровня – размером . Данный алгоритм ориентирован на поиск связей между глобальными оптимумами и оптимумами с «сильным» ядром – т.е. структур с таким расположением атомов, которое доказало свою оптимальность для кластеров меньших размерностей. Очевидно, что процедура отбора будет сильно влиять на обнаружение путей от некоторого устойчивой конформации меньшей размерности к оптимальной конформации большей размерности. Поэтому составляющие процедуры могут сильно варьироваться в зависимости от накладываемых ограничений. Многие критерии обнаруживаются экспериментально и используют знания предметной области (например, наблюдения о вкладе энергии соседних атомов).

## Локальная оптимизация

Большинство существующих методов глобальной оптимизации атомных кластеров являются двухэтапными, используя технику локальной оптимизации кластера из близкого к оптимальному решению положения, поскольку сложность алгоритмов локальной оптимизации имеет полиномиальный характер.

В предложенном алгоритме так же присутствуют этапы, включающие фазу локальной оптимизации:

, (3.9)

где – совокупность координат атомов, образующих конформацию.

В данной работе в качестве метода локальной оптимизации использовался метод BFGS (Broyden, Fletcher, Goldfarb, Shanno) с ограниченной памятью. Версия алгоритма BFGS с ограниченной памятью была опубликована Liu и Nocedal в их совместной работе [44]. Преимущество модификации заключается в эффективной работе с большим числом оптимизируемых переменных.

L-BFGS на основе последних значений функции и ее градиента определяет положительно определенную аппроксимацию Гессиана, которая в дальнейшем используется для совершения шага по методу Ньютона. За счет использования фиксированного количества значений функции существенно снижается трудоемкость итерации. Обычно значение – небольшое число порядка 3-10, много меньшее размерности исходной задачи.

# Вычислительные эксперименты

## Программная реализация

В процессе работы было реализовано программное приложение поиска глобально-оптимальных структур кластеров Морса.

Программа написана языке программирования Java в среде разработки IntelliJ IDEA 14. Java предоставляет интерфейсы для использования таких структур, как *BigInteger* и *BigDecimal*, что позволило использовать в данном приложении арифметику произвольной размерности. Как упоминалось ранее, разработанный алгоритм хорошо поддается распараллеливанию, поэтому при его реализации был задействован пул потоков *ThreadPool*.

Более того, программы на Java транслируются в байт-код, выполняемый виртуальной машиной (JVM) – программой, обрабатывающей код и передающей инструкции оборудованию как интерпретатор. Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина.

Графический интерфейс приложения реализован с помощью библиотеки JavaFX с применением шаблона проектирования Модель-Представление-Контроллер (MVC). Библиотека автоматически задействует визуальные темы под каждую из распространенных операционных систем: семейство систем Windows, Mac OS, Unix.

### Последовательность действий работы с приложением

Для более удобного управления параметрами алгоритма, было решено вместо задания условий в диалоговых окнах графического интерфейса использовать текстовые конфигурационные файлы формата \*.json. Пример такого файла представлен на рисунке 10. Подробная информация о всех возможных параметрах приведена в справочной информации к приложению.

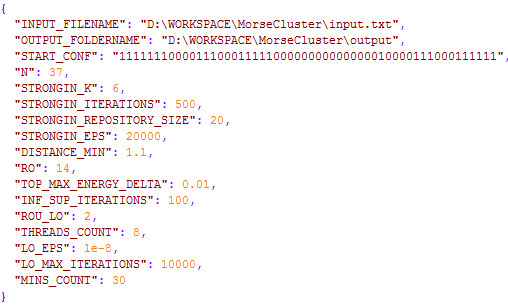


Рисунок 10 – Конфигурационный файл

Для предоставления пользователю удобного доступа к функциям приложения был разработан графический интерфейс.

При запуске программы открывается окно, изображенное на рисунке 11.

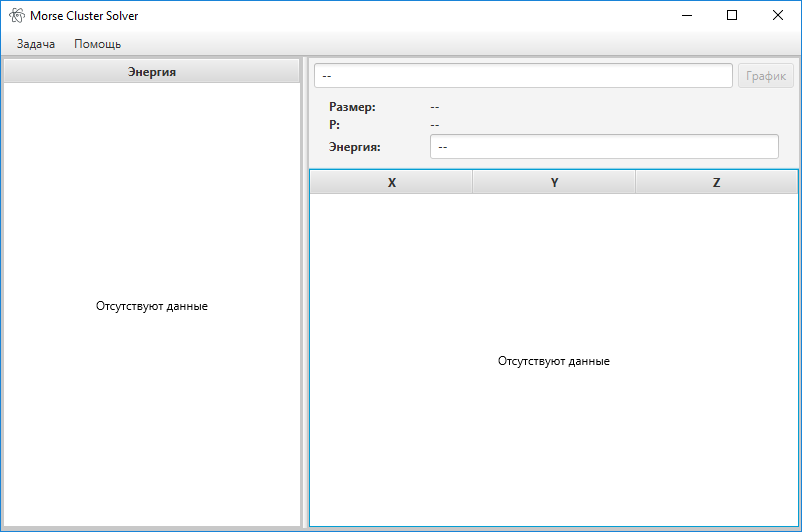


Рисунок 11 – Главное окно приложения

Главная форма приложения состоит из основного меню, списка найденных конформаций в левой части, панели с подробными сведениями о конформации в правой части.

В основном меню содержатся необходимые функции для работы с приложением. В пункте меню «Задача» содержатся кнопки запуска нового расчета «Расчет» и выхода из приложения. В пункте меню «Помощь» содержится кнопка «Справка».

Для пользователя на этом шаге доступны только кнопка «Расчет» и «Справка», остальные кнопки и поля будут доступны только после того, как система произведет хотя бы один расчет. При нажатии на кнопку «Справка», открывается диалоговое окно со справочной информацией о системе и ее разработчиках (рисунок 12).

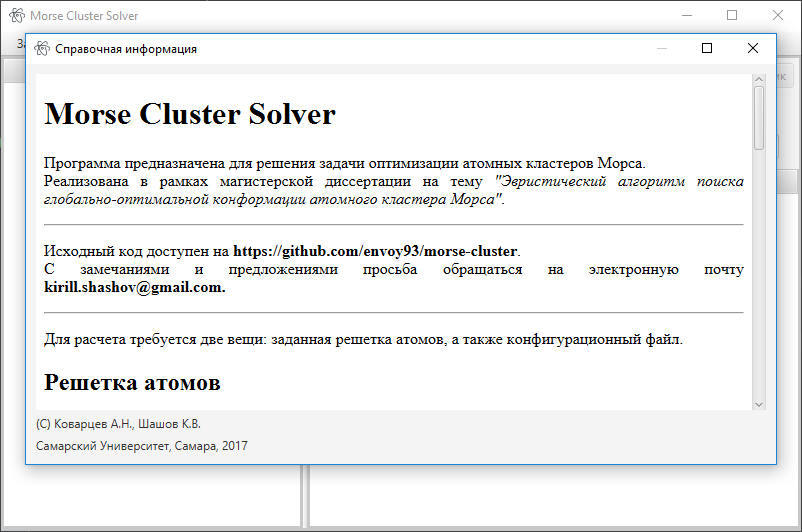


Рисунок 12 – Диалоговое окно со справочной информацией

После того, как пользователь нажал на кнопку «Расчет», приложение начинает вычисления. В нижней части окна появляется панель с отображением состояния прогресса задачи для каждого из вычислительных потоков, а также для координирующего потока, собирающего результаты в конце расчетов. Во время вычислений пользователь не может поменять исходные данные, начать новый расчет, выйти из приложения. Окно приложения приобретает вид, изображенный на рисунке 13.

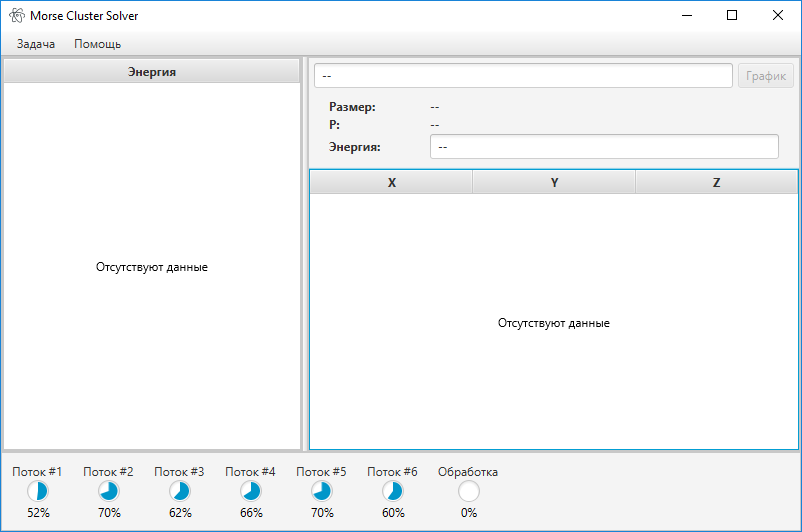


Рисунок 13 – Вид окна приложения при запущенном расчете

После того, как программа закончит расчеты, кнопки «Расчет» и «Выход» снова станут доступны. Нижняя панель с отображением прогресса вычислений скрывается, и выводится диалоговое окно с сообщением о конце расчетов и затраченном времени на них. В левой части экрана отображается список найденных конформаций (рисунок 14).

На этом этапе пользователь может посмотреть подробную информацию о каждой из найденных конформаций, для этого достаточно нажать на интересующую конформацию.

При выборе конформации в правой части экрана отобразится интересующая пользователя информация (рисунок 15).

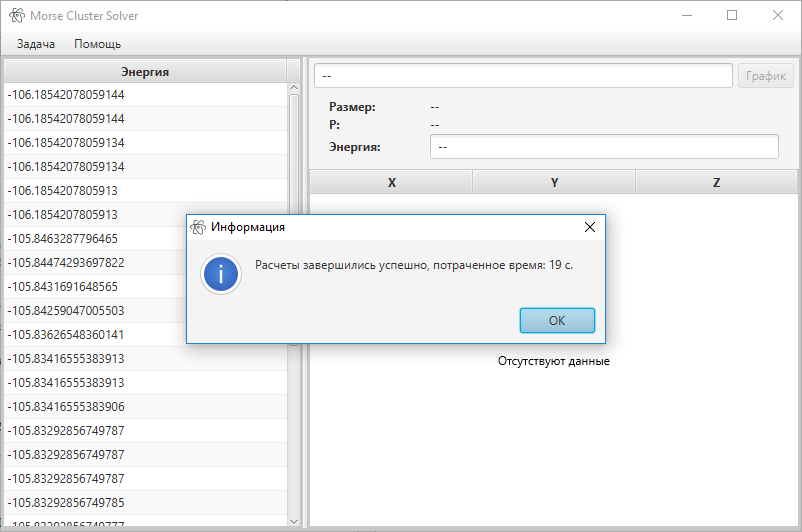


Рисунок 14 – Окончание вычислений

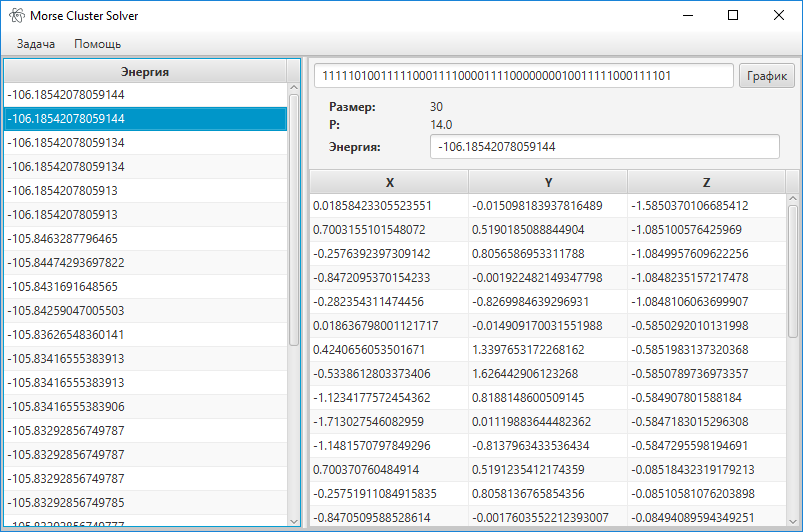


Рисунок 15 – Отображение подробной информации об атомной структуре

Координаты атомов, верхнее текстовое поле, отображающее битовое представление атомной структуры, и поле, отображающее значение энергии, доступны для быстрого копирования в буфер обмена.

Для любой конформации пользователь может открыть трехмерный график (рисунок 16) по нажатию на кнопку «График».

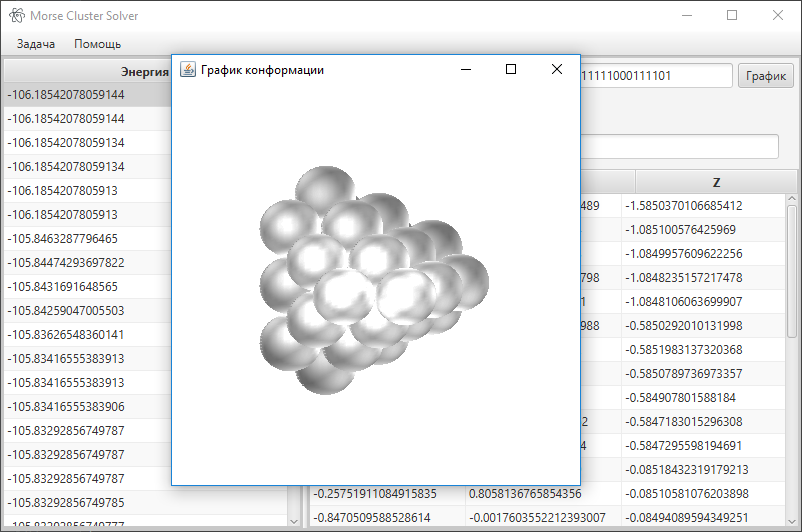


Рисунок 16 – Построение графика конформации

## Результаты поиска оптимальных конформаций

Первыми тестовыми испытаниями алгоритма и его программной реализации были задачи оптимизации кластеров размерностью менее 40 атомов. При поиске конформаций малых размеров наиболее удобно проводить тестирование не только всей разработанной технологии, но и отдельных ее составляющих: генерации решетки атомов, метода Стронгина, алгоритма развития, локальной оптимизации.

Результаты данных экспериментов по поиску оптимальных структур представлены в таблице 3.

Таблица 3 – Результаты поиска кластеров малой размерности

|  |  |
| --- | --- |
|  |  |
| 10 | -26.132735234616742 |
| 11 | -29.596054324057125 |
| 12 | -33.33230475638694 |
| 13 | -37.25887680231351 |
| 14 | -40.79834795841109 |
| 15 | -44.80643749849561 |
| 20 | -64.79195340669686 |
| 30 | -106.83578974677665 |
| 35 | -129.7373597747828 |
| 36 | -133.7446659471386 |
| 37 | -138.70858233923377 |

В качестве начальной решетки атомов использовалась сетка из 54 атомов, изображенная на рисунке 17. Эксперименты проводились при заданном параметре .

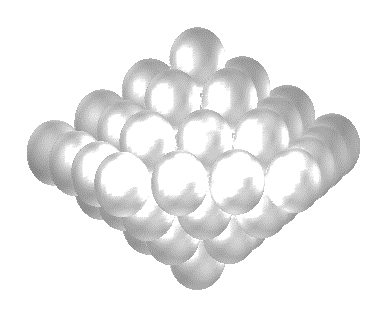


Рисунок 17 – Решетка из 54 атомов

Все найденные конформации являются глобально-оптимальными, их значения потенциала Морса совпадают с информацией из базы Cambridge Cluster Database [15].

Графики конформаций представлены на рисунках 18-28.

Конформации 10, 14, 15, 20 атомов были найдены исключительно благодаря алгоритму развития, сумевшему отыскать оптимум без метода Стронгина и начальной конфигурации.

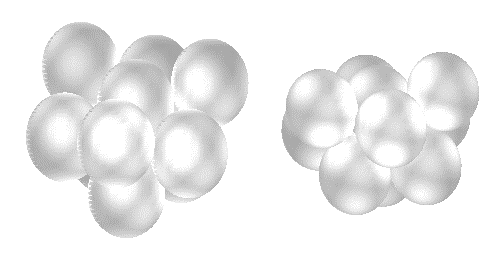


Рисунок 18 – Оптимальные конформации 10 атомов

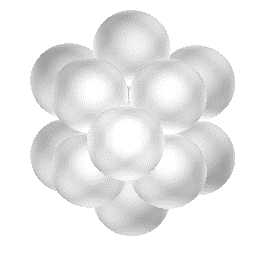


Рисунок 19 – Оптимальная конформация 14 атомов

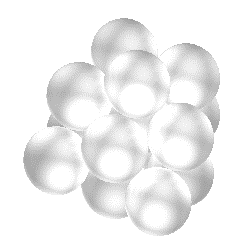


Рисунок 20 – Оптимальная конформация 15 атомов

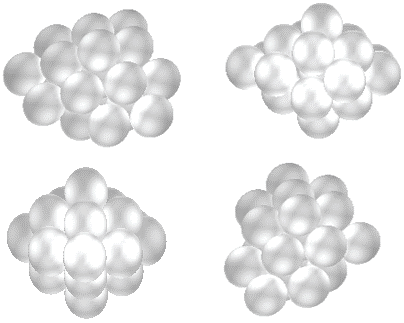


Рисунок 21 – Оптимальные конформации 20 атомов

Структуры 13 и 30 атомов потребовали задания начальной конфигурации: 2 атома в первом случае и 8 – во втором. Оба кластера были достроены алгоритмом развития.

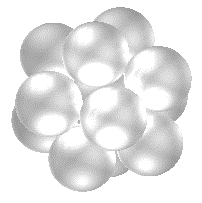


Рисунок 22 – Оптимальная конформация 13 атомов

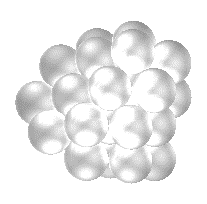


Рисунок 23 – Оптимальная конформация 30 атомов

Кластеры 11, 12 атомов найдены преимущественно за счет метода Стронгина, определившего 8 атомов, остальные атомы достраивались алгоритмом развития.

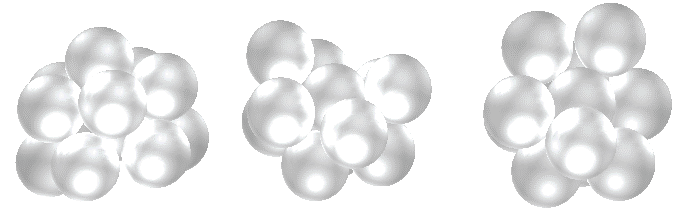


Рисунок 24 – Оптимальные конформации 11 атомов

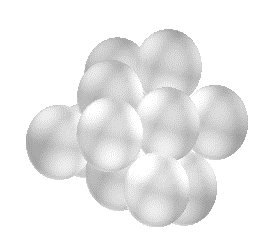


Рисунок 25 – Оптимальная конформация 12 атомов

Задача оптимизации кластеров 35-37 атомов потребовала анализа особенностей формирования атомных структур предыдущих размеров для выделения значимых атомов, которые в дальнейшем использовались в качестве начальной конфигурации (5-6 атомов).

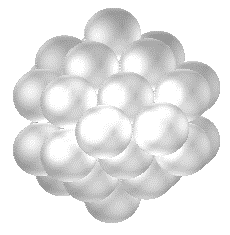


Рисунок 26 – Оптимальная конформация 35 атомов

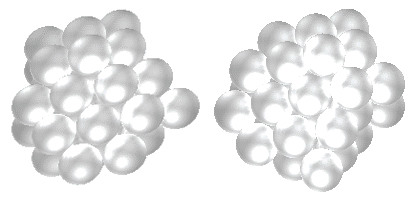


Рисунок 27 – Оптимальные конформации 36 атомов

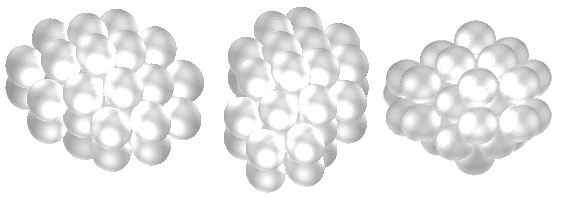


Рисунок 28 – Оптимальные конформации 37 атомов

В качестве более трудной задачи, требующей детальной настройки всех аспектов алгоритма для их эффективного взаимодействия, была выбрана задача поиска глобально-оптимальной конформации кластера Морса размером 209 атомов при .

Вычислительные эксперименты над кластерами предыдущих размерностей показали, что выгодной начальной конфигурацией в данном случае может стать плотная упаковка меньшего размера (например, 199 атомов). С выбранными параметрами алгоритм нашел сразу две структуры, которым соответствует глобальный минимум .

Одна из найденных конформаций изображена на рисунке 29, а ее представление в виде слоев – на рисунке 30.

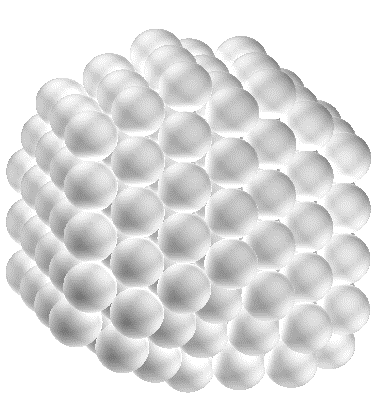


Рисунок 29 – Оптимальная конформация 209 атомов

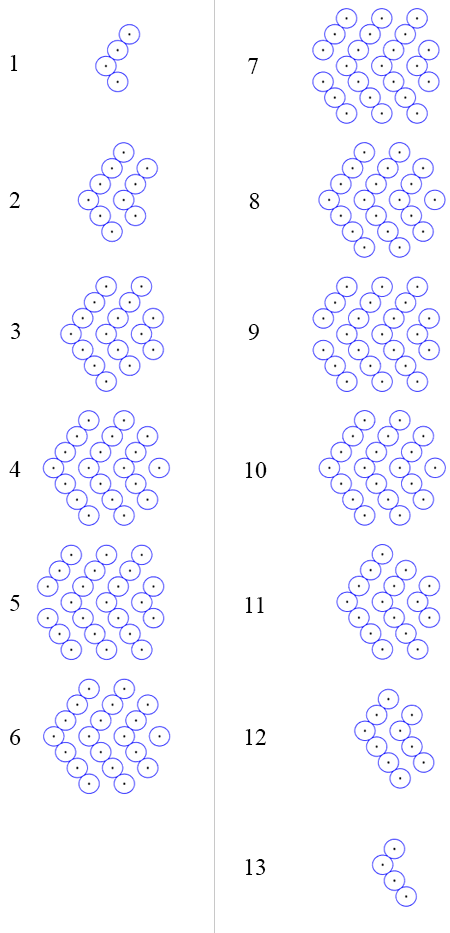


Рисунок 30 – Структура конформации 209 атомов в виде слоев

### Экспериментальное исследование параметров алгоритма

Разработанный алгоритм имеет множество параметров, которые существенно влияют на получаемый результат: начальная конфигурация, количество итераций на том или ином участке, точность локальной оптимизации и т.д. Тем не менее, описанные параметры достаточно легко подбираются исходя из размеров задачи и доступного времени для вычислений.

Наиболее неочевидным моментом является выбор количества атомов, которое будет определять метод Стронгина (2.2). На рисунке 31 представлен график, отображающий зависимость величины от ,   
где – наименьшее найденное алгоритмом значение энергии, – глобальный минимум энергии из базы данных [15], – количество атомов, определяемых методом Стронгина, – размер искомых конформаций.

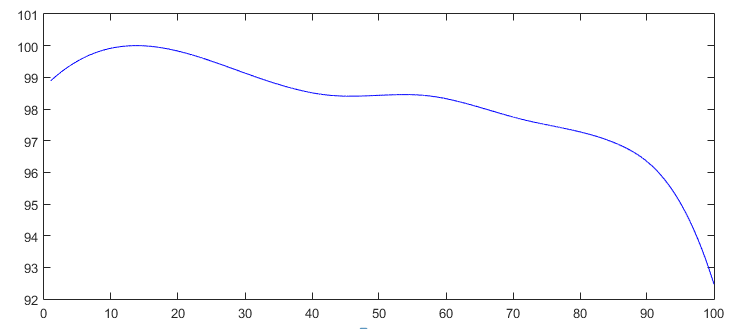


Рисунок 31 – Зависимость качества решения от выбора параметра

Значения, отображенные на графике, получены экспериментально при поиске атомных структур размерностью 20-35 атомов без использования начальной конфигурации. Хотя полученные результаты зависят также от размера задачи и прочих параметров алгоритма, выбор значения параметра в интервале можно использовать в качестве рекомендации при проведении вычислительных экспериментов.

# Заключение

В процессе работы были выполнены все поставленные задачи:

1. Рассмотрены особенности формирования атомных структур.
2. Сделан обзор существующих методов оптимизации.
3. Предложена методика поиска глобально-оптимальных конформаций кластеров Морса на основе использования заранее сформированной решетки атомов, модификации алгоритма Стронгина, алгоритма развития, процедур локальной оптимизации.
4. Разработана программная реализация алгоритма глобальной оптимизации атомных структур.
5. Проведены вычислительные эксперименты.

Результаты экспериментальных исследований с помощью разработанного программного приложения подтвердили на практике применимость предложенного алгоритма для решения поставленной задачи. Были найдены новые глобальные оптимумы кластеров Морса, отсутствующие в известных базах данных.

Практически все существующие методы оптимизации атомных кластеров пытаются осуществить пространственную оптимизацию напрямую, однако в предложенной методике используется оригинальный подход, позволяющий существенно упростить вычислительную задачу, сведя ее к оптимизации функции одной переменной. Такой подход в совокупности с используемыми алгоритмами и эвристиками позволяет решать оптимизационные задачи огромных (600 переменных и более) размерностей, имея при этом скромные вычислительные ресурсы.

Сложность и комплексность решаемых задач определяют актуальность данной работы, ее новизну и оригинальность.

Помимо академического интереса математиков, использующих задачу оптимизации кластеров Морса как полигон для испытания методов оптимизации, опыт практических расчетов стационарных точек на потенциальных поверхностях сложной структуры может быть использован во многих других областях. Особенно активно это научное направление развивается в последнее десятилетие в связи с быстрым прогрессом в области нанотехнологий. Например, в химии – для исследования структуры биомолекул, моделирования новых биологически активных соединений, в том числе для целей создания лекарственных препаратов.

Результаты работы, приведённые в диссертации, были опубликованы в сборнике трудов международной конференции «Инновационные исследования: проблемы внедрения результатов и направления развития».

# Список использованных источников

1. Коварцев А.Н. Эволюционный детерминированный алгоритм глобальной оптимизации атомных кластеров Морса // Компьютерная оптика. 2015. №39. Т. 2. С. 234-240.
2. Levinthal C. How to Fold Graciously // Mossbauer Spectroscopy in Biological Systems: Proceedings of a meeting held at Allerton House. 1969.   
   P. 22-24.
3. Lee J., Pillardy J., Czaplewski C., Arnautova,Y.A., Ripoll D.R., Liwo A., Gibson K.D., Wawak R.J., Scheraga H.A. Efficient parallel algorithms in global optimization of potential energy functions for peptides, proteins, and crystals // Computer Physics Communications. 2000. V. 128. P. 399-411.
4. Ripoll D.R., Liwo A., Scheraga H.A. Global optimization in protein folding // Encyclopedia of Optimization. 2009. P. 1392-1411.
5. Klepeis J. L.; Ierapetritou M. G.; Floudas C. A. Protein Folding and Peptide Docking: A Molecular Modeling and Global Optimization Approach // Computers and Chemical Engineering. 1998. V. 22. P. 3-10.
6. Leary R.H. Global optima of Lennard–Jones clusters // Journal of Global Optimization. 1997. P. 35-53.
7. Сергеев Я.Д., Квасов Д.Е. Краткое введение в теорию липшицевой глобальной оптимизации. Нижний Новгород: Изд-во ННГУ, 2016. 48 с.
8. Wales D.J., Scheraga H.A. Global optimization of clusters, crystals, and biomolecules // Science. 1999. V. 285. P. 1368-1372.
9. Leary R.H. Global optimization on funneling landscapes // Journal of Global Optimization. 2000. V. 18. P. 367-383.
10. Doye J. P. K., Wales D. J. Structural consequences of the range of the interatomic potential: A menagerie of clusters // Journal of Chemical Society. 1997. V. 3. P. 4233-4244.
11. Radford S. E., Dobson C. M., Evans P. A. The folding of hen lysozyme involves partially structured intermediates and multiple pathways // Nature. 1992. V. 358. P. 302-307.
12. Abkevich V. I., Gutin A. M., Shakhnovich E. I. Free energy landscape for protein folding kinetics: Intermediates, traps and multiple pathways in theory and lattice model simulations // J. Chem. Phys. 1994. V. 101. P. 6052-6062.
13. Levy Y., Becker O. M. Effect of conformational constraints on the complex potential energy surfaces // Phys. Rev. Lett. 1998. V. 81. P. 1126.
14. Miller M. A., Wales D. J., Energy landscape of a model protein // J. Chem. Phys. 1999. V. 111. P. 6610-6616.
15. The Cambridge Energy Landscape Database [Электронный ресурс] // Wales group home page: [сайт]. URL: http://www-wales.ch.cam.ac.uk/CCD.html (дата обращения: 01.05.2017).
16. Heiles S., Johnston R. L. Global optimization of clusters using electronic structure methods // Quantum Chemistry. 2013. V. 113. P. 2091-2109.
17. Батурин В. С. Структура, стабильность и термодинамические свойства нанокластеров. Липецк: ЛГТУ, 2002. 103 с.
18. Deaven D.M. Molecular Geometry Optimization with a Genetic Algorithm // Physical Review Letters. 1995. V.75. P. 288.
19. Muller S.D., Schraudolph N.N., Koumoutsakos P., Evolutionary and Gradient-Based Algorithms for Lennard-Jones Cluster Optimization // Genetic and Evolutionary Computation Conference. 2003. P. 160-165.
20. Roberts C., Johnston R.L., Wilson N.T. A genetic algorithm for the structural optimization of Morse clusters // Theor. Chem. Acc. 2000. V. 104.   
    P.123-130.
21. Storn R., Price K. Differential Evolution. A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces // Journal of Global Optimization. 1997. V. 11. P. 341-359.
22. Moloi N.P., Ali M.M. An Iterative Global Optimization Algorithm for Potential Energy Minimization // Journal of Computational Optimization and Applications. 2005. V. 30. P. 1-14.
23. Byrd R., Hansen S., Nocedal J., Singer Y. Stochastic Quasi-Newton Method for Large-Scale Optimization // Math. 2014. P. 1-22.
24. Wales, D.J., Oakley, M.T. Energy landscape and global optimization for a frustrated model protein // Journal of Physical Chemistry. 2011. V. 115. No. 39. P. 11525-11529.
25. Wales, D.J., Oakley, M.T. Symmetrisation schemes for global optimisation of atomic clusters // Physical Chemistry Chemical Physics. 2013. V. 15. No. 11. P. 3965-3976.
26. Locatelli M., Schoen F. Global Optimization: Theory, Algorithms and Applications // SIAM. 2013. P. 433.
27. Locatelli M., Schoen F. Local search based heuristics for global optimization: Atomic clusters and beyond // European Journal of Operational Research. 2012. V. 222. No. 1. P. 1-9.
28. Locatelli M., Addis B. A global optimization method for the design of space trajectories // Computational Optimization and Applications. 2011. V. 48. No. 3.   
    P. 635-652.
29. Conway B.A. Spacecraft Trajectory Optimization // Cambridge University Press. 2010. P. 298.
30. Marques J.M.C., Pereira F. An evolutionary algorithm for global minimum search of binary atomic clusters // Chemical Physics Letters. 2010. V. 485.   
    P. 211-216.
31. Locatelli M., Cassioli A. Global optimization of binary Lennard-Jones clusters // Optimization Methods and Software. 2009. V. 24. No. 4-5. P. 819-835.
32. Liuzzi G., Lucidi S., Piccialli V. A DIRECT-based approach exploiting local minimizations for the solution of large-scale global optimization problems // Computational Optimization and Applications. 2010. V. 45. P. 353-375.
33. Locatelli M., Grosso A. Solving molecular distance geometry problems by global optimization algorithms // Computational Optimization and Applications. 2009. V. 43. P. 23-37.
34. Larsson H.R., van Duin A.C.T., Hartke B. Global optimization of parameters in the reactive force field ReaxFF for SiOH // J.Comput.Chem. 2013.   
    V. 34. P. 2178-2189.
35. Lourenco N., Pereira F.B. Pso-cgo: a particle swarm algorithm for cluster geometry optimization // International Journal of Natural Computing Research. 2011. V. 2. No. 1. P. 1-20.
36. Cheng L., Feng Y., Yang J., Yang J. Funnel hopping: searching the cluster potential energy surface over the funnels // Journal of Chemical Physics. 2009. V. 130. No. 21. P. 2141-2186.
37. Wales D.J., Oakley M.T. Symmetrisation schemes for global optimisation of atomic clusters // Physical Chemistry Chemical Physics. 2013. V. 15. No. 11. P. 3965-3976.
38. Wu X, Cai W., Shao X. A dynamic lattice searching method with rotation operation for optimization of large clusters // Chem Phys. 2009. V. 363. P. 72-77.
39. Logsdail A.J., Li Z., Johnston R. L. Development and Optimisation of a Novel Genetic Algorithm for Identifying Nanoclusters from Scanning Transmission Electron Microscopy Images // J. Comput. Chem. 2012. V. 33. P. 391-400.
40. Shao N. Probing the structural evolution of medium-sized gold clusters: Au n- (n = 27-35) // Journal of the American Chemical Society. 2010. V. 132. No. 18. P.6596-6605.
41. Коварцев А.Н. Геометрически обоснованный метод формирования атомных кластеров Морса больших размеров // Компьютерная оптика. 2017. №41. Т. 1. C. 118-125.
42. Стронгин Р.Г., Гергель В. П., Баркалов К. А. Параллельные методы решения задач глобальной оптимизации // Изв. вузов. Приборостроение. 2009. №52. Т. 10. С. 25-33.
43. Hoare M.R., Pal P. Physical cluster mechanism: statistic and energy surface for monotonic systems // Advances in Physics. 1971. V. 20. P. 161-196.
44. Liu D.C., Nocedal J. On the limited memory BFGS method for large scale optimization // Mathematical Programming. 1989. V. 45. P. 503-528.

# Приложение А Листинг исходного кода Java

**public** **class** **ClusterMath** {

**private** **static** String startConf;

**private** **static** ArrayList<Integer> indexes;

**private** **static** ArrayList<Vertex> vertices;

**private** **static** Map<String, Conformation> optCache = **new** HashMap<>();

**public** **static** **void** **init**(String startConf, ArrayList<Vertex> vertices, ArrayList<Integer> indexes) {

ClusterMath.startConf = startConf;

ClusterMath.vertices = vertices;

ClusterMath.indexes = indexes;

optCache.clear();

}

**public** **static** Conformation **calcWithStartConf**(String stronginBits, **boolean** isLocalOpt) {

**return** **calcE**(getFullBits(stronginBits), isLocalOpt);

}

**public** **static** Conformation **calc**(String fullBits, **boolean** isLocalOpt) {

**return** **calcE**(fullBits, isLocalOpt);

}

**public** **static** **int** **calcAdjacentNunberWithStartConf**(String stronginBits, **int** atomIndex) {

**if** (atomIndex >= stronginBits.length()) {

**throw** **new** **IllegalArgumentException**("Invalid bits size");

}

String fullBits = getFullBits(stronginBits);

Vertex vertex = vertices.get(indexes.get(atomIndex));

**int** count = **0**;

**for** (**int** i = **0**; i < fullBits.length(); i++) {

**if** ((i != indexes.get(atomIndex)) && (fullBits.charAt(i) == '1')) {

**if** (vertex.distanceTo(vertices.get(i)) < Configuration.get().getDISTANCE\_MIN()) {

count++;

}

}

}

**return** count;

}

**private** **static** String **getFullBits**(String stronginBits) {

**if** (stronginBits == **null**) {

**return** startConf;

}

**if** (indexes.size() != stronginBits.length()) {

**throw** **new** **IllegalArgumentException**("Invalid bits size");

}

StringBuilder sb = **new** StringBuilder(startConf);

**for** (**int** i = **0**; i < stronginBits.length(); i++) {

sb.setCharAt(indexes.get(i), stronginBits.charAt(i));

}

**return** sb.toString();

}

**private** **static** Conformation **calcE**(String fullBits, **boolean** isLocalOpt) {

**if** (fullBits.length() != vertices.size()) {

**throw** **new** **IllegalArgumentException**("Invalid bits size");

}

String key = fullBits;

**if** (optCache.containsKey(key)) {

**return** optCache.get(key);

}

ArrayList<Vertex> verticesConf = **new** ArrayList<>();

**for** (**int** i = **0**; i < fullBits.length(); i++) {

**if** (fullBits.charAt(i) == '1') {

verticesConf.add(**new** Vertex(vertices.get(i)));

}

}

Conformation conf;

**if** (isLocalOpt) {

conf = Lbfgs.minimize(**new** Conformation(**new** Bits(fullBits), verticesConf, **0**));

} **else** {

**double** energy = getEnergy(verticesConf);

conf = **new** Conformation(**new** Bits(fullBits), verticesConf, energy);

}

**if** (isLocalOpt && !optCache.containsKey(key)) {

optCache.put(key, conf);

}

**return** conf;

}

**public** **static** **double** **getEnergy**(ArrayList<Vertex> verticesConf) {

**double** r;

**double** energy = **0**;

**for** (**int** i = **0**; i < verticesConf.size() - **1**; i++) {

**for** (**int** j = i + **1**; j < verticesConf.size(); j++) {

r = verticesConf.get(i).distanceTo(verticesConf.get(j));

energy += Math.exp(Configuration.get().getRO() \* (**1** - r)) \* (Math.exp(Configuration.get().getRO() \* (**1** - r)) - **2**);

}

}

**return** energy;

}

}

**public** **class** **GrowthAlg** {

**public** **static** Conformation **buildBestConf**(Bits bits, **int** n, **int** iterations) {

Conformation conf = **null**;

Map<String, Conformation> results = **new** HashMap<>();

buildConfRecursive(bits.getBites().toString(), n, iterations, results);

**for** (Conformation conformation : results.values()) {

**if** ((conf == **null**) || (conf.getEnergy() > conformation.getEnergy())) {

conf = conformation;

}

}

**return** conf;

}

**private** **static** **void** **buildConfRecursive**(**final** String bits, **final** **int** n, **int** iterations, **final** Map<String, Conformation> results) {

**if** (results.containsKey(bits)) {

**return**;

}

**int** currSize = **0**;

**for** (**int** i = **0**; i < bits.length(); i++) {

**if** (bits.charAt(i) == '1') currSize++;

}

**if** (currSize == n) {

results.put(bits, ClusterMath.calcWithStartConf(bits, **false**));

**return**;

}

ArrayList<String> temp = findBestAdjacentAtom(bits);

**int** itersMin = **1**;

**if** (iterations > **0**) {

itersMin = (temp.size() < iterations) ? temp.size() : iterations;

iterations /= itersMin;

}

**for** (**int** i = **0**; i < itersMin; i++) {

buildConfRecursive(temp.get(i), n, iterations, results);

}

}

**public** **static** ArrayList<String> **findBestAdjacentAtom**(String startBits) {

ArrayList<String> list = **new** ArrayList<>();

ArrayList<String> adjacentList = **new** ArrayList<>();

**int** adjacentMax = **0**;

StringBuilder sb = **new** StringBuilder(startBits);

**for** (**int** i = **0**; i < startBits.length(); i++) {

**if** (startBits.charAt(i) == '0') {

**int** temp = ClusterMath.calcAdjacentNunberWithStartConf(startBits, i);

**if** (temp > adjacentMax) {

adjacentMax = temp;

adjacentList.clear();

}

**if** (temp == adjacentMax) {

sb.setCharAt(i, '1');

adjacentList.add(sb.toString());

sb.setCharAt(i, '0');

}

}

}

Map<String, Conformation> conformations = **new** HashMap<>();

Conformation conf;

**double** minEnergy = **0**;

**for** (String bits : adjacentList) {

conf = ClusterMath.calcWithStartConf(bits, **false**);

**if** (conf.getEnergy() < minEnergy) {

minEnergy = conf.getEnergy();

}

conformations.put(bits, conf);

}

**for** (Map.Entry<String, Conformation> entry : conformations.entrySet()) {

**if** (((Math.abs(entry.getValue().getEnergy() - minEnergy)) / minEnergy) < Configuration.get().getTOP\_MAX\_ENERGY\_DELTA()) {

list.add(entry.getKey());

}

}

**return** list;

}

}

**public** **class** **InfSupFinder** {

**public** **static** StringBuilder[] **findInfSup**(String x, **int** Natom, **int** Mcl) {

StringBuilder sbX = **new** StringBuilder(x);

**int** N = **0**;

**int** M = Natom;

**for** (**int** i = **0**; i < sbX.length(); i++) {

**if** (sbX.charAt(i) == '1') {

N++;

}

}

StringBuilder zeros = **new** Bits(Mcl).getBites();

**if** (N == M) {

**return** **new** StringBuilder[]{**new** StringBuilder(sbX), **new** StringBuilder(sbX)};

}

StringBuilder sbInf;

StringBuilder sbSup;

**if** (N > M) {

**int** NM = N - M;

//xInf

sbInf = **new** StringBuilder(zeros);

**int** k = **0**;

**for** (**int** i = Mcl - **1**; i >= **0**; i--) {

**if** ((sbX.charAt(i) == '1') && (k != NM)) {

sbInf.setCharAt(i, '0');

k++;

} **else** {

sbInf.setCharAt(i, sbX.charAt(i));

}

}

//xSup

sbSup = **new** StringBuilder(sbX);

k = N - M + **1**;

**int** j = **0**;

**int** i;

**for** (i = Mcl - **1**; i >= **0**; i--) {

**if** (sbSup.charAt(i) == '1') {

j++;

} **else** {

**if** (j >= k) {

**break**;

}

}

}

sbSup.setCharAt(i, '1');

j = **0**;

**for** (**int** z = i + **1**; z < Mcl; z++) {

**if** (j == k) {

**break**;

}

**if** (sbSup.charAt(z) == '1') {

sbSup.setCharAt(z, '0');

j++;

}

}

toRight(sbSup, Mcl, i, '1');

} **else** {

**int** MN = M - N;

//xSup

sbSup = **new** StringBuilder(zeros);

**int** k = **0**;

**for** (**int** i = Mcl - **1**; i >= **0**; i--) {

**if** ((sbX.charAt(i) == '0') && (k != MN)) {

sbSup.setCharAt(i, '1');

k++;

} **else** {

sbSup.setCharAt(i, sbX.charAt(i));

}

}

//xInf

k = M - N + **1**;

**int** j = **0**;

sbInf = **new** StringBuilder(sbX);

**int** np = Mcl;

**for** (**int** i = Mcl - **1**; i >= **0**; i--) {

**if** (sbInf.charAt(i) == '1') {

np = i; //позиция в векторе

}

}

**int** i;

**for** (i = Mcl - **1**; i >= np; i--) {

**if** (sbInf.charAt(i) == '0') {

j++;

} **else** {

**if** (j >= k) {

**break**;

}

}

}

**if** (j < k) {

sbInf = zeros; //TODO

} **else** {

sbInf.setCharAt(i, '0');

j = **0**;

**for** (**int** z = i + **1**; z < Mcl; z++) {

**if** (j == k) {

**break**;

}

**if** (sbInf.charAt(z) == '0') {

sbInf.setCharAt(z, '1');

j++;

}

}

toRight(sbInf, Mcl, i, '0');

}

}

**return** **new** StringBuilder[]{sbSup, sbInf};

}

**private** **static** **void** **toRight**(StringBuilder v, **int** Mcl, **int** index, **char** bit) {

**int** j = **0**;

**for** (**int** i = index + **1**; i < Mcl; i++) {

**if** (v.charAt(i) != bit) {

j++;

}

}

**for** (**int** i = index + **1**; i < Mcl; i++) {

**if** (j > **0**) {

v.setCharAt(i, (bit == '1') ? '0' : '1');

j--;

} **else** {

v.setCharAt(i, bit);

}

}

}

}

**public** **class** **Lbfgs** {

**public** **static** Conformation **minimize**(Conformation conformation) {

ArrayList<Vertex> vertices = conformation.getVertices();

**double**[] input = **new** **double**[vertices.size() \* **3**];

**int** j = **0**;

**for** (**int** i = **0**; i < vertices.size(); i++) {

input[j++] = vertices.get(i).getX();

input[j++] = vertices.get(i).getY();

input[j++] = vertices.get(i).getZ();

}

LbfgsConstant.LBFGS\_Param params = **new** LbfgsConstant.LBFGS\_Param(com.github.lbfgs4j.liblbfgs.Lbfgs.defaultParams());

params.epsilon = Configuration.get().getLO\_EPS();

params.max\_iterations = Configuration.get().getLO\_MAX\_ITERATIONS();

MorseFunction morseFunction = **new** MorseFunction(input.length);

LbfgsMinimizer minimizer = **new** LbfgsMinimizer(params, **false**);

**double**[] output = minimizer.minimize(morseFunction, input);

**double** min = morseFunction.valueAt(output);

ArrayList<Vertex> verticesOpt = arrayToCollection(output);

**return** **new** **Conformation**(conformation.getBits(), verticesOpt, min);

}

**private** **static** ArrayList<Vertex> **arrayToCollection**(**double**[] x) {

ArrayList<Vertex> vertices = **new** ArrayList<>(x.length / **3**);

**int** j = **0**;

**for** (**int** i = **0**; i < x.length / **3**; i++) {

vertices.add(**new** Vertex(x[j++], x[j++], x[j++]));

}

**return** vertices;

}

**public** **static** **class** **MorseFunction** **implements** Function {

**private** **int** size;

**public** **MorseFunction**(**int** size) {

**this**.size = size;

}

**@Override**

**public** **int** **getDimension**() {

**return** size;

}

**@Override**

**public** **double** **valueAt**(**double**[] x) {

**return** ClusterMath.getEnergy(arrayToCollection(x));

}

**@Override**

**public** **double**[] **gradientAt**(**double**[] x) {

**double**[] grad = **new** **double**[x.length];

ArrayList<Vertex> vertices = arrayToCollection(x);

**for** (**int** k = **0**; k < vertices.size(); k++) {

**for** (**int** i = **0**; i < vertices.size(); i++) {

**if** (k != i) {

Vertex xk = **new** Vertex(vertices.get(k).getX(), vertices.get(k).getY(), vertices.get(k).getZ());

Vertex xi = **new** Vertex(vertices.get(i).getX(), vertices.get(i).getY(), vertices.get(i).getZ());

**double** rki = xk.distanceTo(xi);

**double**[] aki = **new** **double**[**3**];

aki[**0**] = (xk.getX() - xi.getX()) / rki;

aki[**1**] = (xk.getY() - xi.getY()) / rki;

aki[**2**] = (xk.getZ() - xi.getZ()) / rki;

**double** fki = **2** \* Configuration.get().getRO() \* (Math.exp(Configuration.get().getRO() \* (**1** - rki)) - Math.exp(**2** \* Configuration.get().getRO() \* (**1** - rki)));

grad[**3** \* k + **0**] += fki \* aki[**0**];

grad[**3** \* k + **1**] += fki \* aki[**1**];

grad[**3** \* k + **2**] += fki \* aki[**2**];

}

}

}

**return** grad;

}

}

}

**public** **class** **Efficiency** {

**private** MinsRepository rep;

**private** Bits x;

**private** Bits xInf;

**private** Bits xSup;

**private** **double** z;

**public** **Efficiency**(MinsRepository repository, Bits x) {

**this**.rep = repository;

**this**.x = x;

updateData();

}

**private** **void** **updateData**() {

**int** N = Configuration.get().getSTRONGIN\_N();

**int** M = Configuration.get().getSTRONGIN\_M();

**int** K = Configuration.get().getSTRONGIN\_K();

StringBuilder[] res = InfSupFinder.findInfSup(x.getBites().toString(), N, M);

xSup = **new** Bits(res[**0**]);

xInf = **new** Bits(res[**1**]);

**int** k = **0**;

StringBuilder sb = **new** StringBuilder();

**for** (**int** i = **0**; i < M; i++) {

**if** ((k < K) && (xInf.get(i) == xSup.get(i)) && (xInf.get(i) == '1')) {

k++;

sb.append('1');

} **else** {

sb.append('0');

}

}

Conformation conf;

**if** (k == K) {

conf = findBestConf(**new** Bits(sb), Configuration.get().getINF\_SUP\_ITERATIONS());

z = conf.getEnergy();

} **else** {

Conformation confInf = findBestConf(getFirstAtoms(xInf, K), Configuration.get().getINF\_ITERATIONS());

Conformation confSup = findBestConf(getFirstAtoms(xSup, K), Configuration.get().getSUP\_ITERATIONS());

conf = (confInf.getEnergy() < confSup.getEnergy()) ? confInf : confSup;

BigDecimal t = **new** BigDecimal(confSup.getEnergy() - confInf.getEnergy()).setScale(Configuration.get().getBIG\_DECIMAL\_SCALE(), BigDecimal.ROUND\_HALF\_UP);

z = confInf.getEnergy() + t.multiply(**new** BigDecimal(x.getNumber().subtract(xInf.getNumber()).divide(xSup.getNumber().subtract(xInf.getNumber()))).setScale(Configuration.get().getBIG\_DECIMAL\_SCALE())).doubleValue(); //TODO not log?

}

rep.tryAddConf(conf);

}

//call GrowthAlg or use cache

**private** Conformation **findBestConf**(Bits bits, **int** iterations) {

String key = bits.getBites().toString();

**if** (!rep.getCache().containsKey(key)) {

rep.getCache().put(key, GrowthAlg.buildBestConf(bits, Configuration.get().getSTRONGIN\_N(), iterations));

}

**return** rep.getCache().get(key);

}

//get subBits with first k ones

**private** Bits **getFirstAtoms**(Bits bits, **int** K) {

StringBuilder sb = **new** StringBuilder();

**int** k = **0**;

**for** (**int** i = **0**; i < bits.getSize(); i++) {

sb.append(k < K ? bits.get(i) : '0');

**if** (bits.get(i) == '1') {

k++;

}

}

**return** **new** **Bits**(sb);

}

**public** Bits **getX**() {

**return** x;

}

**public** Bits **getXInf**() {

**return** xInf;

}

**public** Bits **getXSup**() {

**return** xSup;

}

**public** **double** **getZ**() {

**return** z;

}

}

**public** **class** **ExecutorService** {

**private** **static** PrintStream log;

**private** **static** String DEFAULT\_CONFIG = "config.json";

**private** **static** String DEFAULT\_LOG = "log.txt";

**private** **static** String DEFAULT\_INPUT = "input.txt";

**private** **static** String DEFAULT\_OUTPUT = "output";

**public** **static** PrintStream **initLog**() **throws** FileNotFoundException {

**if** (log == **null**) {

String userDir = System.getProperty("user.dir") + File.separator;

String logPath = userDir + DEFAULT\_LOG;

log = **new** PrintStream(**new** File(logPath));

}

**return** log;

}

**public** **static** Config **init**(String[] args) **throws** IOException {

String userDir = System.getProperty("user.dir") + File.separator;

String configPath = userDir + DEFAULT\_CONFIG;

String inputPath = userDir + DEFAULT\_INPUT;

String outputPath = userDir + DEFAULT\_OUTPUT;

**if** (args.length > **0**) {

configPath = args[**0**].replace("\\", "\\\\");

}

log = **null**;

initLog();

Configuration.setupConfig(**new** BufferedReader(**new** FileReader(configPath)), inputPath, outputPath);

**return** Configuration.get();

}

**public** **static** **void** **process**(StronginTask.ProgressCallBack progressCallBack, OnFinishCallBack finishCallback) **throws** ExecutionException, InterruptedException, FileNotFoundException {

**if** (Configuration.get() == **null**) {

**throw** **new** **RuntimeException**("ExecutorService#init function is not be called");

}

Map<String, Conformation> output = **new** HashMap<>();

**long** time = System.currentTimeMillis();

**if** (Configuration.get().getM() - Configuration.get().getSTRONGIN\_M() == Configuration.get().getN()) {

progressCallBack.onFinish(**10**);

output.put(Configuration.get().getSTART\_CONF(), ClusterMath.calcWithStartConf(**null**, **true**));

ExecutorService.log.close();

finishCallback.onFinish(saveResults(output, progressCallBack), System.currentTimeMillis() - time);

**return**;

}

//interval

StringBuilder stronginOnes = **new** StringBuilder();

**for** (**int** i = **0**; i < Configuration.get().getSTRONGIN\_N(); i++) {

stronginOnes.append('1');

}

StringBuilder stronginZeros = **new** StringBuilder();

**for** (**int** i = **0**; i < Configuration.get().getSTRONGIN\_M() - Configuration.get().getSTRONGIN\_N(); i++) {

stronginZeros.append('0');

}

Bits a = **new** Bits(**new** StringBuilder(stronginZeros).append(stronginOnes));

Bits b = **new** Bits(**new** StringBuilder(stronginOnes).append(stronginZeros));

java.util.concurrent.ExecutorService executor = Executors.newFixedThreadPool(Configuration.get().getTHREADS\_COUNT());

ArrayList<StronginTask> tasks = **new** ArrayList<>(Configuration.get().getTHREADS\_COUNT());

ArrayList<BigInteger> points = **new** ArrayList<>(Configuration.get().getTHREADS\_COUNT() + **1**);

points.add(a.getNumber());

**for** (**int** i = **0**; i < Configuration.get().getTHREADS\_COUNT() - **1**; i++) {

points.add(b.getNumber().subtract(a.getNumber()).divide(BigInteger.valueOf(Configuration.get().getTHREADS\_COUNT())).multiply(BigInteger.valueOf(i + **1**)).add(a.getNumber()));

}

points.add(b.getNumber());

**for** (**int** i = **0**; i < Configuration.get().getTHREADS\_COUNT(); i++) {

Strongin strongin = **new** Strongin(**new** Bits(Configuration.get().getSTRONGIN\_M(), points.get(i)), **new** Bits(Configuration.get().getSTRONGIN\_M(), points.get(i + **1**)), Configuration.get().getSTRONGIN\_ITERATIONS(), Configuration.get().getSTRONGIN\_REPOSITORY\_SIZE());

StronginTask.ProgressCallBack progressCallBack1 = progressCallBack.clone();

progressCallBack1.setId(i);

StronginTask task = **new** StronginTask(strongin, progressCallBack1);

tasks.add(task);

executor.execute(task);

}

**for** (StronginTask task : tasks) {

Strongin strongin = task.get();

String key;

**for** (Conformation variant : strongin.getRep().getMins()) {

key = variant.getBits().getBites().toString();

**if** (!output.containsKey(key)) {

output.put(key, variant);

} **else** {

**if** (variant.getEnergy() < output.get(key).getEnergy()) {

output.put(key, variant);

}

}

}

logIntervals(strongin);

task.getProgressCallBack().onProgress(**100**);

}

progressCallBack.onFinish(**10**);

executor.shutdown();

ExecutorService.log.close();

finishCallback.onFinish(saveResults(output, progressCallBack), System.currentTimeMillis() - time);

}

**private** **static** List<Conformation> **saveResults**(Map<String, Conformation> map, StronginTask.ProgressCallBack progressCallBack) **throws** FileNotFoundException {

List<Conformation> output = **new** ArrayList<>();

File directory = **new** File(Configuration.get().getOUTPUT\_FOLDERNAME());

**if** (!directory.exists()) {

directory.mkdir();

}

**for** (Conformation conformation : map.values()) {

output.add(ClusterMath.calc(conformation.getBits().getBites().toString(), **true**));

}

progressCallBack.onFinish(**50**);

output.sort(**new** Comparator<Conformation>() {

**@Override**

**public** **int** **compare**(Conformation left, Conformation right) {

**if** (left.getEnergy() == right.getEnergy()) {

**return** **0**;

}

**return** left.getEnergy() > right.getEnergy() ? **1** : -**1**;

}

});

List<Conformation> mins = output.subList(**0**, Math.min(output.size(), Configuration.get().getMINS\_COUNT()));

**int** i = **1**;

**for** (Conformation conformation : mins) {

saveConformation(directory, conformation, "[" + (i++) + "]");

progressCallBack.onFinish(**50** + **50** \* ((i - **1**) / mins.size()));

}

**return** mins;

}

**private** **static** **void** **logIntervals**(Strongin strongin) **throws** FileNotFoundException {

initLog();

strongin.getIntervals().sort(**new** Comparator<Interval>() {

**@Override**

**public** **int** **compare**(Interval left, Interval right) {

**return** left.getA().getNumber().compareTo(right.getA().getNumber());

}

});

ExecutorService.log.println("THREAD");

**for** (Interval interval : strongin.getIntervals()) {

ExecutorService.log.println(interval);

}

ExecutorService.log.flush();

}

**private** **static** **void** **saveConformation**(File dir, Conformation conf, String prefixName) **throws** FileNotFoundException {

String name = prefixName + " N" + Configuration.get().getN() + " M" + Configuration.get().getM() + " " + conf.getEnergy();

PrintWriter out = **new** PrintWriter(dir.getAbsolutePath() + File.separator + name.replace(".", ",") + ".txt");

out.print(conf);

out.flush();

out.close();

}

**public** **static** **void** **logError**(Exception e) {

**try** {

initLog();

} **catch** (FileNotFoundException e1) {

e1.printStackTrace();

}

**if** (ExecutorService.log != **null**) {

e.printStackTrace(ExecutorService.log);

ExecutorService.log.close();

}

}

**public** **interface** **OnFinishCallBack** {

**void** **onFinish**(List<Conformation> results, **long** milliseconds);

}

}

**public** **class** **Strongin** {

**private** **static** **final** **long** m = **150**;

**public** **static** **final** BigDecimal log2 = BigDecimalMath.log(**new** BigDecimal(**2**).setScale(Configuration.get().getBIG\_DECIMAL\_SCALE()));

**private** ArrayList<Interval> intervals;

**private** MinsRepository rep;

**private** **int** iterations;

**private** **int** sizeMins;

**private** Bits b;

**private** Bits a;

**private** StronginTask.ProgressCallBack progressCallBack;

**private** **Strongin**() {

}

**public** **Strongin**(Bits a, Bits b, **final** **int** iterations, **int** sizeMins) {

**this**.a = a;

**this**.b = b;

**this**.iterations = iterations;

**this**.sizeMins = sizeMins;

}

**public** **static** **double** **calcF**(BigInteger a, BigInteger b, **double** zA, **double** zB) {

**double** logA = BigDecimalMath.log(**new** BigDecimal(a).setScale(Configuration.get().getBIG\_DECIMAL\_SCALE())).divide(log2, RoundingMode.HALF\_UP).doubleValue();

**double** logB = BigDecimalMath.log(**new** BigDecimal(b).setScale(Configuration.get().getBIG\_DECIMAL\_SCALE())).divide(log2, RoundingMode.HALF\_UP).doubleValue();

**return** m \* (logB - logA) + (Math.pow(zB - zA, **2**) / (m \* (logB - logA))) - **2** \* (zA + zB);

}

**public** MinsRepository **solve**(StronginTask.ProgressCallBack progressCallBack) {

**if** ((a == **null**) || (b == **null**) || (iterations <= **0**) || (sizeMins <= **0**)) {

**throw** **new** **IllegalArgumentException**("Strongin params not exist");

}

**this**.progressCallBack = progressCallBack;

**return** **solve**(a, b, iterations, sizeMins);

}

**private** MinsRepository **solve**(Bits a, Bits b, **final** **int** iterations, **int** sizeMins) {

rep = **new** MinsRepository(sizeMins);

intervals = **new** ArrayList<>(iterations);

intervals.add(**new** Interval(rep, a, b));

**if** (a.getNumber().compareTo(b.getNumber()) == **0**) {

rep.tryAddConf(ClusterMath.calcWithStartConf(a.getBites().toString(), **true**));

**return** rep;

}

**int** ind = **0**;

Interval interval = **null**;

Efficiency zX = **null**;

Bits bitsX = **null**;

**boolean** isUsed = **false**;

Bits cacheB = **null**;

Efficiency cacheZB = **null**;

**int** progressDelta = **2** \* iterations / **100**;

**if** (progressDelta <= **0**) {

progressDelta = **2**;

}

**for** (**int** i = **0**; i < iterations; i++) {

**if** (i % progressDelta == **0**) {

**if** (progressCallBack != **null**) {

progressCallBack.onProgress(Math.min(i \* **100** / iterations, **99**));

}

}

interval = intervals.get(ind);

**double** logA = BigDecimalMath.log(**new** BigDecimal(interval.getA().getNumber()).setScale(Configuration.get().getBIG\_DECIMAL\_SCALE())).divide(log2, RoundingMode.HALF\_UP).doubleValue();

**double** logB = BigDecimalMath.log(**new** BigDecimal(interval.getB().getNumber()).setScale(Configuration.get().getBIG\_DECIMAL\_SCALE())).divide(log2, RoundingMode.HALF\_UP).doubleValue();

BigInteger x = **new** BigDecimal(interval.getB().getNumber().add(interval.getA().getNumber())).setScale(Configuration.get().getBIG\_DECIMAL\_SCALE()).divide(**new** BigDecimal(**2**)).setScale(**0**, BigDecimal.ROUND\_HALF\_UP).toBigInteger(); //Math.ceil(Math.pow(2, (logA + logB) / 2.0));

bitsX = **new** Bits(Configuration.get().getSTRONGIN\_M(), x);

zX = **new** Efficiency(rep, bitsX);

isUsed = **false**;

**double** logSize = BigDecimalMath.log(**new** BigDecimal(Configuration.get().getSTRONGIN\_EPS()).setScale(Configuration.get().getBIG\_DECIMAL\_SCALE()).divide(**new** BigDecimal(interval.getA().getNumber()), RoundingMode.HALF\_UP).add(BigDecimal.ONE)).divide(log2, RoundingMode.HALF\_UP).doubleValue();

**if** ((logB - logA) / **2.0** < logSize) { //todo check

intervals.remove(ind);

} **else** {

cacheB = interval.getB();

cacheZB = interval.getZB();

**if** (!(zX.getXInf().getNumber().compareTo(interval.getA().getNumber()) < **0**) && !(zX.getXInf().getNumber().compareTo(interval.getB().getNumber()) > **0**)) {

interval.setB(bitsX, zX);

isUsed = **true**;

}

**if** (!(zX.getXSup().getNumber().compareTo(bitsX.getNumber()) < **0**) && !(zX.getXSup().getNumber().compareTo(cacheB.getNumber()) > **0**)) {

**if** (isUsed) {

intervals.add(**new** Interval(rep, bitsX, cacheB, zX, cacheZB));

} **else** {

interval.setA(bitsX, zX);

interval.setA(bitsX, zX);

isUsed = **true**;

}

}

**if** (!isUsed) {

intervals.remove(ind);

}

}

**if** (intervals.size() == **0**) {

**break**;

}

ind = **0**;

interval = intervals.get(ind);

**for** (**int** j = **0**; j < intervals.size(); j++) {

**if** (intervals.get(j).getF() > interval.getF()) {

ind = j;

}

}

}

**return** rep;

}

**public** ArrayList<Interval> **getIntervals**() {

**return** intervals;

}

**public** MinsRepository **getRep**() {

**return** rep;

}

}