

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Департамент программной инженерии

УДК 004.023

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия»,
профессор департамента программной
инженерии, к.т.н.

_____ В.В. Шилов

«___» _____ 2017 г.

Выпускная квалификационная работа

на тему Реализация и оценка качества ресурсно-эффективных алгоритмов для
метрической задачи коммивояжера

по направлению подготовки 09.03.04 «Программная инженерия»

Научный руководитель профессор, руководитель департамента программной инженерии, к.т.н. С.М. Авдошин	Выполнила студентка группы БПИ131 4 курса бакалавриата образовательной программы «Программная инженерия» Е.Н. Береснева
Подпись, Дата	Подпись, Дата

Москва 2017

Реферат

Отчет 113 с., 13 рис., 7 табл., 14 схем, 67 источников, 11 прил.

Ключевые слова: *метрическая задача коммивояжера, ресурсно-эффективный алгоритм, эвристический алгоритм, NP-трудная задача.*

Представлены результаты выпускной квалификационной работы «Реализация и оценка качества ресурсно-эффективных алгоритмов для метрической задачи коммивояжера», выполненной в соответствии с Приказом Национального исследовательского университета «Высшая школа экономики» № 2.3-02/0812-01 от 08.12.2016.

Объект исследования – метрическая задача коммивояжера.

Предмет исследования – эвристические алгоритмы решения метрической задачи коммивояжера.

Цель исследования – определить группу Парето-оптимальных алгоритмов решения метрической задачи коммивояжера по критериям времени работы и точности решения.

Задачи исследования:

- реализация существующих эвристических алгоритмов решения метрической задачи коммивояжера;
- разработка методики оценки качества алгоритмов;
- модификация эвристических алгоритмов решения метрической задачи коммивояжера для получения реализации с лучшими оценками качества;
- экспериментальная оценка качества реализованных алгоритмов.

Методы исследования:

- изучение монографических публикаций и статей;
- методы дискретной оптимизации;
- сравнительный анализ.

Научная новизна:

1. Предложены модификации эвристических алгоритмов метрической задачи коммивояжера, которые позволили улучшить временные оценки для алгоритмов NN, DENN, NA, NI, CI, FI, AI, NSI, DMST, DMST-M, CHR, 2-Opt, qCABC и повысить оценки точности для алгоритма DMST.
2. Найдены Парето-оптимальные алгоритмы решения метрической задачи коммивояжера.

Практическая значимость:

Создана программная реализация эвристических алгоритмов решения метрической задачи коммивояжера.

Программные реализации Парето-оптимальных алгоритмов могут быть использованы при решении задач маршрутизации в следующих областях:

- доставка писем и товаров;
- перевозка людей;
- управление спутниками, телескопами, микроскопами и лазерами;
- сборка генома;
- кластеризация массивов данных;
- рентгеноструктурный анализ;
- проектирование телекоммуникационных сетей;
- соединение ряда пунктов кольцевыми линиями энергопередач, газоснабжения;
- обработка деталей на станке;
- проектирование топологии сверх больших интегральных схем.

Результаты работы:

- изучены существующие эвристические алгоритмы решения метрической задачи коммивояжера;
- приведена классификация эвристических алгоритмов решения метрической задачи коммивояжера;
- предложены модификации алгоритмов;
- проведена экспериментальная оценка алгоритмов;
- выявлена группа Парето-оптимальных алгоритмов решения метрической задачи коммивояжера.

Апробация работы:

Основные положения и результаты докладывались и обсуждались на следующих научных конференциях:

1. Межвузовская научно-техническая конференция студентов, аспирантов и молодых исследователей им. Е.В. Арменского. 20 февраля 2017. МИЭМ НИУ ВШЭ, Москва. Получен диплом 2-ой степени за лучший доклад в секции «Математическое моделирование», см. рис. Л.1 в Приложении Л.
2. Научно-практическая конференция по компьютерным наукам CoCoS'2017. 4 апреля 2017. НИУ ВШЭ, Москва.

3. Spring/Summer Researches' Colloquim on Software Engineering, SYRCoSE 5 -7 June 2017. Innopolis. (Статья «Pareto-optimal Algorithms for Metric TSP» прошла экспертизу, рекомендована к представлению на конференции).

Публикации:

1. Е.Н. Береснева. Апостериорные оценки точности и временной сложности эвристических алгоритмов решения евклидовой задачи коммивояжера. Материалы межвузовской научно-технической конференции студентов, аспирантов и молодых исследователей им. Е.В. Арменского, – М.: – МИЭМ НИУ ВШЭ, 2017 год, стр. 26-27, ISBN 978-5-94768-075-1.
2. Avdoshin, S. Pareto-optimal Algorithms for Metric TSP / S.M. Avdoshin, E.N. Beresneva // International Journal of Open Information Technologies (INJOIT). – 2017. – N 5. – C. 16-24, ISSN: 2307-8162 [1].
3. Spring/Summer Researches' Colloquim on Software Engineering, SYRCoSE 5 -7 June 2017. Innopolis. (Статья «Pareto-optimal Algorithms for Metric TSP» прошла экспертизу, рекомендована к публикации).

Регистрация программ в РОСПАТЕНТ:

1. Программа решения метрической задачи коммивояжера на основе метода ближайшей вставки (The program for Metric TSP based on Nearest Insertion). Заявка на государственную регистрацию программы для ЭВМ в федеральную службу по интеллектуальной собственности, патентам и товарным знакам.
2. Программа решения метрической задачи коммивояжера на основе метода "выгодной" вставки (The program for Metric TSP based on Cheapest Insertion). Заявка на государственную регистрацию программы для ЭВМ в федеральную службу по интеллектуальной собственности, патентам и товарным знакам.
3. Программа решения метрической задачи коммивояжера на основе метода ближайшего отрезка (The program for Metric TSP based on Nearest Segment Insertion). Заявка на государственную регистрацию программы для ЭВМ в федеральную службу по интеллектуальной собственности, патентам и товарным знакам.
4. Программа решения метрической задачи коммивояжера на основе удвоенного минимального оствового дерева (The program for Metric TSP based on double minimum spanning tree). Заявка на государственную регистрацию программы для ЭВМ в федеральную службу по интеллектуальной собственности, патентам и товарным знакам.
5. Программа решения метрической задачи коммивояжера на основе минимальных паросочетаний вершин нечетной степени минимального оствового дерева (The

program for Metric TSP based on minimum-weight perfect matching of odd vertices of minimum spanning tree). Заявка на государственную регистрацию программы для ЭВМ в федеральную службу по интеллектуальной собственности, патентам и товарным знакам.

6. Программа решения метрической задачи коммивояжера на основе 2-Opt (The program for Metric TSP based on 2-Opt). Заявка на государственную регистрацию программы для ЭВМ в федеральную службу по интеллектуальной собственности, патентам и товарным знакам.
7. Программа решения метрической задачи коммивояжера на основе метода поисковой оптимизации, инспирированном роем пчёл (The program for Metric TSP based on qCABC). Заявка на государственную регистрацию программы для ЭВМ в федеральную службу по интеллектуальной собственности, патентам и товарным знакам.
8. Программа решения метрической задачи коммивояжера на основе кривой Мура (The program for Metric TSP based on Moore Curve). Заявка на государственную регистрацию программы для ЭВМ в федеральную службу по интеллектуальной собственности, патентам и товарным знакам.

Abstract

The Travelling Salesman Problem (TSP) is a fundamental task in a class of combinatorial optimization problems which consists of finding a Hamiltonian circuit of minimal length. Solutions of the TSP are generally used for costs minimization tasks in logistics and manufacturing, such as finding the best tour for round-the-world trip or construction of very large-scale integration schemes. A special case of the TSP is Metric TSP, where the triangle inequality holds.

The objective of this article is to determine a group of Pareto optimal algorithms for Metric TSP under criteria of run time efficiency and qualitative performance as a part of the experimental study. In this paper, we consider only heuristic algorithms providing near optimal solutions in polynomial time, since the TSP is NP-hard.

Classification of algorithms for Metric TSP is presented. Groups of feasible heuristic algorithms and their modifications are described. The data structure and the details of the research methodology are provided. Two real-life kinds of inputs are used for the experimental study: very large scale integration schemes and geographic coordinated of cities. Finally, results and prospective research are discussed.

The paper contains 113 pages, 13 illustrations, 7 tables, 14 schemes, 67 bibliography items, 11 appendices.

Keywords: *travelling salesman problem, resource-efficient algorithm, heuristic algorithm, Metric TSP, computational experiment, NP-hard problem.*

Содержание

Определения и обозначения.....	10
Сокращения.....	12
Введение.....	13
1. Постановка задачи.....	16
2.1. Метрики.....	16
2.2. Математическая постановка метрической задачи коммивояжера.....	17
2.3. Параметры ресурсной эффективности	18
2.4. Оптимальность по Парето	18
Выводы по главе	19
2. Обзор эвристических алгоритмов.....	20
2.1. Эвристики построения маршрута	20
2.1.1. Алгоритмы упорядоченных последовательностей (Order Sequence algorithms).....	20
2.1.2. Алгоритмы наращивания пути (Increasing Path algorithms)	21
2.1.3. Алгоритмы вставки в подцикл (Subtour Insertion algorithms)	22
2.1.4. Алгоритмы объединения подциклов (Merged Multiple Subtours algorithms)	22
2.1.5. Комбинированные алгоритмы построения маршрута (Combined tour construction algorithms).....	23
2.2. Эвристики улучшения маршрута	23
2.2.1. Локально-оптимальные алгоритмы (Local-optimal algorithms).....	23
2.2.2. Алгоритмы имитации отжига (Simulated Annealing algorithms).....	24
2.2.3. Эволюционные алгоритмы (Evolutionary algorithms)	25
2.2.4. Алгоритмы роевого интеллекта (Swarm Intelligence algorithms)	25
Выводы по главе	26
3. Выбранные алгоритмы и особенности реализации	28
3.1. Входные данные	28
3.2. Описание приближенных алгоритмов	30

3.2.1. Алгоритм ближайшего соседа (Nearest Neighbour, NN)	30
3.2.2. Модифицированный алгоритм ближайшего соседа (Double Ended Nearest Neighbour, DENN)	31
3.2.3. Алгоритм на основе кривой Мура (Moore Curve, MC).....	33
3.2.4. Алгоритм на основе кривой Серпинского (Sierpinski Curve, SC).....	34
3.2.5. Алгоритм добавления «ближайшего» (Nearest Addition, NA)	36
3.2.6. Алгоритм вставки «ближайшего» (Nearest Insertion, NI)	37
3.2.7. Алгоритм «выгодной» вставки (Cheapest Insertion, CI).....	39
3.2.8. Алгоритм вставки « дальнейшего» (Farthest Insertion, FI)	40
3.2.9. Алгоритм вставки «случайного» (Arbitrary Insertion, AI)	41
3.2.10. Алгоритм «ближайшего» до отрезка (Nearest Segment Insertion, NSI).....	42
3.2.11. Жадный алгоритм (Greedy, GRD)	44
3.2.12. Алгоритм на основе удвоенного минимального оствнового дерева (Double Minimal Spanning Tree, DMST)	45
3.2.13. Улучшенный алгоритм на основе удвоенного минимального оствнового дерева (Double Minimal Spanning Tree Modified, DMST-M)	45
3.2.14. Алгоритм на основе минимальных паросочетаний вершин нечетной степени минимального оствнового дерева (Christofides algorithm, CHR).....	46
3.2.15. Алгоритм 2-Opt	47
3.2.16. Алгоритм LKH (Helsgaun's Lin and Kernighan algorithm, LKH).....	47
3.2.17. Улучшенный алгоритм поисковой оптимизации, инспирированный роем пчёл (quick Combinatorial Artificial Bee Colony algorithm, qCABC)	48
3.3. Особенности реализации	50
3.3.1. Волновая функция как структура хранения данных.....	50
3.3.2. Вычисление целочисленного значения квадратного корня	51
3.3.1. Алгоритм GRD.....	51
Выводы по главе	52
4. Организация экспериментального исследования.....	53
Выводы по главе	56

5. Результаты исследования	57
Выводы по главе	63
Заключение	64
Список использованных источников	66
Приложение А. Сравнение структур данных	71
Приложение Б. Алгоритмы вычисления целочисленного значения квадратного корня	73
Приложение В. Система непересекающихся множеств	76
Приложение Г. Текст программы	77
Приложение Д. Оценки точности алгоритмов	78
Приложение Е. Время работы алгоритмов	80
Приложение Ж. Графики Парето-оптимальных алгоритмов (VLSI Data Sets)	81
Приложение З. Графики Парето-оптимальных алгоритмов (National TSPs)	98
Приложение И. Сводные таблицы с результатами эксперимента (VLSI Data Sets)	108
Приложение К. Сводные таблицы с результатами эксперимента (National TSPs)	111
Приложение Л. Диплом 2 степени за лучшую работу на конференции Е.В. Арменского	113

Определения и обозначения

В настоящей ВКР используются следующие термины с соответствующими определениями:

Временная сложность алгоритма – асимптотическая оценка функции трудоёмкости [2].

Гамильтонов цикл – это замкнутый путь, проходящий через каждую вершину графа G ровно один раз.

Задача коммивояжера заключается в нахождении в данном полном взвешенном графе гамильтонова цикла минимального веса.

Задача является **NP-полной** (NP-Complete), если она принадлежит к классу NP, и к ней должны полиномиально сводиться все задачи класса NP [2].

Задача является **NP-трудной** (NP-Hard), если эта задача в постановке принятия решения принадлежит к классу NP-полных задач [2].

Класс NP – класс задач с полиномиально-роверяемым решением или класс полиномиально-роверяемых задач. Задача относится к классу NP, если её решение может быть проверено с полиномиальной временной сложностью [2].

Кратчайшее связывающее дерево (минимальное оствовное дерево) – связный подграф связного неориентированного взвешенного графа минимального веса, в который входят все его вершины.

Неориентированным графом $G(V, E)$ называется совокупность множества вершин $V = V(G)$ и множества $E = E(G)$ неупорядоченных пар $u = \{v_i, v_j\}$, называемых ребрами, где v_i, v_j – вершины, инцидентные ребру u .

Неориентированным мультиграфом $G(V, E)$ называется совокупность множества вершин $V = V(G)$ и семейства $E = E(G)$ неупорядоченных пар $u = \{v_i, v_j\}$, называемых ребрами, где v_i, v_j – вершины, инцидентные ребру u .

Подцикл – это цикл, содержащий k вершин, где $k < N$.

Степень вершины в неориентированном графе – число ребер, инцидентных данной вершине.

Функция трудоемкости – это отношение, связывающее входные данные алгоритма с количеством элементарных операций [2].

Эйлеров цикл – цикл графа, проходящий через каждое ребро графа G ровно по одному разу.

В настоящей ВКР используются следующие обозначения:

Обозначение	Значение
N	Количество вершин $ V $
n	Количество измерений
$G = (V, V^2)$	Граф G , состоящий из множества вершин V и множества ребер V^2
$d(v_i, v_j)$	Расстояние между вершинами v_i и v_j
M	Множество реализованных эвристических алгоритмов решения метрической задачи коммивояжера
$f_\varepsilon(m)$	Апостериорная оценка точности алгоритма m
$f_t(m)$	Апостериорное время работы алгоритма m
$f_\varepsilon^{apr}(m)$	Априорная оценка точности алгоритма m
$f_t^{apr}(m)$	Априорная оценка временной сложности алгоритма m
$\sigma(f_t)$	Стандартное отклонение среди значений времени работы алгоритма
$E(f_\varepsilon)$	Математическое ожидание среди оценок точности алгоритма
$\sigma(f_\varepsilon)$	Стандартное отклонение среди оценок точности алгоритма
$\max(f_\varepsilon)$	Максимальное значение среди оценок точности алгоритма
$\min(f_\varepsilon)$	Минимальное значение среди оценок точности алгоритма

Сокращения

В настоящей ВКР используются следующие сокращения:

Обозначение	Значение
AI	Алгоритм вставки «случайного» (Arbitrary Insertion)
CHR	Алгоритм на основе минимальных паросочетаний вершин нечетной степени минимального остовного дерева (Christofides algorithm)
CI	Алгоритм «выгодной» вставки (Cheapest Insertion)
DENN	Модифицированный алгоритм ближайшего соседа (Double Ended Nearest Neighbour)
DMST	Алгоритм на основе удвоенного минимального остовного дерева (Double Minimal Spanning Tree)
DMST-M	Улучшенный алгоритм на основе удвоенного минимального остовного дерева (Double Minimal Spanning Tree Modified)
DSU	Система непересекающихся множеств (Disjoint Set Union)
FI	Алгоритм вставки «далнейшей» (Farthest Insertion)
GRD	Жадный алгоритм (Greedy)
LKH	Алгоритм LKH (Helsgaun's Lin and Kernighan algorithm)
MC	Алгоритм на основе кривой Мура (Moore Curve)
MTSP	Метрическая задача коммивояжера (Metric Travelling Salesman Problem)
NA	Алгоритм добавления «ближайшего» (Nearest Addition)
NI	Алгоритм вставки «ближайшего» (Nearest Insertion)
NN	Алгоритм ближайшего соседа (Nearest Neighbour)
NSI	Алгоритм «ближайшего» до отрезка (Nearest Segment Insertion)
qCABC	Улучшенный алгоритм поисковой оптимизации, инспирированный роем пчёл (quick Combinatorial Artificial Bee Colony algorithm)
SC	Алгоритм на основе кривой Серпинского (Sierpinski Curve)
TSP	Задача коммивояжера (Travelling Salesman Problem)
VLSI	Высоко интегральные вычислительные схемы (Very large scale integer schemes)

Введение

Задача коммивояжера (Travelling Salesman Problem) – бесспорно, одна из самых широко исследуемых задач комбинаторной оптимизации, суть которой состоит в отыскании гамильтонова цикла наименьшей длины. Простым языком, задача коммивояжера может быть описана как задача нахождения кратчайшей дистанции между N городами, при условии, что каждый город необходимо посетить единожды, и важно вернуться в первоначальный город.

Несмотря на простоту формулировки задачи, разработка алгоритмов для решения задачи коммивояжера осуществляется на протяжении многих лет, и, по-прежнему, остается актуальной, поскольку задача является NP-трудной. Наивный метод перебора всех возможных вариантов имеет вычислительную сложность $O(N!)$ и позволяет получить $\frac{(N-1)!}{2}$ возможных различных циклов. Однако уже при $N = 15$ необходимо перебрать более 87 млрд различных маршрутов. Для этого необходимо затратить более 12 часов, при условии, что один маршрут вычисляется за 1 микросекунду. Для $N = 18$ на определение кратчайшего маршрута понадобится уже более 5 лет. Полученные подсчеты послужили причиной появления более тысячи публикаций, посвященных анализу алгоритмов, нацеленных решить задачу коммивояжера более эффективно.

Существуют следующие вариации классической задачи коммивояжера, основанные на типе расстояний между вершинами:

1. Симметричная задача коммивояжера (Symmetric TSP) отличается от классической постановки тем, что расстояния между вершинами удовлетворяют аксиоме симметричности - они одинаковы для одних и тех же пар вершин вне зависимости от направления ребер.
2. Асимметричная задача коммивояжера (Asymmetric TSP) может быть представлена в виде ориентированного графа. В отличие от симметричной задачи коммивояжера, расстояния между одними и теми же парами вершин в одном направлении могут отличаться от противоположного направления.

Объектом исследования в данной работе является частный случай симметричной задачи коммивояжера – задача коммивояжера в метрическом пространстве (Metric TSP). В метрической задаче коммивояжера веса ребер удовлетворяют аксиомам симметричности, неотрицательности, тождественности, и неравенства треугольника. Иными словами, в метрической задаче коммивояжера выполняется свойство метрики.

Метрическая задача коммивояжера может быть применена для решения множества практических задач в нашей повседневной жизни. Наиболее популярная сфера применения задачи коммивояжера – логистика [3-5]. В этой области рассматриваются задачи определения

эффективных маршрутов для школьных автобусов [4], доставки почты [4], грузовиков [6], доставки заказов на складах [7], туристов [8], минимизация затрат (физических, материальных, временных) при доставке различных грузов по конкретным адресам [9].

Другие направления практического применения задачи коммивояжера:

- управление спутниками, телескопами, микроскопами и лазерами [10];
- сборка генома [4];
- кластеризация массивов данных [11];
- рентгеноструктурный анализ [12, 13];
- проектирование телекоммуникационных сетей [14];
- соединение ряда пунктов кольцевыми линиями энергопередач, газоснабжения, например, задача подводки электроэнергии к рабочим местам [4];
- определение оптимальной последовательности обработки деталей на станке [15];
- проектирование топологий сверх больших интегральных схем [16].

Развитие практического применения задачи коммивояжера более подробно описано в статье [17].

Задача коммивояжера играет большую роль в разработке многих широко используемых эвристических методов. Подобные алгоритмы рассчитаны на быструю работу и получение решений с предположительно небольшим отклонением от оптимума. Эвристики не подходят для поиска точного решения задачи коммивояжера, но рассчитаны на получение базового начального приближения.

Предметом исследования являются эвристические алгоритмы решения метрической задачи коммивояжера.

Цель исследования – определить группу Парето-оптимальных алгоритмов решения метрической задачи коммивояжера по критериям времени работы и точности решения.

Для достижения данной цели были сформулированы следующие задачи:

- реализовать существующие эвристические алгоритмы решения метрической задачи коммивояжера;
- разработать методику оценки качества алгоритмов;
- предложить модификации эвристических алгоритмов решения метрической задачи коммивояжера с целью получения реализаций с лучшими оценками качества;
- произвести экспериментальную оценку качества реализованных алгоритмов.

Актуальность данной работы состоит в определении Парето-оптимальных алгоритмов решения метрической задачи коммивояжера для достижения ресурсной эффективности при

заданных ограничениях. Ранее алгоритмы решения метрической задачи коммивояжера не были проанализированы на основе оптимальности по Парето.

Работа структурирована следующим образом. Глава 1 включает теоретические основы работы – математическую постановку метрической задачи коммивояжера, параметры ресурсной эффективности и определение оптимальности по Парето. В главе 2 рассматривается классификация алгоритмов решения метрической задачи коммивояжера, приведены известные подходы. Глава 3 описывает выбранные входные данные, алгоритмы, особенности реализации и структуры данных. Глава 4 описывает планирование экспериментов. Глава 5 содержит сравнительный анализ полученных результатов. В заключении приводятся перспективы дальнейших исследований по данной тематике.

В Приложениях к работе приведены:

- сравнение времени работы алгоритмов с использованием двух различных структур данных;
- алгоритмы вычисления целочисленного значения квадратного корня и их оценки времени работы;
- реализация системы непересекающихся множеств на языке C++;
- текст программы;
- графики, иллюстрирующие оценки точности алгоритмов;
- график, отображающий время работы алгоритмов;
- графики Парето-оптимальных алгоритмов для двух баз данных;
- сводные таблицы с результатами принадлежности алгоритмов к Парето-оптимальным алгоритмам;
- диплом 2 степени за лучшую работу, представленную на конференции Е.В. Арменского.

1. Постановка задачи

В данной главе описываются теоретические основы работы. Приводится классификация распространенных на практике метрик. Описывается математическая постановка метрической задачи коммивояжера, используемая в данной работе. Указываются параметры ресурсной эффективности, позволяющие осуществить сравнение алгоритмов с использованием оптимальности по Парето.

2.1. Метрики

Расстояние между двумя вершинами v_i и v_j рассчитывается с помощью функции $d(v_i, v_j)$. Данная функция возвращает вещественное значение и удовлетворяет следующим свойствам [18]:

1. $d(v_i, v_j) \geq 0$ (*аксиома неотрицательности*).
2. $d(v_i, v_j) = 0$ тогда и только тогда, когда $v_i = v_j$ (*аксиома тождества*).
3. $d(v_i, v_j) = d(v_j, v_i)$ (*аксиома симметрии*).
4. $d(v_i, v_j) \leq d(v_i, v_k) + d(v_k, v_j)$ (*аксиома неравенства треугольника*).

Пусть множество точек V задано своими координатами в евклидовом пространстве R^n . Тогда расстояние Минковского порядка p , где $p \geq 1$, между двумя вершинами $v = (x_1, x_2, \dots, x_n)$ и $w = (x_1, x_2, \dots, x_n)$ определяется по формуле:

$$d(v, w) = \sqrt[p]{\sum_{i=1}^D |x_i(v) - x_i(w)|^p}$$

На практике распространены следующие метрики, основанные на расстоянии Минковского (рис. 1):

- «Манхэттенское» расстояние, или метрика L_1 , при $p = 1$ (1).
- Евклидово расстояние, или метрика L_2 , при $p = 2$ (2).
- Максимальная метрика, или метрика Чебышева, или метрика L_∞ , при $p \rightarrow \infty$ (3).

$$d_{Manh}(v, w) = \sum_{i=1}^D |x_i(v) - x_i(w)| \quad (1)$$

$$d_{Euc}(v, w) = \sqrt{\sum_{i=1}^D (x_i(v) - x_i(w))^2} \quad (2)$$

$$d_{Max}(v, w) = \max_{i=1, \dots, D} |x_i(v) - x_i(w)| \quad (3)$$

L_1 и L_∞ метрики используются, к примеру, сверлильными станками, которые просверливают заданное множество отверстий на печатной плате. L_1 метрика используется в

станках, координаты перемещения которого настраиваются поочередно – сначала по одной оси, затем по другой. Таким образом, общее время достижения новой точки состоит из суммы последовательных перемещений. В свою очередь L_∞ метрика используется в станках, координаты перемещения которого перестраиваются одновременно. Поэтому общее время достижения новой точки является временем максимального из перемещений.

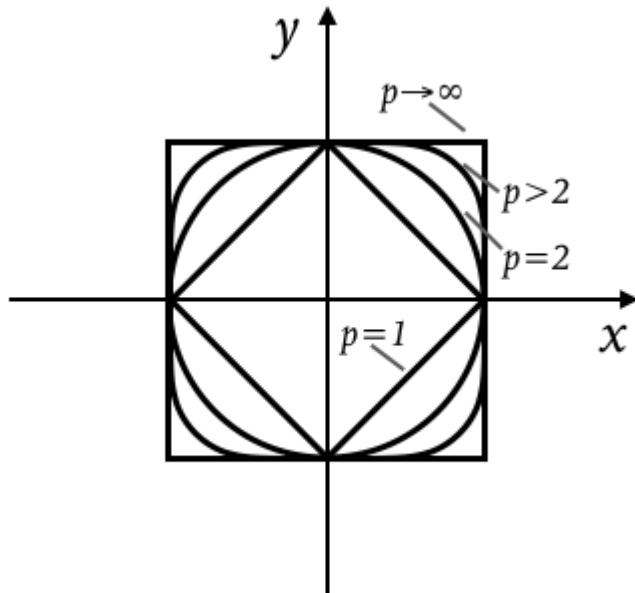


Рисунок 1. Визуальное представление значения параметра p метрики Минковского

2.2. Математическая постановка метрической задачи коммивояжера

Дан полный взвешенный неориентированный граф $G = (V, V^2)$, содержащий $N = |V|$ вершин. Пусть $I = \{1, 2, \dots, N\}$. На множестве вершин графа зададим индексацию $index = V \rightarrow I$, $(\forall v_i \in V)(\forall v_j \in V) v_i \neq v_j \Rightarrow i \neq j$. Здесь $i = index(v_i)$.

Обозначим через $S = \{p: V \rightarrow V | (p(1) = 1 \& (\forall i \in V)(\forall j \in V)(p(i) = p(j) \Rightarrow i = j)\}$ множество кодов всех гамильтоновых циклов $s = (p_1, p_2, \dots, p_N)$ графа G . Здесь p_i используется в качестве сокращенной записи для $p(i)$.

В этих обозначениях вес гамильтонова цикла $s \in S$ определяется его $p_1 p_2 p_3 \dots p_N$ по следующей формуле:

$$f(s) = d(p_1, p_N) + \sum_{i=1}^{N-1} d(p_i, p_{i+1})$$

Для заданной метрики $d \in \{d_{Euc}, d_{Manh}, d_{Max}\}$ найти:

$$s_0: f(s_0) = \min_{s \in S} f(s)$$

2.3. Параметры ресурсной эффективности

Пусть M – множество реализованных эвристических алгоритмов решения метрической задачи коммивояжера. Определим два параметра оценки ресурсной эффективности для $m \in M$:

- $f_\varepsilon(m)$ – оценка точности алгоритма, выраженная в процентах (qualitative performance, error rate).
- $f_t(m)$ – время работы алгоритма, измеряемое в секундах.

Оценка точности алгоритма вычисляется по формуле:

$$f_\varepsilon(m) = \frac{f(s) - f(s_0)}{f(s_0)},$$

где $f(s)$ – длина гамильтонова цикла, полученного в ходе выполнения алгоритма;

$f(s_0)$ – оптимальная длина гамильтонова цикла.

Априорная оценка точности алгоритма в худшем случае определяется как отношение наихудшей (наибольшей) длины гамильтонова цикла к оптимальному значению:

$$f_\varepsilon^{apr}(m) \leq \frac{f(s)}{f(s_0)}$$

2.4. Оптимальность по Парето

В данной работе рассматривается двухкритериальная оптимизация по Парето.

Алгоритм $m_0 \in M$ является Парето-оптимальным, если:

$$(\forall m \in M) \left((m \neq m_0) \Rightarrow (f_\varepsilon(m) > f_\varepsilon(m_0)) \vee (f_t(m) > f_t(m_0)) \right)$$

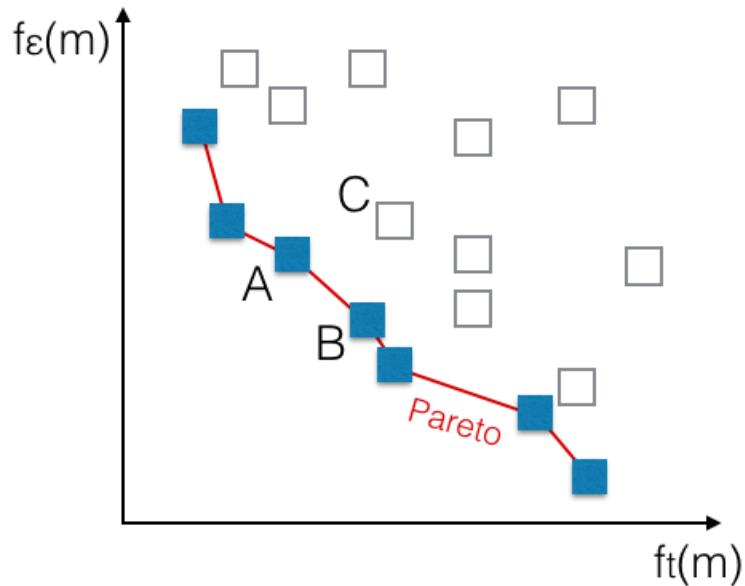


Рисунок 2. Пример множества Парето-оптимальных алгоритмов

На рис. 2 показан график, который содержит множество Парето-оптимальных алгоритмов, расположенных на красной ломаной линии. Квадраты обозначают полученные решения. Решение С не является оптимальным по Парето, так как оба решения А и С обладают лучшими (меньшими) численными значениями.

В данной работе требуется найти множество

$$M_0 = \left\{ (\forall m \in M) \left((m \neq m_0) \Rightarrow (f_\varepsilon(m) > f_\varepsilon(m_0)) \vee (f_t(m) > f_t(m_0)) \right) \right\}$$
 Парето-оптимальных алгоритмов решения метрической задачи коммивояжера по критериям времени работы и точности решения.

Выходы по главе

Приведена классификация распространенных на практике метрик. Описана математическая постановка метрической задачи коммивояжера, используемая в данной работе. Выбраны параметры ресурсной эффективности, позволяющие осуществить сравнение алгоритмов с использованием оптимальности по Парето. Приведена постановка задачи, исследуемая в данной работе.

2. Обзор эвристических алгоритмов

В данной главе приводится классификация эвристических алгоритмов решения метрической задачи коммивояжера, включающая ссылки на публикации по данной тематике. В первой половине указаны существующие эвристики построения маршрута с кратким описанием методов. Во второй половине описываются основные эвристики улучшения уже найденных маршрутов, приводятся известные подходы. Приводятся априорные оценки точности $f_\epsilon^{apr}(m)$ и априорные асимптотические временные оценки $f_t^{apr}(m)$ алгоритмов.

Алгоритмы решения метрической задачи коммивояжера можно разделить на две группы:

- точные алгоритмы, и
- эвристические алгоритмы.

В данном исследовании рассматриваются только эвристические алгоритмы решения метрической задачи коммивояжера, позволяющие получить хорошее (в определенном смысле) решение за время, ограниченное полиномом от размера задачи (входных данных). В свою очередь эвристики делятся на:

- алгоритмы построения маршрута (tour construction algorithms);
- алгоритмы улучшения маршрута (tour improving algorithms).

2.1. Эвристики построения маршрута

Алгоритмы построения маршрута имеют одну общую особенность – построение цикла происходит итеративно, очередная вершина добавляется на каждом шаге. Алгоритмы, входящие в данную группу, состоят из следующих трех основных частей.

1. Инициализация – правило определения первоначального подцикла (initial sub-tour) или начальной вершины (starting point).
2. Выбор – критерий выбора очередного элемента для добавления к текущему маршруту.
3. Вставка – определение позиции вставки нового элемента в текущий маршрут.

Эвристики построения маршрута подразделяются на пять подгрупп, классификация впервые была представлена в 1988 году и до сих пор признается многими исследователями [19].

2.1.1. Алгоритмы упорядоченных последовательностей (Order Sequence algorithms)

Алгоритмы, входящие в данную подгруппу, отличаются тем, что в первую очередь все ребра ранжируются (сортируются) по определенному критерию. На каждой итерации лучшее в

соответствии с неким критерием ребро (с наибольшим или наименьшим рангом) добавляется к текущему решению. Позиция (ранг) ребер не обязательно изменяется на каждом шаге. Ребра, которые нарушают установленные правила, не учитываются и не добавляются к маршруту.

К алгоритмам упорядоченных последовательностей относятся алгоритмы группы «механизм сбережений» (Clarke and Wright Savings) [20, 21] и жадный алгоритм Greedy [22].

На данный момент известны следующие модификации алгоритмов группы «механизм сбережений» [23, 24].

В данной работе среди двух представленных алгоритмов будет рассматриваться только алгоритм Greedy. Несмотря на одинаковые оценки временной сложности $O(N^2 \log N)$ и точности $\log N$ обоих алгоритмов, экспериментальные результаты жадного алгоритма являются более точными [25]. В качестве входных данных использовалась генерация 150 случайных вершин.

В дальнейшем планируется реализация «механизма сбережений» с целью экспериментального сравнения алгоритмов подгруппы упорядоченных последовательностей на других тестовых данных.

2.1.2. Алгоритмы наращивания пути (Increasing Path algorithms)

В алгоритмах, входящих в данную подгруппу, в качестве отправного элемента выбирается одна вершина или ребро. На каждой итерации одно ребро выбирается согласно определенному критерию и добавляется к одному из концов текущего маршрута. Недостатком такого подхода является вероятность возникновения такой ситуации, при которой после добавления всех ребер конечные точки маршрута оказываются очень далеко друг от друга. В результате для превращения маршрута в цикл приходится добавлять очень «тяжелое» (с большим весом) ребро, что может значительно повлиять на итоговый результат.

К алгоритмам наращивания пути относятся алгоритм «ближайшего соседа» – Nearest Neighbour [26] и модифицированный алгоритм «ближайшего соседа» Double Ended Nearest Neighbour [27], а также алгоритмы из группы заполняющих пространство кривых (Space-Filling Curve Heuristic [28, 29]) – Moore Curve [30], Sierpinski Curve [31].

Алгоритмы «ближайшего соседа» имеют временную оценку сложности $O(N^2)$ и оценку точности, равную $0.5[\log N + 1]$ [19].

Алгоритмы из группы заполняющих пространство кривых имеют временную оценку сложности $O(N \log N)$ и оценку точности $\log N$ [19].

В данной работе рассматриваются все вышеперечисленные алгоритмы из подгруппы алгоритмов наращивания пути.

2.1.3. Алгоритмы вставки в подцикл (Subtour Insertion algorithms)

В алгоритмах, входящих в данную подгруппу, в качестве первоначального подцикла выбираются две вершины. На каждой итерации выбирается ранее не использованная вершина согласно установленным правилам добавляется в подцикл определенным образом. При добавлении вершины одно из ребер текущего подцикла удаляется и добавляются два не входящих в подцикл ребра, содержащих новую вершину. После каждой итерации алгоритма число вершин и ребер должно увеличиваться.

К методам вставки в подцикл относятся алгоритмы группы «вставка» [26, 32]:

- алгоритм добавления «ближайшего» (Nearest Addition);
- алгоритм вставки «ближайшего» (Nearest Insertion);
- алгоритм вставки «далнейшего» (Farthest Insertion);
- алгоритм «выгодной» вставки (Cheapest Insertion);
- алгоритм «случайной» вставки (Arbitrary Insertion).

Также в данную подгруппу входит алгоритм Convex Hull [33, 34], в качестве первоначального подцикла строящий выпуклую оболочку вершин.

В данной работе рассматриваются все известные алгоритмы группы «вставка». Временная оценка сложности алгоритмов составляет $O(N^2)$, за исключением Cheapest Insertion $O(N^2 \log N)$. Оценки точности алгоритмов Nearest Addition, Nearest Insertion и Cheapest Insertion лучше, чем у Farthest Insertion и Arbitrary Insertion (2 против $2 \lg N + 0.16$) [19].

Алгоритм Convex Hull имеет временную оценку сложности, равную $O(N^2 \log N)$. Математически выведенного значения оценки точности для данного алгоритма нет. В данной работе алгоритм не рассматривается, в связи с его применимостью для решения только евклидовой задачи коммивояжера на R^2 .

2.1.4. Алгоритмы объединения подциклов (Merged Multiple Subtours algorithms)

В алгоритмах, входящих в данную подгруппу, согласно выбранным критериям создается множество непересекающихся подциклов, которые впоследствии объединяются по установленным правилам.

Наиболее известными алгоритмами объединения подциклов являются алгоритмы Nearest Merger [35] и Patching [36].

Временная оценка сложности алгоритма Nearest Merger равна $O(N^2)$, алгоритма Patching – $O(N^3)$. Оценка точности алгоритма Nearest Merger равна двум, для алгоритма Patching доказанной оценки точности нет [19].

Алгоритмы объединения подциклов применяются в основном для решения асимметричной задачи коммивояжера. Не найдено ни одного источника, показывающего практическую значимость алгоритмов данной подгруппы при решении метрической задачи коммивояжера. В связи с этим, в данной работе решено не исследовать данные алгоритмы. В будущем возможна их реализация с целью определения применимости к метрической задаче коммивояжера.

2.1.5. Комбинированные алгоритмы построения маршрута (Combined tour construction algorithms)

К данной подгруппе относятся алгоритмы, основанные на построении кратчайшего связывающего дерева, за авторством Кристофидеса. Первый алгоритм Double MST заключается в удвоении кратчайшего связывающего дерева, составлении эйлерова цикла и сведению его к гамильтонову циклу путем удаления повторяющихся вершин [37]. Второй алгоритм Christofides является наиболее известным и основывается на минимальных паросочетаниях вершин нечетной степени минимального остовного дерева [38].

Алгоритм Double MST имеет квадратичную временную сложность $O(N^2)$ и позволяет получить решения, не превосходящие оптимальные более чем в 2 раза. Второй алгоритм Кристофидеса имеет кубическую временную сложность $O(N^3)$, решения полученные с его помощью хуже оптимальных не более чем в 1,5 раза [19]. В данной работе рассматриваются оба алгоритма.

2.2. Эвристики улучшения маршрута

Отличительной особенностью эвристических алгоритмов улучшения маршрута является то, что на вход данных алгоритмов подаются циклы, построенные с помощью одного из алгоритмов построения маршрутов. Задачей данных эвристик является изменение полученного цикла с целью улучшения (уменьшения его веса).

2.2.1. Локально-оптимальные алгоритмы (Local-optimal algorithms)

Алгоритмы данной подгруппы основаны на удалении пересечений ребер в цикле. 2-Opt [39] сводится к удалению двух пересекающихся рёбер из маршрута и вставке двух новых рёбер, в том случае если сумма длин новых ребер меньше суммы длин старых.

Алгоритм k -Opt [40] является обобщением алгоритма 2-Opt и позволяет сменить k пар ребер. Цикл, найденный с помощью данного алгоритма, называется k -оптимальным, если более ни одна k -Opt операция не может привести к улучшению маршрута. Временная сложность алгоритма составляет $O(N^k)$. Согласно [26] для любого алгоритма k -Opt, где $k \leq N/4$, оценка точности решения приблизительно равна двум.

Локально-оптимальный алгоритм Lin and Kernighan Heuristic [41] основан на алгоритме *k*-Opt и в дополнении ко всему является адаптивным, т.е. решение о том, сколько ребер должно поменяться, принимается на каждом шаге.

Позднее была предложена модификация алгоритма Lin and Kernighan Heuristic, получившая название LKH Heuristic (Helsgaun's Lin and Kernighan Heuristic) и позволившая значительно повысить точность решения [41]. Данный алгоритм уже не является локально-оптимальным, он считается *составным*. Это означает, что он и самостоятельно создает первоначальную перестановку, и впоследствии производит итеративные улучшения. Временная сложность алгоритма составляет приблизительно $O(N^{2,2})$ [42].

В данной работе рассматривается базовая локально-оптимальная эвристика – 2-Opt и самый эффективный на данный момент алгоритм – LKH Heuristic.

2.2.2. Алгоритмы имитации отжига (Simulated Annealing algorithms)

Оптимизационные эвристические алгоритмы, к примеру, локально-оптимальные, стремятся к итеративному улучшению текущего решения. На каждой итерации алгоритма вес цикла улучшается (уменьшается), и в том случае, когда дальнейшее исполнение алгоритма не может привести к улучшению решения, алгоритм останавливается. Полученное значение может оказаться локальным минимумом. Алгоритмы имитации отжига [43] предполагают периодическое (согласно определенному критерию) увеличение стоимости маршрута, что способствует снижению вероятности получения решения, «застрявшего» в локальном минимуме. Данный алгоритм основывается на имитировании природы физического процесса, происходящего во время кристаллизации вещества (при отжиге металлов, в частности). Отжиг – процесс плавки материала, после которого температуру снижают очень медленно. Это сделано для того, чтобы предотвратить повреждение кристаллической решетки материала с целью увеличения прочности металла. В том случае если понижение температуры выполняется постепенно и правильно, то достигается минимум энергии атомов, что в комбинаторной оптимизации соответствует достижению глобального минимума.

Алгоритм имитации отжига похож на метод поиска локального экстремума функции с помощью движения вдоль градиента (градиентный спуск). Однако ввиду случайности выбора промежуточного решения вероятность попадания в локальный минимумы снижается.

Детальное описание алгоритма приведено в книге [44]. Исходный алгоритм имитации отжига получил ряд модификаций и описан в статьях [45, 46].

Временная оценка сложности алгоритма – $O(N^2)$ [44]. В работе [47] показано, что алгоритмы имитации отжига значительно уступают по точности решения локально-

оптимальному алгоритму Lin and Kernighan Heuristic, в связи с чем в данной работе они не рассматриваются.

2.2.3. Эволюционные алгоритмы (Evolutionary algorithms)

Эволюционные алгоритмы инспирированы природой биологической эволюции и включают в себя следующие этапы:

- репродукция – унаследование черт обоих решений-родителей;
- мутация и рекомбинация генов – изменение согласно определенным правилам;
- отбор – выбор решений, соответствующих определенному критерию «годности», и отсев «неудачных» решений.

Качество решений, полученных с помощью эволюционных алгоритмов, описывается фитнес-функцией (функцией приспособленности).

Алгоритмы данной подгруппы подразделяются на генетические (Genetic Algorithms) и нейросетевые (Neural Networks Algorithms) алгоритмы. Обзор генетических алгоритмов представлен в книге [25]. Обзор нейросетевых алгоритмов приведен в статье [48].

В других работах эволюционные алгоритмы рассматриваются в основном на малых размерностях ($N \leq 300$), что связано с высоким временем работы алгоритмов.

В данной работе эволюционные алгоритмы не рассматриваются.

2.2.4. Алгоритмы роевого интеллекта (Swarm Intelligence algorithms)

При решении задач комбинаторной оптимизации весьма перспективным является использование методов, основанных на моделировании интеллектуального поведения колоний агентов. Алгоритмы роевого интеллекта основаны на коллективном популяционном поведении агентов, в целом рассматриваемых как распределенная самоорганизующаяся система. Данные алгоритмы являются эвристическими итеративными методами случайного поиска.

Муравьиный алгоритм (Ant Colony optimization) – базируется на применении специальных феромонов муравьями-фуражирами, которые позволяют пометить наиболее удачные маршруты [49].

Пчелиный алгоритм (Combinatorial Artificial Bee Colony algorithm) – основан на поведении медоносных пчёл-разведчиков, которые исполняют определенные танцевальные движения при обнаружении хорошего (в определенном смысле) источника нектара. В комбинаторной оптимизации подnectаром подразумевается решение (гамильтонов цикл) [50].

Алгоритм кукушки (Cuckoo Search algorithm) инспирирован паразитическим поведением кукушек, откладывающих свои яйца в гнезда птиц другого вида. Данный алгоритм предусматривает «отсев» решений, неудовлетворяющих определенному критерию, подобно некоторым видам птиц, выбрасывающим яйца кукушки или покидающим свои гнезда [51].

Алгоритм на основе роя частиц (Particle Swarm optimization) – основан на стайном инстинкте животных [52]. Перемещение в стаях подчиняется принципу наилучшего найденного в этом пространстве состояния и постоянно изменяется при нахождении агентами (частицами) улучшенных состояний.

Временные оценки сложности алгоритмов из данной подгруппы составляют $O(N^3)$. Оценки точности не определены [4].

В данной работе применяется один из пчелиных алгоритмов, как самый эффективный из группы алгоритмов роевого интеллекта, согласно [50].

В табл. 1 изложена классификация эвристических алгоритмов решения метрической задачи коммивояжера, описанная в данной главе. Приведены оценки временной сложности и точности алгоритмов.

Выводы по главе

В главе приведена классификация эвристических алгоритмов решения метрической задачи коммивояжера, указаны ссылки на публикации по данной тематике. В первой половине приведена краткая характеристика существующих эвристик построения маршрута. Во второй половине описаны основные эвристики улучшения уже найденных маршрутов, приведены известные подходы. Указаны временные оценки сложности и оценки точности алгоритмов. Отмечены алгоритмы, выбранные для исследования.

Таблица 1.

Классификация эвристических алгоритмов решения метрической задачи коммивояжера

Эвристические алгоритмы			
Алгоритмы построения маршрута (Tour construction methods)			
Название подгруппы	Название алгоритма	Временная оценка сложности, $f_{\varepsilon}^{apr}(m)$	Оценка точности, $f_t^{apr}(m)$
Алгоритмы упорядоченных последовательностей	Clarke and Wright Savings	$O(N^2)$	$\log N$
	Greedy	$O(N^2 \log N)$	$\log N$
Алгоритмы построения маршрута	Nearest Neighbour, Double Ended Nearest Neighbour	$O(N^2)$	$0.5(\lceil \log N + 1 \rceil)$
	Moore Curve Sierpinski Curve	$O(N \log N)$	$\log N$
Алгоритмы вставки в подцикл	Nearest Addition, Nearest Insertion, Farthest Insertion, Arbitrary Insertion	$O(N^2)$	$2 - \frac{2}{N}$
	Cheapest Insertion	$O(N^2 \log N)$	$2 - \frac{2}{N}$
	Convex Hull	$O(N^2 \log N)$?
Алгоритмы объединения подциклов	Nearest merger	$O(N^2)$	$2 - \frac{2}{N}$
	Patching	$O(N^3)$?
Комбинированные алгоритмы построения маршрута	Double MST	$O(N^2)$	$2 - \frac{2}{N}$
	Christofides	$O(N^3)$	$\frac{3}{2}$
Алгоритмы улучшения маршрута (Tour improving methods)			
Локально-оптимальные алгоритмы	2-Opt	$O(N^2)$	≈ 2
	k-Opt	$O(N^k)$	≈ 2
	LKH Heuristic	$\approx O(N^{2,2})$	≈ 2
Алгоритмы имитации отжига	Simulated Annealing	$O(N^2)$?
Эволюционные алгоритмы	Genetic Algorithms, Neural Networks Algorithms	$O(N^3)$?
Алгоритмы роевого интеллекта	Ant Colony optimization, Combinatorial Artificial Bee Colony, Cuckoo Search algorithm, Particle Swarm optimization	$O(N^3)$?

3. Выбранные алгоритмы и особенности реализации

В данной главе приводятся выбранные входные данные, алгоритмы, особенности реализации и структуры данных. В первой части описываются типы входных данных для используемых алгоритмов, их количество, приведен формат тестовых файлов. Во второй части приводятся группы алгоритмов и отдельные алгоритмы приближенного решения метрической задачи коммивояжера, использованные в данной работе. В третьей части указываются особенности реализации алгоритмов, включающие в себя описание используемых структур данных и исследование скорости вычисления расстояния между вершинами.

3.1. Входные данные

В работе используются два типа тестовых данных. Первый тип содержит наборы, имитирующие топологии сверх больших интегральных схем (VLSI Data Sets) [53]. Второй тип состоит из множества географических координат городов в определенных странах (National TSPs) [54].

VLSI Data Sets состоят из 102 наборов вершин, количество точек варьируется от 131 до 744710. Для каждого N есть только один набор вершин. Координаты каждой точки представлены целочисленными значениями. Каждый набор точек записан в отдельном файле формата “*.tsp”. Пример формата представления данных для набора из 395 точек представлен на рис. 3:

```
NAME : pb1395
COMMENT : Bonn VLSI data set with 395 points
COMMENT : Uni Bonn, Research Institute for Discrete Math
COMMENT : Contributed by Andre Rohe
TYPE : TSP
DIMENSION : 395
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 0 15
2 0 22
3 0 29
4 0 42
5 0 43
6 0 44
```

Рисунок 3. Формат файла VLSI Data Sets [53]

Входные данные National TSPs состоят из 25 наборов вершин, количество точек варьируется от 29 до 71009. Для каждого N есть только один набор вершин. Координаты каждой точки представлены двумя вещественными значениями, обозначающими высоту и

широту, с 4 знаками после запятой. Каждый набор точек записан в отдельном файле формата “*.tsp”. Пример формата представления данных для набора из 980 точек представлен на рис. 4.

```
NAME : lu980
COMMENT : 980 locations in Luxembourg
COMMENT : Derived from National Imagery and Mapping Agency data
TYPE : TSP
DIMENSION : 980
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 49525.5556 5940.5556
2 49525.5556 5940.5556
3 49738.8889 6345.0000
4 49608.3333 6405.8333
5 49796.6667 6155.5556
6 49828.6111 5764.7222
```

Рисунок 4. Формат файла National TSP [54]

Для каждого набора данных из VLSI Data Sets и National TSPs есть визуальное представление расположения точек (рис. 5а, рис. 6а) и оптимального цикла (рис. 5б, рис. 6б). Для каждого набора данных указано целочисленное значение оптимального (точно просчитанного) либо псевдо-оптимального решения, на основе которого можно определять априорные точностные оценки эвристических алгоритмов решения метрической задачи коммивояжера. Под псевдо-оптимальным решением понимается наилучшее значение решения, полученное ранее другими исследователями.

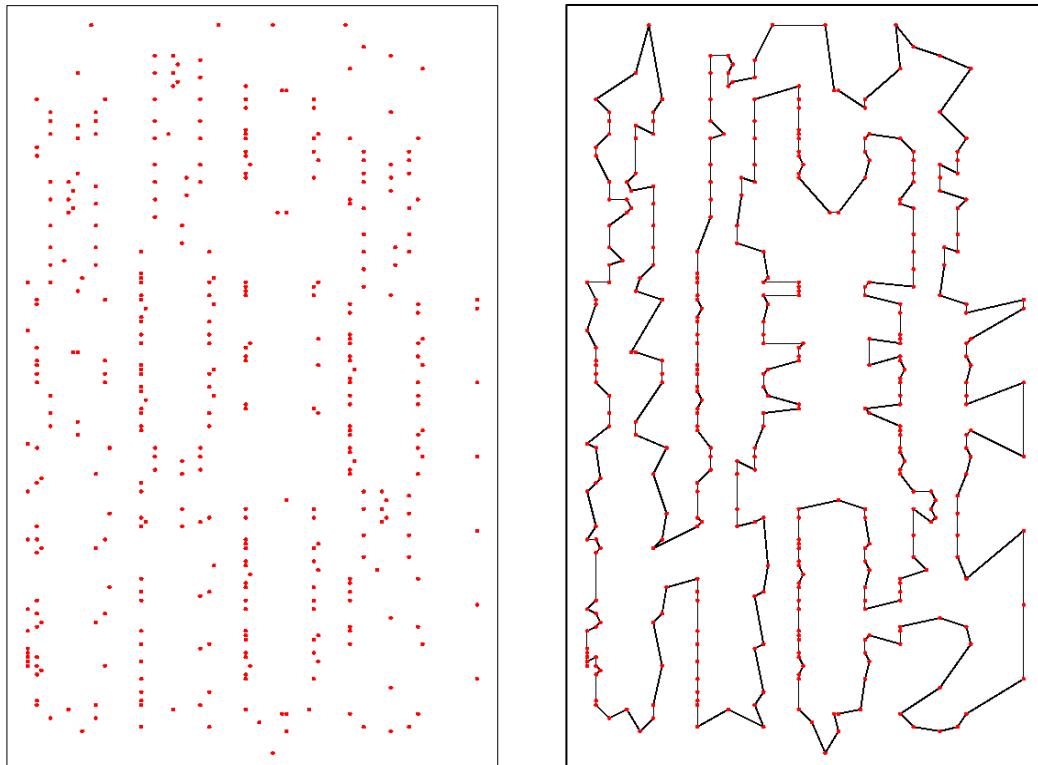


Рисунок 5. VLSI Data Sets: (а) набор точек; (б) оптимальный цикл [53]

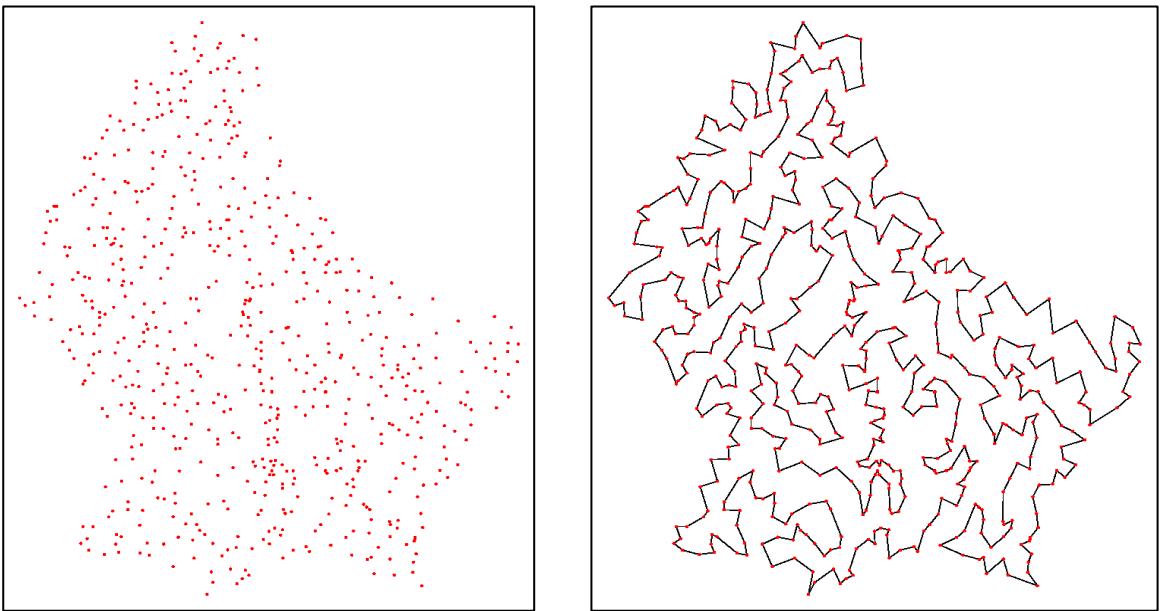


Рисунок 6. National TSPs: (а) набор точек; (б) оптимальный цикл [54]

Расстояние между вершинами рассчитывается одинаково для двух типов входных данных и представляет собой целочисленную евклидову метрику (EUC_2D), с округлением до ближайшего целого. Таким образом, в данном исследовании вместо формулы 2 будет использоваться следующая формула:

$$d_{Euc_2D}(v, w) = \left\lceil \sqrt{|x(v) - x(w)|^2 + |y(v) - y(w)|^2} + 0.5 \right\rceil \quad (4)$$

3.2. Описание приближенных алгоритмов

Ниже приведены алгоритмы приближенного решения метрической задачи коммивояжера, использованные в данной работе.

3.2.1. Алгоритм ближайшего соседа (Nearest Neighbour, NN)

Данный алгоритм относится к группе алгоритмов наращивания пути, описанных в Главе 2 настоящего отчета. NN является базовой и самой интуитивно понятной эвристикой для приближенного решения задачи коммивояжера. Основная идея заключается в поиске ближайшей, ещё не пройденной вершины, относительно последней добавленной [26].

Алгоритм начинается с выбора вершины v_0 и её ближайшего соседа – вершины v_1 , расстояние между которыми минимально среди всех возможных пар вершин. Обе вершины образуют ребро и отмечаются как пройденные. Следующей выбирается ещё не пройденная ранее вершина v_i , ближайшая к последней добавленной вершине v_1 . Степень близости выражается как $\min d(v_i, v_1)$, $i \neq 1$. Между v_i и v_1 добавляется ребро, v_i отмечается как пройденная. Аналогичным образом добавляются остальные вершины. Построение завершается, как только все вершины будут пройдены. Маршрут необходимо замкнуть с целью получения

цикла, для этого на последнем шаге алгоритма добавляется ребро между первой и последней добавленными вершинами.

Временная сложность алгоритма и в лучшем, и в худшем случае составляет $O(N^2)$.

Известно, что данный алгоритм в среднем даёт 25% отклонение от оптимума для N нормально распределенных вершин на плоскости [55]. Однако существует множество наборов вершин, для которых NN найдет худшее решение. Определено, что решение, полученное с помощью NN для набора вершин, удовлетворяющего свойствам метрики, не может быть хуже оптимального более чем в $0.5(\lceil \log_2 N + 1 \rceil)$ раз [26].

Псевдокод алгоритма NN приведен в схеме 1.

```

Input: given array of vertices
1: // find minimum edge
2: for all vertices i
3:   for all vertices j = i + 1
4:     distance = d(i, j)
5:     if (distance < min_distance)
6:       min_distance = distance
7:       first = i
8:       last = j
9: add first and last to the tour
10: for all vertices i not in the tour
11:   // find closest i to last
12:   distance = d(i, last)
13:   if (distance < min_distance)
14:     min_distance = distance
15:     closest = i
16:   add closest to the tour
17:   last = closest
18: // make a cycle C from the tour
19: C = connect first and last
20: return C
Output: cycle C

```

Схема 1. Псевдокод алгоритма NN

3.2.2. Модифицированный алгоритм ближайшего соседа (Double Ended Nearest Neighbour, DENN)

Модификацией алгоритма NN является алгоритм DENN, в котором поиск ближайшей, ещё не пройденной вершины, осуществляется относительно крайних вершин в текущем маршруте [56].

Алгоритм начинается с выбора вершины v_0 и её ближайшего соседа – вершины v_1 , расстояние между которыми минимально среди всех возможных пар вершин. Обе вершины образуют ребро и отмечаются как пройденные. Затем среди всех точек v_2, \dots, v_{n-1} , не

пройденных ранее, определяются вершины v'_0 и v'_1 , ближайшие к вершинам v_0 и v_1 , соответственно. В том случае, если $d(v'_0, v_0) < d(v'_1, v_1)$, то между вершинами v'_0 и v_0 добавляется ребро, и v'_0 вместе с v_1 становятся новыми крайними вершинами маршрута. Иначе ребро добавляется между вершинами v'_1 и v_1 , тогда новыми крайними точками маршрута будут вершины v'_1 вместе с v_0 . Предположим, что на i -той итерации мы имеем маршрут вида $v_0, v_1, v_2, \dots, v_i$. Среди всех ранее не пройденных вершин определяются вершины v'_0 и v'_i , которые являются ближайшими к вершинам v_0 (начальная точка маршрута) и v_i (конечная точка маршрута), соответственно. Выбор вершины для добавления аналогичен описанному ранее для вершин v'_0 и v'_1 . Построение завершается, как только не останется ни одной не пройдённой вершины. Маршрут необходимо замкнуть с целью получения цикла, для этого на последнем шаге алгоритма добавляется ребро между первой и последней добавленными вершинами.

Псевдокод алгоритма DENN приведен в схеме 2.

```

Input: given array of vertices
1: // find minimum edge
2: for all vertices i
3:   for all vertices j = i + 1
4:     distance = d(i, j)
5:     if (distance < min_distance)
6:       min_distance = distance
7:       first = i
8:       last = j
9: add first and last to the tour
10: for all vertices i not in the tour
11:   // find closest i to first
12:   distance = d(i, last)
13:   if (distance < min_distance_to_first)
14:     min_distance_to_first = distance
15:     closest_to_first = i
16:   // find closest i to last
17:   distance = d(i, last)
18:   if (distance < min_distance_to_last)
19:     min_distance_to_last = distance
20:     closest_to_last = i
21:   if (min_distance_to_first <= min_distance_to_last)
22:     closest = closest_to_first
23:     first = closest
24:   else
25:     closest = closest_to_last
26:     last = closest
27:   add closest to the tour
28: // make a cycle C from the tour
29: C = connect first and last

```

```
30: return C
Output: cycle C
```

Схема 2. Псевдокод алгоритма DENN

Временная оценка сложности алгоритма и в лучшем, и в худшем случае также составляет $O(N^2)$. Известно, что данный алгоритм в среднем даёт 25% отклонение от оптимума для N нормально распределенных вершин на плоскости [55]. Однако существует множество наборов вершин, для которых DENN найдет худшее решение. Было определено, что решение, полученное с помощью алгоритма DENN для набора вершин, удовлетворяющего свойствам метрики, будет хуже оптимального не более чем в $0.5(\lceil \log N + 1 \rceil)$ раз [56].

3.2.3. Алгоритм на основе кривой Мура (Moore Curve, MC)

Данный алгоритм относится к группе алгоритмов наращивания пути, описанных в Главе 2 настоящего отчета. МС основан на свойстве заполняющей пространство кривой (Space-Filling Curve) - кривой, область значений которой содержит весь двумерный единичный квадрат. На рис. 7 показана кривая Мура после первого, второго и третьего шага деления. Данная кривая определяет порядок обхода, на основе которого соединяются вершины для решения задачи коммивояжера.

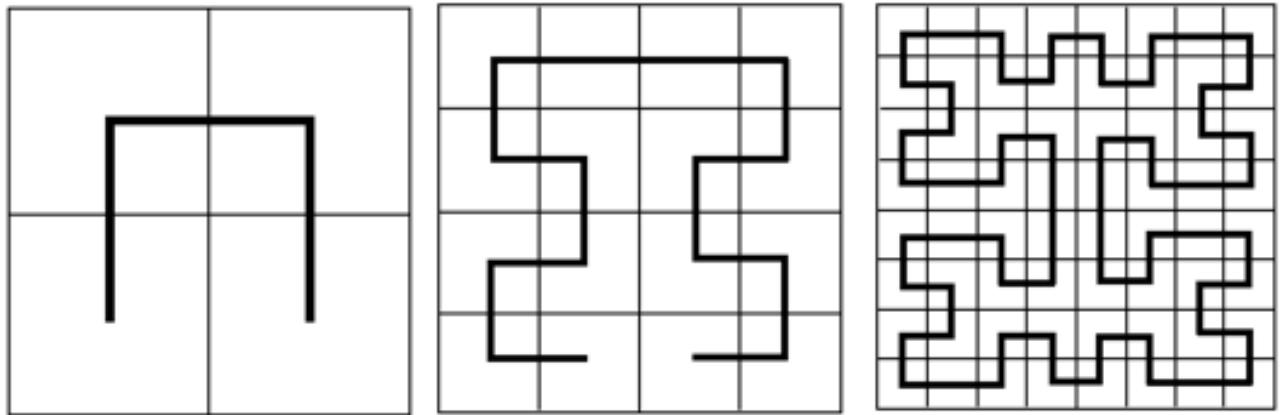


Рисунок 7. Вид кривой Мура после 1, 2 и 3 шагах рекурсивного деления

Построение алгоритма решения метрической задачи коммивояжера на плоскости, основанного на данной кривой, осуществляется в два этапа. На первом шаге для каждой вершины v_i вычисляется её позиция θ вдоль кривой. Индекс θ принадлежит промежутку $[0, 1]$. На втором шаге вершины v_i сортируются по возрастанию θ . Стоит отметить, что θ не означает процентное отношение пройденного пути. Однако сортировка данных индексов позволяет получить необходимую последовательность вершин.

Временная оценка сложности алгоритма составляет $O(N \log N)$. Определено, что для равномерно распределенных вершин оценка точности решения равна 1,25. Во всех других случаях оценка точности равна $\log N$ [57].

Псевдокод определения индекса θ в алгоритме МС приведен в схеме 3.

```

Input: x, y - coordinates of vertex
        max_input - max(x, y) for all vertices
1: array rank[5][4] = {
2:     {1, 0, 2, 3},
3:     {2, 3, 1, 0},
4:     {2, 1, 3, 0},
5:     {0, 3, 1, 2},
6:     {0, 1, 3, 2}
7: };
8: array next_state[5][4] = {
9:     {2, 2, 3, 3},
10:    {1, 3, 1, 2},
11:    {2, 2, 4, 1},
12:    {4, 1, 3, 3},
13:    {3, 4, 2, 4}
14: };
15: result = 0
16: state = 0
17: loop_index = max_input - 1
18: while (loop_index > 0) do
19:     ax = x >= 0.5 ? 1 : 0
20:     ay = y >= 0.5 ? 1 : 0
21:     aq = 2 * ax + ay
22:     result = 4 * result + rank[state][aq]
23:     state = next_state[state][aq]
24:     x = 2 * x - ax
25:     y = 2 * y - ay
26:     loop_index = loop_index - 1
27: return result
Output: index  $\theta$  for the given vertex

```

Схема 3. Псевдокод определения индекса θ в алгоритме МС [58]

3.2.4. Алгоритм на основе кривой Серпинского (Sierpinski Curve, SC)

Данный алгоритм также входит в семейство Space-Filling Curves. Кривая Серпинского является более симметричной по сравнению с кривой Мура. На рис. 8 показана кривая Серпинского после первого, второго и третьего шага деления. На рис. 9 показано, как с помощью кривой Серпинского можно решить метрическую задачу коммивояжера графически.

Построение алгоритма решения метрической задачи коммивояжера на плоскости, основанного на данной кривой, осуществляется в два этапа. На первом шаге для каждой вершины v_i вычисляется её позиция θ вдоль кривой. Индекс θ принадлежит промежутку $[0, 1]$.

На втором шаге вершины v_i сортируются по возрастанию θ . Стоит отметить, что θ не означает процентное отношение пройденного пути. Однако сортировка данных индексов позволяет получить необходимую последовательность вершин.

Псевдокод определения индекса θ в алгоритме SC приведен в схеме 4.

```

Input: x, y - coordinates of vertex
        max_input - max(x, y) for all vertices
1: loop_index = max_input
2: result = 0
3: if (x > y)
4:     result = result + 1
5:     x = max_input - x
6:     y = max_input - y
7: while (loop_index > 0) do
8:     result = result + result
9:     if (x + y > max_input)
10:        result = result + 1
11:        old_x = x
12:        x = max_input - y
13:        y = old_x
14:        x = x + x
15:        y = y + y
16:        result = result + result
17:        if (y > max_input)
18:            result = result + 1
19:            old_x = x
20:            x = y - max_input
21:            y = max_input - old_x
22:        loop_index = loop_index / 2
23: return result
Output: index  $\theta$  for the given vertex

```

Схема 4. Псевдокод определения индекса θ в алгоритме SC [58]

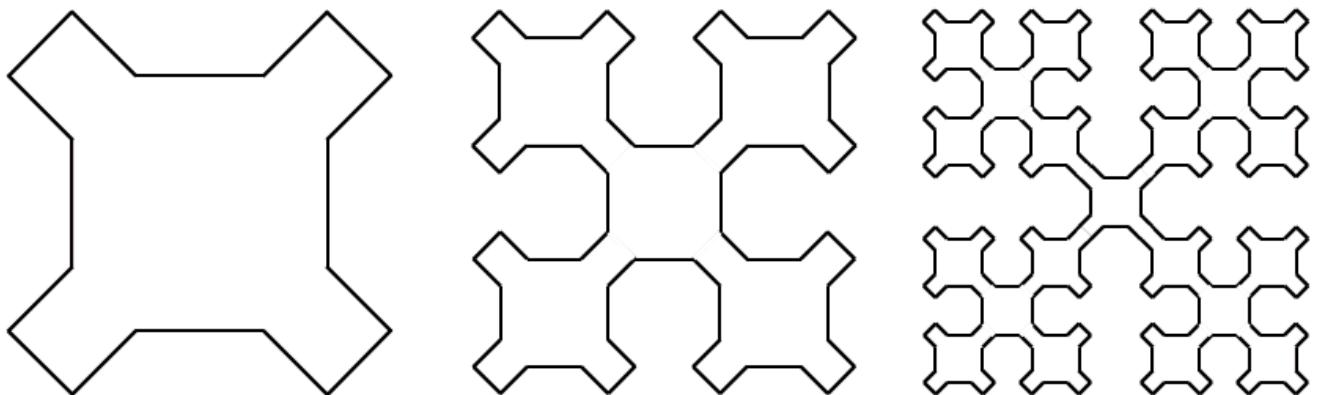


Рисунок 8. Вид кривой Серпинского на 1, 2 и 3 шагах рекурсивного деления

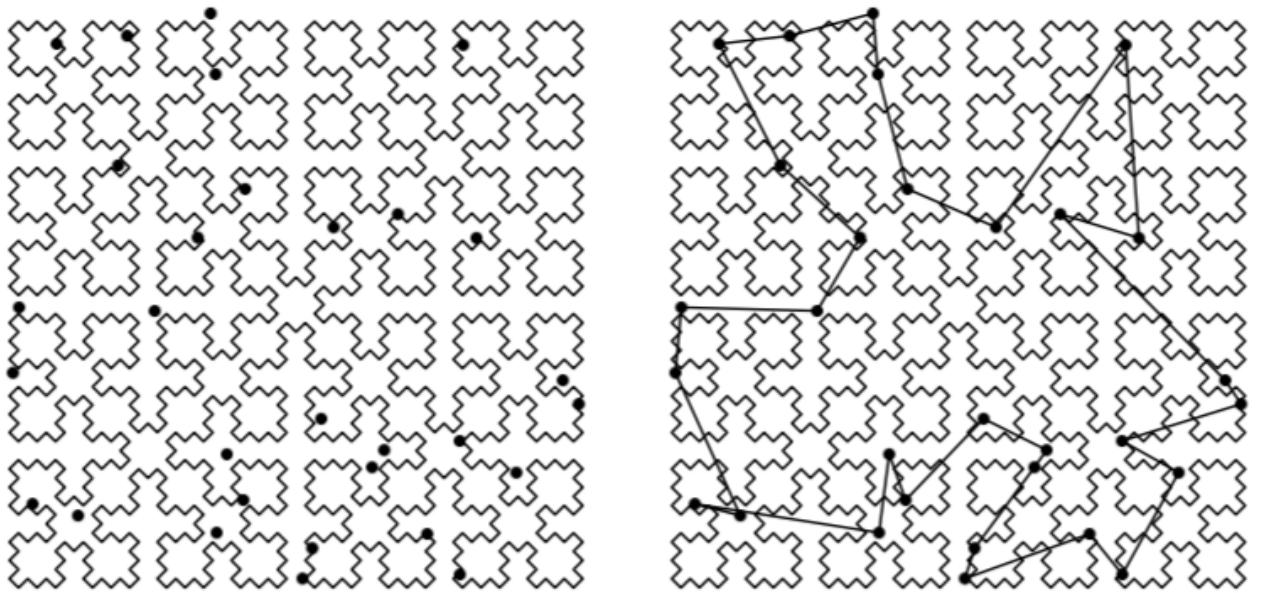


Рисунок 9. Построение гамильтонова цикла с помощью кривой Серпинского [58]

Временная оценка сложности алгоритма составляет $O(N \log N)$. Определено, что для равномерно распределенных вершин оценка точности решения равна 1,25. Во всех других случаях оценка точности равна $\log N$ [57].

3.2.5. Алгоритм добавления «ближайшего» (Nearest Addition, NA)

Данный алгоритм относится к группе алгоритмов вставки в подцикл, описанных в Главе 2 настоящего отчета. Основная идея NA заключается в выборе очередной вершины, не пройденной ранее и являющейся ближайшей к одной из вершин, входящих в маршрут.

Алгоритм начинается с выбора случайной вершины v_0 и её ближайшего соседа – вершины v_1 . Обе вершины образуют цикл и отмечаются как пройденные. Затем для каждой вершины из списка ещё не пройденных v_2, v_3, \dots, v_{N-1} определяется минимальное расстояние до цикла – любой из вершин v_0, v_1 . Вершина v_i , $i = \overline{2..N-1}$, с минимальным расстоянием до цикла, ранее не включенная в него, вставляется перед своим ближайшим соседом, входящим в цикл. Предположим, что на i -той итерации мы имеем маршрут вида $v_0, v_1, v_2, \dots, v_i$, затем среди всех ранее не пройденных вершин определяется вершина v'_j , которая является ближайшей к вершине v_j , $j = \overline{0..i}$. Найденная вершина v'_j отмечается как пройденная и вставляется в цикл перед вершиной v_j , образуя новый цикл $v_0, v_1, \dots, v'_j, v_j, \dots, v_i$. Построение завершается, как только не останется ни одной не пройденной вершины.

Временная сложность алгоритма и в лучшем, и в худшем случае составляет $O(N^2)$. Результат, полученный с помощью NA, для N вершин, удовлетворяющих свойствам метрики, не может быть хуже оптимального более чем в $2 - \frac{2}{N}$ раза.

Псевдокод алгоритма NA приведен в схеме 5.

```

Input: given array of vertices
1: // find minimum edge
2: choose random vertex i
3: for all vertices j, j != i
4:     distance = d(i, j)
5:     if (distance < min_distance)
6:         min_distance = distance
7:         nearest_j = j
8: add i and nearest_j to the cycle C // now C consists of 2 vertices
9: for it = 1 to N - 2
10:    for all vertices i not in the C
11:        for all vertices j in the C
12:            // find nearest i to any vertex in C
13:            distance = d(i, j)
14:            if (distance < min_distance)
15:                min_distance = distance
16:                nearest_vertex = i
17:                nearest_to = j
18:    insert nearest_vertex in C before nearest_to
19: return C
Output: cycle C

```

Схема 5. Псевдокод алгоритма NA

3.2.6. Алгоритм вставки «ближайшего» (Nearest Insertion, NI)

NI является дополненным вариантом алгоритма NA, описанного ранее. Основная идея алгоритма совпадает с NA, однако дополнительно на этапе инициализации в качестве первоначально подцикла выбирается ребро наименьшего веса и на этапе вставки определяется лучшее место для вставки очередной вершины в цикл.

Алгоритм начинается с выбора вершины v_0 и её ближайшего соседа – вершины v_1 , расстояние между которыми минимально среди всех возможных пар вершин. Обе вершины образуют цикл и отмечаются как пройденные. Затем определяется минимальное расстояние до любой из вершин $v_0, v_1, v_2, \dots, v_i$ цикла, построенного на предыдущих итерациях – от каждой вершины из списка ещё не пройденных v_{i+1}, \dots, v_{N-1} с указанием ближайшего соседа $v_j, j = \overline{0..i}$ из цикла. Найденная вершина v'_j отмечается как пройденная и вставляется в цикл, образуя новый – $v_0, v_1, \dots, v'_j, v_j, \dots, v_i$.

Место вставки определяется следующим образом:

- если ближайший сосед для выбранной вершины v'_j не изменился с прошлой итерации, то c'_j вставляется между такими соседними вершинами v_j и v_{j+1} цикла, чтобы минимизировать увеличение длины цикла $\{ \min d(v'_j, v_j) + d(v'_j, v_{j+1}) - d(v_j, v_{j+1}), j = \overline{0..i} \}$;

- иначе выбранная вершина v'_j вставляется до или после своего ближайшего соседа v_j из цикла так, чтобы минимизировать увеличение длины цикла $\{ \min(d(v'_j, v_j) + d(v'_j, v_{j+1}) - d(v_j, v_{j+1}), d(v'_j, v_{j-1}) + d(v'_j, v_j) - d(v_{j-1}, v_j) \}$;

Построение завершается, как только не останется ни одной не пройденной вершины.

Временная сложность алгоритма составляет $O(N^2)$. Результат, полученный с помощью NI, для N вершин, удовлетворяющих свойствам метрики, не может быть хуже оптимального более чем в $2 - \frac{2}{N}$ раза. И, тем не менее, доказано, что существуют такие наборы данных, для которых данный алгоритм может получить наихудшее решение [26].

Псевдокод алгоритма NI приведен в схеме 6.

```

Input: given array of vertices
1: // find minimum edge
2: for all vertices i
3:     for all vertices j = i + 1
4:         distance = d(i, j)
5:         if (distance < min_distance)
6:             min_distance = distance
7:             first = i
8:             last = j
9: add first and last to the cycle C // now C consists of 2 vertices
10: for it = 1 to N - 2
11:     for all vertices i not in the C
12:         for all vertices j in the C
13:             // find nearest i to any vertex in C
14:             distance = d(i, j)
15:             if (distance < min_distance)
16:                 min_distance = distance
17:                 nearest_vertex = i
18:                 nearest_to = j
19:             // find best place for insertion
20:             // a = nearest_vertex (not in C)
21:             // b = nearest_to (in C)
22:             // before_b = vertex in C before nearest_to
23:             // after_b = vertex in C after nearest_to
24:             d1 = d(a, before_b) + d(a, b) - d(before_b, b)
25:             d2 = d(a, after_b) + d(a, b) - d(b, after_b)
26:             if (d1 <= d2)
27:                 insert nearest_vertex in C before nearest_to
28:             else
29:                 insert nearest_vertex in C after nearest_to
30: return C
Output: cycle C

```

Схема 6. Псевдокод алгоритма NI

3.2.7. Алгоритм «выгодной» вставки (Cheapest Insertion, CI)

Данный алгоритм относится к группе алгоритмов вставки в подцикл, описанных в Главе 2 настоящего отчета. Основная идея CI заключается в выборе такой очередной вершины, добавление которой в уже имеющийся цикл даст наименьший прирост его длины.

Алгоритм начинается с выбора вершины v_0 и её ближайшего соседа – вершины v_1 , расстояние между которыми минимально среди всех возможных пар вершин. Обе вершины образуют цикл и отмечаются как пройденные. На каждой очередной итерации it подбираются такие вершины v_i, v_j и v' , такие что v_i и v_j – соседние вершины в построенном цикле, v' – ещё не пройденная вершина и $d(v', v_i) + d(v', v_j) - d(v_i, v_j)$ – минимально для всех возможных пар i и j . Для всех ещё не пройденных вершин v на предыдущем шаге запоминается номер вершины v_i , вставка после которой даст минимальный прирост. Вершина v' отмечается как пройденная и вставляется между вершинами v_i и v_j , образуя новый цикл $v_0, \dots, v_i, v', v_j, \dots, v_{it+1}$. Построение завершается, как только не останется ни одной не пройденной вершины.

Псевдокод алгоритма CI приведен в схеме 7.

```

Input: given array of vertices
1: // find minimum edge
2: for all vertices i
3:     for all vertices j = i + 1
4:         distance = d(i, j)
5:         if (distance < min_distance)
6:             min_distance = distance
7:             first = i
8:             last = j
9: add first and last to the cycle C // now C consists of 2 vertices
10: for it = 1 to N - 2
11:     for all vertices i not in the C
12:         for all vertices j in the C
13:             // find nearest i to any vertex in C so its
14:             // addition in cycle length is minimal
15:             // after_j = vertex in C after j
16:             distance = d(i, j) + d(i, after_j) - d(j, after_j)
17:             if (distance < min_distance)
18:                 min_distance = distance
19:                 nearest_vertex = i
20:                 nearest_to = j
21:             insert nearest_vertex in C after nearest_to
22:         return C
Output: cycle C

```

Схема 7. Псевдокод алгоритма CI

Асимптотическая времененная сложность алгоритма составляет $O(N^2 \log N)$. Результат, полученный с помощью CI, для N вершин, удовлетворяющих свойствам метрики, не может быть хуже оптимального более чем в $2 - \frac{2}{N}$ раза [26].

3.2.8. Алгоритм вставки «далнейшего» (Farthest Insertion, FI)

Данный алгоритм относится к группе алгоритмов вставки в подцикл, описанных в Главе 2 настоящего отчета. Основная идея FI заключается в выборе очередной вершины, не пройденной ранее и являющейся дальнейшей относительно любой вершины, входящей в маршрут. Дополнительно на этапе инициализации в качестве первоначально подцикла выбирается ребро наибольшего веса и на этапе вставки определяется лучшее место для вставки очередной вершины в цикл.

Алгоритм начинается с выбора вершины v_0 и её ближайшего соседа – вершины v_1 , расстояние между которыми максимально среди всех возможных пар вершин. Обе вершины образуют цикл и отмечаются как пройденные. На каждой очередной итерации it подбираются такие вершины v_i и v_j при условии, что v_i входит в построенный подцикл, а v_j – не входит, и расстояние $d(v_i, v_j)$ максимально для всех возможных пар i и j . Для всех ещё не пройденных вершин v_j на предыдущем шаге запоминается номер вершины v_i , вставка после которой даст минимальный прирост, это значит, что расстояние $d(v_i, v_j) + d(v_{i+1}, v_j) - d(v_i, v_{i+1})$ должно быть минимальным для всех возможных пар i и j , где v_{i+1} – соседняя вершина для v_i . Вершина v_j отмечается как пройденная и вставляется между вершинами v_i и v_{i+1} , образуя новый цикл $v_0, \dots, v_i, v_j, v_{i+1}, \dots, v_{it+1}$. Построение завершается, как только не останется ни одной не пройденной вершины.

Асимптотическая времененная сложность алгоритма составляет $O(N^2)$. Результат, полученный с помощью FI, для N вершин, удовлетворяющих свойствам метрики, не может быть хуже оптимального более чем в $2 \log N + 0.16$ раз [26].

Псевдокод алгоритма FI приведен в схеме 8.

```

Input: given array of vertices
1: // find maximum edge
2: for all vertices i
3:   for all vertices j = i + 1
4:     distance = d(i, j)
5:     if (distance > max_distance)
6:       max_distance = distance
7:       first = i
8:       last = j
9: add first and last to the cycle C // now C consists of 2 vertices
10: for it = 1 to N - 2

```

```

11:   for all vertices i not in the C
12:     for all vertices j in the C
13:       // find farthest i to any vertex in C
14:       distance = d(i, j)
15:       if (distance > max_distance)
16:         max_distance = distance
17:         farthest_vertex = i
18:         farthest_to = j
19:   // find best place for insertion
20:   for all vertices j in the C
21:     // after_j = vertex in C after j
22:     distance = d(farthest_vertex, after_j) +
23:               d(farthest_vertex, j) - d(j, after_j)
24:     if (distance < min_distance)
25:       min_distance = distance
26:       nearest_to = j
27:   insert nearest_vertex in C after nearest_to
28: return C
Output: cycle C

```

Схема 8. Псевдокод алгоритма FI

3.2.9. Алгоритм вставки «случайного» (Arbitrary Insertion, AI)

Данный алгоритм относится к группе алгоритмов вставки в подцикл, описанных в Главе 2 настоящего отчета. Основная идея AI заключается в случайном выборе очередной вершины, не пройденной ранее, и добавлении её в цикл наилучшим образом – прирост длины цикла должен быть минимальным.

Алгоритм начинается с выбора вершины v_0 и её ближайшего соседа – вершины v_1 , расстояние между которыми минимально среди всех возможных пар вершин. Обе вершины образуют цикл и отмечаются как пройденные. На каждой очередной итерации it случайнным образом выбирается вершина v_{rand} , не входящая в цикл. Затем среди всех вершин из цикла выбирается такая вершина v_j , вставка после которой вершины v_{rand} даст минимальный прирост, это значит, что расстояние $d(v_{rand}, v_j) + d(v_{rand}, v_{j+1}) - d(v_j, v_{j+1})$ должно быть минимальным для всех возможных j , v_{j+1} – вершина, следующая за v_j в цикле. Вершина v_{rand} отмечается как пройденная и вставляется между вершинами v_j и v_{j+1} , образуя новый цикл $v_0, \dots, v_j, v_{rand}, v_{j+1}, \dots, v_{it+1}$. Построение завершается, как только не останется ни одной не пройденной вершины.

Асимптотическая времененная сложность алгоритма составляет $O(N^2)$. Результат, полученный с помощью AI, для N вершин, удовлетворяющих свойствам метрики, не может быть хуже оптимального более чем в $2 \log N + 0.16$ раз [26].

Псевдокод алгоритма AI приведен в схеме 9.

```

Input: given array of vertices
1: // find minimum edge
2: for all vertices i
3:     for all vertices j = i + 1
4:         distance = d(i, j)
5:         if (distance < min_distance)
6:             min_distance = distance
7:             first = i
8:             last = j
9: add first and last to the cycle C // now C consists of 2 vertices
10: rand = random vertex not in the C
11: // find best place for insertion
12: for all vertices j in the C
13:     // after_j = vertex in C after j
14:     distance = d(rand, after_j) + d(rand, j) - d(j, after_j)
15:     if (distance < min_distance)
16:         min_distance = distance
17:         nearest_to = j
18: insert rand in C after nearest_to
19: return C
Output: cycle C

```

Схема 9. Псевдокод алгоритма AI

3.2.10. Алгоритм «ближайшего» до отрезка (Nearest Segment Insertion, NSI)

Данный алгоритм относится к группе алгоритмов вставки в подцикл, описанных в Главе 2 настоящего отчета. Этапы инициализации и вставки алгоритмов NSI и CI совпадают. Авторской модификацией является определение очередной вершины для вставки в подцикл.

Алгоритм начинается с выбора вершины v_0 и её ближайшего соседа – вершины v_1 , расстояние между которыми минимально среди всех возможных пар вершин. Обе вершины образуют цикл и отмечаются как пройденные. На каждой очередной итерации подбираются такие вершины v_i, v_j и v' , что v_i и v_j – соседние вершины в построенном цикле, являющиеся концами отрезка Z , v' – ещё не пройденная вершина и $d(v', Z)$ – минимально для всех возможных пар i и j . Для всех ещё не пройденных вершин v на предыдущем шаге запоминается номер вершины v_i – начало отрезка, расстояние до которого минимально. Вершина v' отмечается как пройденная и вставляется между вершинами v_i и v_j , образуя новый цикл $v_0, \dots, v_i, v', v_j, \dots, v_{it+1}$.

Место вставки определяется следующим образом:

- если ближайший сосед для выбранной вершины v' не изменился с прошлой итерации, то v' вставляется между такими соседними вершинами v_i и v_j цикла, чтобы минимизировать расстояние D от v' до отрезка Z , образованного вершинами v_i и v_j $\{D = \min d(v', Z), i, j \in \text{цикл}\}$. В том случае, когда выбор D не однозначен

- необходимо выбрать тот отрезок D_i , вставка вершины v' между концами которого даст минимальный прирост длины цикла;
- иначе место вставки выбранной вершины v' определяется её ближайшим соседом v_i из цикла. Вершина вставляется до v_i , если расстояние D_1 от v' до отрезка Z_1 , образованного вершинами v_i и v_{i-1} , меньше расстояния D_2 от v' до отрезка Z_2 , образованного вершинами v_i и v_{i+1} . Иначе вершина вставляется после v_i . В общем случае – $\{\min(d(v', Z_1), d(v', Z_2))\}$. В том случае, когда $D_1 = D_2$ – необходимо выбрать тот отрезок D_i , вставка вершины v' между концами которого даст минимальный прирост длины цикла.

Построение завершается, как только не останется ни одной не пройденной вершины.

Временная сложность алгоритма и в лучшем, и в худшем случае составляет $O(N^2)$.

Псевдокод алгоритма NSI приведен в схеме 10.

```

Input: given array of vertices
1: // find minimum edge
2: for all vertices i
3:     for all vertices j = i + 1
4:         distance = d(i, j)
5:         if (distance < min_distance)
6:             min_distance = distance
7:             first = i
8:             last = j
9: add first and last to the cycle C // now C consists of 2 vertices
10: for it = 1 to N - 2
11:     for all vertices i not in the C
12:         for all vertices j in the C
13:             // find nearest i to any segment in C
14:             // after_j = vertex in C after j
15:             distance = find_dist_to_segment(j, after_j, i)
16:             if (distance < min_distance)
17:                 min_distance = distance
18:                 vertex_to_insert = i
19:             // find best place for insertion
20:             for all vertices j in the C
21:                 // after_j = vertex in C after j
22:                 distance = d(vertex_to_insert, after_j) +
23:                             d(vertex_to_insert, j) - d(j, after_j)
24:                 if (distance < min_distance)
25:                     min_distance = distance
26:                     nearest_to = j
27:             insert vertex_to_insert in C after nearest_to
28: return C
Output: cycle C

```

Схема 10. Псевдокод алгоритма NSI

Псевдокод алгоритма нахождения расстояния от вершины С до отрезка АВ приведен в схеме 11.

```

Procedure find_dist_to_segment(A, B, C)
Input: a_x, a_y - coordinates of start vertex A in segment
        b_x, b_y - coordinates of end vertex B in segment
        c_x, c_y - coordinates of vertex C not in cycle
        max_input - max(x, y) for all vertices
1: v_x = b_x - a_x;
2: v_y = b_y - a_y;
3: w_x = c_x - a_x;
4: w_y = c_y - a_y;
5: // dot(w,v) ((w).x * (v).x + (w).y * (v).y)
6: c1 = w_x * v_x + w_y * v_y;
7: if (c1 <= 0)
8:     return d(C, A);
9: c2 = v_x * v_x + v_y * v_y;
10: if (c2 <= c1)
11:     return d(C, B);
12: b = (double)c1 / c2;
13: new_x = (int)(A.x + b * v_x + 0.5)
14: new_y = (int)(A.y + b * v_y + 0.5)
15: vertex Pb = (new_x, new_y)
16: return d(C, Pb);
Output: length from point C to segment AB

```

Схема 11. Псевдокод алгоритма нахождения расстояния от вершины до отрезка

3.2.11. Жадный алгоритм (Greedy, GRD)

GRD относится к классу жадных алгоритмов, входящих в группу алгоритмов упорядоченных последовательностей, описанных в Главе 2 настоящего отчета. Основная идея заключается в последовательном добавлении отсортированных в порядке возрастания ребер, не увеличивающих степень ни одной из вершин до 3 и не образующих подциклов.

Алгоритм начинается с сортировки всех потенциальных ребер в порядке возрастания их длин. Затем выбирается кратчайшее ребро и добавляется в список ребер, составляющих будущий цикл, при условии, что оно не нарушает условий, описанных выше. Процедура повторяется, пока маршрут не будет состоять из N ребер. Очевидно, что одно ребро не может быть добавлено более одного раза.

Временная сложность алгоритма составляет $O(N^2 \log N)$. Аналогично NN и DENN доказано, что решение алгоритмом Greedy для набора вершин, удовлетворяющего свойствам метрики, будет хуже оптимального не более чем в $0.5(\lceil \log_2 N + 1 \rceil)$ раз [26].

Псевдокод алгоритма GRD приведен в схеме 12.

```
Input: given array of vertices
1: edges = find all edges (lengthes + indices)
2: sort edges in ascent way (from shortest to largest)
3: for counter = 1 to N
4:     // look for suitable edge
5:     for all edges e
6:         bool_1 = all vertices from e has degree <= 1 ? 1 : 0
7:         bool_2 = addition of e does not make any cycle less N ? 1 : 0
8:         if (bool_1 and bool_2)
9:             add e
10:            break
11:
12: return C
Output: cycle C
```

Схема 12. Псевдокод алгоритма GRD

3.2.12. Алгоритм на основе удвоенного минимального оствовного дерева (Double Minimal Spanning Tree, DMST)

Данный алгоритм относится к группе комбинированных алгоритмов построения маршрута, описанных в Главе 2 настоящего отчета. DMST предложен Кристофидесом в 1974 году [37]. Алгоритм состоит из следующих шагов:

1. Построить минимальное оствовное дерево T для полного графа $G(V, V^2)$.
2. Построить ориентированный граф T' на основе дерева T путем замены каждого ребра из T парой противоположно направленных дуг.
3. Построить в графе T' эйлеров цикл E .
4. Осуществить приведение к решению задачи коммивояжера – удалить из E все повторяющиеся вершины «жадным» способом – начиная с первого элемента в эйлеровом цикле, исключать ранее встречающиеся вершины.

Временная сложность алгоритма DMST равна $O(N^2)$. Результат, полученный с помощью DMST, для N вершин, удовлетворяющих свойствам метрики, не может быть хуже оптимального более чем в $2 - \frac{2}{N}$ раза [26].

3.2.13. Улучшенный алгоритм на основе удвоенного минимального оствовного дерева (Double Minimal Spanning Tree Modified, DMST-M)

DMST-M является авторской модификацией алгоритма DMST. Основным отличием между этими двумя алгоритмами является 4 шаг алгоритма DMST. Улучшенный вариант алгоритма требует удаления из эйлерова цикла E всех повторяющихся вершин с помощью аксиомы неравенства треугольника. На каждой очередной итерации it подбираются такие

подряд идущие в E вершины v_i, v_j и v_k , что число вхождений вершины v_j больше одного, и удаление вершины v_j приведет к наибольшему улучшению длины текущего эйлерова цикла $\{\max d(v_i, v_j) + d(v_k, v_j) - d(v_i, v_k)\}$.

Временная сложность алгоритма DMST-M равна $O(N^2)$. Результат, полученный с помощью DMST-M, для N вершин, удовлетворяющих свойствам метрики, не может быть хуже оптимального более чем в $2 - \frac{2}{N}$ раза [4].

3.2.14. Алгоритм на основе минимальных паросочетаний вершин нечетной степени минимального оствового дерева (Christofides algorithm, CHR)

Данный алгоритм также относится к группе комбинированных алгоритмов построения маршрута, описанных в Главе 2 настоящего отчета. CHR предложен Кристофидесом на базе алгоритма на основе удвоенного минимального оствового дерева в 1976 году [38]. Алгоритм состоит из следующих шагов:

1. Построить минимальное оствовое дерево T для полного графа $G(V, V^2)$.
2. Пусть O – множество вершин нечетной степени дерева T . Согласно лемме о рукопожатиях $|O| = 2k, k \in Z$. Таким образом, число вершин нечетной степени будет четным.
3. Определить паросочетание минимального веса P среди всех вершин из O . Паросочетанием минимального веса графа G называется множество

$$P_0 = \arg \min_{P \in P(G)} d(P),$$

где $P(G)$ – множество всех паросочетаний графа G ;

$d(P)$ – вес паросочетания P , равный сумме весов ребер из которого состоит данное паросочетание.

4. Построить мультиграф H , являющийся объединением ребер P и дерева T . Все вершины H будут иметь четные степени.
5. Построить в мультиграфе H эйлеров цикл E .
6. Осуществить приведение к гамильтонову циклу путем удаления из E всех повторяющихся вершин «ожадным» способом – исключить из эйлерового цикла все ранее встречающиеся вершины, начиная с первой.

Временная сложность алгоритма CHR равна $O(N^3)$. Результат, полученный с помощью CHR, для N вершин, удовлетворяющих свойствам метрики, не может быть хуже оптимального более чем в $\frac{3}{2}$ раза [26].

3.2.15. Алгоритм 2-Opt

Данный алгоритм относится к группе локально-оптимальных эвристических алгоритмов решения метрической задачи коммивояжера, описанных в Главе 2 настоящего отчета. Он сводится к удалению одной пары рёбер и вставке новой пары, если сумма длин новых ребер меньше суммы длин старых (рис. 10). Цикл сводится к локальному минимуму, если в нём не существует ни одной пары рёбер, замена которых привела бы к улучшению решения.

Временная сложность алгоритма 2-Opt равна $O(N^2)$ при заданном максимальном количестве итераций. Согласно [4] оптимально применять не более 1000 итераций для достижения приемлемого времени работы алгоритмов при больших размерностях ($N \geq 5\,000$).

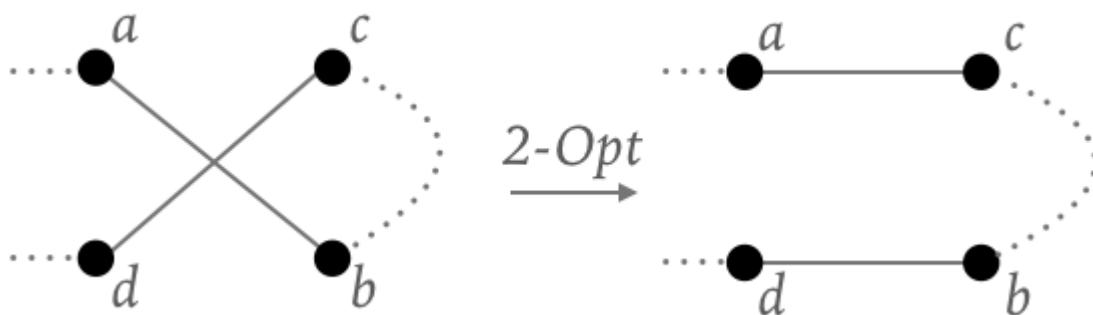


Рисунок 10. Итерация алгоритм 2-Opt

Псевдокод алгоритма 2-Opt приведен в схеме 13.

```

Input: given cycle C of vertices
1: limit = 0
2: while (no more intersections or limit < const)
3:   for each vertex a
4:     for each vertex c
5:       // b - vertex after a
6:       // d - vertex after c
7:       if (d(a, b) + d(c, d) > d(a, c) + d(b, d))
8:         // replace two edges
9:         reverse order of vertices from b to c
10:        break
11:    limit = limit + 1
12: return C
Output: cycle C

```

Схема 13. Псевдокод алгоритма 2-Opt

3.2.16. Алгоритм LKH (Helsgaun's Lin and Kernighan algorithm, LKH)

Алгоритм LKH относится к составным эвристическим алгоритмам решения метрической задачи коммивояжера. Алгоритм основывается на обменах (ребер), позволяющих получить другое решение (перестановку) на базе существующего. Суть алгоритма заключается

в нахождении такого множества ребер, перестановка которых позволит получить решение меньшего веса.

В создании первоначального цикла используется алгоритм построения маршрута NN. LKH обобщает алгоритм 2-Opt, используя метод k -Opt, который подразумевает обмен k ребер, где $k = \overline{2.. \sqrt{N}}$:

Пусть C – текущий цикл. На каждой итерации определяются два множества ребер $X = \{x_1, x_2, \dots, x_k\}$ (входящих в цикл) и $Y = \{y_1, y_2, \dots, y_k\}$ (не входящих в цикл). Основное условие – в результате удаления ребер из множества X и последующего добавления ребер из множества Y должен получиться новый замкнутый цикл C' , при условии $f(C) < f(C')$.

На рис. 11 показана одна итерации алгоритма 3-Opt. Были удалены ребра из множества $X = \{x_1, x_2, x_3\}$ и добавлены новые ребра из множества $Y = \{y_1, y_2, y_3\}$.

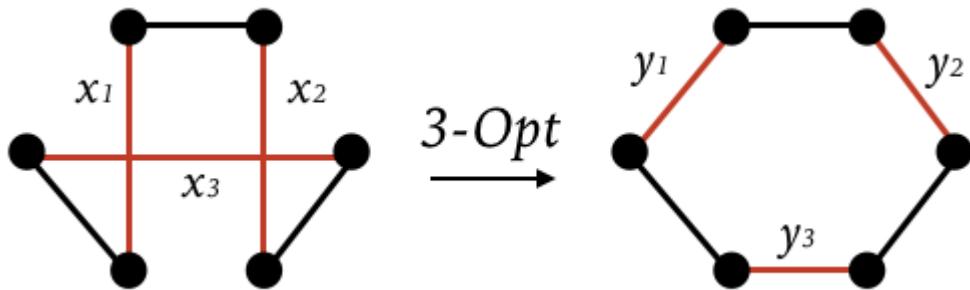


Рисунок 11. Итерация алгоритма 3-Opt

В данной работе данный алгоритм является единственным, который не был реализован лично автором ВКР. Готовая реализация и все пояснения к алгоритму находятся в открытом доступе на сайте автора данного метода [42].

3.2.17. Улучшенный алгоритм поисковой оптимизации, инспирированный роем пчёл (quick Combinatorial Artificial Bee Colony algorithm, qCABC)

Данный алгоритм относится к группе алгоритмов роевого интеллекта, описанных в Главе 2 настоящего отчета. Алгоритм qCABC [50] предполагает разделение всего улья на пчёл, закрепленных в одной из следующих трех категорий:

- пчелы-разведчики (scouts) – осуществляют случайный поиск участка сnectаром;
- занятые пчёлы (employed bees) – хранят информацию об источнике нектара, делятся полученной информацией с пчёлами-наблюдателями
- пчёлы-наблюдатели (onlooker bees) – получают информацию от занятых пчёл и исследуют соседние участки у выбранного нектара.

Алгоритм qCABC заключается в выполнении следующих процедур.

1. Инициализация.

В качестве первоначального цикла используется цикл, построенный с помощью одного из алгоритмов построения маршрута (tour construction methods), описанных ранее.

2. Работа занятых пчел.

На данном этапе каждый агент на основе определенного опыта генерирует новое решение, похожее на текущее, и оценивает его. В случае если длина полученного цикла оказалась меньше исходного, то полученное решение запоминается, а старое забывается.

3. Работа пчел-наблюдателей.

На данном этапе вычисляются вероятностные оценки и фитнес-функции, которые позволяют определить степень качества каждого из найденных решений. От этого зависит направления дальнейших «поисков» улучшений. В природе данная фаза отражается в выполнении занятыми пчелами танцевальных движений по возвращению в улей. Подобными движениями пчелы предоставляют информацию другим пчелам об обнаруженных нектарах, на основании которой другие пчелы-наблюдатели решают, какую область для поисков им стоит выбрать.

4. Работа пчел-разведчиков.

Если полезность источника нектара не может быть улучшена и количество попыток, рассчитанных на улучшение, превышает предельно допустимое значение, то данный источник нектара покидается пчелой-разведчиком. Затем происходит инициализация нового источника нектара.

Псевдокод алгоритма qCABC приведен в схеме 14.

```
Input: given array of vertices
1: initialization
2: while not stop_criteria do
3:     employed bees phase
4:     onlooker bees phase
5:     scouts phase
```

Схема 14. Псевдокод алгоритма qCABC

В табл. 2 приведены оценки точности и временной сложности исследуемых в работе алгоритмов.

Таблица 2.

Оценки точности и временной сложности применяемых алгоритмов

№	Название	Оценка точности, $f_t^{apr}(m)$	Временная оценка сложности, $f_\varepsilon^{apr}(m)$
1	NN	0.5 $\lceil \log_2 N + 1 \rceil$	$O(N^2)$
	DENN		
2	GRD	0.5 $\lceil \log_2 N + 1 \rceil$	$O(N^2 \log N)$
3	NA	$2 - \frac{2}{N}$	$O(N^2)$
	NI		
	NSI		
	DMST		
	DMST-M		
4	FI	$2 \log N + 0.16$	$O(N^2)$
	AI		
5	CI	$2 - \frac{2}{N}$	$O(N^2 \log N)$
6	CHR	$\frac{3}{2}$	$O(N^3)$
7	2-Opt	≈ 2	$O(N^2)$
8	LKH	≈ 2	$O(N^{2.2})$
9	MC	$\log N$	$O(N \log N)$
	SC		
10	qCABC	?	$O(N^3)$

3.3. Особенности реализации

3.3.1. Волновая функция как структура хранения данных

С целью улучшения временных характеристик алгоритмов (уменьшения времени работы) принято решение в качестве структуры данных использовать волновую функцию, в которой гамильтонов цикл представляется в виде одномерного массива *wave* размера N [59]. Основная идея заключается в том, что каждая вершина v_i с уникальным порядковым номером $i = \overline{0..N-1}$ имеет ссылку на вершину v_j с уникальным порядковым номером $j = \overline{0..N-1}, j \neq i$. Данное правило кодируется как $wave[i] = j$.

Волновая функция используется при реализации следующих алгоритмов:

- NN;
- DENN;
- NA;
- NI;
- CI;
- FI;
- AI;

- NSI;
- DMST;
- DMST-M;
- CHR;
- 2-Opt;
- qCABC.

Введение такой структуры данных позволяет сократить время вычисления указанных алгоритмов решения метрической задачи коммивояжера по сравнению со стандартным контейнером $std::vector<T>$ языка C++. Сравнение времен работы алгоритмов 2-opt, NN и DENN с использованием двух вышеперечисленных способов хранения гамильтонова цикла отражено в Приложении А. На графиках по оси абсцисс отложено количество вершин в каждом из наборов данных, по оси ординат время работы алгоритма в секундах с использованием одной из структур данных. В ходе экспериментов были просчитаны значения, полученные на основе всех наборов данных из базы VLSI Data Sets (от 131 до 744710 вершин). Для наглядности на графиках приведена только часть результатов. В целом, определено, что использование волновой функции позволяет сократить время работы алгоритмов NN и DENN в среднем на 28% и 36%, соответственно. Для алгоритма 2-Opt улучшение по всей базе наборов в среднем составляет 62%.

3.3.2. Вычисление целочисленного значения квадратного корня

Для целочисленной евклидовой метрики EUC_2D необходимо вычислять квадратный корень, округленный до ближайшего целого, согласно формуле (4). При выполнении данной работы было дополнительно проведено сравнительное исследование существующих реализаций вычисления квадратного корня с последующим округлением результата до ближайшего целого. Сравнительная таблица скорости вычисления для 10 реализаций отображена в Приложении Б. В качестве входных данных используется набор FQM5087 из базы VLSI Data Sets, количество вершин в котором равно 5087. Каждый алгоритм запускался по 11 раз. Время работы алгоритма в ходе первого запуска не учитывалось ввиду особенностей языка разработки C++. Время работы в ходе оставшихся десяти запусков усреднялось.

Лучшие временные характеристики были достигнуты с помощью встроенного в C++ алгоритма вычисления квадратного корня из вещественного числа $std::sqrt()$ с последующим округлением до ближайшего целого.

3.3.1. Алгоритм GRD

Во время работы GRD для каждой вершины необходимо хранить индекс поддерева, к которому она принадлежит на каждой итерации алгоритма. В простейшей реализации

принадлежность вершины к тому или иному поддереву хранится просто с помощью массива, в котором для каждой вершины хранится номер дерева, к которому она принадлежит. Для каждого ребра за $O(1)$ можно определить, принадлежат ли его вершины к разным деревьям. Наконец, объединение двух деревьев осуществляется за $O(N)$ простым проходом по массиву. Учитывая, что всего операций объединения $N - 1$, асимптотика работы с поддеревьями составляет $O(N^2)$.

При выполнении работы была реализована структура данных «Система непересекающихся множеств» (Disjoint Set Union, DSU), позволяющая получить константное время работы с поддеревьями $O(1)$, что теоретически снижает мультипликативную константу временной сложности алгоритма [60] [61].

Базовый интерфейс данной структуры данных состоит всего из трёх операций:

- `make_set(x)` — добавляет новый элемент x , помещая его в новое множество, состоящее из одного него.
- `union_sets(x, y)` — объединяет множество, в котором находится элемент x , и множество, в котором находится элемент y .
- `find_set(x)` — возвращает, в каком множестве находится указанный элемент x .

Полная реализация на языке C++ находится в Приложении В.

Выводы по главе

В первой части главы описаны типы и количество применяемых входных данных. Во второй части приведены алгоритмы приближенного решения метрической задачи коммивояжера, использованные в данной работе. В третьей части указаны особенности реализации алгоритмов, включающие в себя описание используемых структур данных и исследование скорости вычисления расстояния между вершинами.

4. Организация экспериментального исследования

В данной главе описываются цель и план экспериментального исследования, приводится описание аппаратного и программного окружения. Указываются временные ограничения, приводится обоснование количества запусков каждого алгоритма, определяются наборы данных для выявления Парето-оптимальных алгоритмов.

Цель эксперимента – определить Парето-оптимальные алгоритмы решения метрической задачи коммивояжера на основе баз данных VLSI Data Sets и National TSPs.

Для исследования полученных алгоритмов была написана программа на языке C++ (код программы изложен в Приложении Г).

Тестирование проводилось на ноутбуке Macbook Air “13 (4 ГБ ОЗУ, 128 ГБ SSD, Core i5 Dual-core 1.4 GHz) с использованием Microsoft Visual Studio Professional 2015.

Эксперимент начинается с фиксации определенного алгоритма построения маршрута m , к которому относится один из алгоритмов NN, DENN, MC, SC, NA, NI, CI, FI, AI, NSI, DMST, DMST-M или CHR. Затем выбирается один из алгоритмов улучшения маршрута m' (2-Opt, qCABC, LKH). Далее выбирается база наборов данных DT – VLSI Data Sets или National TSPs. После определения DT считывается один из файлов D, содержащих одну проблему для решения.

На первой стадии для каждого D алгоритм m запускается 11 раз. На каждой итерации i , начиная со второй, вычисляются оценка точности $f_{\varepsilon_i}(m, D)$ и время работы алгоритма $f_{t_i}(m, D)$. На первой итерации осуществляется прогон «вхолостую», что объясняется особенностями компилятора языка C++. После 11 запусков полученные временные оценки усредняются $f_{t_{avg}}(m, D)$, среди оценок точности выбирается лучшая $f_{\varepsilon_{min}}(m, D)$. Одновременно с этим определяется стандартное отклонение среди полученных 10 временных оценок $\sigma(f_{t_i}(m, D))$, что позволяет определить разброс значений в представленном множестве. Также запоминается перестановка s_0 , для которой было получено лучшее решение.

На второй стадии для каждого D алгоритм m' также запускается 10 раз. В качестве первоначальной перестановки используется s_0 , полученное на первой стадии. Аналогично с первой стадией, на каждой итерации, начиная со второй, вычисляются оценка точности $f_{\varepsilon_i}(s_0 + m', D)$ и время работы алгоритма $f_{t_i}(s_0 + m', D)$. Затем полученные временные оценки усредняются $f_{t_{avg}}(s_0 + m', D)$, среди оценок точности выбирается лучшая $f_{\varepsilon_{min}}(s_0 + m', D)$, определяется стандартное отклонение среди полученных 10 временных оценок $\sigma(f_{t_i}(s_0 + m', D))$.

После того, как для всех D одного DT получены результаты, вычисляются:

- $E(f_{\varepsilon_{min}}(m, D))$, $E(f_{\varepsilon_{min}}(s_0 + m', D))$ – математическое ожидание среди всех оценок точности для алгоритмов m и m' , соответственно, внутри одного DT.
- $\sigma(f_{\varepsilon_{min}}(m, D))$, $\sigma(f_{\varepsilon_{min}}(s_0 + m', D))$ – стандартное отклонение среди всех оценок точности для алгоритмов m и m' , соответственно, внутри одного DT.
- $\max(f_{\varepsilon_{min}}(m, D))$, $\max(f_{\varepsilon_{min}}(s_0 + m', D))$ – максимальное значение среди всех оценок точности для алгоритмов m и m' , соответственно, внутри одного DT.
- $\min(f_{\varepsilon_{min}}(m, D))$, $\min(f_{\varepsilon_{min}}(s_0 + m', D))$ – минимальное значение среди всех оценок точности для алгоритмов m и m' , соответственно, внутри одного DT.

План выполнения эксперимента представлен в виде алгоритма, приведенного в схеме 15:

```
Input: Algorithms, input datasets (VLSI Data Sets, National TSPs)
1: foreach tour construction and composite algorithm m
2:   foreach tour improving algorithm m'
3:     foreach dataset type DT from input datasets
4:       foreach dataset D form DT
5:         for i ∈ {1...11} // tour construction stage
6:           solution s = run algorithm m on D
7:           if (i > 1)
8:             calculate  $f_{\varepsilon_i}(m, D)$ ,  $f_{t_i}(m, D)$ 
9:           calculate  $f_{\varepsilon_{min}}(m, D)$ 
10:          remember best solution  $s_0$ 
11:          calculate  $\sigma(f_{t_i}(m, D))$ 
12:          calculate  $f_{t_{avg}}(m, D) = \frac{f_{t_1}(m, D) + \dots + f_{t_{10}}(m, D)}{10}$ 
13:          if (m is composite) continue
14:            for i ∈ {1...11} // improvement stage on  $s_0$ 
15:              solution s = run algorithm m' on D
16:              if (i > 1)
17:                calculate  $f_{\varepsilon_i}(s_0 + m', D)$ ,  $f_{t_i}(s_0 + m', D)$ 
18:              calculate  $f_{\varepsilon_{min}}(s_0 + m', D)$ 
19:              calculate  $\sigma(f_{t_i}(s_0 + m', D))$ 
20:              calculate  $f_{t_{avg}}(s_0 + m', D) = \frac{f_{t_1}(s_0 + m', D) + \dots + f_{t_{10}}(s_0 + m', D)}{10}$ 
21:              calculate  $E(f_{\varepsilon_{min}}(m, D))$ ,  $E(f_{\varepsilon_{min}}(s_0 + m', D))$  for all D
22:              calculate  $\sigma(f_{\varepsilon_{min}}(m, D))$ ,  $\sigma(f_{\varepsilon_{min}}(s_0 + m', D))$  for all D
23:              calculate  $\max(f_{\varepsilon_{min}}(m, D))$ ,  $\max(f_{\varepsilon_{min}}(s_0 + m', D))$  for all D
24:              calculate  $\min(f_{\varepsilon_{min}}(m, D))$ ,  $\min(f_{\varepsilon_{min}}(s_0 + m', D))$  for all D
```

Схема 15. План выполнения эксперимента

Введено ограничение на длительность вычислений для одного алгоритма, равное 11 800 секундам. Таким образом, любой алгоритм не может исполняться более 3 часов и 20 минут. Это означает, что время эксперимента для одного алгоритма не может превышать $11\,800 \text{ с} * 11 \text{ запусков} = 129\,800 \text{ с} = 36,05 \text{ ч} \approx 1,5 \text{ суток}$.

Стоит отметить, что количество значимых запусков каждого алгоритма $N_{runs} = 10$ выбрано на основе расчета доверительного интервала $\left(\bar{f}_t - 1.96 \frac{\sigma(f_t)}{\sqrt{N_{runs}}}; \bar{f}_t + 1.96 \frac{\sigma(f_t)}{\sqrt{N_{runs}}}\right)$ для средних временных оценок \bar{f}_t при известном стандартном отклонении $\sigma(f_t)$. Здесь $1.96 = Z_{1-\frac{\alpha}{2}}$ при уровне значимости доверительного интервала 95% (табл. 3). Ширина полученных доверительных интервалов Δf_t мала, что свидетельствует о том, что значения в множестве сгруппированы вокруг среднего значения. Полученные экспериментальные результаты значений временных оценок можно считать репрезентативными.

Таблица 3.
Стандартное отклонение времени работы алгоритмов для 10 запусков при $N = 10150$

№	Название алгоритма	$\bar{f}_t = E(f_t)$	$\sigma(f_t) = \sqrt{E((f_t - \bar{f}_t)^2)}$	$f_t = \bar{f}_t \pm 1.96 \frac{\sigma(f_t)}{\sqrt{N_{runs}}}$
1.	NSI + 2-Opt	993,582	2,362	(992,118; 995,046)
2.	NI + 2-Opt	1120,140	1,962	(1118,924; 1121,356)
3.	CHR + 2-Opt	1552,690	1,934	(1551,491; 1553,889)
4.	LKH	221,195	1,874	(220,033; 222,356)
5.	DMST-M + 2-Opt	653,006	1,764	(652,006; 654,006)
6.	MC + 2-Opt	373,806	1,654	(372,381; 374,831)
7.	DENN + 2-Opt	367,586	1,481	(366,668; 368,504)
8.	GRD + 2-Opt	1177,910	1,421	(1177,029; 1178,791)
9.	FI + 2-Opt	131,901	1,362	(131,057; 132,745)
10.	SC + 2-Opt	353,806	1,334	(352,979; 354,633)
11.	DMST + 2-Opt	511,945	1,278	(511,153; 512,737)
12.	NA + 2-Opt	227,789	1,264	(227,006; 228,572)
13.	NN + 2-Opt	703,114	1,219	(702,358; 703,870)
14.	AI + 2-Opt	89,600	1,004	(88,977; 90,222)
15.	CI + 2-Opt	1349,130	0,971	(1348,528; 1349,732)
16.	CHR	420,288	0,920	(419,718; 420,858)
17.	DENN	0,850	0,481	(0,552; 1,148)
18.	CI	5,095	0,471	(4,803; 5,387)
19.	NSI	4,011	0,398	(3,765; 4,258)
20.	GRD	11,591	0,380	(11,355; 11,827)
21.	DMST-M	8,170	0,365	(7,944; 8,396)
22.	NI	2,044	0,362	(1,820; 2,268)
23.	DMST	1,185	0,339	(0,975; 1,395)
24.	AI	0,754	0,214	(0,622; 0,887)
25.	FI	1,147	0,198	(1,024; 1,270)
26.	NN	0,471	0,165	(0,368; 0,573)
27.	NA	0,501	0,101	(0,439; 0,536)
28.	MC	0,003	0,002	(0,002; 0,004)
29.	SC	0,007	0,002	(0,006; 0,008)

Для определения Парето-оптимальных алгоритмов используются 19 наборов данных из базы VLSI Data Sets и только 10 наборов из National TSPs, количество вершин в которых приблизительно соотносится с количеством вершин в других наборах из VLSI (табл. 4). Данный выбор связан с желанием определить, насколько чувствительны Парето-оптимальные алгоритмы к различным формам представления данных.

Таблица 4.

Количество вершин в наборах, используемых для определения Парето-оптимальных алгоритмов

	№	VLSI Data Sets	National TSPs
Количество вершин в наборе	1.	237	-
	2.	737	734
	3.	984	980
	4.	1083	-
	5.	1 973	1 979
	6.	3 386	3 496
	7.	5087	-
	8.	7 168	7 146
	9.	10150	9 976
	10.	14 233	14 185
	11.	16 928	16 862
	12.	22 777	22 775
	13.	33 203	33 708
	14.	43 748	-
	15.	52 057	-
	16.	104 814	-
	17.	238 025	-
	18.	498 378	-
	19.	744 710	-

Далее будем полагать, что $E(f_{\varepsilon_{min}}(m, D))$ будет обозначаться как $E(f_\varepsilon)$. Аналогично со стандартным отклонением, наибольшими и наименьшими значениями.

Выводы по главе

В данной главе описана цель экспериментального исследования, приведен план проведения эксперимента, представленный также в виде псевдокода. Описано аппаратное и программное окружение. Указано временное ограничение – максимально возможное время, отведенное на полное исполнение эксперимента для одного алгоритма, равное 11 800 с. Приведено обоснование репрезентативности выбранного количества запусков каждого алгоритма. Определены 29 наборов данных для выявления Парето-оптимальных алгоритмов.

5. Результаты исследования

В данной главе представлены результаты экспериментов, проведенных в рамках исследования. Приводится их анализ как в целом, так и внутри подгрупп, к которым они принадлежат. Определяются Парето-оптимальные алгоритмы. Подробные результаты приводятся в приложениях Д, Е, Ж, З, И и К.

Стоит отметить, что результаты эксперимента показали, что при использовании алгоритма qCABC в качестве алгоритма улучшения, затрачиваемое время на исполнение алгоритма значительно превосходит самый «медленный» алгоритм CHR + 2-Opt (рис. 12), а вес исходного цикла уменьшается не более чем на 4,35% (табл. 5).

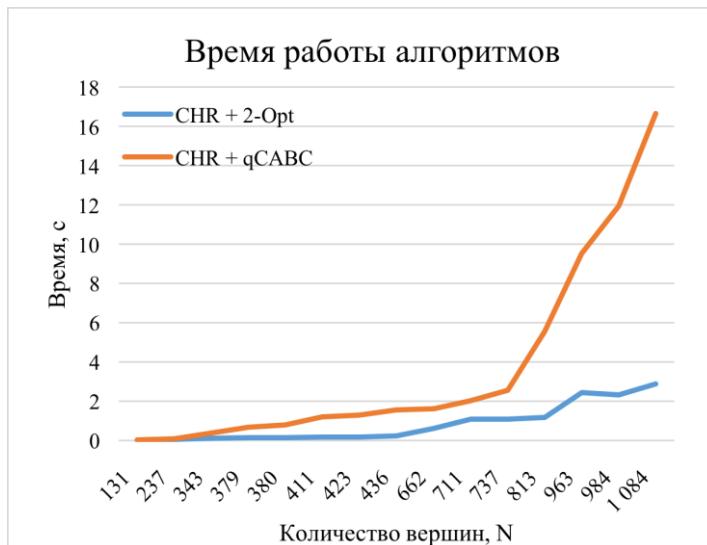


Рисунок 12. Сравнение времени работы алгоритма CHR с улучшениями 2-Opt и qCABC

Таблица 5.
Математическое ожидание оценок точности алгоритмов без применения алгоритма улучшения qCABC и с ним.

Название алгоритма m	$E(f_\varepsilon(m))$	$E(f_\varepsilon(m + qCABC))$	$\Delta E(f_\varepsilon)$
NSI	36,23%	31,88%	4,35%
MC	64,49%	60,91%	3,58%
SC	66,16%	62,74%	3,42%
NI	28,07%	25,43%	2,64%
DMST-M	32,46%	30,01%	2,45%
CHR	12,60%	10,43%	2,17%
DENN	22,82%	20,84%	1,98%
AI	85,20%	83,35%	1,85%
NA	51,30%	49,78%	1,52%
CI	20,28%	18,95%	1,33%
FI	56,88%	55,98%	0,90%
NN	25,38%	24,99%	0,39%
GRD	17,31%	17,11%	0,20%
LKH	0,08%	0,08%	0,00%

Обратившись к источнику, в котором был представлен алгоритм, было выяснено, что все экспериментальные результаты проводились на малых размерностях $N \leq 150$. Это объясняет, почему данный алгоритм хоть и считается одним из лучших среди алгоритмов роевого интеллекта, однако в ходе данного эксперимента, в котором акцент сделан на больших размерностях, он показал столь плохие результаты. В связи с полученными результатами было принято решение далее исключить данный алгоритм из рассмотрения, как «нежизнеспособный».

Полученные в ходе эксперимента результаты по оценкам точности остальных алгоритмов занесены в таблицу 6. Алгоритмы отсортированы в порядке возрастания значений математического ожидания относительной точности решения $E(f_\varepsilon)$, выраженных в процентах (наименьшее значение в столбце зеленого цвета, наибольшее – красного).

Таблица 6.

Сводная таблица с результатами алгоритмов по оценкам точности

Название алгоритма	$E(f_\varepsilon)$	$\max f_\varepsilon$	$\min f_\varepsilon$	$\sigma(f_\varepsilon)$
LKH	0,08%	0,23%	0,00%	0,07%
CHR + 2-Opt	6,14%	12,14%	3,47%	1,59%
GRD + 2-Opt	6,79%	10,82%	4,69%	1,57%
DENN + 2-Opt	11,06%	22,26%	4,39%	5,30%
NN + 2-Opt	11,89%	24,91%	3,90%	2,36%
CHR	12,60%	16,82%	9,31%	1,41%
CI + 2-Opt	13,04%	21,86%	6,74%	2,83%
NI + 2-Opt	14,60%	29,66%	5,86%	6,33%
DMST-M + 2-Opt	16,08%	35,78%	4,80%	9,29%
GRD	17,31%	31,34%	10,30%	3,83%
NSI + 2-Opt	17,63%	33,65%	8,92%	6,87%
DMST + 2-Opt	19,08%	39,12%	6,91%	10,52%
CI	20,28%	25,05%	12,46%	1,96%
DENN	22,82%	33,38%	11,97%	2,47%
NN	25,38%	32,68%	13,94%	2,62%
NA + 2-Opt	27,04%	57,90%	6,79%	17,84%
NI	28,07%	35,29%	14,89%	2,87%
FI + 2-Opt	28,90%	58,05%	4,01%	17,68%
DMST-M	32,46%	41,68%	18,55%	4,32%
SC + 2-Opt	36,20%	166,45%	8,11%	31,69%
NSI	36,23%	48,17%	19,15%	5,46%
MC + 2-Opt	36,47%	177,83%	6,21%	39,70%
DMST	40,09%	48,88%	33,16%	3,07%
AI + 2-Opt	50,23%	77,85%	5,26%	23,43%
NA	51,30%	59,23%	35,38%	4,67%
FI	56,88%	66,09%	31,59%	5,98%
MC	64,49%	242,41%	33,07%	41,08%
SC	66,16%	246,64%	30,76%	42,13%
AI	85,20%	100,92%	65,78%	6,81%

Анализ таблицы 6 позволяет сделать следующие выводы:

- наименьшее $E(f_\varepsilon)$, равное 0,08%, принадлежит алгоритму LKH, что свидетельствует о том, что алгоритм выдает наилучшие результаты по точности по сравнению с другими алгоритмами в среднем для всех N. Данный алгоритм опережает своего ближайшего «конкурента» - CHR + 2-Opt более чем на 6%.
- наибольшее $E(f_\varepsilon)$, равное 85,20%, принадлежит одному из алгоритмов группы вставка – AI. Следом за данным алгоритмом идут алгоритмы из группы заполняющих пространство кривых с математическим ожиданием, меньшим чем у AI приблизительно на 20%.
- наибольшая максимальная оценка точности $\max f_\varepsilon$ у Space-Filling Curve алгоритмов - MC и SC. Разница между ними составляет порядка 2 %, большее значение принадлежит алгоритму SC – 246,64%.
- наименьшая максимальная оценка точности $\max f_\varepsilon$ у алгоритма с наименьшим математическим ожиданием LKH.
- наибольшая минимальная оценка точности $\min f_\varepsilon$ у алгоритма AI. Это означает, что полученные в ходе эксперимента решения на основе данного алгоритма превосходили оптимальное решение не менее чем на 65,78%.
- наименьшая минимальная оценка точности $\min f_\varepsilon$, равная 0%, только у алгоритма LKH, что свидетельствует о получении решения, равного оптимальному.
- наиболее «непредсказуемыми» алгоритмами по оценкам точности решений являются алгоритмы группы Space-Filling Curves – стандартное отклонение колеблется в пределах от 31,69% до 42,13%, в расчет берется как применение алгоритма улучшения 2-Opt, так и нет.

Анализ качественных характеристик алгоритмов построения маршрута внутри их подгрупп позволяет сделать следующие выводы:

- Алгоритмы наращивания пути:

Алгоритм DENN предсказуемо имеет такие оценки точности, которые не хуже чем у NN. Эксперимент показал, что оценки точности всегда лучше, но в среднем они не более чем на 3% лучше NN. Временные оценки DENN превосходят NN, в среднем первый алгоритм работает медленнее второго на 6,5%.

Алгоритмы из семейства заполняющих пространство кривых по оценкам точности полностью уступают алгоритмам из группы «ближайших соседей», однако превосходят по времени работы – они быстрее приблизительно в 10000 раз при больших размерностях. Такой разброс по оценкам точности решения связан с формой

$$\text{распределения входных данных} - \frac{\max(w, h)}{\min(w, h)},$$

где w – ширина минимального покрывающего все точки прямоугольника,
 h – высота минимального покрывающего все точки прямоугольника,
тем хуже получаемая точность решения для любого N . На рис. 13 представлено визуальное распределение точек при $N = 4966$, на этих входных данных алгоритм показывает наихудшую относительную точность решения $f_\varepsilon(MC) = 242,41\%$, $f_\varepsilon(SC) = 246,64\%$.

– Алгоритмы вставки в подцикл:

Алгоритм AI предсказуемо дает решения с наибольшим весом относительно других алгоритмов данной подгруппы. Время его работы меньше приблизительно в 3 раза по сравнению с NI, NSI, в 4 раза по сравнению с CI и в 2 раза по сравнению с FI, однако не является самым лучшим, так как уступает алгоритму NA (больше в 1,3 раза). Самым точным и быстрым внутри данной подгруппы является алгоритм CI. По точности он превосходит ближайшего «соперника» NI в среднем в 1,4 раза.

– Комбинированные алгоритмы:

При сравнении алгоритмов DMST и DMST-M можно заметить, что точность решения при использовании модифицированной версии повышается в среднем на 8%, однако время работы значительно увеличивается (в 5-8 раз).

Алгоритм CHR является безусловным лидером по точности решения в данной подгруппе, он превосходит DMST-M в среднем на 20%. По времени решения CHR уступает DMST-M в 15-20 раз.



Рисунок 13. Визуальное распределение точек для XQD4966 [53]

Для всех алгоритмов характерно улучшение оценок точности решения после применения алгоритма 2-Opt. Замечено постепенное увеличение f_ε при $N \geq [5000 - 7000]$ в связи с ограничением числа итераций до 1000. Временная сложность при этом увеличивается в несколько десятков раз.

На основе результатов, полученных в ходе проведения эксперимента, построены графики относительной точности решения и времени работы алгоритмов для наборов из VLSI Data Sets (Приложения Д и Е).

В Приложении Д на рис. Д.1 изображен график с оценками точности для всех алгоритмов, на рис. Д.2. показаны те же результаты, за исключением алгоритмов MC, MC + 2-Opt, SC, SC + 2-Opt, AI и AI + 2-Opt. По оси абсцисс отложено количество вершин в различных наборах данных. По оси ординат указаны оценки точности решений, выраженные в процентах.

В Приложении Е по оси абсцисс отложено количество вершин в различных наборах данных. По оси ординат указано время работы алгоритмов в секундах.

Также построены графики, позволяющие определить Парето-оптимальные алгоритмы. Графики для базы данных VLSI Data Sets отражены в Приложении Ж, для базы данных National TSPs в Приложении З. По оси абсцисс отложено значение оценки точности алгоритмов, по оси ординат указано время работы алгоритмов. В названии графика указано название используемого набора данных, число в названии означает количество вершин в наборе.

Анализ полученных графиков показывает, что Парето-оптимальные алгоритмы являются устойчивыми к разным типам входных данных – они одинаковы для двух различных баз данных VLSI Data Sets (Приложение И) и National TSPs (Приложение К). В таблицах символом «+» отмечено вхождение соответствующего алгоритма в группу Парето-оптимальных для указанного набора данных. Ячейка в таблице окрашена в серый цвет в том случае, если для выбранного набора данных решение не было найдено ввиду временных ограничений, описанных в Главе 4 настоящего отчета, или ограничений по памяти, которые имеют место для жадного алгоритма GRD. В ходе его работы необходимо хранить $\frac{N(N-1)}{2}$ пар ребер, что при больших N вызывает переполнение в оперативной памяти.

В результате выделяются следующие Парето-оптимальные алгоритмы, перечисленные в порядке увеличения времени работы:

- **Moore Curve (MC);**
- **Sierpinski Curve (SC)** – результат алгоритма сильно зависит от формы входных данных, оценки точности не зависят строго от N ;
- **Nearest Neighbour (NN);**
- **Double Ended Nearest Neighbour (DENN);**
- **Cheapest Insertion (CI)** является оптимальным по Парето при $N \leq 400\ 000$ ввиду временного ограничения, при $N \leq 3\ 500$ не является Парето-оптимальным для некоторых наборов данных, что можно списать на особенность входных данных;

- **Greedy (GRD)** – является оптимальным по Парето при $N \leq 30\,000$ ввиду ограничений по памяти;
- **Cheapest Insertion and 2-Opt (CI + 2-Opt)** – является оптимальным по Парето при $30\,000 \leq N \leq 100\,000$. Нижняя граница связана с невозможностью получить результаты с использованием алгоритма GRD, верхняя граница определена ввиду ограничений по памяти;
- **Greedy and 2-Opt (GRD + 2-Opt)** – входит в группу Парето-оптимальных алгоритмов при малых $N \leq 800$;
- **Christofides (CHR)** – входит в группу Парето-оптимальных алгоритмов при $N \leq 2\,000$;
- **Helsgaun's Lin and Kernighan Heuristic (LKH)** – данный алгоритм прекрасно работает при $N \leq 55\,000$, для больших размерностей свыше 55 тысяч вершин время работы алгоритма превысило ограничение, введенное в данной работе.

Парето-оптимальные алгоритмы при разных N указаны в таблице 7. В таблицах символом + отмечено вхождение соответствующего алгоритма в группу Парето-оптимальных для указанного набора данных. Символом ± отмечено вероятностное вхождение алгоритму в группу Парето-оптимальных, это означает, что результаты зависят от типа представления входных данных.

Таблица 7.
Парето-оптимальные алгоритмы при различных N

Название алгоритма	Количество вершин, N (в тыс.)							
	(0; 0.8)	[0.8; 2)	[2; 3.5)	[3.5; 30)	[30; 55)	[55; 100)	[100; 400)	[400; 700)
MC	+	+	+	+	+	+	+	+
SC	±	±	±	±	±	±	±	±
NN	+	+	+	+	+	+	+	+
DENN	+	+	+	+	+	+	+	+
CI	±	±	±	+	+	+	+	
GRD	+	+	+	+				
CI + 2-Opt					+	+		
GRD + 2-Opt	+							
CHR	+	+						
LKH	+	+	+	+	+			

Выходы по главе

В данной главе представлены результаты экспериментов, проведенных в рамках исследования. Приведен их анализ как в целом, так и внутри подгрупп, к которым они принадлежат. Определены Парето-оптимальные алгоритмы при различных N . Подробные результаты приведены в приложениях Д, Е, Ж, З, И и К.

Заключение

В ходе выпускной квалификационной работы были реализованы эвристические алгоритмы решения метрической задачи коммивояжера и предложены способы модификации алгоритмов с целью получения реализаций с лучшими временными оценками или оценками точности.

В данной работе используется эффективная с точки зрения сокращения времени работы алгоритмов структура данных, основанная на волновой функции.

Разработана методика оценки качества алгоритмов, проанализированы данные, полученные в результате успешного проведения экспериментов.

В процессе исследования были выявлены Парето-оптимальные алгоритмы по критериям временной сложности и точности решения. Ими оказались алгоритмы:

- Moore Curve (MC);
- Sierpinski Curve (SC);
- Nearest Neighbour (NN);
- Double Ended Nearest Neighbour (DENN);
- Cheapest Insertion (CI);
- Greedy (GRD);
- Cheapest Insertion and 2-Opt (CI + 2-Opt);
- Greedy and 2-Opt (GRD + 2-Opt);
- Christofides (CHR);
- Helsgaun's Lin and Kernighan Heuristic (LKH).

В рамках продолжения исследовательской работы могут быть рассмотрены другие эвристические алгоритмы решения метрической задачи коммивояжера, проведены эксперименты для других метрик, могут использоваться другие наборы данных.

Апробация работы проведена на трех научных конференциях:

1. Межвузовская научно-техническая конференция студентов, аспирантов и молодых исследователей им. Е.В. Арменского. 20 февраля 2017. МИЭМ НИУ ВШЭ, Москва. Получен диплом 2-ой степени за лучший доклад в секции «Математическое моделирование», см. рис. Л.1 в Приложении Л.
2. Научно-практическая конференция по компьютерным наукам CoCoS'2017. 4 апреля 2017. НИУ ВШЭ, Москва.
3. Spring/Summer Researches' Colloquim on Software Engineering, SYRCoSE 5 -7 June 2017. Innopolis. (Статья «Pareto-optimal Algorithms for Metric TSP» прошла экспертизу, рекомендована к представлению на конференции).

По теме работы опубликованы три работы, одна из них находится в журнале, входящем в базу данных RSCI.

1. Е.Н. Береснева. Апостериорные оценки точности и временной сложности эвристических алгоритмов решения евклидовой задачи коммивояжера. Материалы межвузовской научно-технической конференции студентов, аспирантов и молодых исследователей им. Е.В. Арменского, – М.: – МИЭМ НИУ ВШЭ, 2017 год, стр. 26-27, ISBN 978-5-94768-075-1.
2. Avdoshin, S. Pareto-optimal Algorithms for Metric TSP / S.M. Avdoshin, E.N. Beresneva // International Journal of Open Information Technologies (INJOIT). – 2017. – N 5. – C. 16-24, ISSN: 2307-8162 [1].
3. Spring/Summer Researches' Colloquim on Software Engineering, SYRCoSE 5 -7 June 2017. Innopolis. (Статья «Pareto-optimal Algorithms for Metric TSP» прошла экспертизу, рекомендована к публикации).

Поданы заявки на государственную регистрацию программ для ЭВМ в федеральную службу по интеллектуальной собственности, патентам и товарным знакам:

1. Программа решения метрической задачи коммивояжера на основе метода ближайшей вставки (The program for Metric TSP based on Nearest Insertion).
2. Программа решения метрической задачи коммивояжера на основе метода "выгодной" вставки (The program for Metric TSP based on Cheapest Insertion).
3. Программа решения метрической задачи коммивояжера на основе метода ближайшего отрезка (The program for Metric TSP based on Nearest Segment Insertion).
4. Программа решения метрической задачи коммивояжера на основе удвоенного минимального остовного дерева (The program for Metric TSP based on double minimum spanning tree).
5. Программа решения метрической задачи коммивояжера на основе минимальных паросочетаний вершин нечетной степени минимального остовного дерева (The program for Metric TSP based on minimum-weight perfect matching of odd vertices of minimum spanning tree).
6. Программа решения метрической задачи коммивояжера на основе 2-Opt (The program for Metric TSP based on 2-Opt).
7. Программа решения метрической задачи коммивояжера на основе метода поисковой оптимизации, инспирированном роем пчёл (The program for Metric TSP based on qCABC).
8. Программа решения метрической задачи коммивояжера на основе кривой Мура (The program for Metric TSP based on Moore Curve).

Список использованных источников

- [1] Avdoshin, S. Pareto-optimal Algorithms for Metric TSP / S.M. Avdoshin, E.N. Beresneva // International Journal of Open Information Technologies (INJOIT). – 2017. – N 5. – C. 16-24.
- [2] Ульянов, М. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. - М.: Физматлит, 2008.
- [3] Vučmirović, M. The Travelling Salesman Problem in The Function of Transport Network Optimization // Management Research IX. – 2013. – N 1. C. – 19-87.
- [4] Applegate, D. The Traveling Salesman Problem: A Computational Study / D. Applegate, R. Bixby, V. Chvatal, W. Cook Princeton. – Princeton: Princeton University Press, 2011.
- [5] Hanne, T. Computational Intelligence in Logistics and Supply Chain Management / T. Hanne, R. Dornberger. – Switzerland: Springer, 2017.
- [6] Hosseini-Nasab, H. Green routing for trucking systems with classification of path types / H. Hosseini-Nasab, P. Lotfalian // Journal of Cleaner Production. – 2017. – N 14. – C. 228-233.
- [7] Onçan, T. Production, Manufacturing and Logistics: MILP formulations and an Iterated Local Search Algorithm with Tabu Thresholding for the Order Batching Problem // European Journal of Operational Research. – 2015. – N 243. – C. 142-155.
- [8] Noekel, K. Modelling Public Transport Passenger Flows in the Era of Intelligent Transport Systems / K. Noekel, G. Gentile. – UK: Springer, 2016.
- [9] Filip, E. The Travelling Salesman Problem and its Application in Logistic Practice / E. Filip, M. Otakar // WSEAS Transactions on Business and Economics. – 2011. – N 8. – C. 163-173.
- [10] Matai, R. Traveling Salesman Problem, Theory and Applications / R. Matai, S. Singh, M. Mittal. – Shanghai: InTech, 2010.
- [11] Lenstra, J. Clustering a Data Array and the Traveling-Salesman Problem // Operations Research. – 1974. – N 22. – C. 413-414.
- [12] Bland, R. Large travelling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation // Operations Research Letters. – 1989. – N 3. – C. 125-128.
- [13] Bachmann, F. Optimizing the experimental design of texture goniometry / F. Bachmann, H. Schaeben, H. Schaeben // Journal of applied crystallography. – 2012. – N 45. – 1173-1181.
- [14] Funke, S. Power assignment problems in wireless communication: Covering points by disks, reaching few receivers quickly, and energy-efficient travelling salesman tours // AD HOC Networks. – 2011. N – 9. – 1028-1035.

- [15] Hall, N. Scheduling and Sequencing / N. Hall, M. Magazine. – UK: Springer Science and Business Media, 2013.
- [16] Gerez, S. Algorithms for VLSI Design Automation, - New York: John Wiley & Sons, Inc., 1999.
- [17] Bagchi, T. A review of TSP based approaches for flowshop scheduling / T. Bagchi, J. Gupta, C. Sriskandarajah // European Journal Operations Research. – 2006. – N 3. – C. 816–854.
- [18] Reed, M. Methods of Modern Mathematical Physics / M. Reed, B. Simon. – London: Academic Press, 1972.
- [19] Hock, B. An examination of heuristic algorithms for the Travelling Salesman problem. – Cape Town: University of Cape Town, 1988.
- [20] Clarke, G. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points / G. Clarke, J. Wright // Operations Research. – 1994. – N 12. – C. 568-581.
- [21] Golden, B.L. Implementing vehicle routing algorithms / B.L. Golden, T. Magnanti, H. Nguyen // Networks. – 1977. – N 2. – C. 113-148.
- [22] Fisher, M. An analysis of approximations for maximizing submodular set functions / M. Fisher, G. Nemhauser, L. Wolsey. – UK: Springer, 1978.
- [23] Caccetta, L. An Improved Clarke and Wright Algorithm to Solve the Capacitated Vehicle Routing Problem / L. Caccetta, M. Alameen, M. Abdul-Niby // Engineering, Technology & Applied Science Research. – 2013. – N 2. – C. 413-415.
- [24] Corominas, A. Improving parametric Clarke and Wright algorithms by means of iterative empirically adjusted greedy heuristics / A. Corominas, A. Garcia-Villoria, R. Pastor // SORT-Statistics and Operations Research Transactions. – 2014. – N 1. – C. 3-11.
- [25] Johnson, D.S. The traveling salesman problem: A case study / D.S. Johnson, L.A. McGeoch. – Chichester, UK: John Wiley & Sons, 1997.
- [26] Rosenkrantz, D. An analysis of several heuristics for the traveling salesman problem / D. Rosenkrantz, R. Stearns, P. Lewis II // Operations Research. – 1977. – N 6. – C. 563-581.
- [27] Glover, F. The Travelling Salesman Problem: New Solvable Cases and Linkages with the Development of Approximation Algorithms / F. Glover, A.P. Punnen // Operations Research. – 1997. – N 5. – C. 63-81.
- [28] Bartholdi, J.J. A Minimal Technology Routing System for Meals on Wheels / J.J. Bartholdi, L.K. Platzman, R.L. Collins, W.H. Warden // III School of Industrial and Systems Engineering. – 1983. N 3. – C. 1-8.
- [29] Platzman, L.K. Spacefilling curves and the planar travelling salesman problem / L.K. Platzman,

J.J. Bartholdi // J. ACM. – 1989. – N 4. – C. 710-737.

[30] Moore, E.H. On Certain Crinkly Curves // Trans. Amer. Math Soc. – 1990. N 1. – C. 72-90.

[31] Buchin, K. Space-Filling Curves // Free University of Berlin Press. – 2007. – N 1. – C. 5-30.

[32] Hahsler, M. TSP – Infrastructure for the Traveling Salesperson Problem / M. Hahsler, K. Hornik // Journal of Statistical Software. – 2007. – N 2. – C. 1-21.

[33] Golden, B.L. Approximate traveling salesman algorithms / B.L. Golden, L.D. Bodin, T. Doyle, W. Stewart // Operations Research. – 1980. – N 28. – C. 694-711.

[34] Deineko, V.G. The Convex Hull and Line Traveling Salesman Problem: A Solvable Case / V.G. Deineko, R. Dal, G. Rote. – Austria: Christian-Doppler Laboratorium "Discrete Optimization", 1992.

[35] Johnson, D.S. The Traveling Salesman Problem. Computational Complexity / D.S. Johnson, C.H. Papadimitriou. - New York: Wiley, 1985.

[36] Karp, R.M. The Traveling Salesman Problem. Probabilistic analysis of heuristics / R.M. Karp, J.M. Steele. - New York: Wiley, 1985.

[37] Christofides, N. Graph theory - An Algorithmic Approach. - New York: Academic Press, 1974.

[38] Christofides, N. Worst-case analysis of a new heuristic for the travelling salesman problem. – Pittsburgh: Carnegie-Mellon University Press, 1976.

[39] Croes, A. A method for solving travelling salesman problems // Operations Research. – 1958. – N 5. – C. 791-812.

[40] Lin, S. Computer Solutions of the Traveling Salesman Problem // The Bell System Technical Journal. – 1965. – N 10. – C. 2245-2269.

[41] Lin, S. An Effective Heuristic Algorithm for the Traveling-Salesman Problem / S. Lin, B. W. Kernighan // Operations Research. – 1973. – N 2. – C. 498–516.

[42] S. Kirkpatrick, C. D. Gelatt и M. P. Vecchi, «Optimization by Simulated Annealing,» т. 220, pp. 671-680, 1983.

[43] Laarhoven, P.J. Simulated Annealing: Theory and Applications / P.J. Laarhoven, E.H. Aarts. - Heidelberg, Germany: Springer, 1987.

[44] Geng, X. Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search / X. Geng, Z. Chen // Applied Soft Computing. – 2011. – N 11. – C. 3680–3689

[45] Lin, Y. Developing a dynamic neighborhood structure for an adaptive hybrid simulated annealing – tabu search algorithm to solve the symmetrical traveling salesman problem / Y. Lin, B.

Zheyong, L. Zheyong // Applied Soft Computing. – 2016. – N 49. – C. 937-952.

[46] Bertsimas, D. Simulated annealing / D. Bertsimas, J. Tsitsiklis // Statistical Science. – 1993. – N 8. – C. 10-15.

[47] Potvin, J.Y. The Traveling Salesman Problem: A Neural Network Perspective // ORSA Journal on Computing. – 1993. – N 5. – C. 328-348.

[48] Dorigo, M. Ant colony system: a cooperative learning approach to the traveling salesman problem / M. Dorigo, L.M. Gambardella // IEEE Transactions on Evolutionary Computation. – 1997. – N. 1. – C. 53-66.

[49] Gorkemli, B. Quick Combinatorial Artificial Bee Colony -qCABC- Optimization Algorithm for TSP / B. Gorkemli, D. Karaboga // Applied Soft Computing. – 2014. – N 5. – C. 227-238.

[50] Yang, X.S. Cuckoo search: recent advances and applications / X.S. Yang, S. Deb // Neural Comput & Applic. – 2014. – N 24. – C. 169-193.

[51] Hsiao, P. An effective PSO-based algorithm for the traveling-salesman problem // Springer Berlin Heidelberg. - 2013. N – 3. – C. 1151-1156.

[52] Rohe, A. VLSI Data Sets [Электронный ресурс] / UWATERLOO. Режим доступа: <http://www.math.uwaterloo.ca/tsp/vlsi/index.html>, свободный (дата обращения: 02.01.2017).

[53] National Travelling Salesman Problems [Электронный ресурс] // University of Waterloo. Режим доступа: <http://www.math.uwaterloo.ca/tsp/world/countries.html>, свободный (дата обращения: 16.04.2017).

[54] Aarts, E. Local Search in Combinatorial Optimization / E. Aarts и J. Lenstra. – Princeton: Princeton University Press, 1997.

[55] Bentley, J. Fast algorithms for geometric traveling salesman problem // ORSA Journal on Computing. – 1992. – N 4. – C. 387-411.

[56] Lau, S. Solving traveling salesman problems with heuristic learning approach [Электронный ресурс]. – 2002. Режим доступа: <http://ro.uow.edu.au/theses/1455/>, свободный (дата обращения: 20.02.2017).

[57] Bartholdi, J. A Routing System Based on Spacefilling Curves [Электронный ресурс]. - 2003. Режим доступа: <http://www2.isye.gatech.edu/~jjb/research/mow/mow.pdf>, свободный (дата обращения: 01.05.2017).

[58] Helsgaun, K. LKH [Электронный ресурс]. Режим доступа: <http://www.akira.ruc.dk/~keld/research/LKH/>, свободный (дата обращения: 18 03 2017).

[59] Авдошин, С.М. Обобщенный метод "волны" для решения экстремальных задач на графах /

С.М. Авдошин, В.В. Белов // ZHVM and MF. – 1979. – 19. – С. 739-755.

- [60] Galler, B. An improved equivalence algorithm / B. Galler, M. Fisher // Communications of the ACM. – 1964. – N 7. – С. 301-303.
- [61] Tarjan, R. Worst-case analysis of set union algorithms / R. Tarjan, J. Leeuwen // Journal of the ACM. – 1984. – N 31. – С. 245-281.
- [62] Calculate an integer square root [Электронный ресурс]. Режим доступа: http://www.codexcodex.com/wiki/Calculate_an_integer_square_root, свободный (дата обращения: 11.02.2016).
- [63] Programs for computing the integer square root [Электронный ресурс]. Режим доступа: <http://www.hackersdelight.org/hdcodetxt/isqrt.c.txt>, свободный (дата обращения: 12.02.2016).
- [64] Integer Square Root function [Электронный ресурс]. Режим доступа: <http://atoms.alife.co.uk/sqrt/SquareRoot.java>, свободный (дата обращения: 11.02.2016).
- [65] Github open source by vancepym [Электронный ресурс]. Режим доступа: https://github.com/vancepym/vlmcsd/blob/master/dns_srv.c, свободный (дата обращения: 12.02.2016).
- [66] Apple, IOFixed64.cpp [Электронный ресурс]. Режим доступа: <https://opensource.apple.com/source/IOHIDFamily/IOHIDFamily-700.10.1/IONIDSSystem/IOFixed64.cpp>, свободный (дата обращения: 12.02.2016).
- [67] Вычисление квадратного корня из целого числа [Электронный ресурс]. Режим доступа: <http://www.codenet.ru/progr/alg/sqrtnint.php>, свободный (дата обращения: 11.02.2016).

Приложение А

Сравнение структур данных

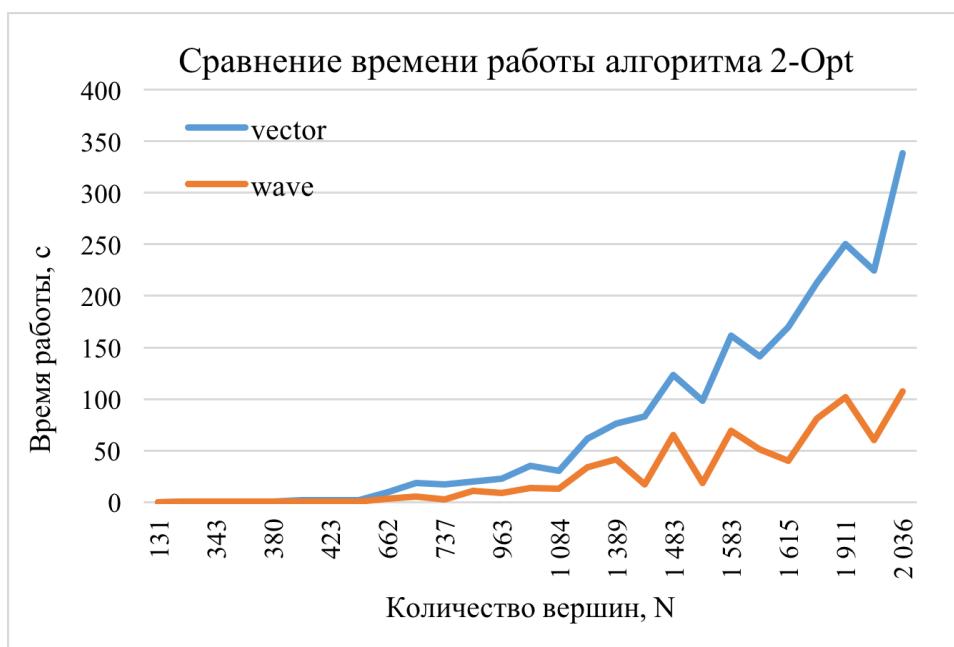


Рисунок А.1. Сравнение времени работы алгоритма 2-Opt для структур данных wave и vector

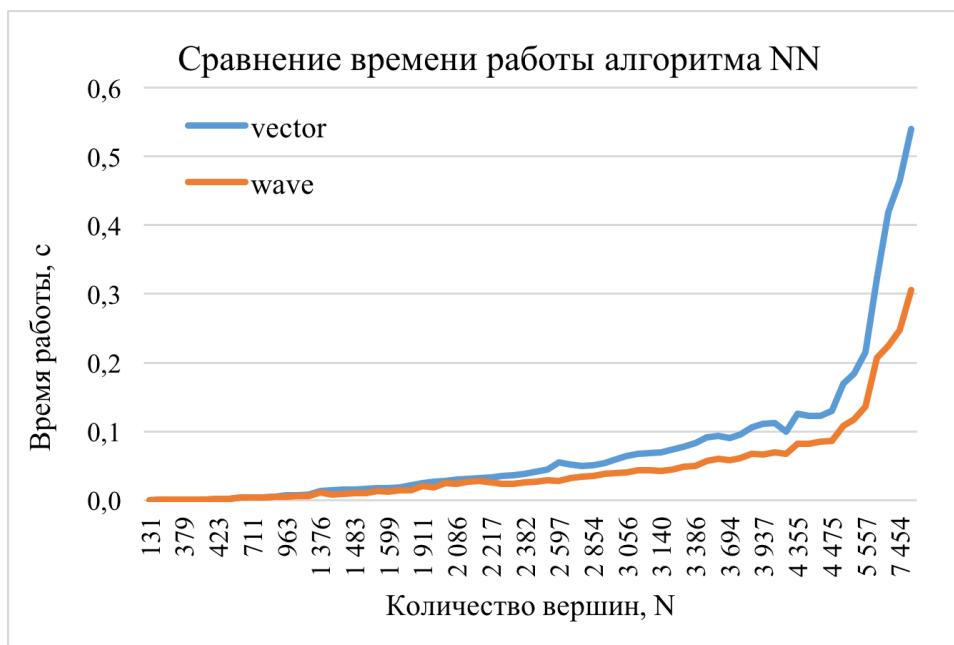


Рисунок А.2. Сравнение времени работы алгоритма NN для структур данных wave и vector

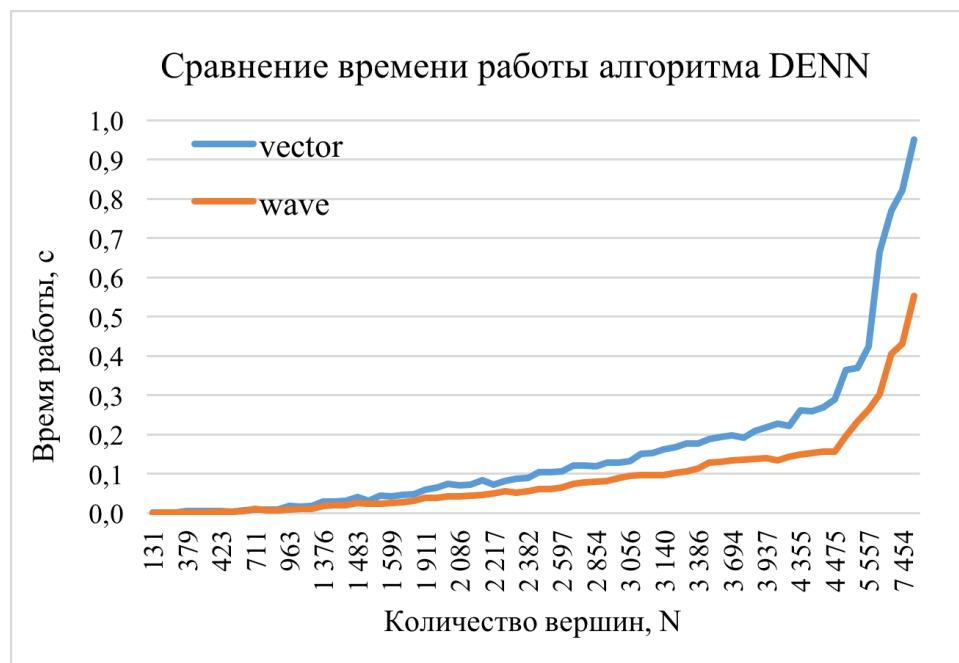


Рисунок А.3. Сравнение времени работы алгоритма DENN для структур данных wave и vector

Приложение Б**Алгоритмы вычисления целочисленного значения квадратного корня**

Таблица Б.1.

Алгоритмы вычисления целочисленного значения квадратного корня

№	Название	Время, мс	Реализация на C++
1	isqrt0 [62]	3541±160	<pre>static int isqrt0(int x) { if (x == 0) return 0; int n = (x / 2) + 1; int n1 = (n + (x / n)) / 2; while (n1 < n) { n = n1; n1 = (n + (x / n)) / 2; } return n; }</pre>
2	isqrt1 [63]	2621±98	<pre>static int isqrt1(unsigned x) { unsigned register x1; register int s, g0, g1; if (x <= 1) return x; s = 1; x1 = x - 1; if (x1 > 0x0000ffff) { s = s+ 8; x1 = x1 >> 16; } if (x1 > 0x000000ff) { s = s + 4; x1 = x1 >> 8; } if (x1 > 0x0000000f) { s = s + 2; x1 = x1 >> 4; } if (x1 > 3) { s = s +1; g0 = 1 << s; g1 = (g0 + (x >> s)) >> 1; while (g1 < g0) { g0 = g1; g1 = (g0 + (x/g0)) >> 1; } return g0; } }</pre>
3	isqrt2 [64]	2652±101	Полная реализация указана в источнике (ввиду большого объема кода – 390 строк).
4	isqrt3 [63]	2730±104	<pre>static int isqrt3(unsigned x) { unsigned a, b, m; a = 1;</pre>

			<pre> b = (x >> 5) + 8; if (b > 65535) b = 65535; do { m = (a + b) >> 1; if (m*m > x) b = m - 1; else a = m + 1; } while (b >= a); return a - 1; } </pre>
5	isqrt4 [63]	2792±123	<pre> static int isqrt4(unsigned x) { unsigned register m, y, b, t; m = 0x40000000; y = 0; while (m != 0) { b = y m; y = y >> 1; t = (int)(x ~(x - b)) >> 31; x = x - (b & t); y = y (m & t); m = m >> 2; } return y; } </pre>
6	isqrt5 [65]	2855±139	<pre> static unsigned int isqrt5(unsigned x) { unsigned int c = 0x8000; unsigned int g = 0x8000; for (;;) { if (g*g > x) { g ^= c; } c >>= 1; if (c == 0) { return g; } g = c; } } </pre>
7	isqrt6 [62]	2746±101	<pre> static unsigned int isqrt6 (unsigned x) { register unsigned root, remainder, place; root = 0; remainder = x; place = 0x40000000; while (place > remainder) place = place >> 2; while (place) { if (remainder >= root + place) { remainder = remainder - root - place; root = root + (place << 1); } root = root >> 1; place = place >> 2; } return root; } } </pre>

8	isqrt7 [66]	2823±78	<pre>static unsigned int isqrt7(unsigned int x) { unsigned int rem = 0; unsigned int root = 0; for (int i = 16; i--;) { root <<= 1; rem = ((rem << 2) + (x >> 30)); a <<= 2; root++; if (root <= rem) { rem -= root; root++; } else root--; } return root >> 1; }</pre>
9	isqrt8 [67]	2450±90	<pre>static unsigned isqrt8(int x) { long temp, div; unsigned rslt = (unsigned)x; if (x <= 0) return 0; else if (1 & 0xFFFF0000L) if (1 & 0xFF000000L) div = 0x3FFF; else div = 0x3FF; else if (x & 0x0FF00L) div = 0x3F; else div = (x > 4) ? 0x7 : 1; while (1) { temp = x / div + div; div = temp >> 1; div += temp & 1; if (rlst > div) rslt = (unsigned)div; else if (x / rslt == rslt - 1 && x % rslt == 0) rslt--; return rslt; } }</pre>
10	std::sqrt	2355±60	Встроенный в язык C++ алгоритм вычисления вещественного корня.

Приложение В

Система непересекающихся множеств

Система непересекающихся множеств на языке C++:

```
static void make_set(int v, int* parent, int* rank) {
    parent[v] = v;
    rank[v] = 0;
}

static int find_set(int v, int* parent) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set (parent[v], parent);
}

static void union_sets(int a, int b, int* parent, int* rank) {
    a = find_set(a, parent);
    b = find_set(b, parent);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            ++rank[a];
    }
}
```

Приложение Г

Текст программы

Текст программы к документации по ВКР на тему «Реализация и оценка качества ресурсно-эффективных алгоритмов для метрической задачи коммивояжера» находится в директории sources/code (носитель информации типа компакт-диск) в связи с большим количеством строк кода.

Приложение Д

Оценки точности алгоритмов

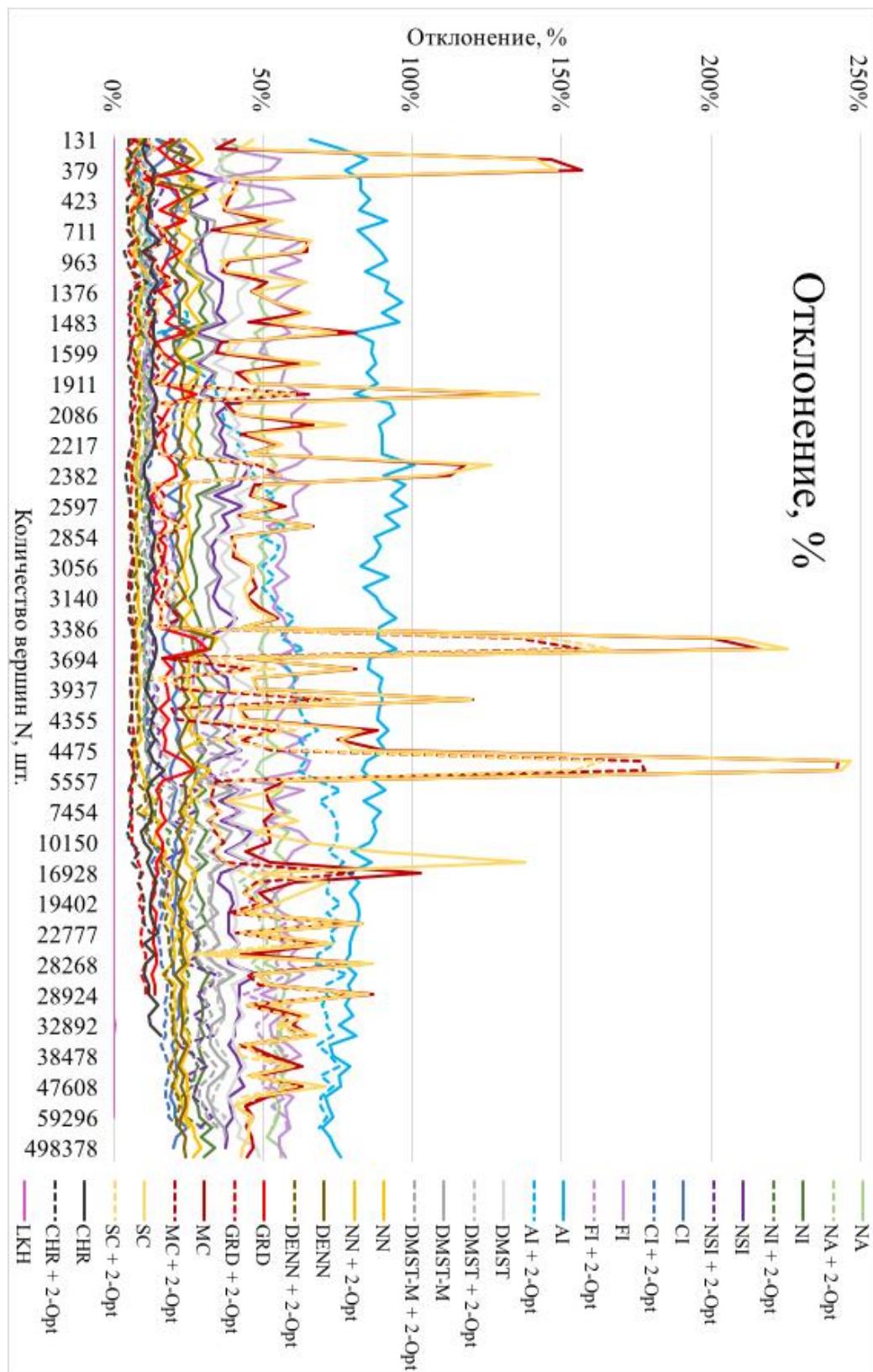


Рисунок Д.1. Отклонение полученных решений от оптимальных для всех алгоритмов

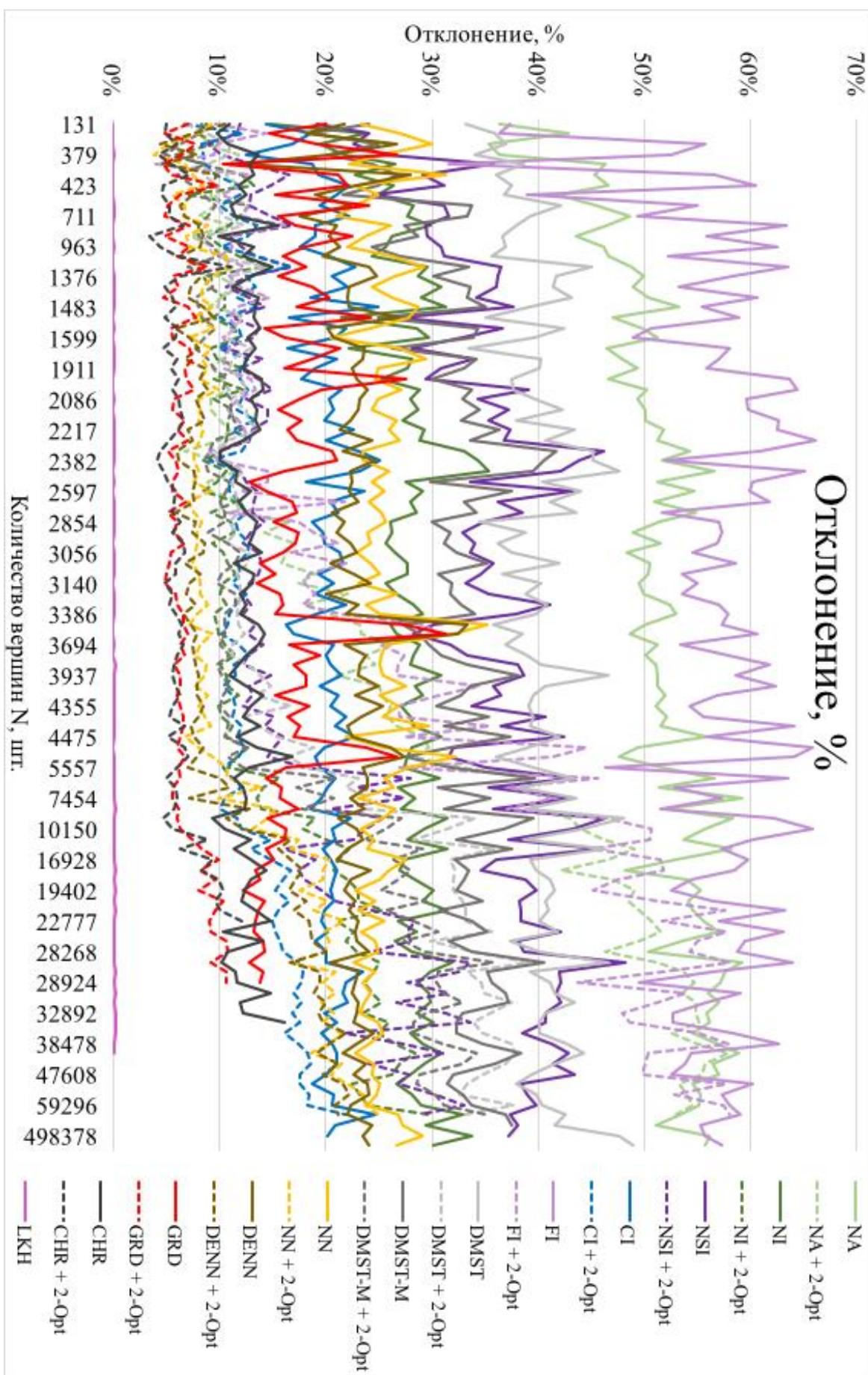


Рисунок Д.2. Отклонение полученных решений от оптимальных для части алгоритмов

Приложение Е

Время работы алгоритмов

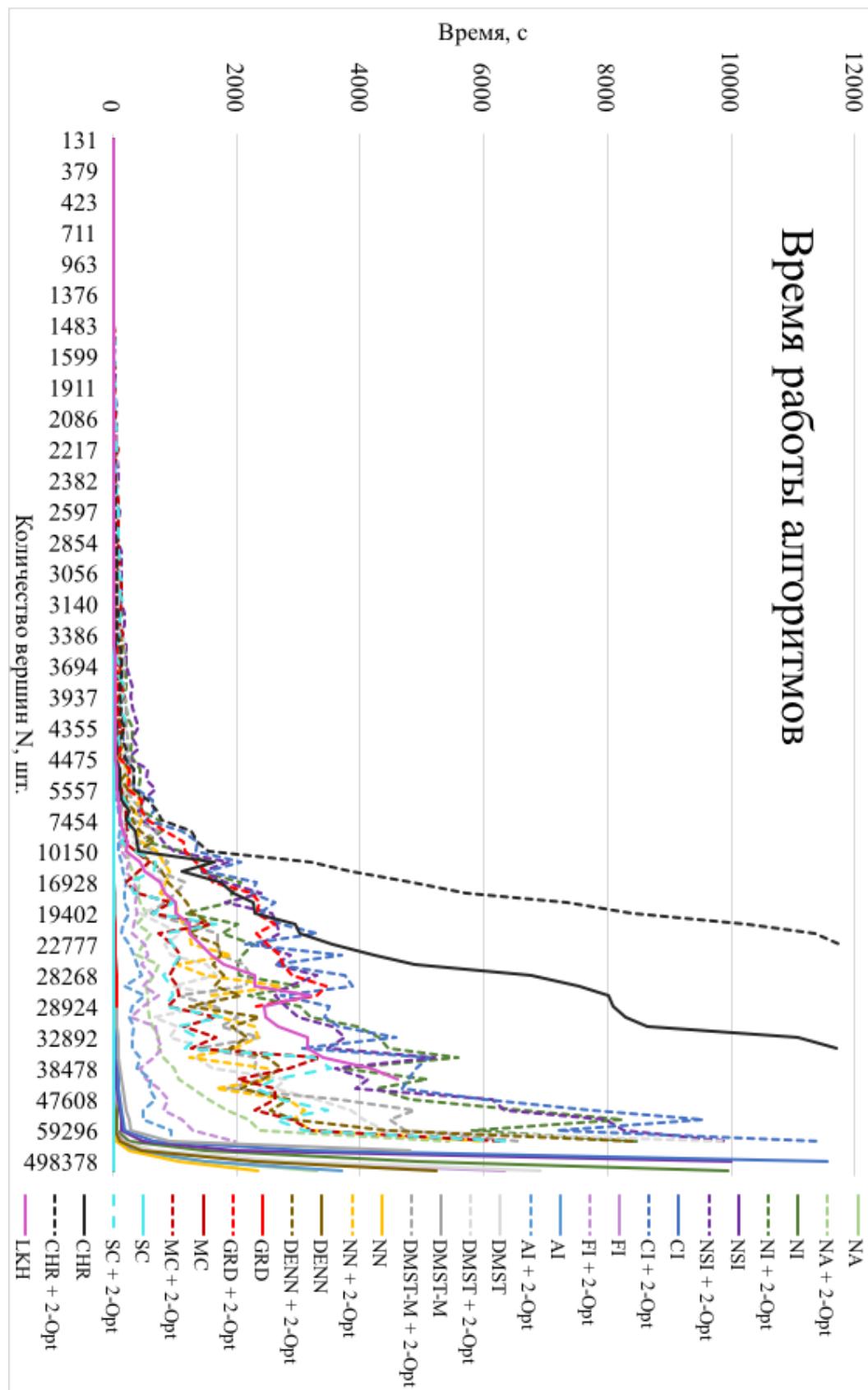


Рисунок Е.1. Время работы алгоритмов

Приложение Ж

Графики Парето-оптимальных алгоритмов (VLSI Data Sets)

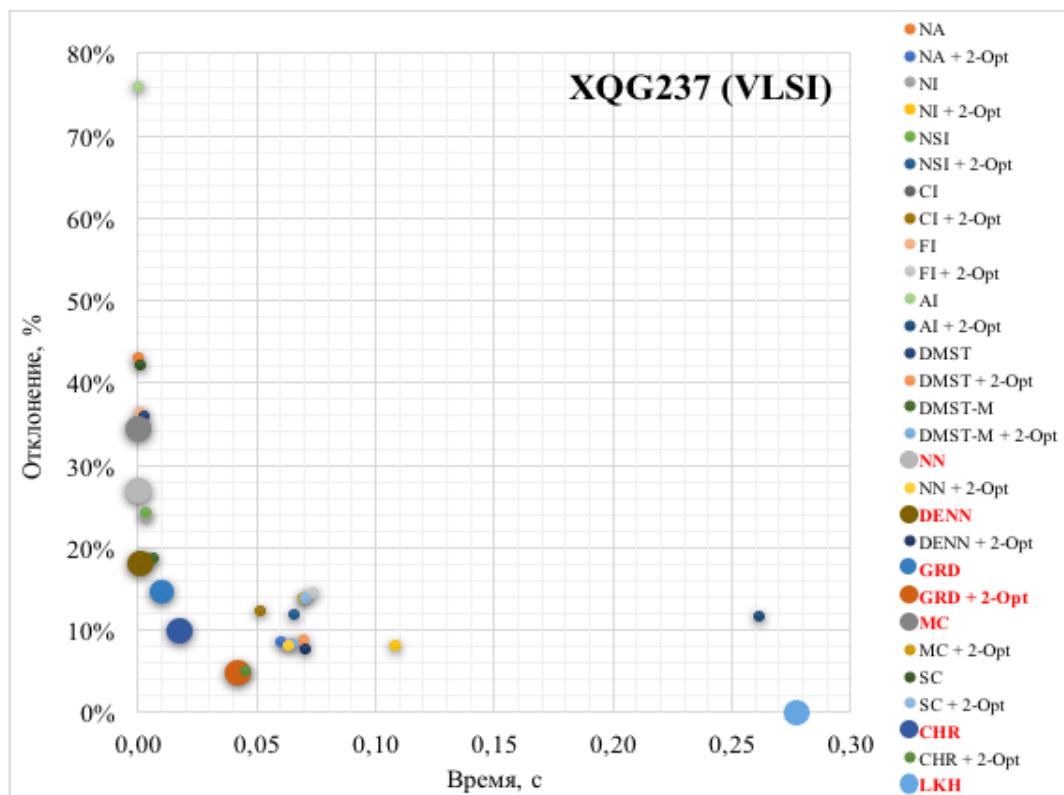


Рисунок Ж.1. Парето-оптимальные алгоритмы для набора данных XQG237

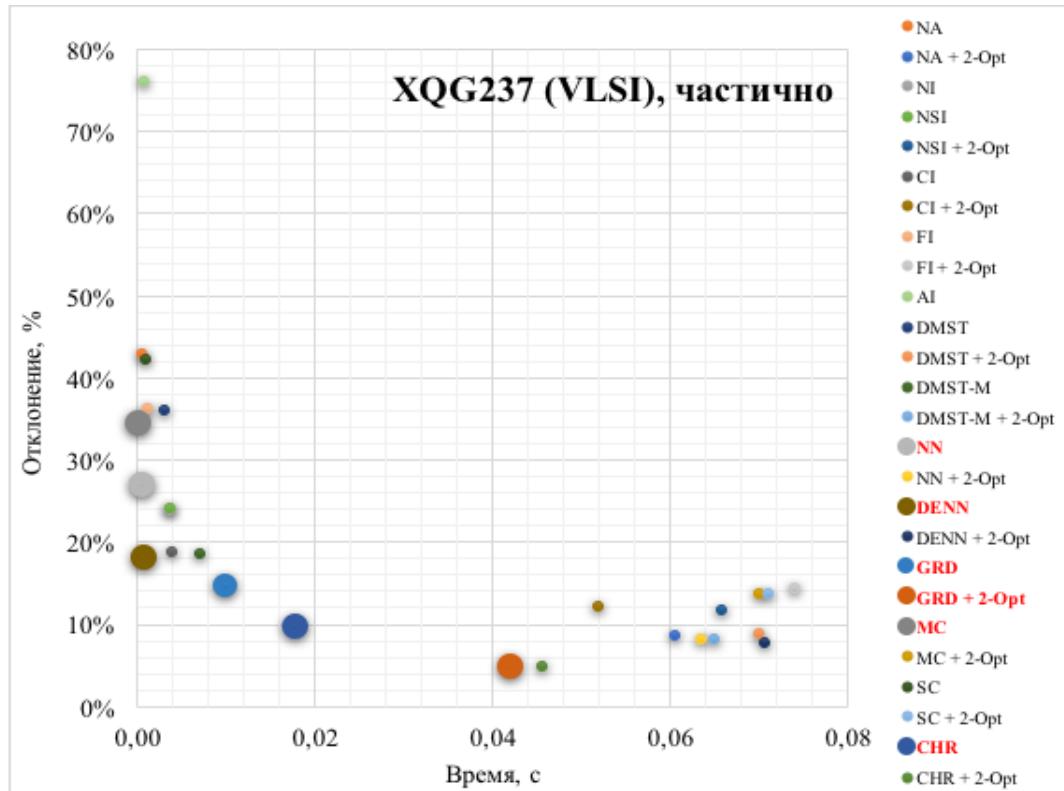


Рисунок Ж.2. Парето-оптимальные алгоритмы для набора данных XQG237 (частично)

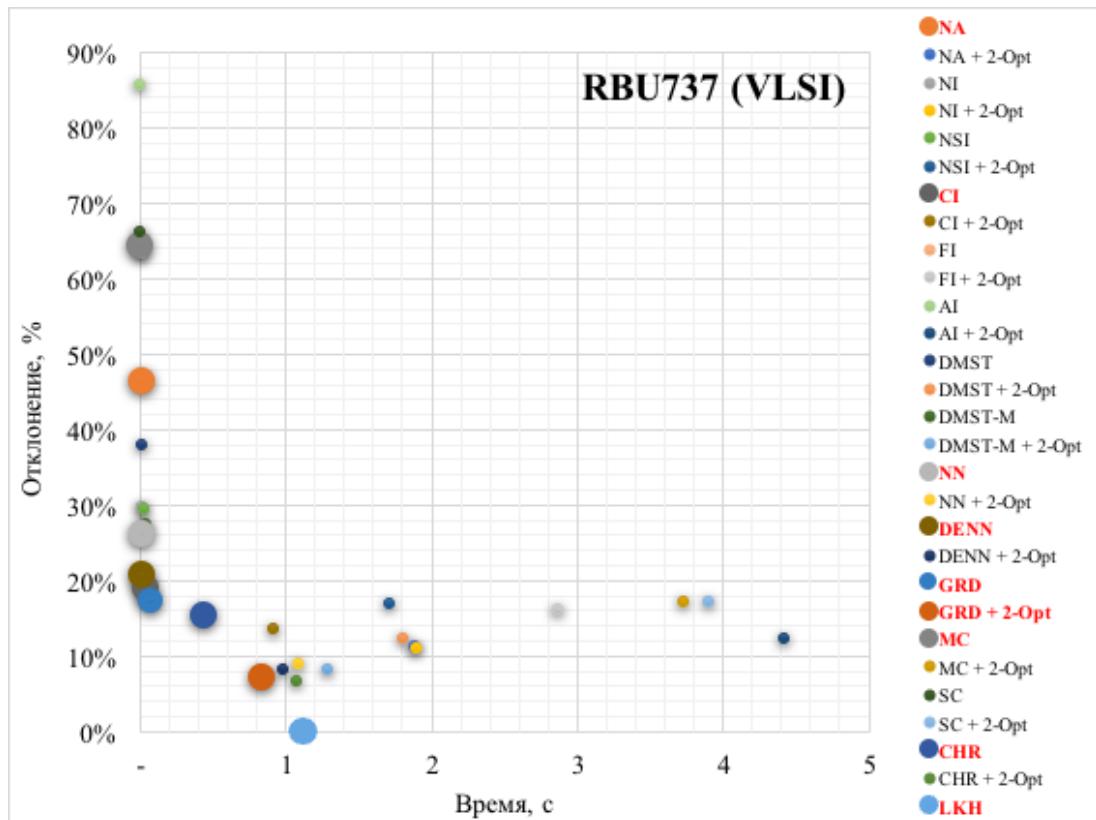


Рисунок Ж.3. Парето-оптимальные алгоритмы для набора данных RBU737

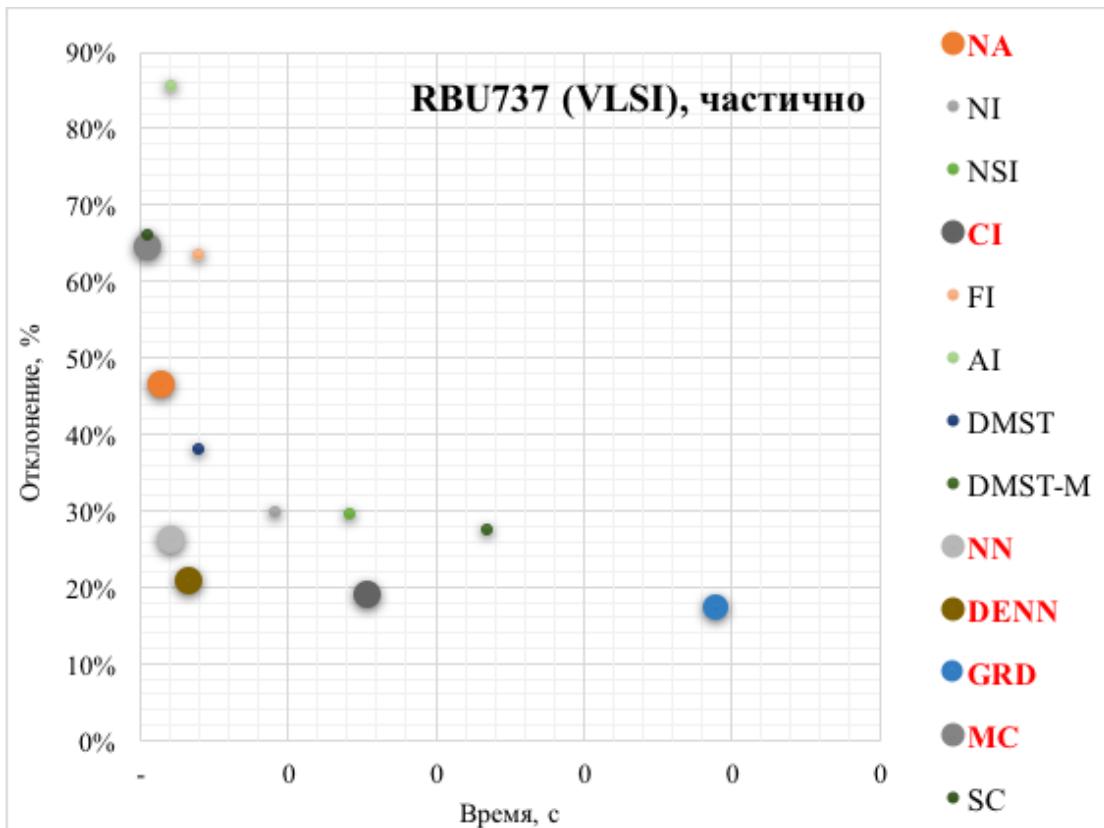


Рисунок Ж.4. Парето-оптимальные алгоритмы для набора данных RBU737 (частично)

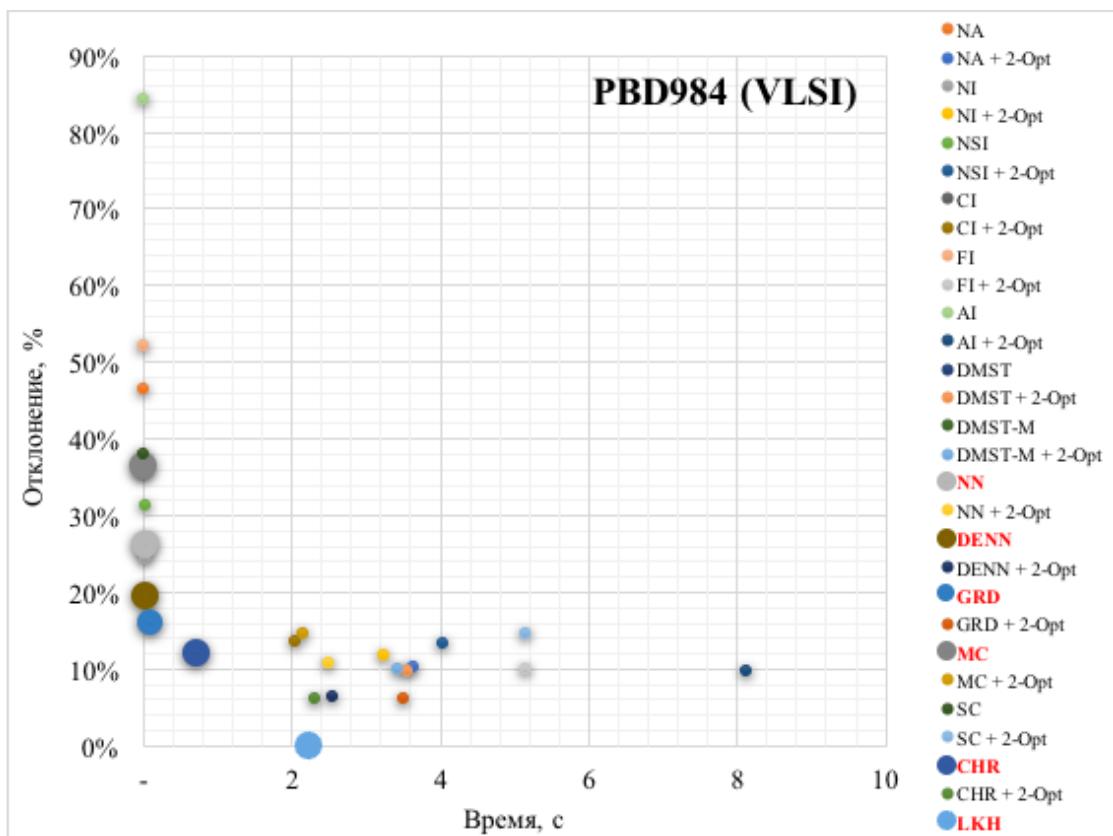


Рисунок Ж.5. Парето-оптимальные алгоритмы для набора данных PBD984

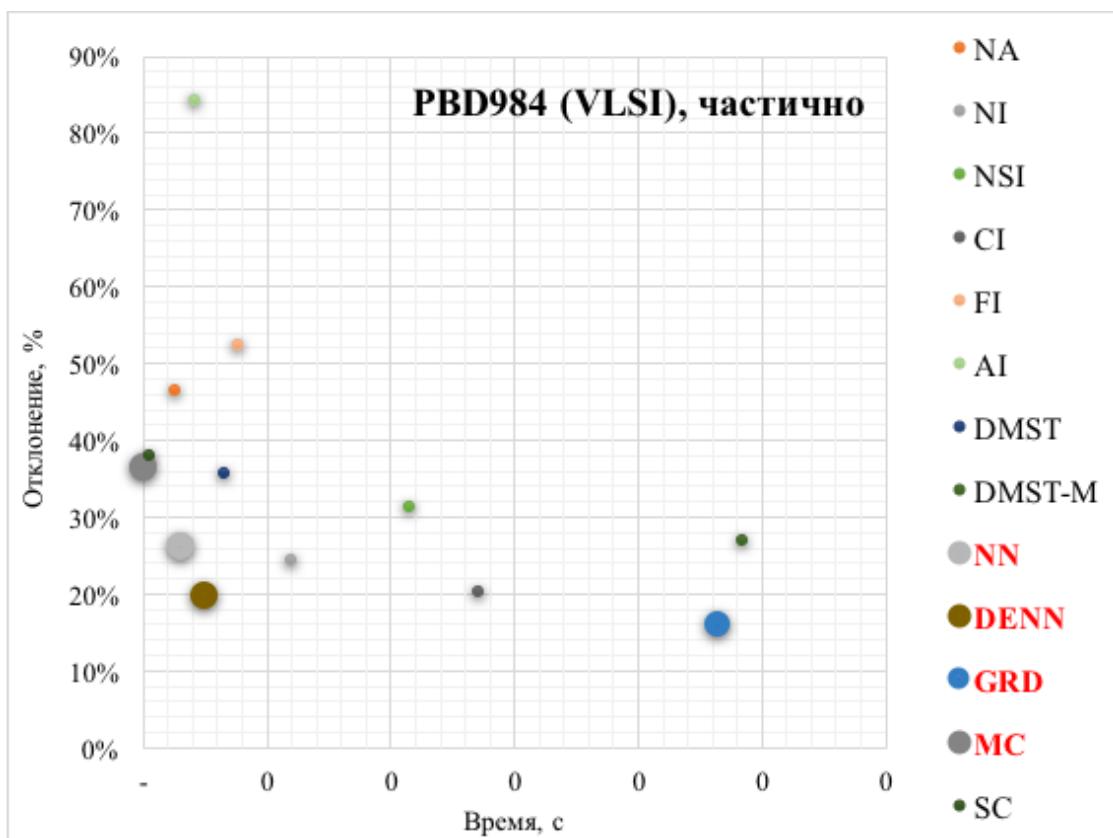


Рисунок Ж.6. Парето-оптимальные алгоритмы для набора данных PBD984 (частично)

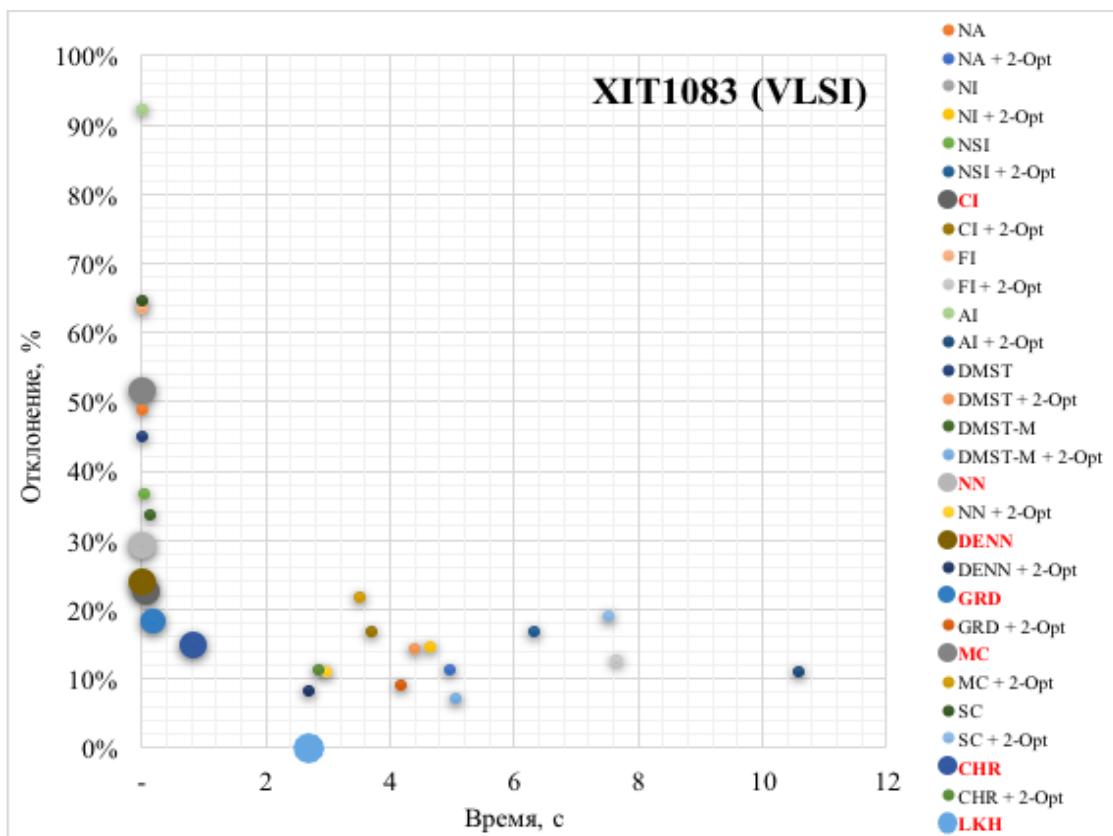


Рисунок Ж.7. Парето-оптимальные алгоритмы для набора данных XIT1083

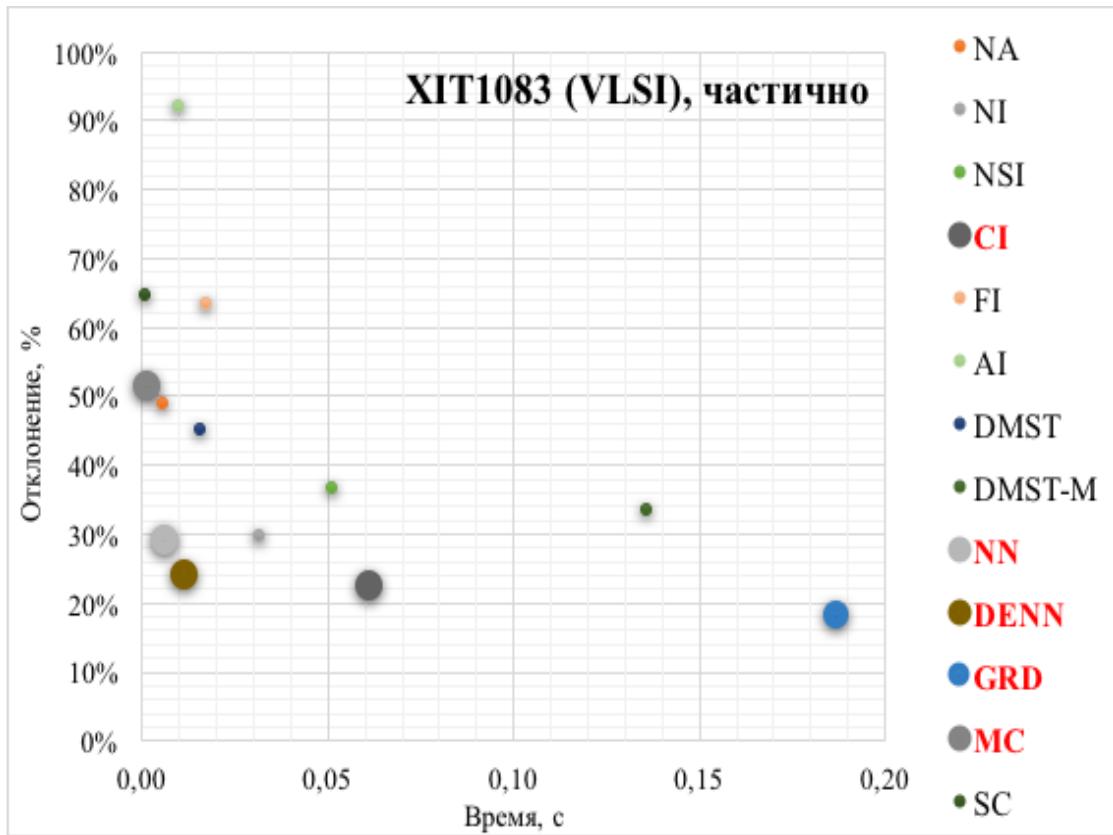


Рисунок Ж.8. Парето-оптимальные алгоритмы для набора данных XIT1083 (частично)

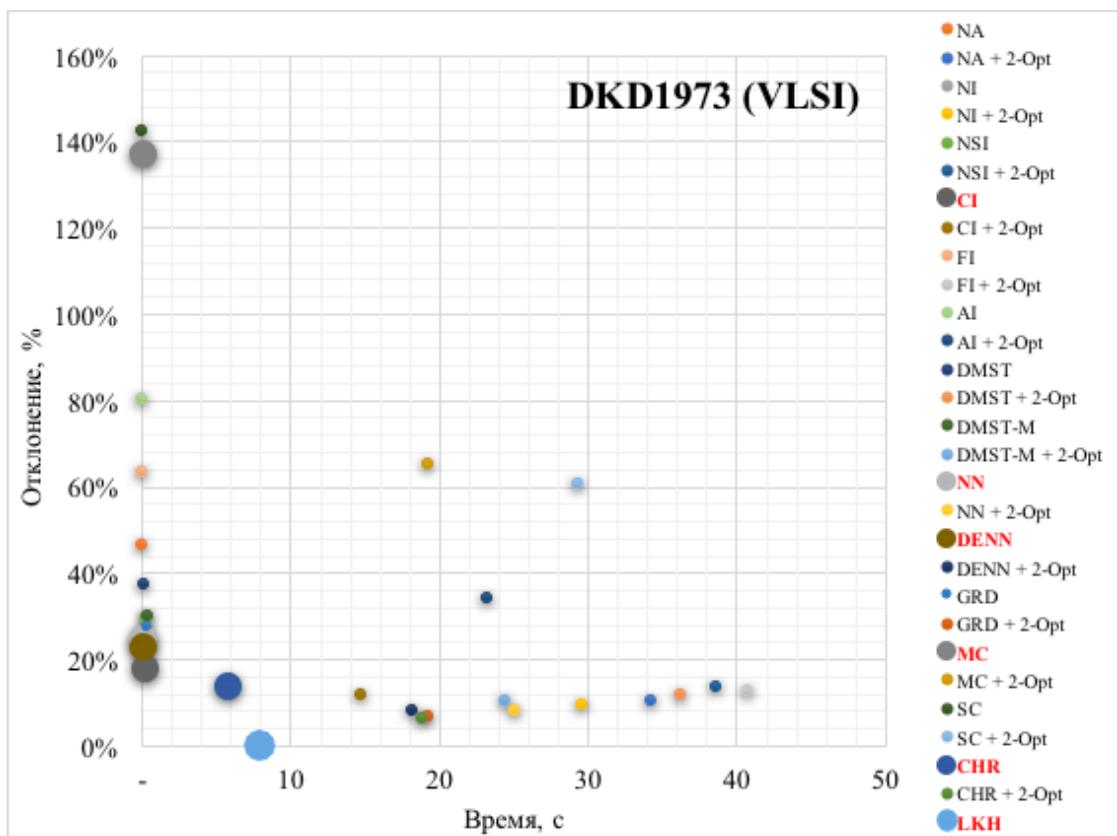


Рисунок Ж.9. Парето-оптимальные алгоритмы для набора данных DKD1973

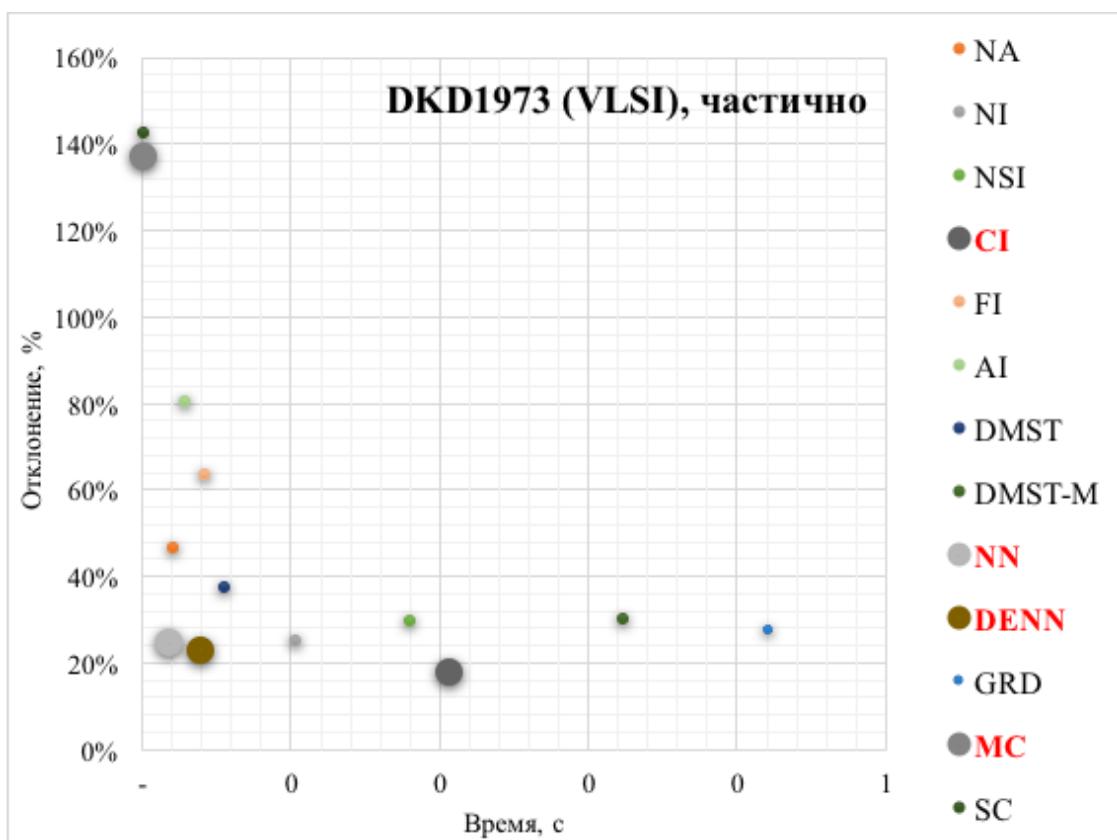


Рисунок Ж.10. Парето-оптимальные алгоритмы для набора данных DKD1973 (частично)

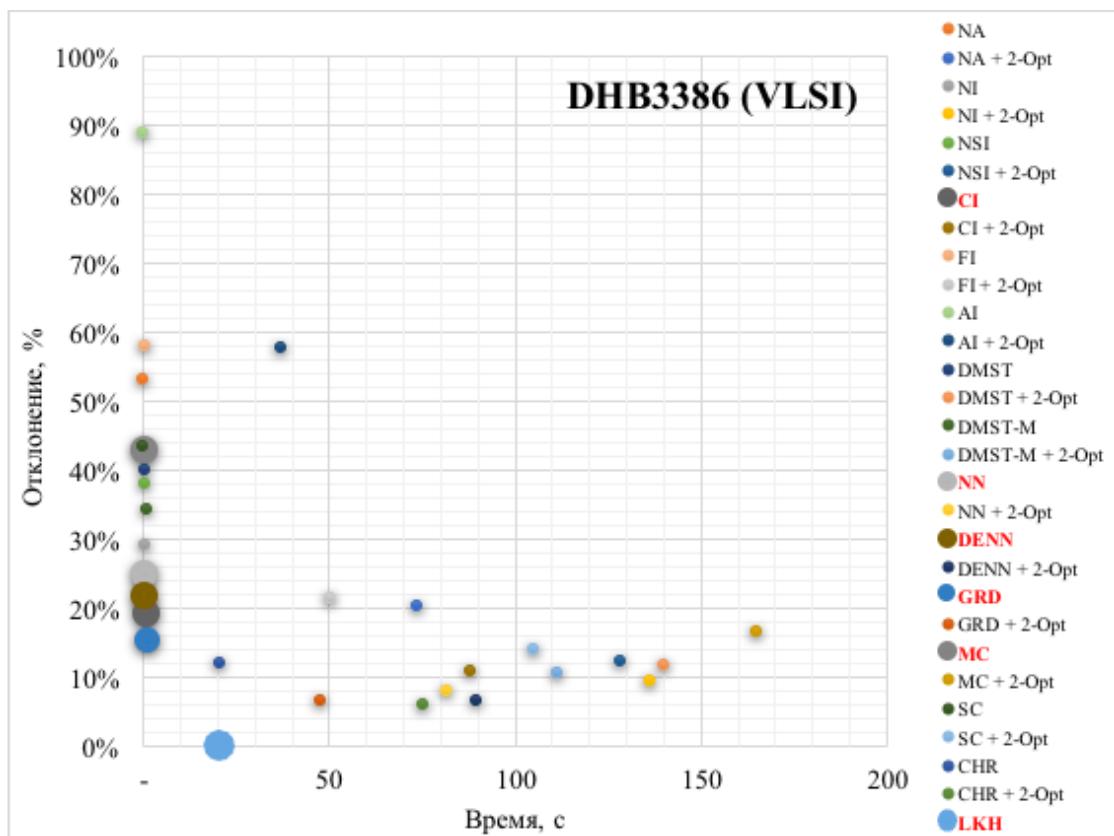


Рисунок Ж.11. Парето-оптимальные алгоритмы для набора данных DHB3386

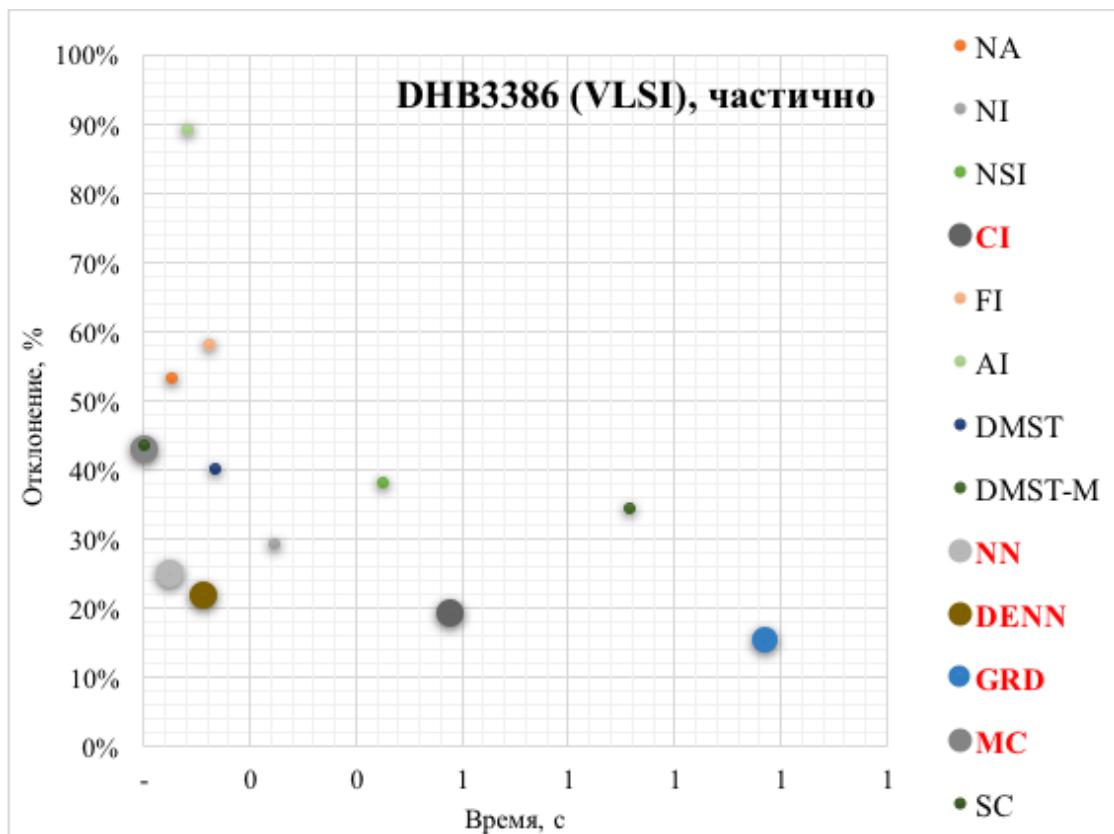


Рисунок Ж.12. Парето-оптимальные алгоритмы для набора данных DHB3386 (частично)

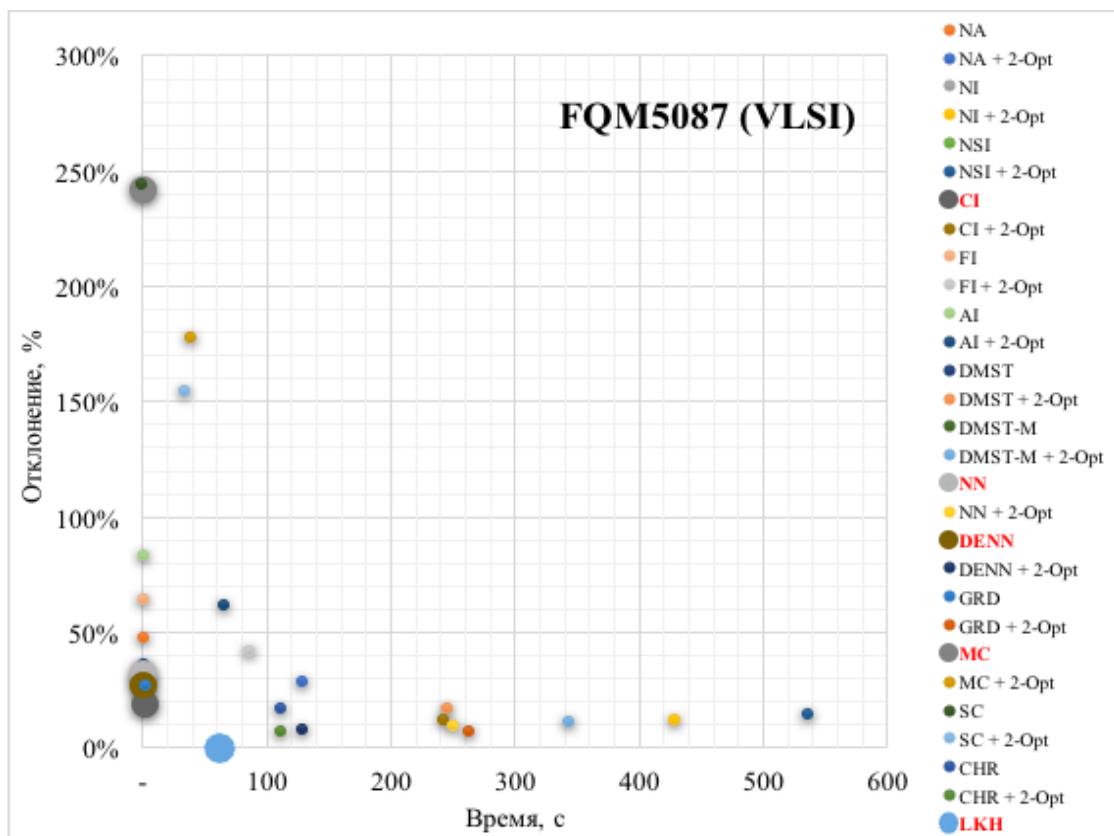


Рисунок Ж.13. Парето-оптимальные алгоритмы для набора данных FQM5087

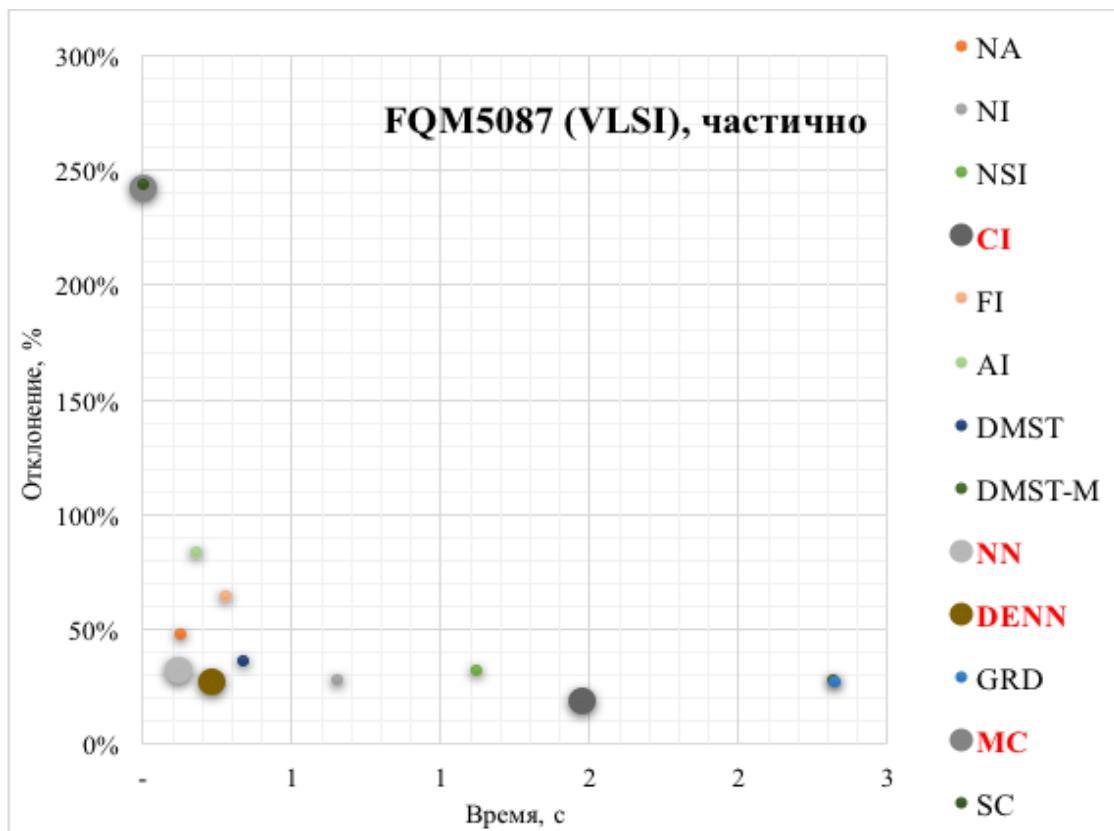


Рисунок Ж.14. Парето-оптимальные алгоритмы для набора данных FQM5087 (частично)

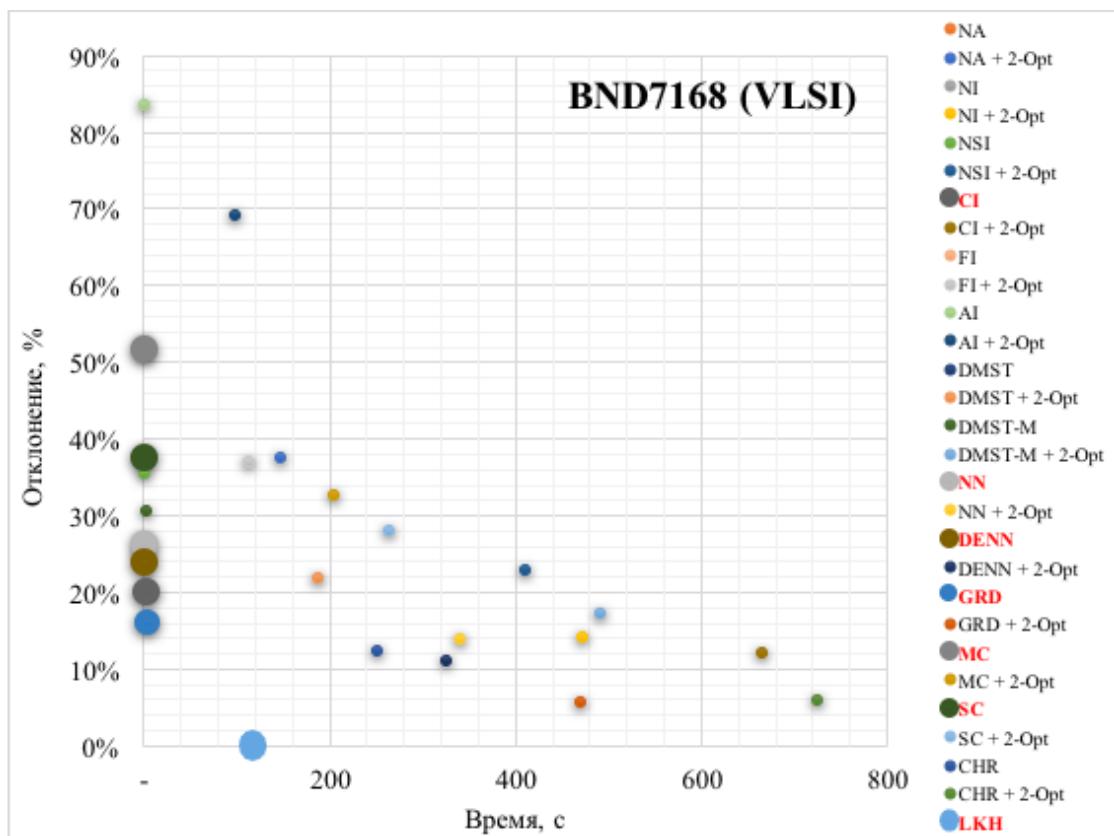


Рисунок Ж.15. Парето-оптимальные алгоритмы для набора данных BND7168

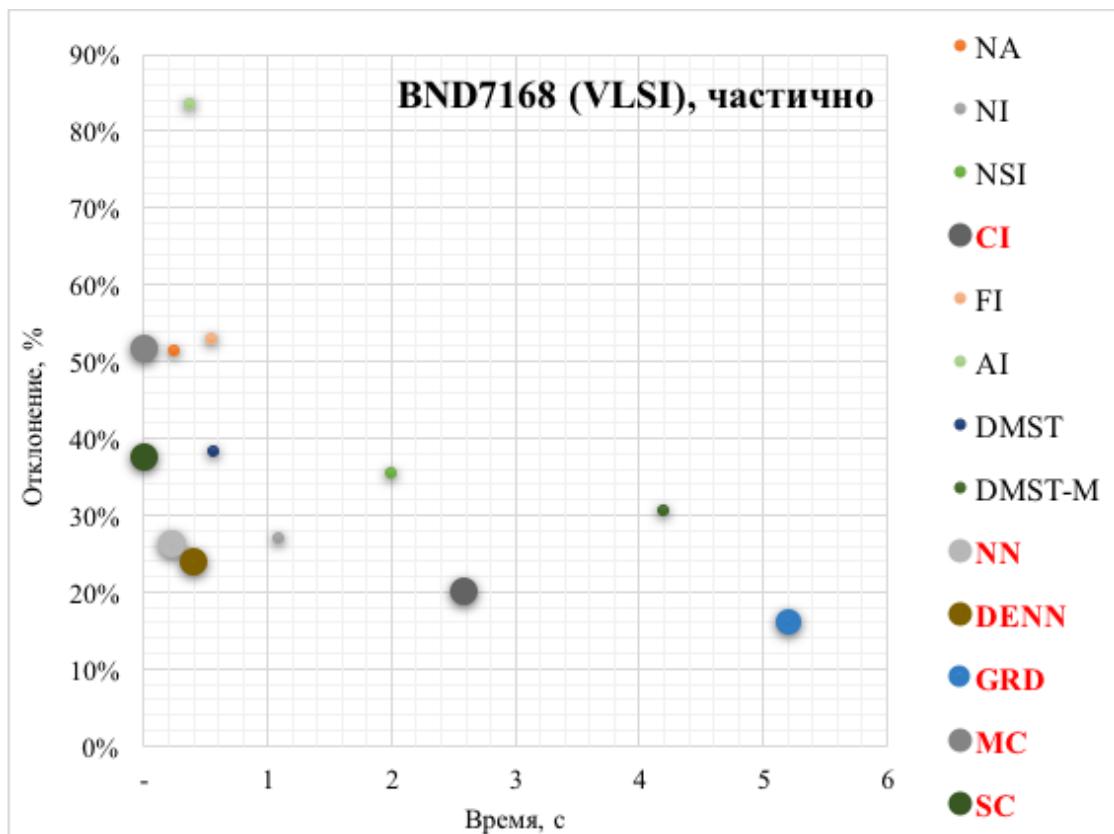


Рисунок Ж.16. Парето-оптимальные алгоритмы для набора данных BND7168 (частично)

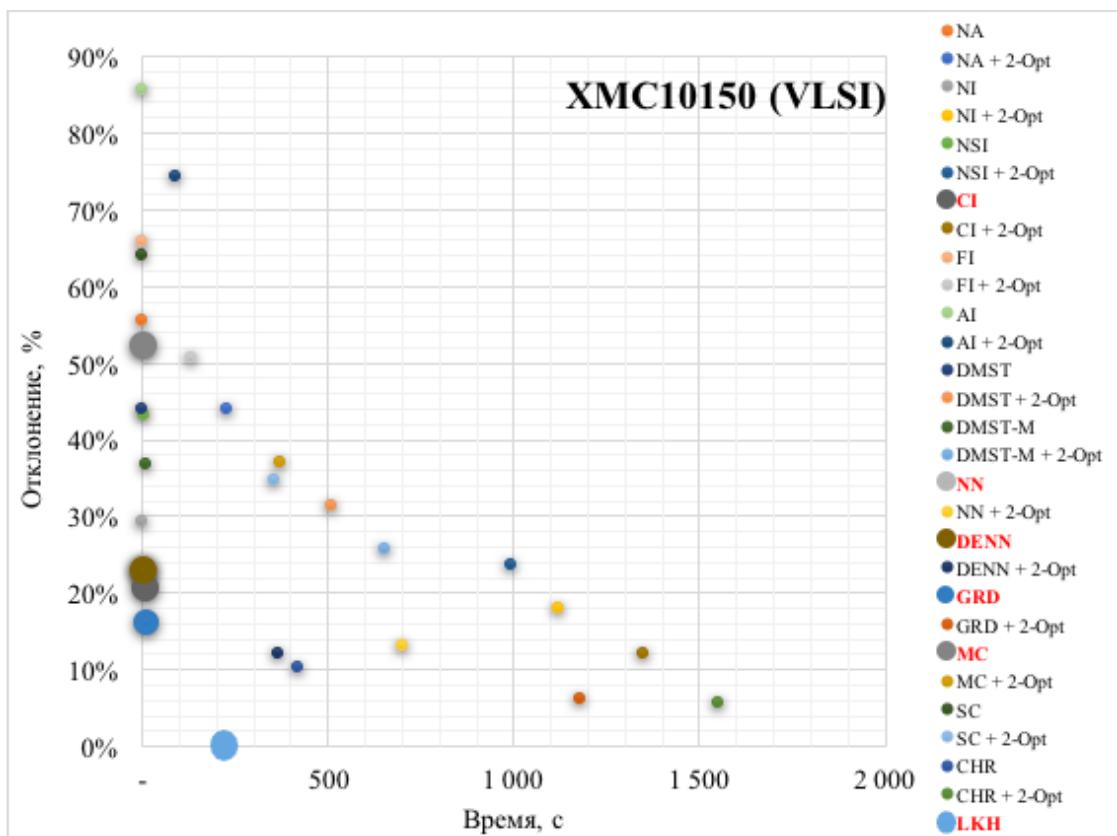


Рисунок Ж.17. Парето-оптимальные алгоритмы для набора данных XMC10150

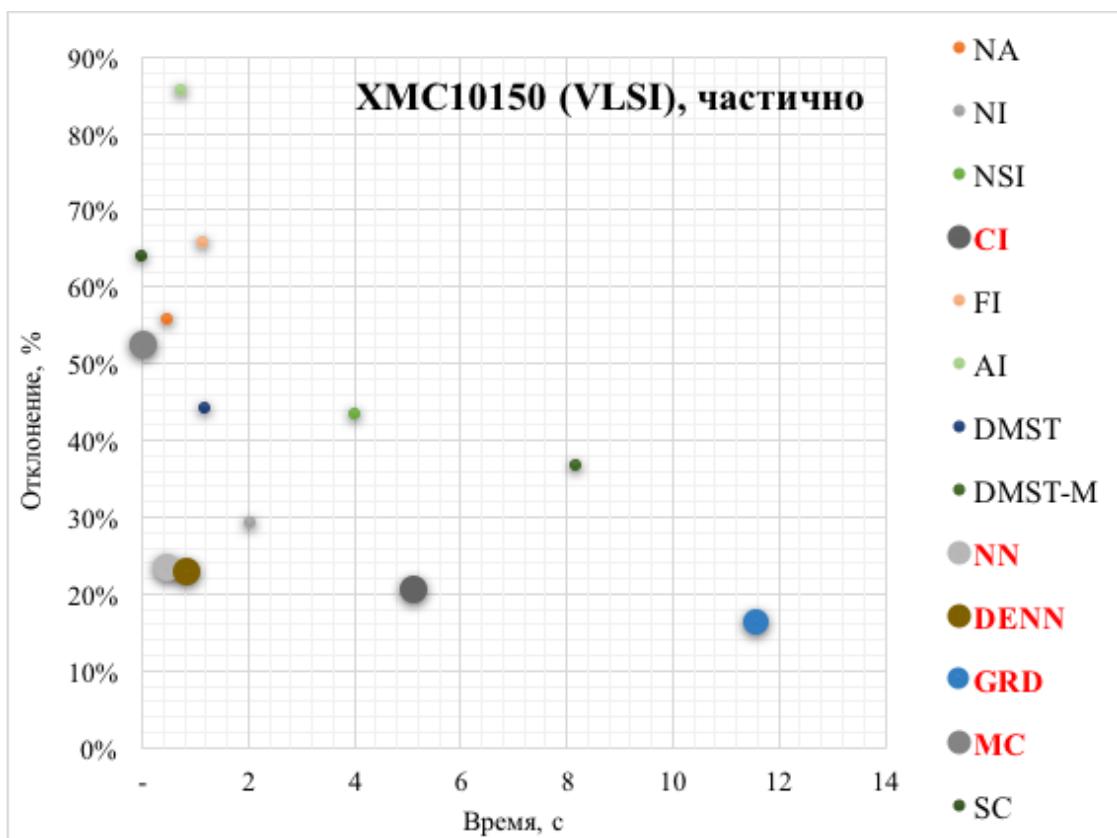


Рисунок Ж.18. Парето-оптимальные алгоритмы для набора данных XMC10150 (частично)

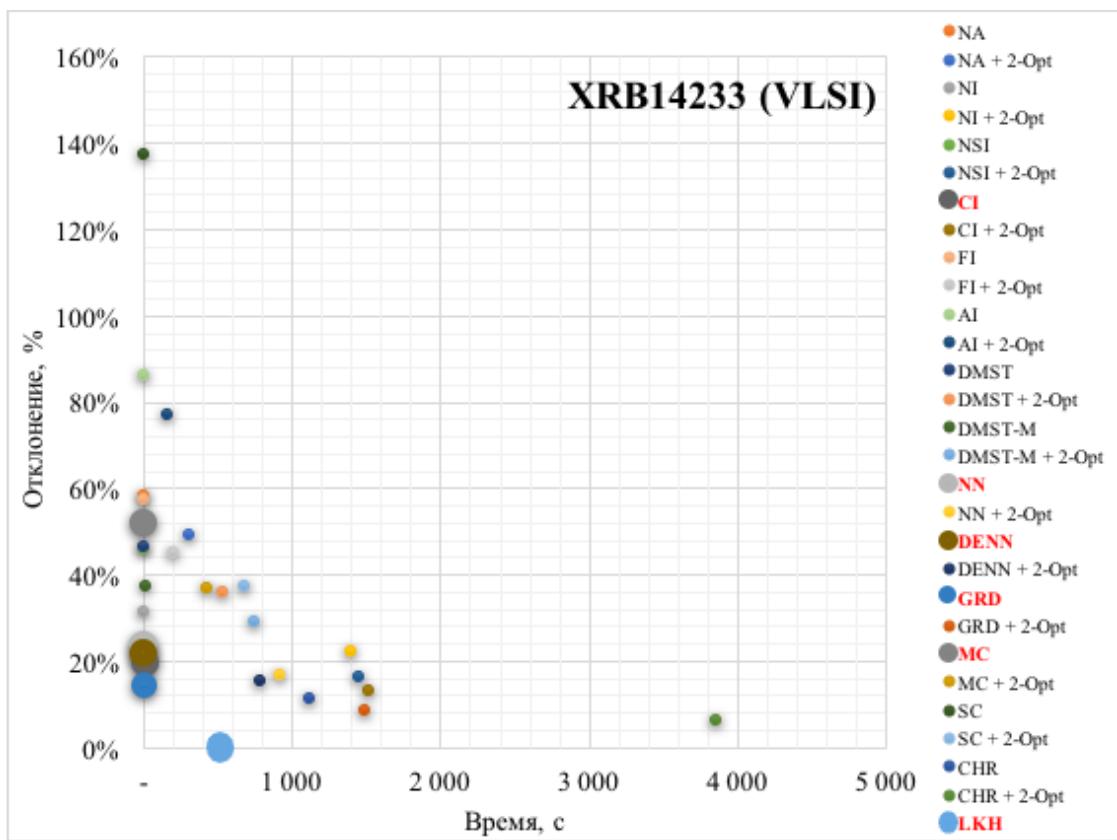


Рисунок Ж.19. Парето-оптимальные алгоритмы для набора данных XRB14233

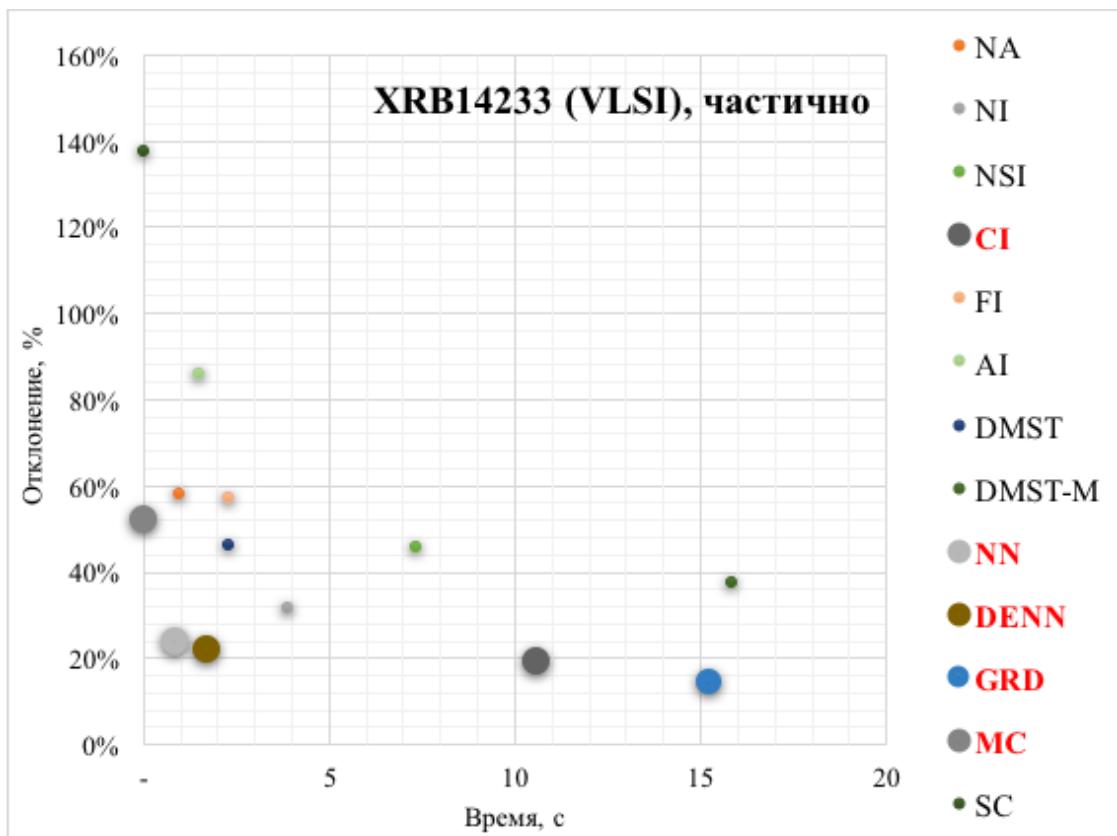


Рисунок Ж.20. Парето-оптимальные алгоритмы для набора данных XRB14233 (частично)

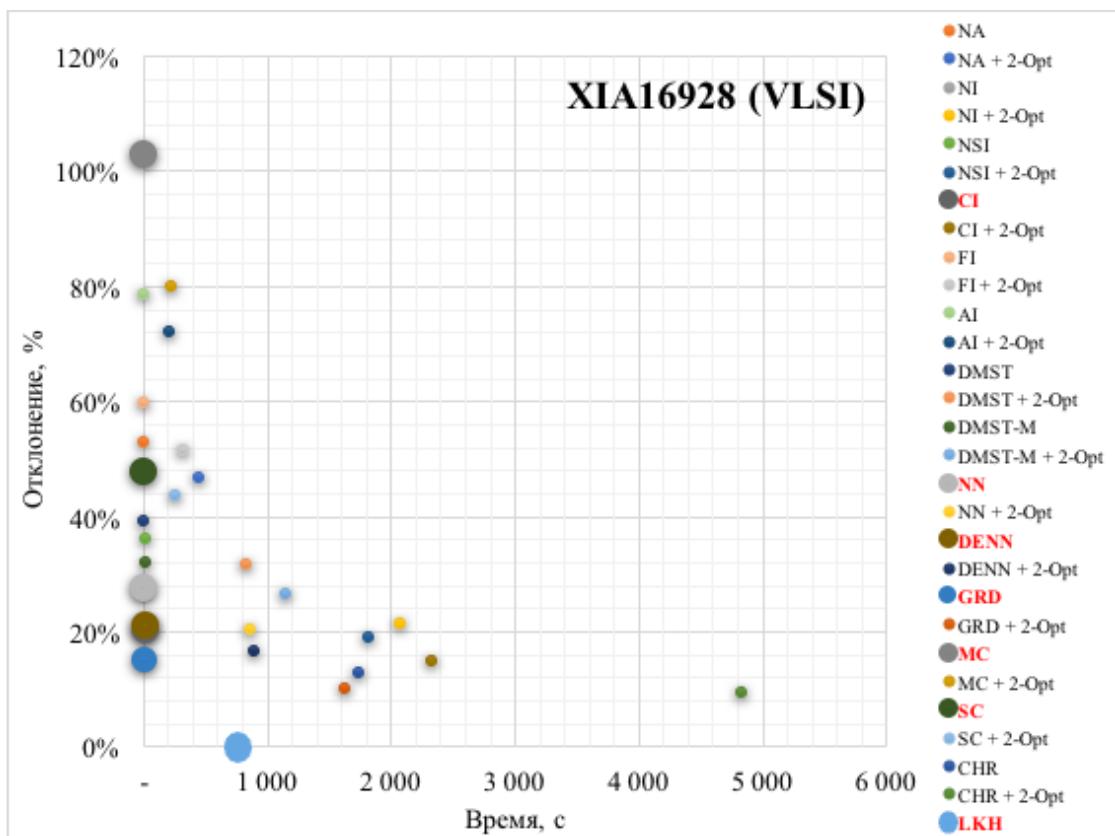


Рисунок Ж.21. Парето-оптимальные алгоритмы для набора данных XIA16928

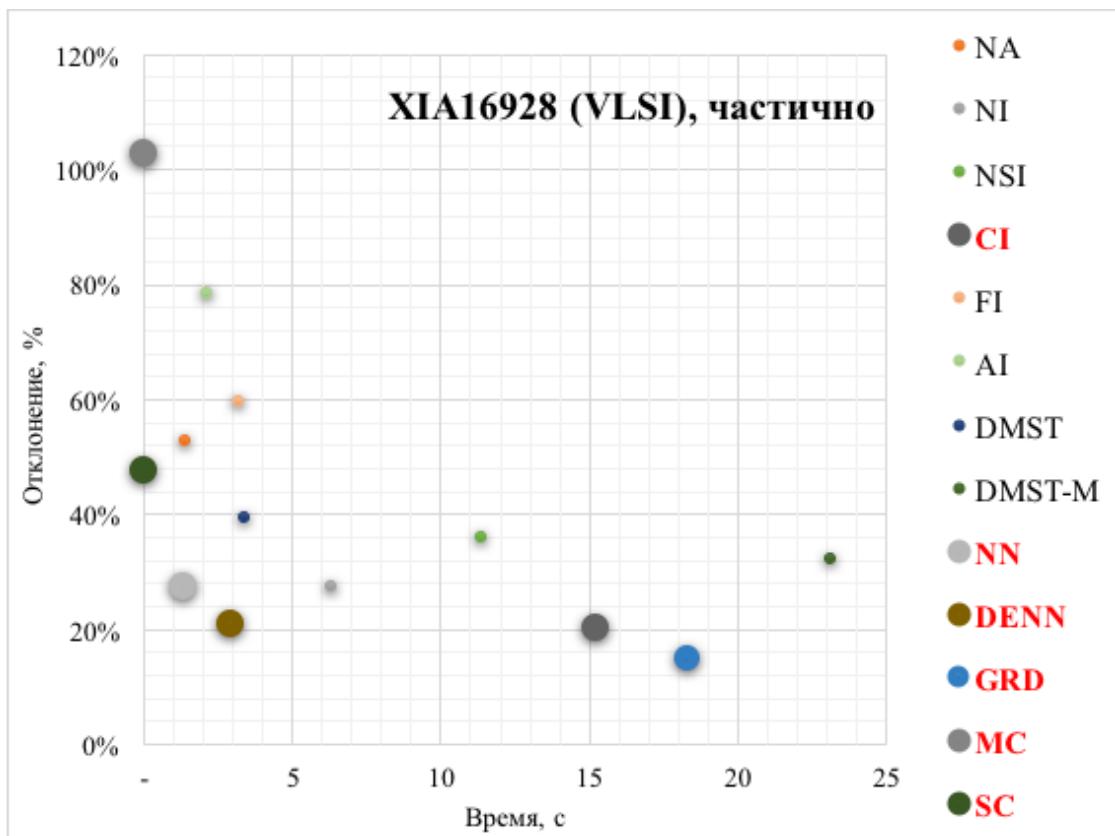


Рисунок Ж.22. Парето-оптимальные алгоритмы для набора данных XIA16928 (частично)

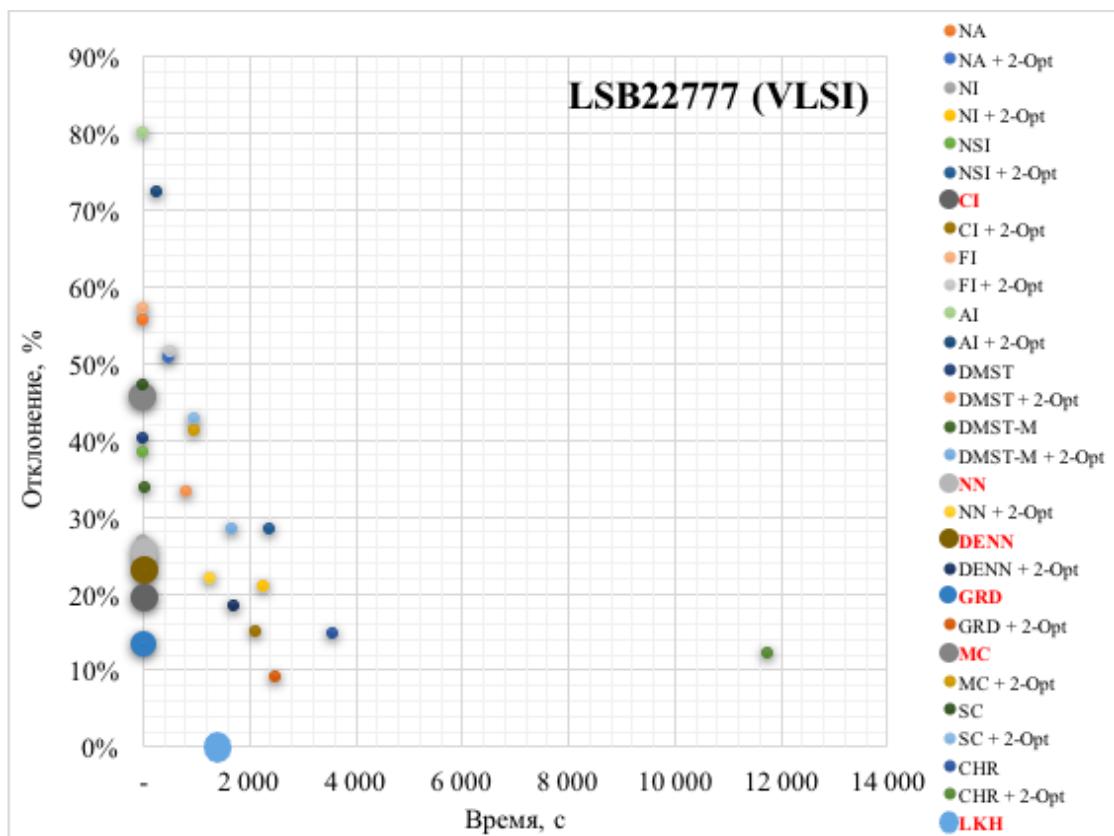


Рисунок Ж.23. Парето-оптимальные алгоритмы для набора данных LSB22777

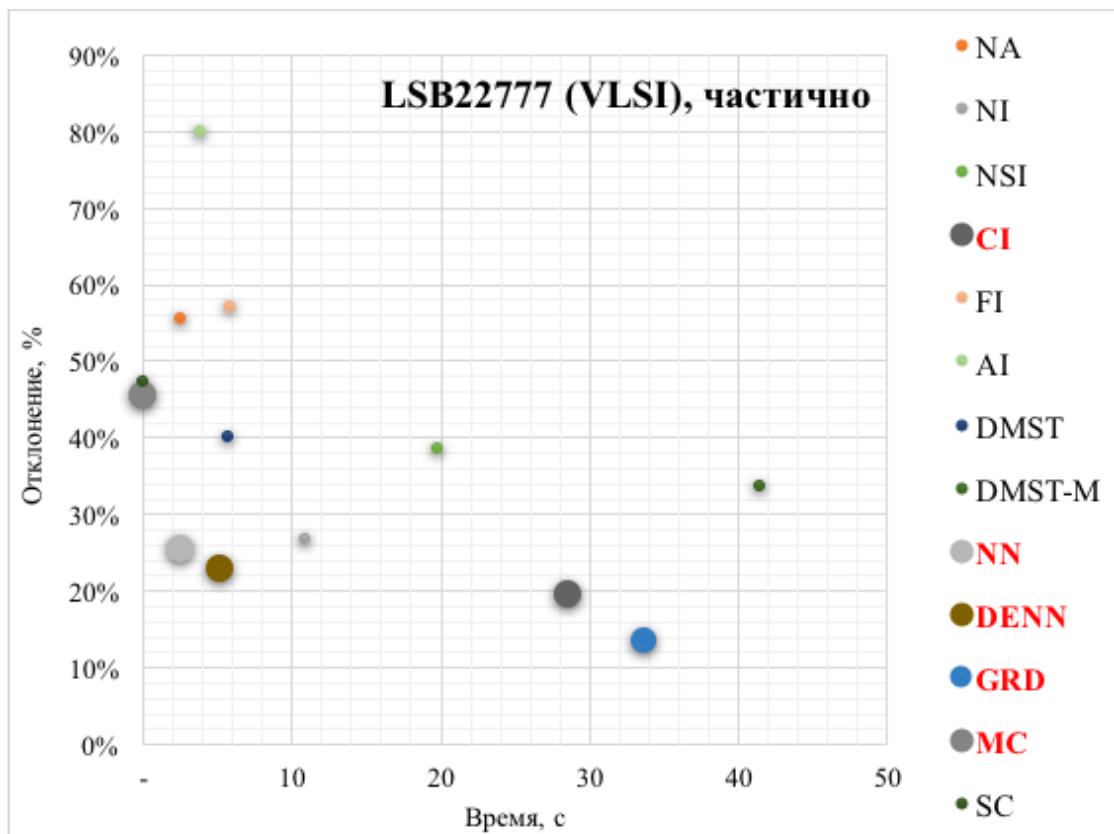


Рисунок Ж.24. Парето-оптимальные алгоритмы для набора данных LSB22777 (частично)

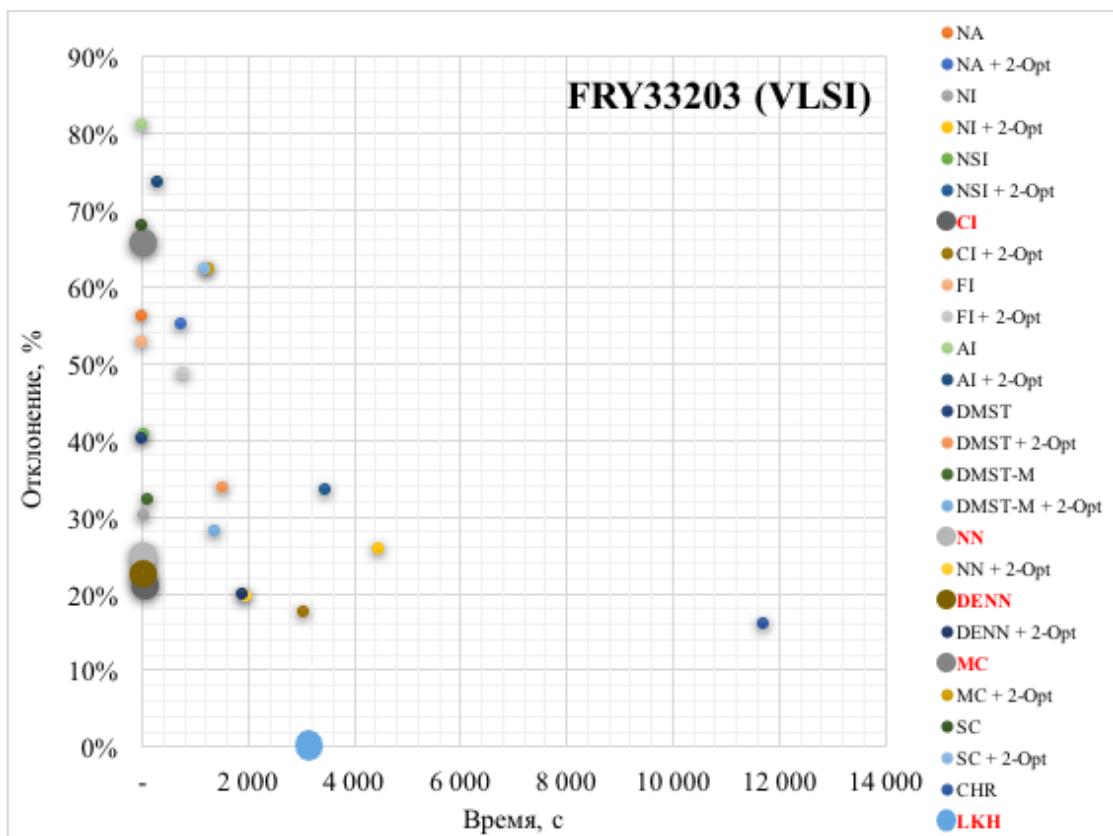


Рисунок Ж.25. Парето-оптимальные алгоритмы для набора данных FRY33203

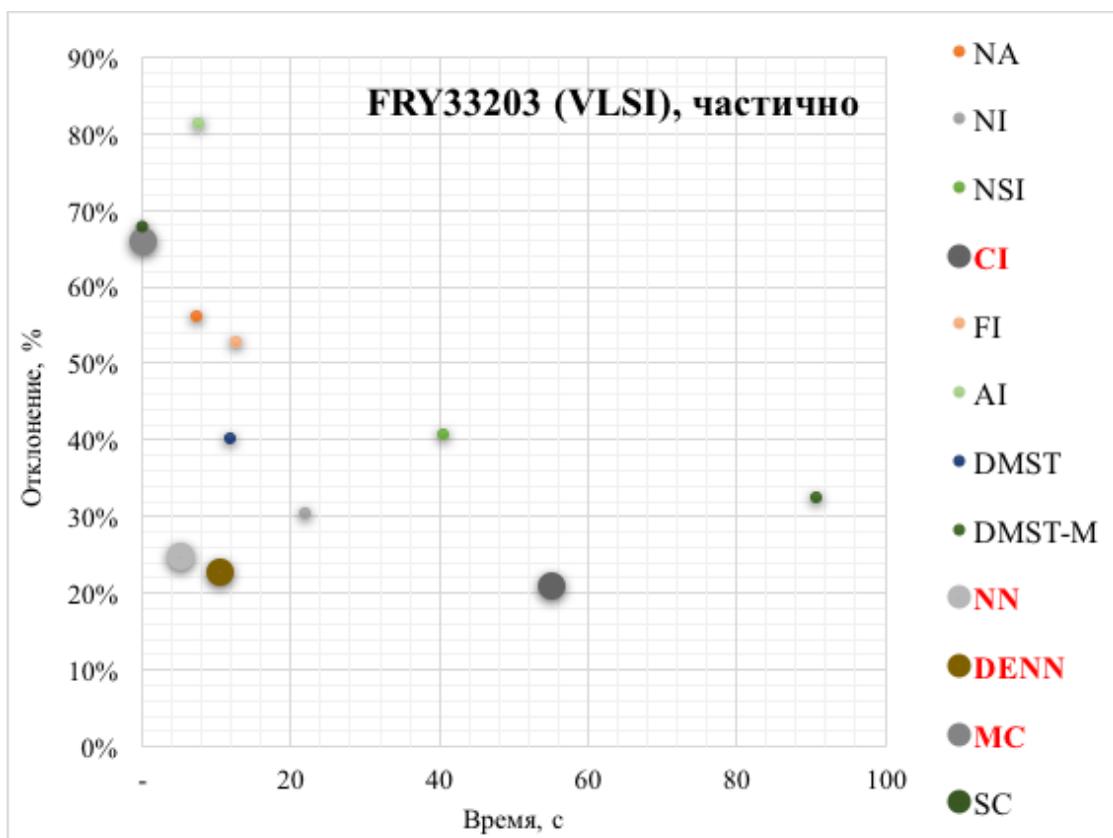


Рисунок Ж.26. Парето-оптимальные алгоритмы для набора данных FRY33203 (частично)

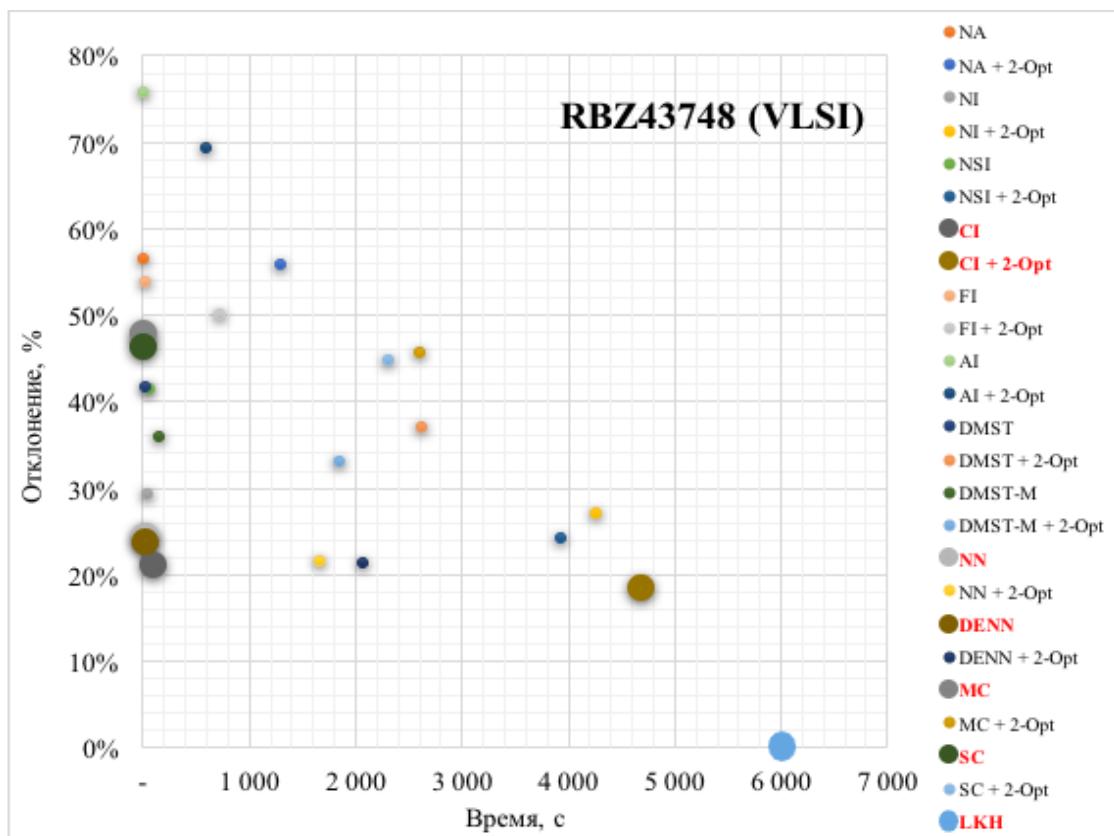


Рисунок Ж.27. Парето-оптимальные алгоритмы для набора данных RBZ43748

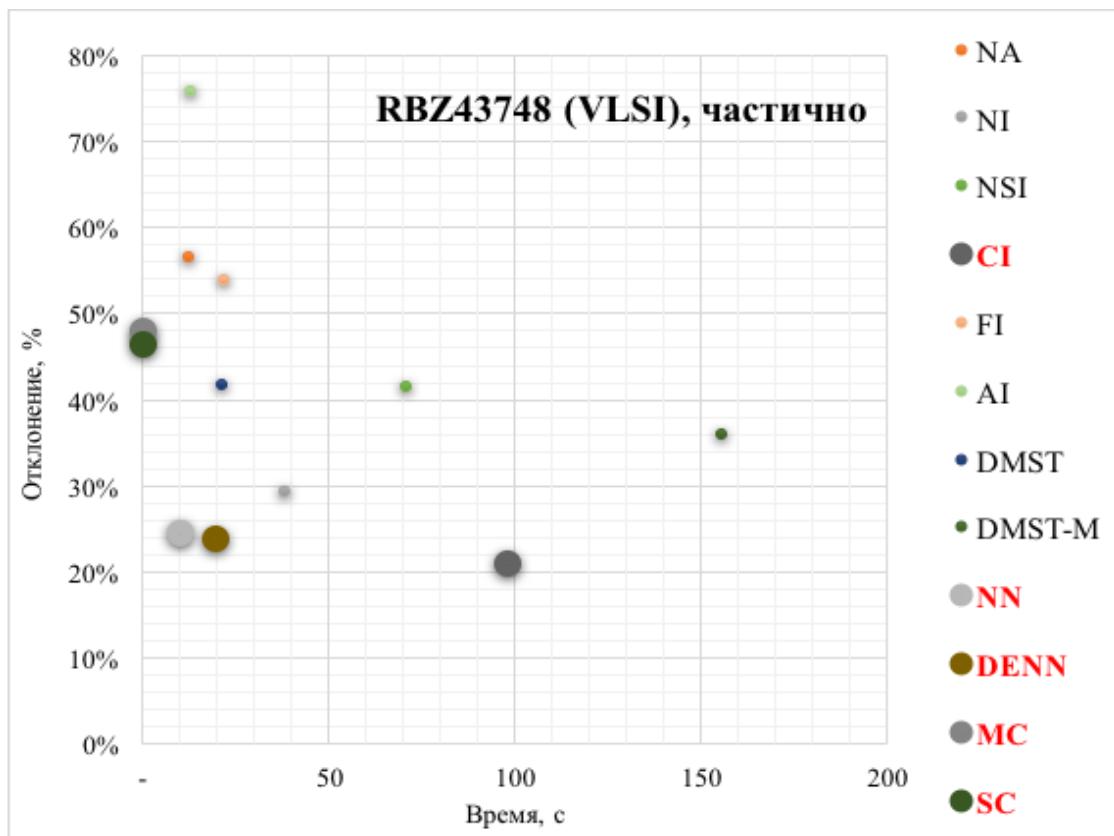


Рисунок Ж.28. Парето-оптимальные алгоритмы для набора данных RBZ43748 (частично)

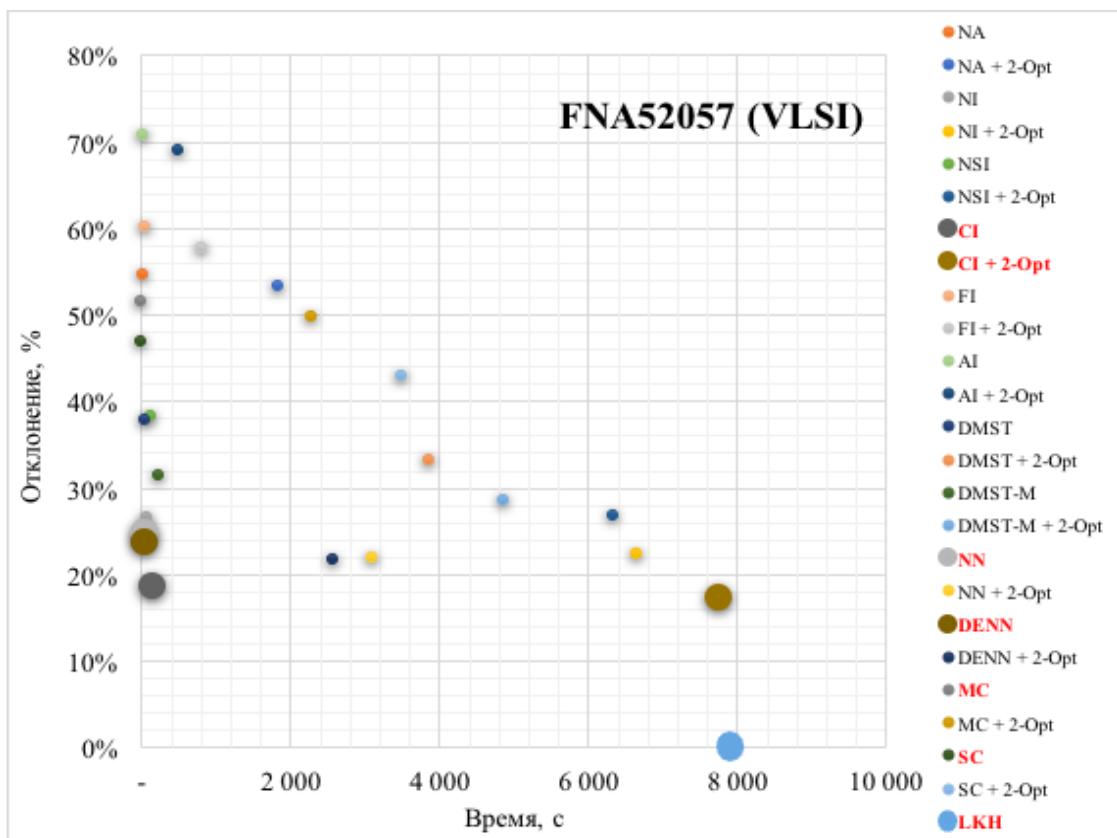


Рисунок Ж.29. Парето-оптимальные алгоритмы для набора данных FNA52057

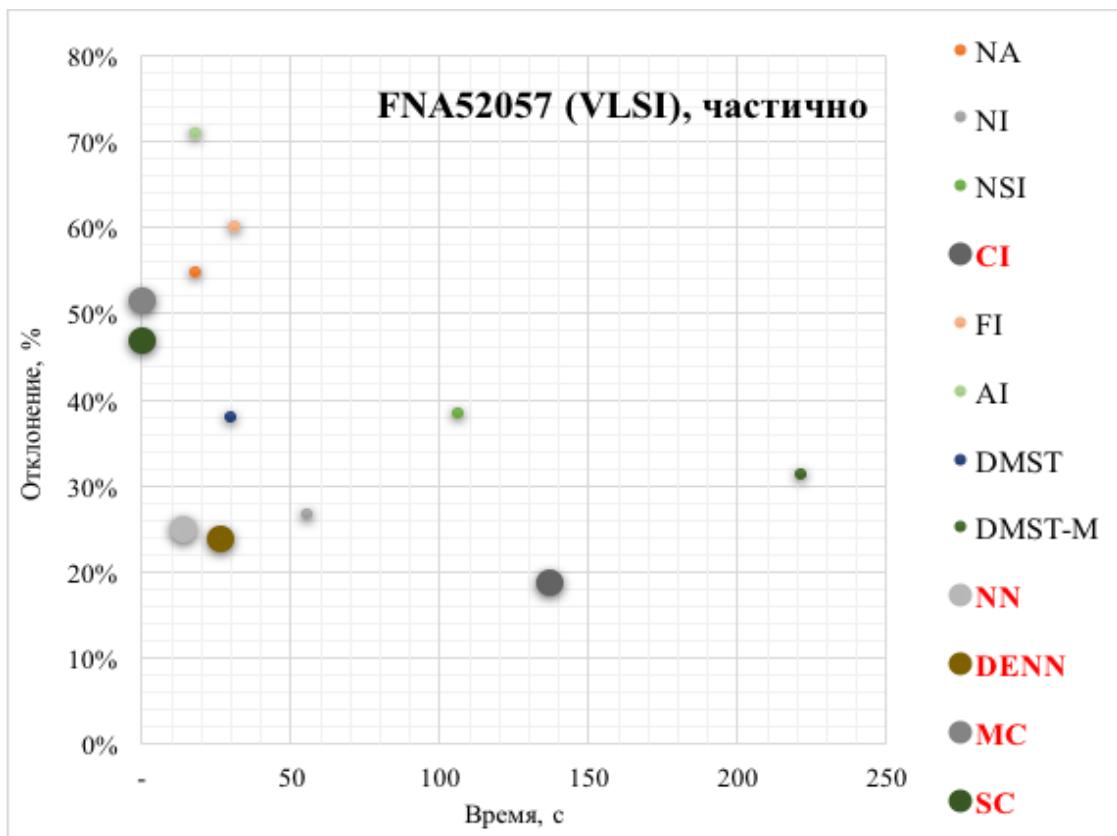


Рисунок Ж.30. Парето-оптимальные алгоритмы для набора данных FNA52057 (частично)

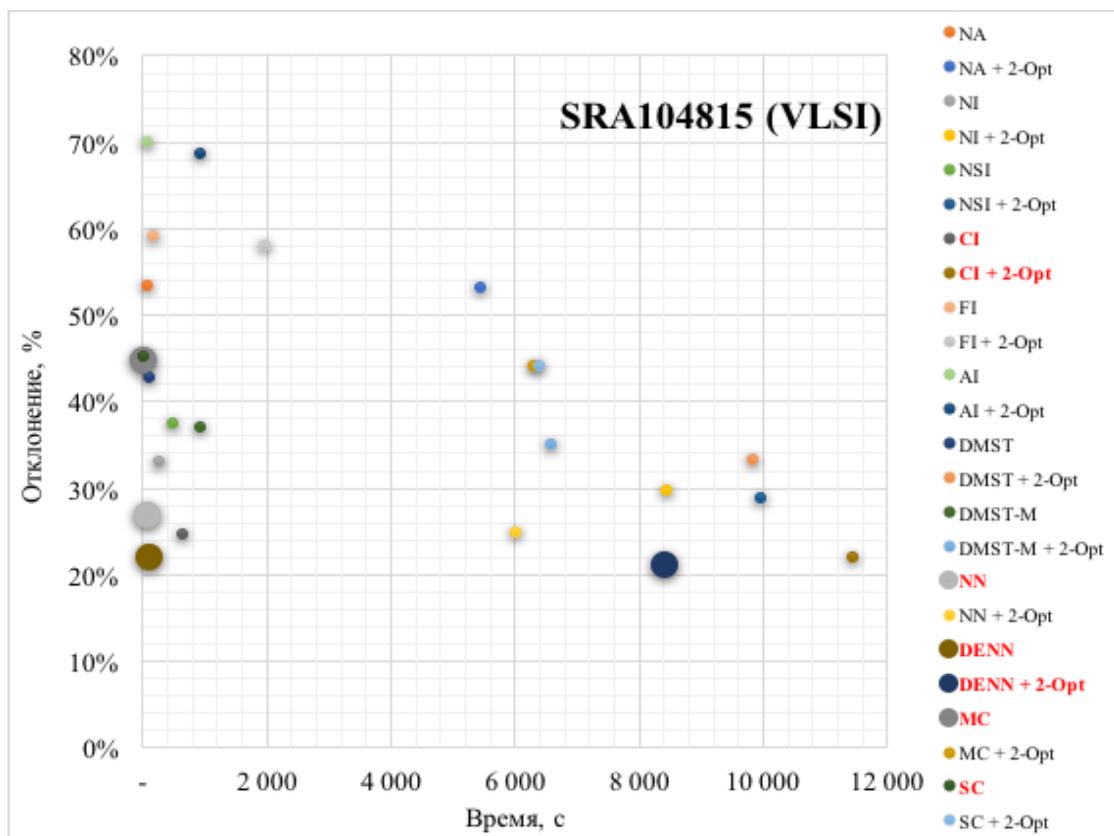


Рисунок Ж.31. Парето-оптимальные алгоритмы для набора данных SRA104815

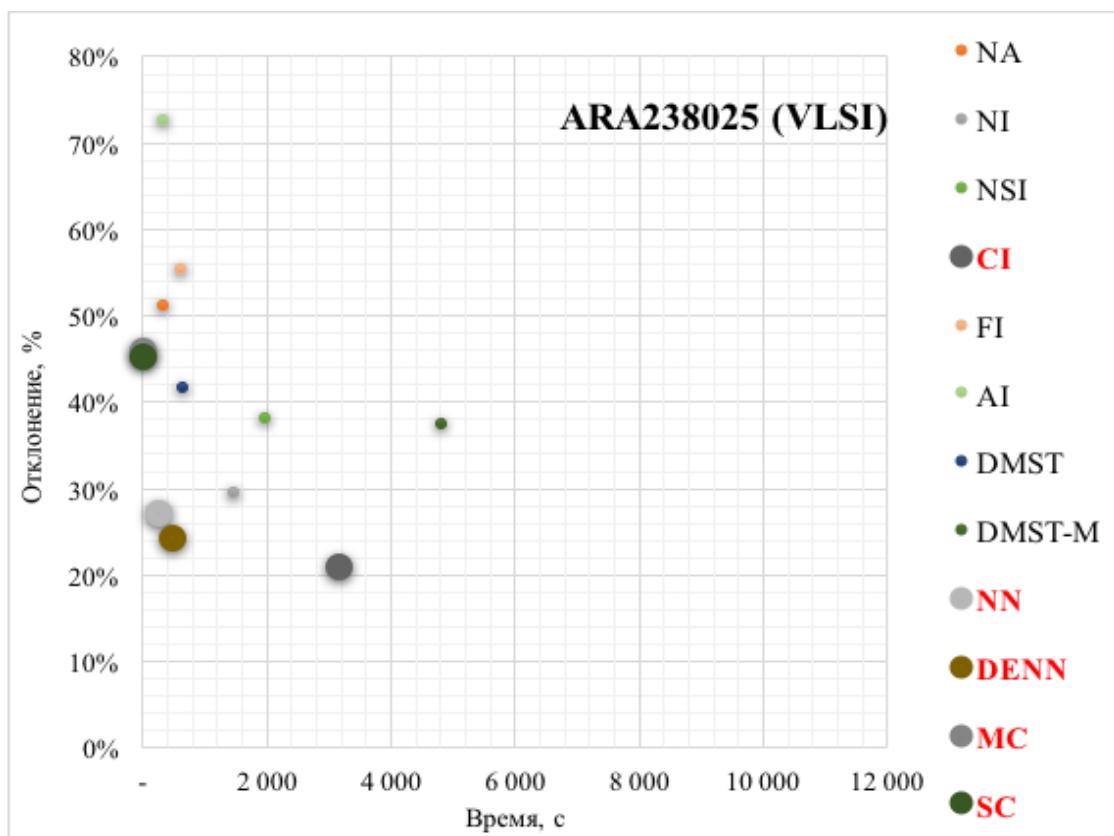


Рисунок Ж.32. Парето-оптимальные алгоритмы для набора данных ARA238025

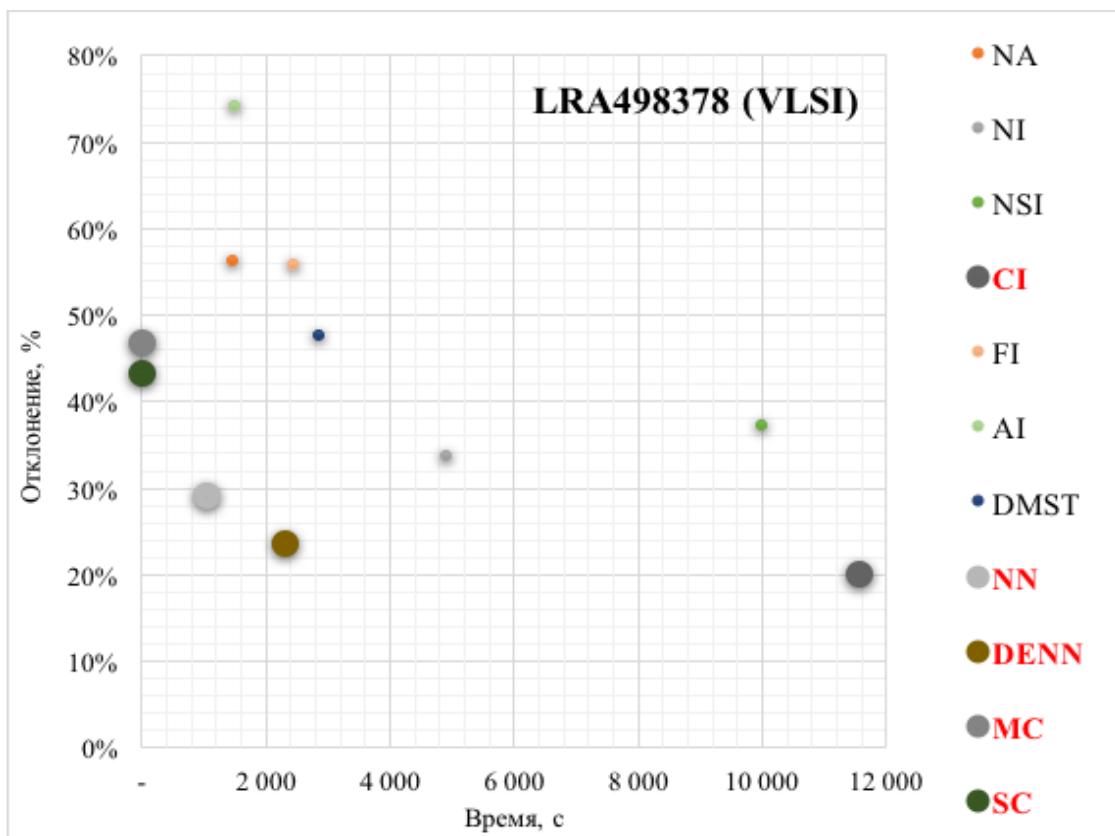


Рисунок Ж.33. Парето-оптимальные алгоритмы для набора данных LRA498378

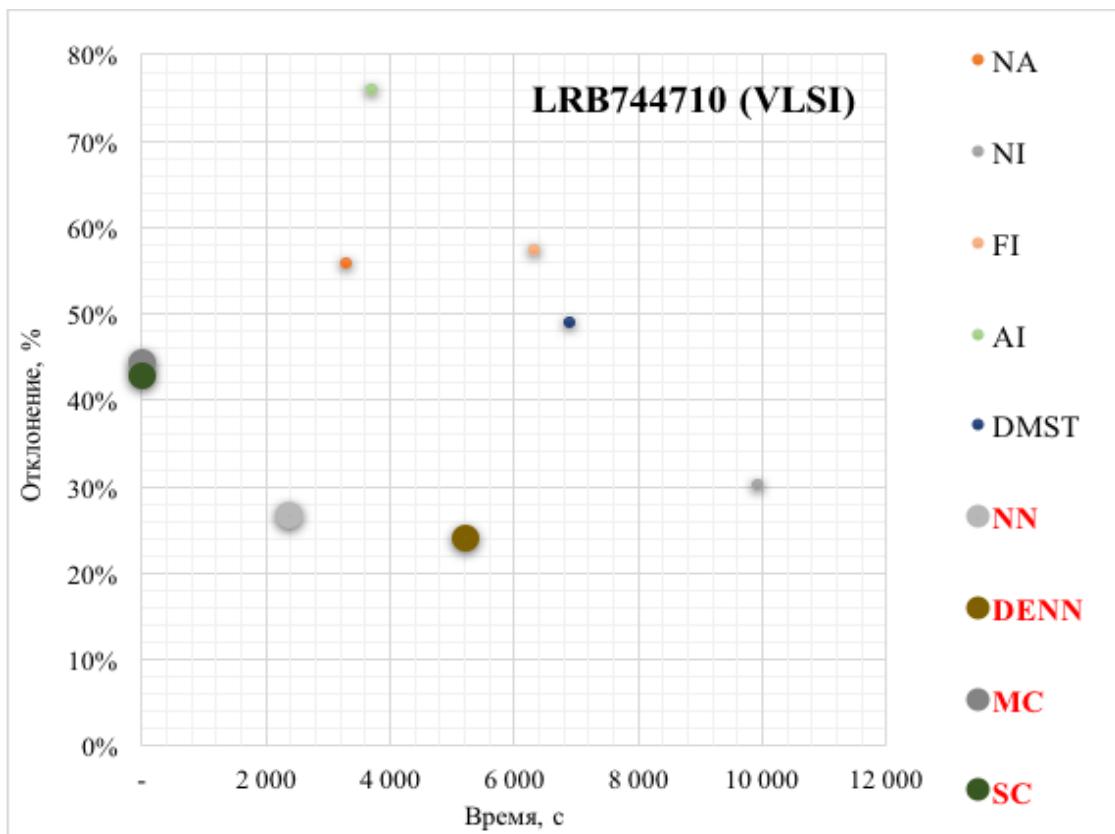


Рисунок Ж.34. Парето-оптимальные алгоритмы для набора данных LRB744710

Приложение 3

Графики Парето-оптимальных алгоритмов (National TSPs)

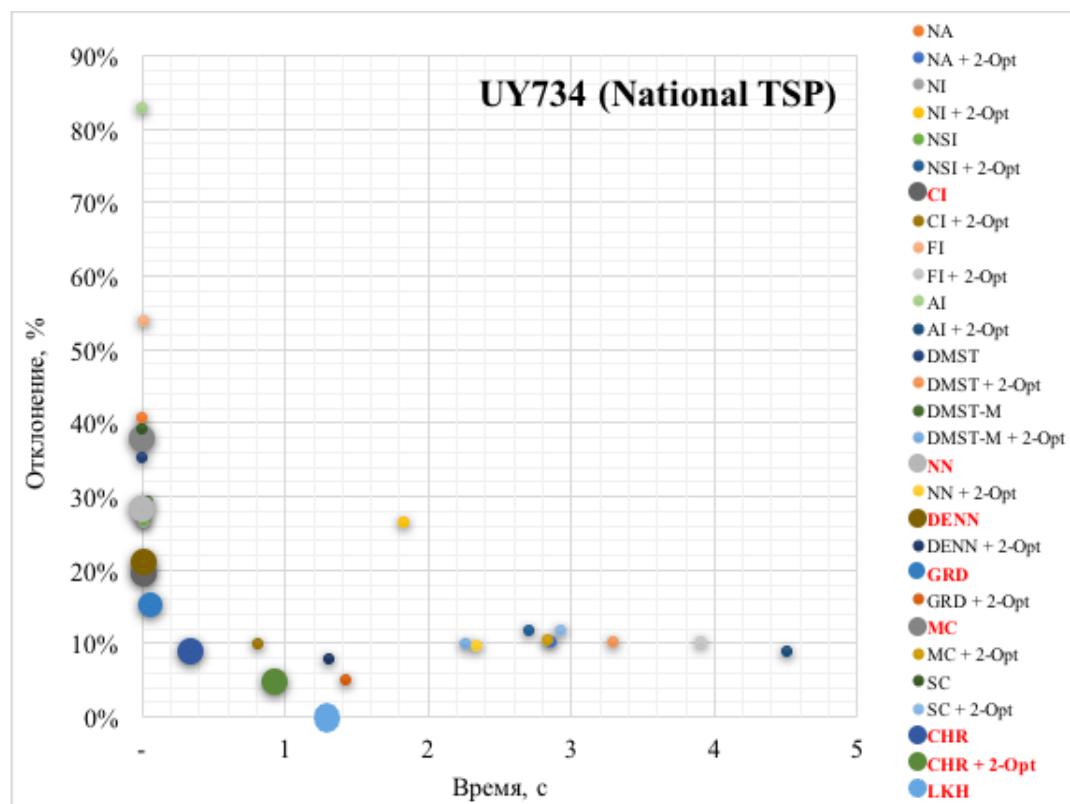


Рисунок 3.1. Парето-оптимальные алгоритмы для набора данных UY734

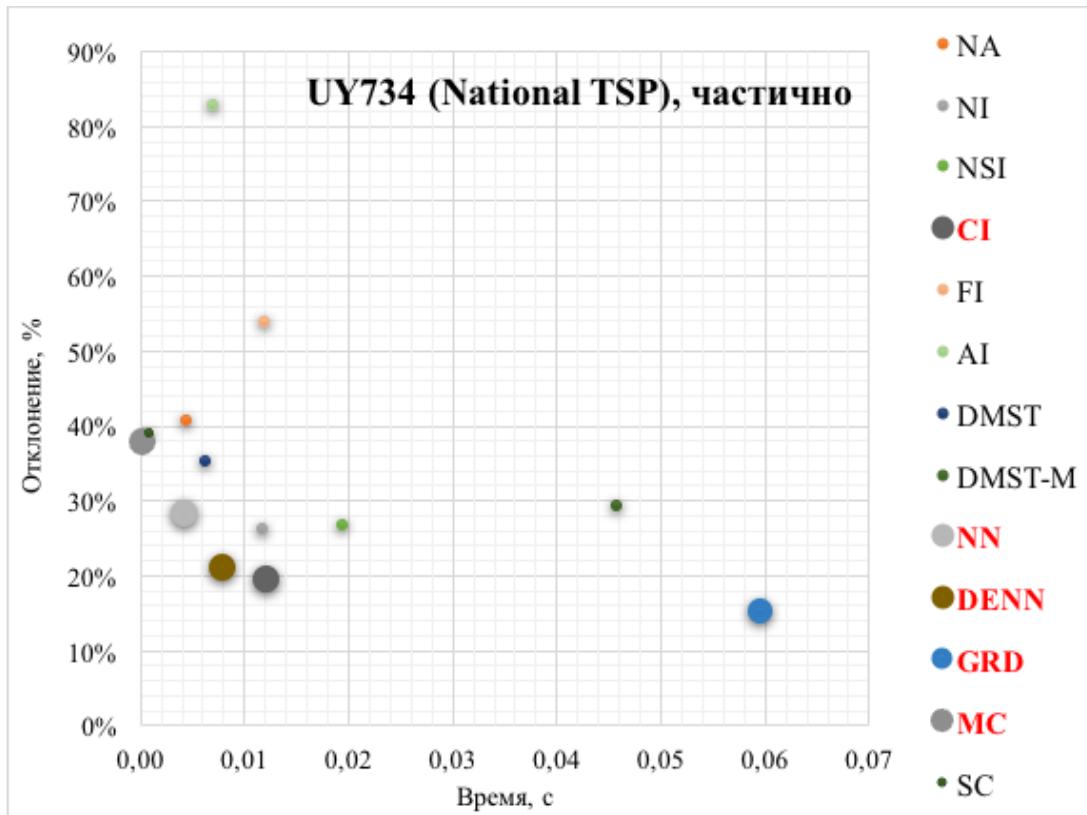


Рисунок Е. 3. Парето-оптимальные алгоритмы для набора данных UY734 (частично)

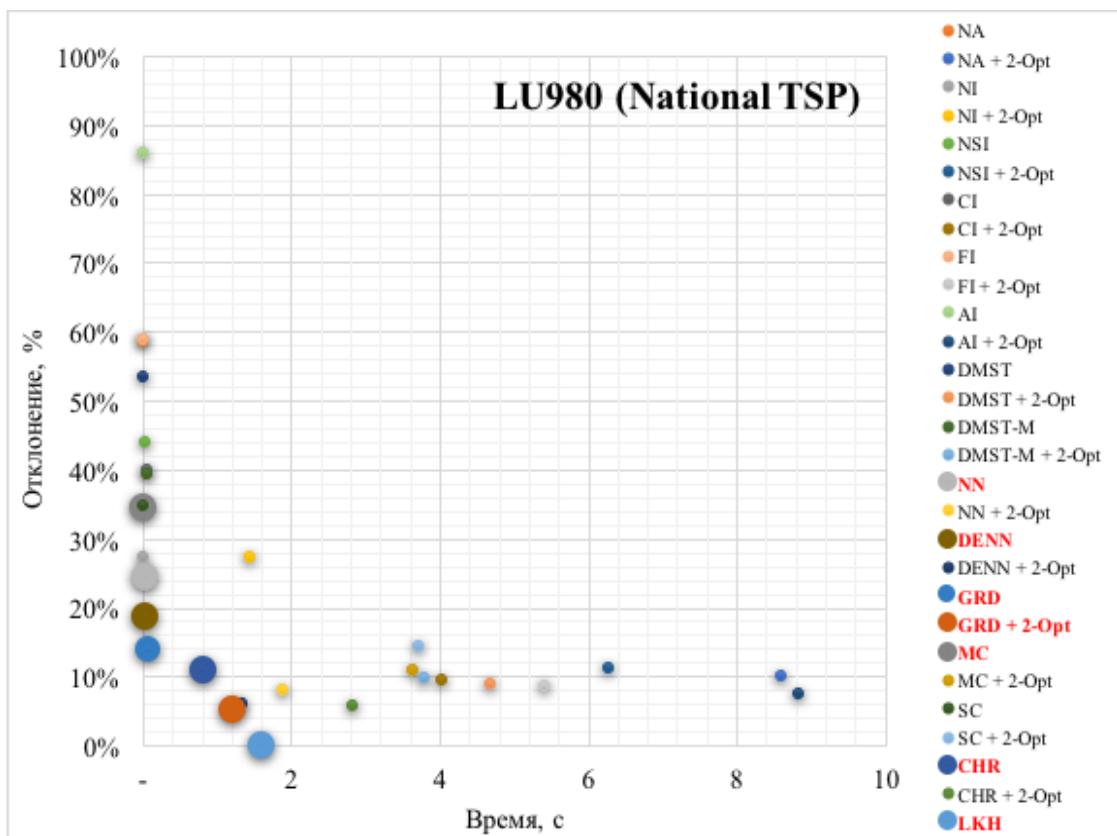


Рисунок 3.3. Парето-оптимальные алгоритмы для набора данных LU980

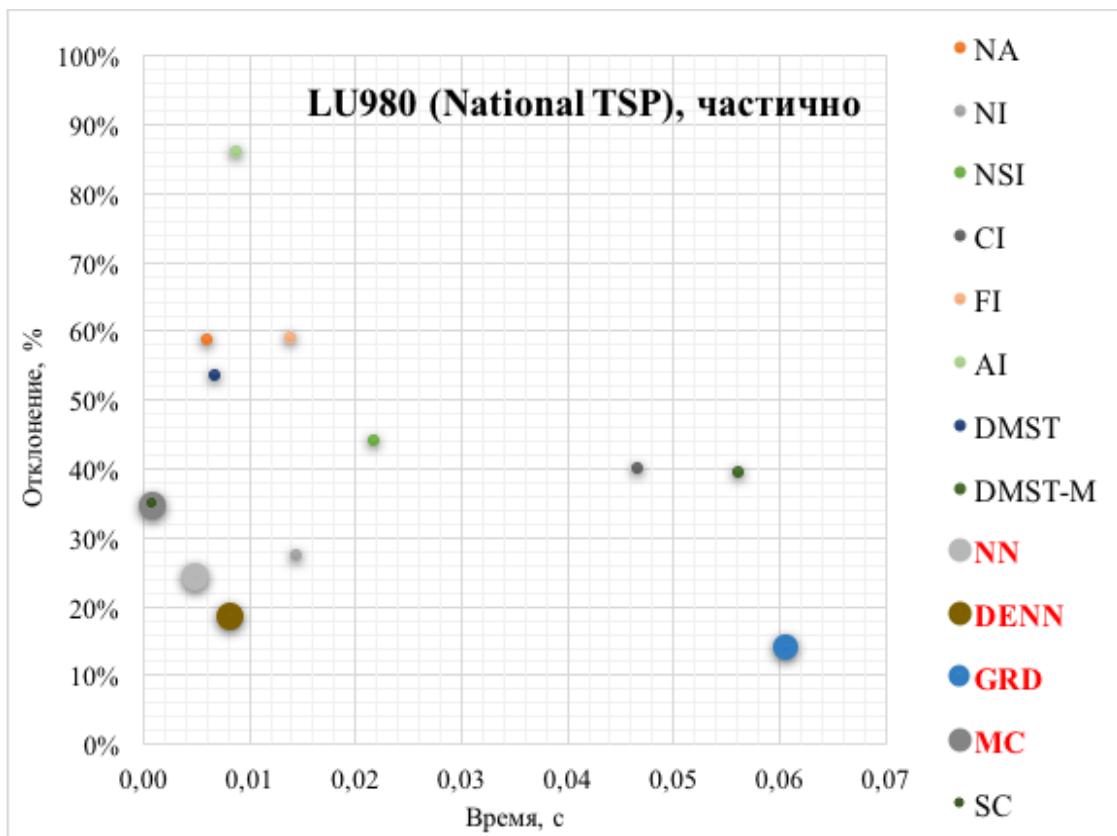


Рисунок 3.4. Парето-оптимальные алгоритмы для набора данных LU980 (частично)

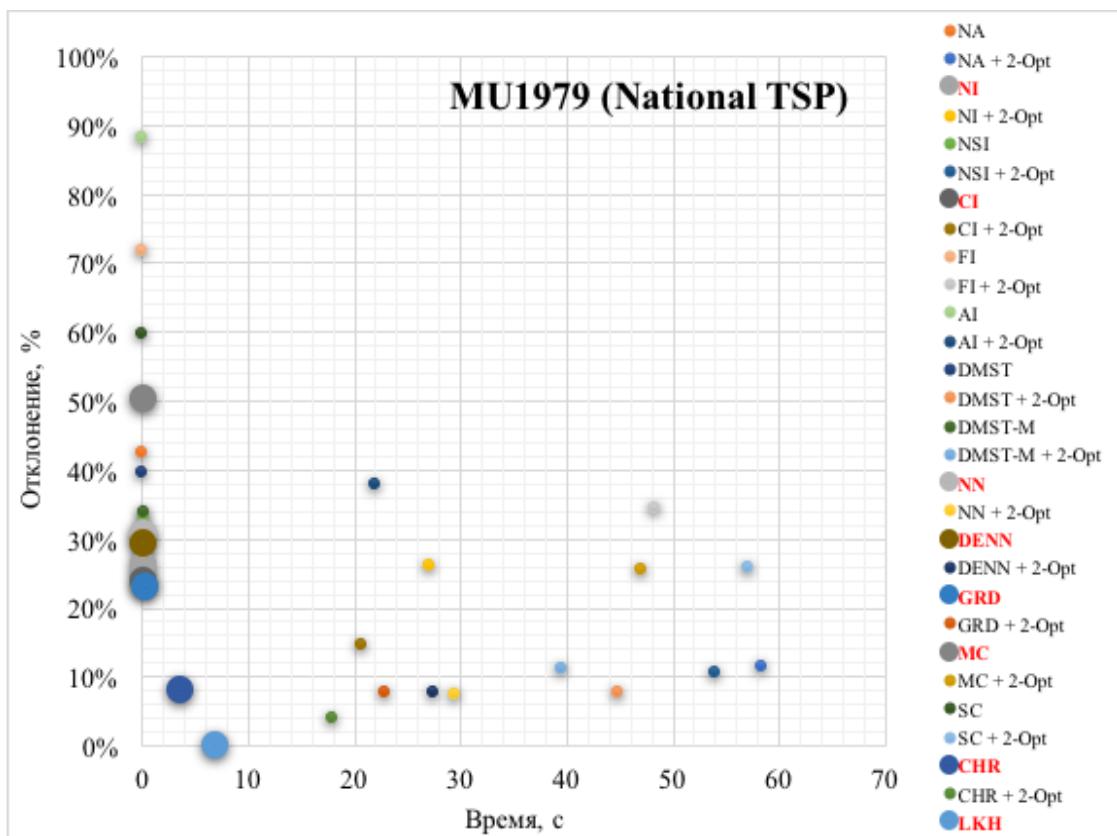


Рисунок 3.5. Парето-оптимальные алгоритмы для набора данных MU1979

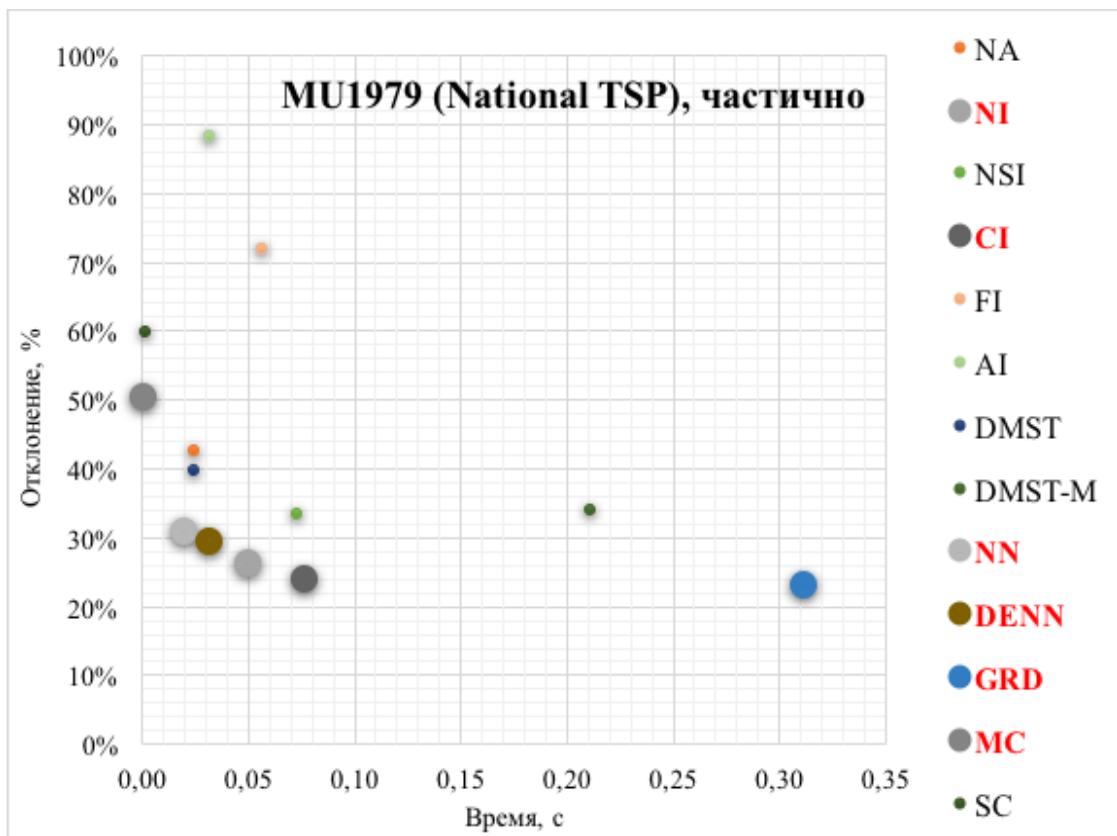


Рисунок 3.6. Парето-оптимальные алгоритмы для набора данных MU1979 (частично)

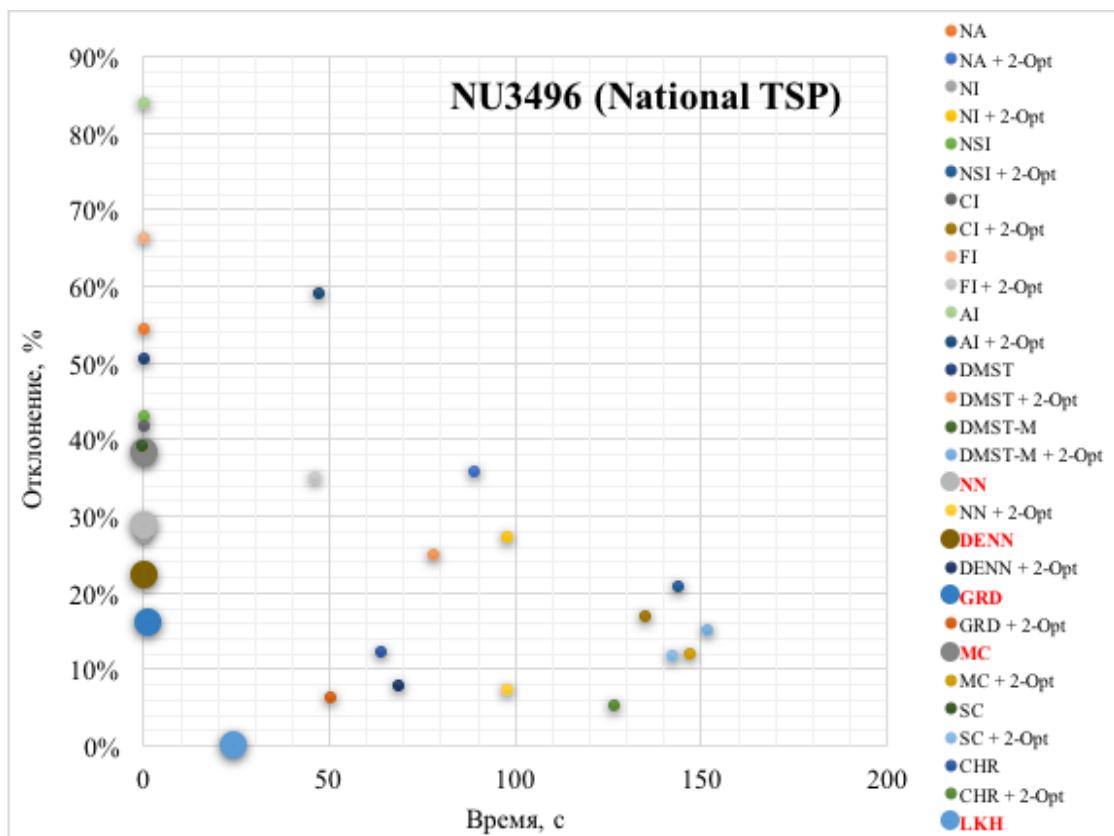


Рисунок 3.7. Парето-оптимальные алгоритмы для набора данных NU3496

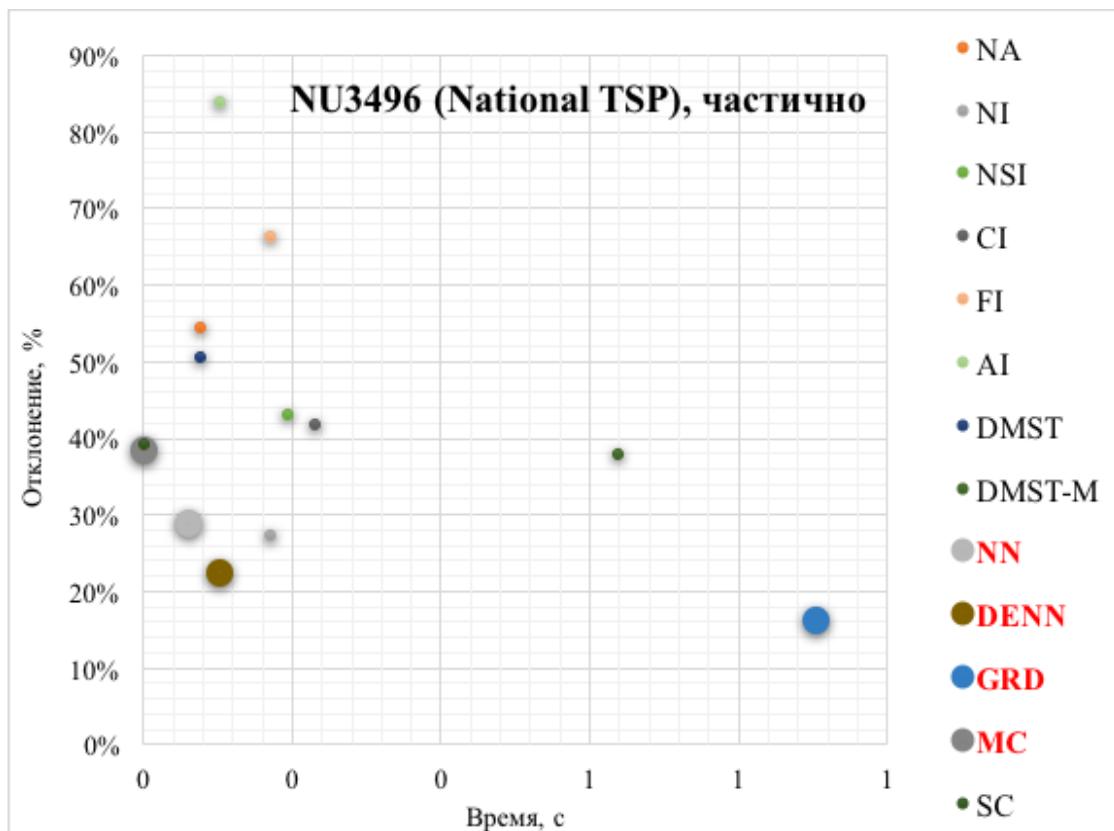


Рисунок 3.8. Парето-оптимальные алгоритмы для набора данных NU3496 (частично)

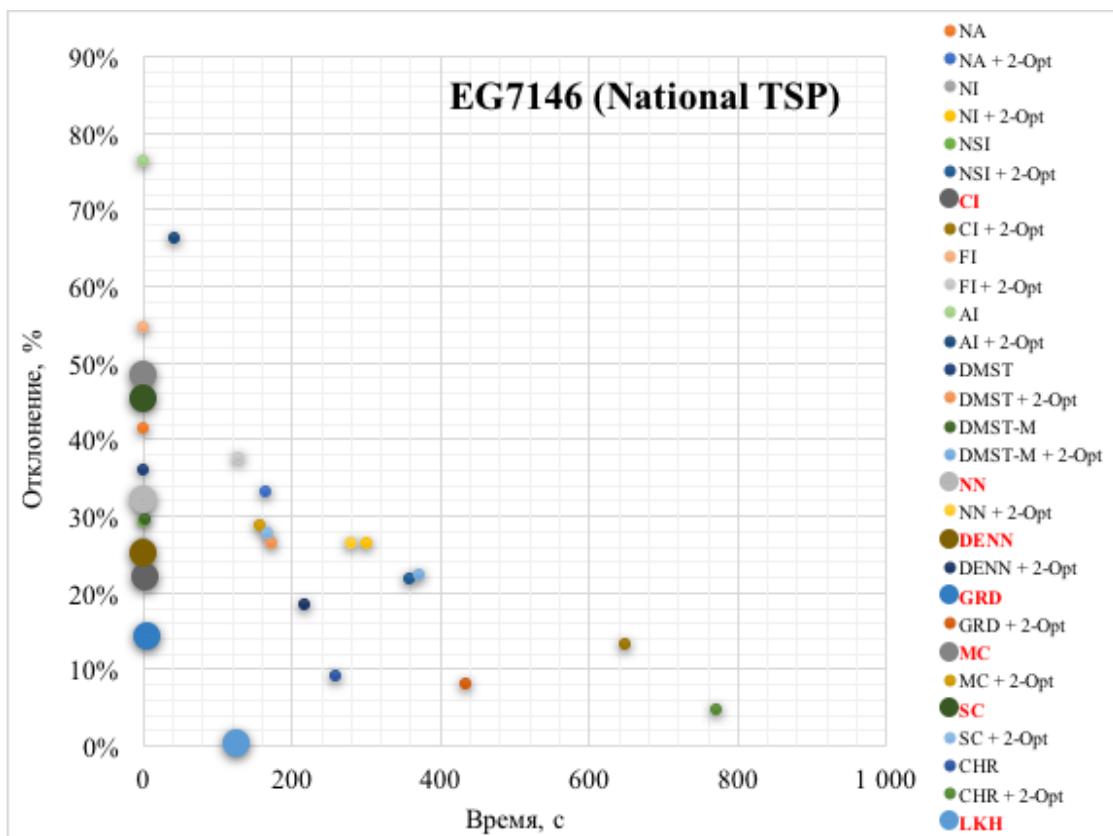


Рисунок 3.9. Парето-оптимальные алгоритмы для набора данных EG7146

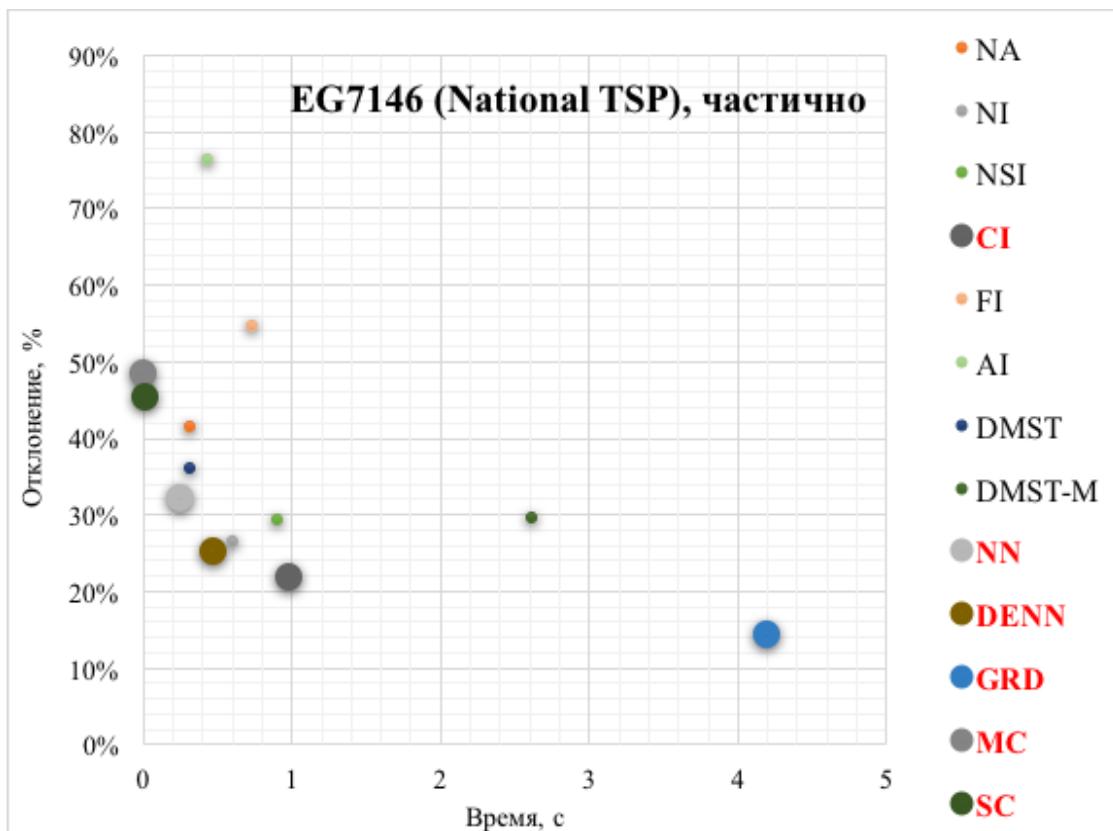


Рисунок 3.10. Парето-оптимальные алгоритмы для набора данных EG7146 (частично)

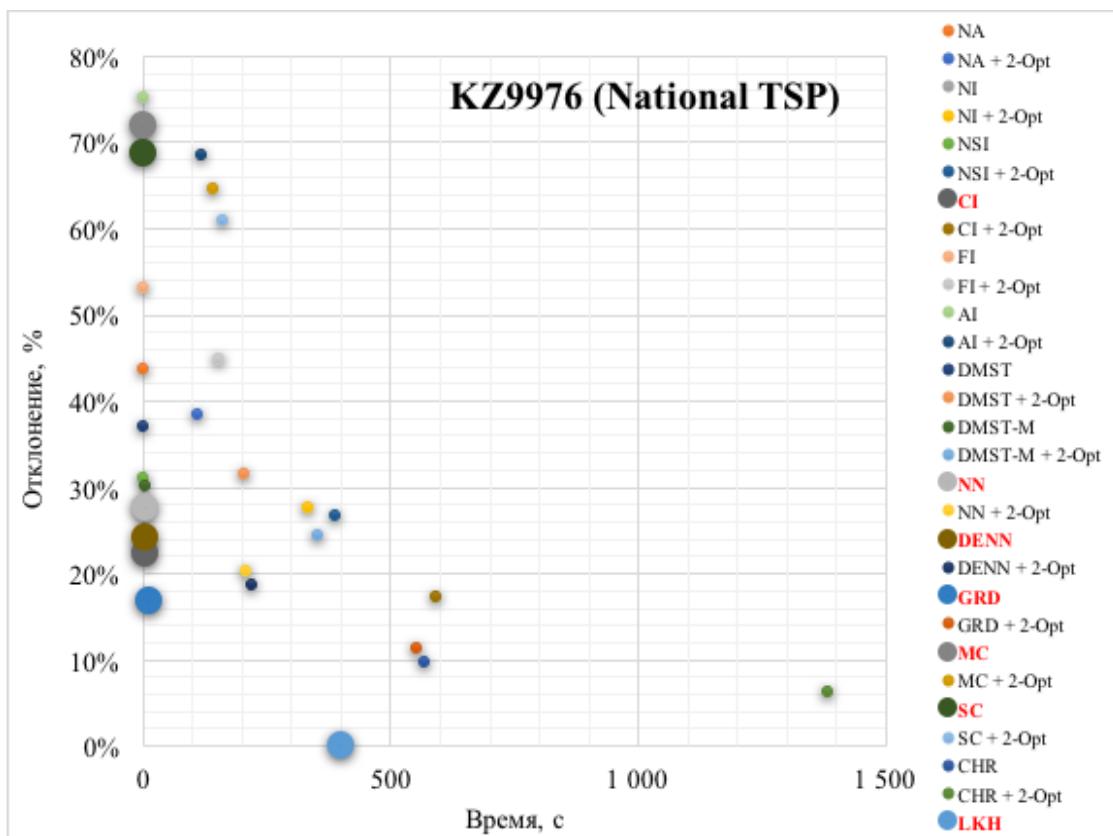


Рисунок 3.11. Парето-оптимальные алгоритмы для набора данных KZ9976

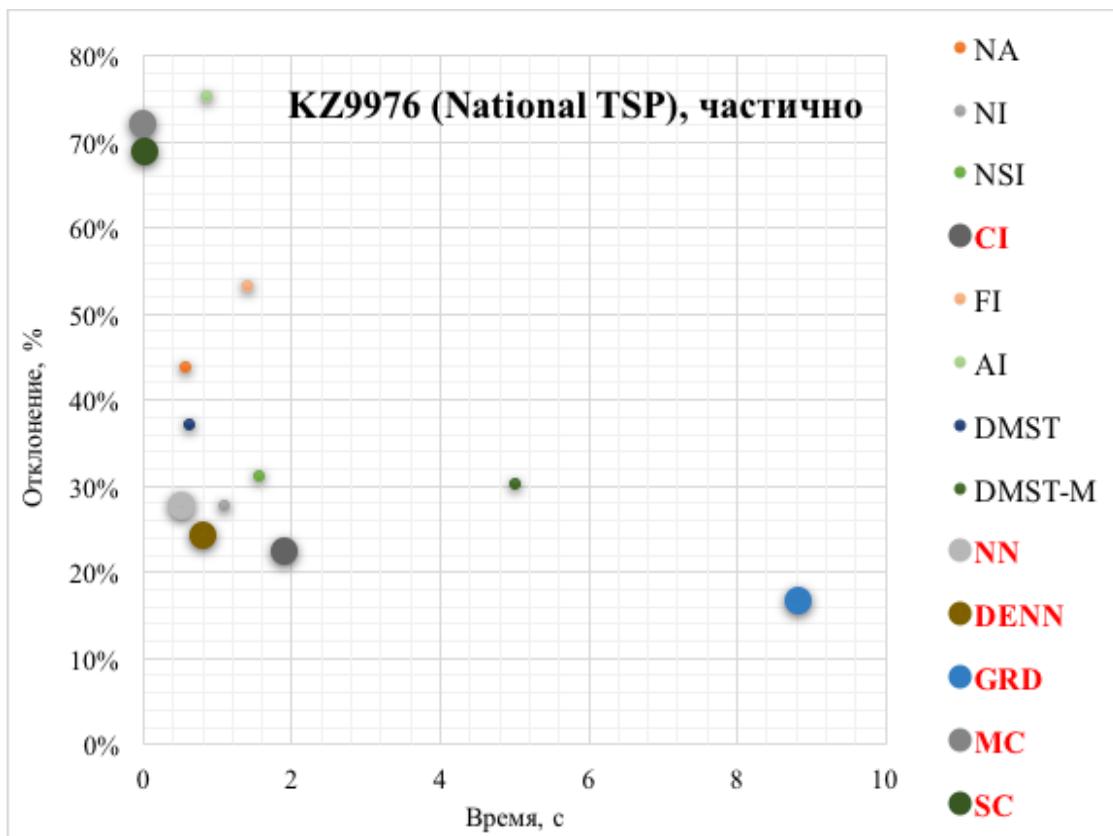


Рисунок 3.12. Парето-оптимальные алгоритмы для набора данных KZ9976 (частично)

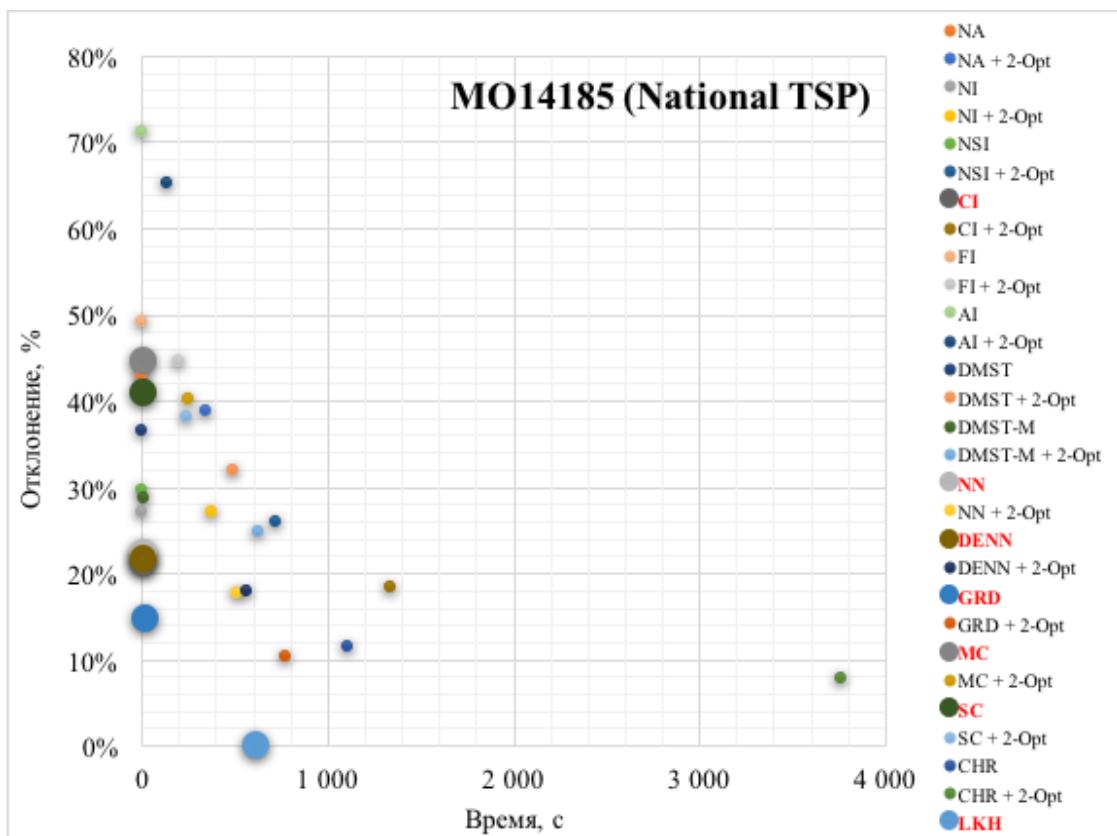


Рисунок 3.13. Парето-оптимальные алгоритмы для набора данных MO14185

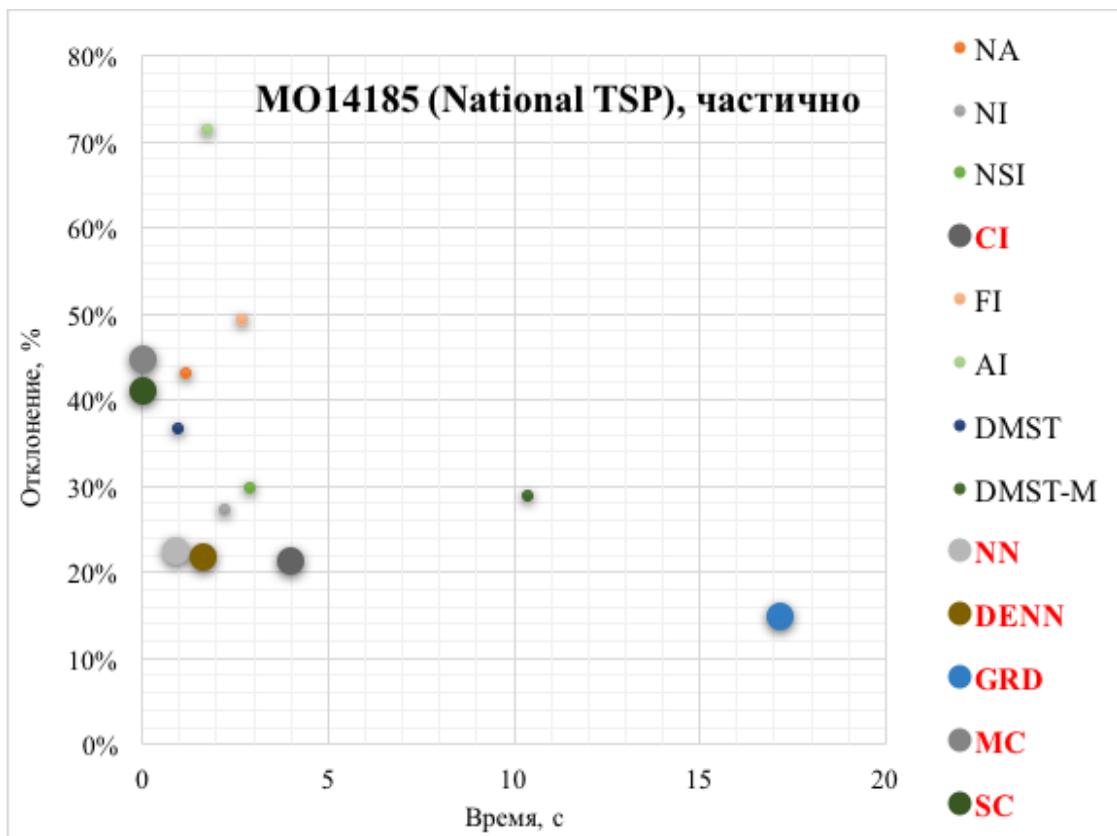


Рисунок 3.14. Парето-оптимальные алгоритмы для набора данных MO14185 (частично)

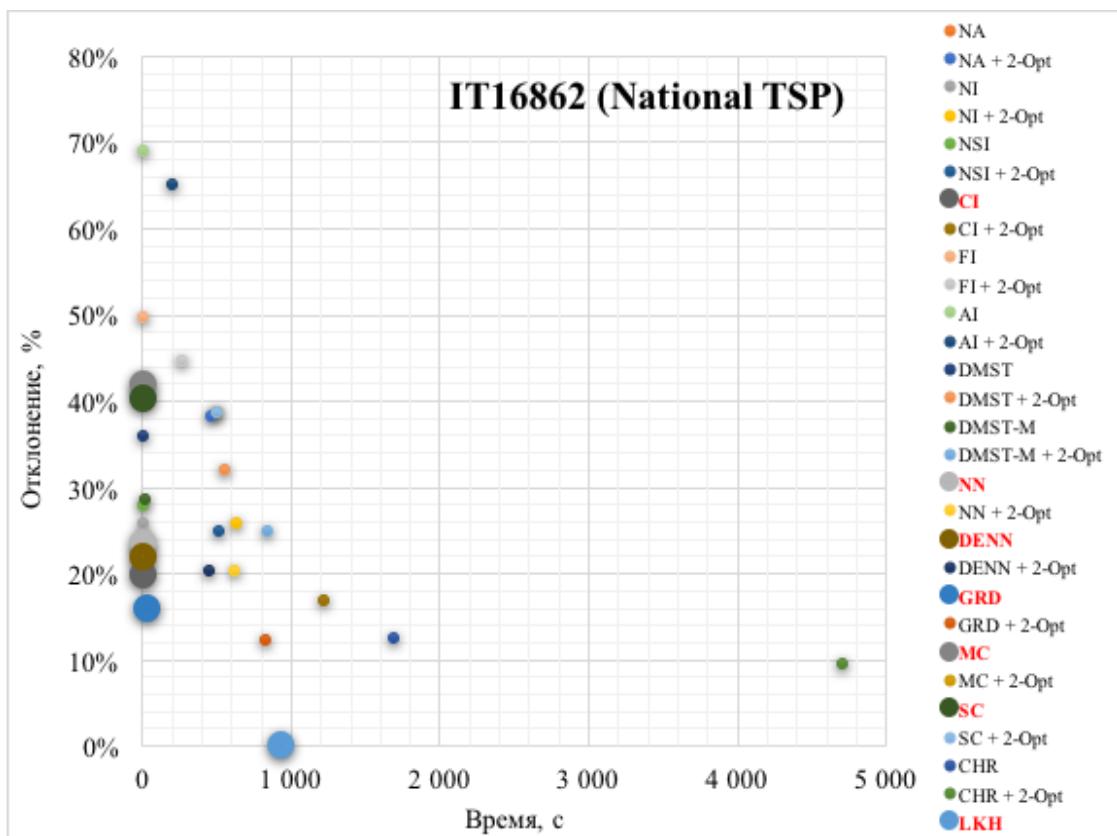


Рисунок 3.15. Парето-оптимальные алгоритмы для набора данных IT16862

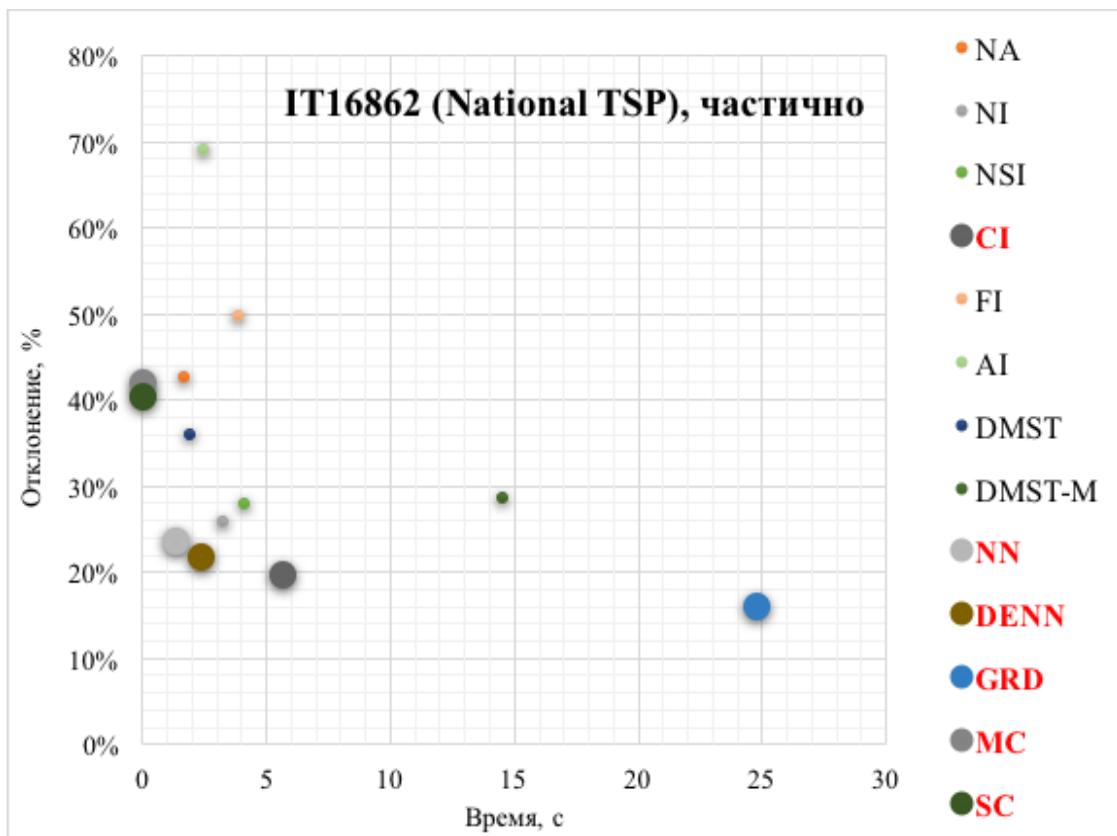


Рисунок 3.16. Парето-оптимальные алгоритмы для набора данных IT16862 (частично)

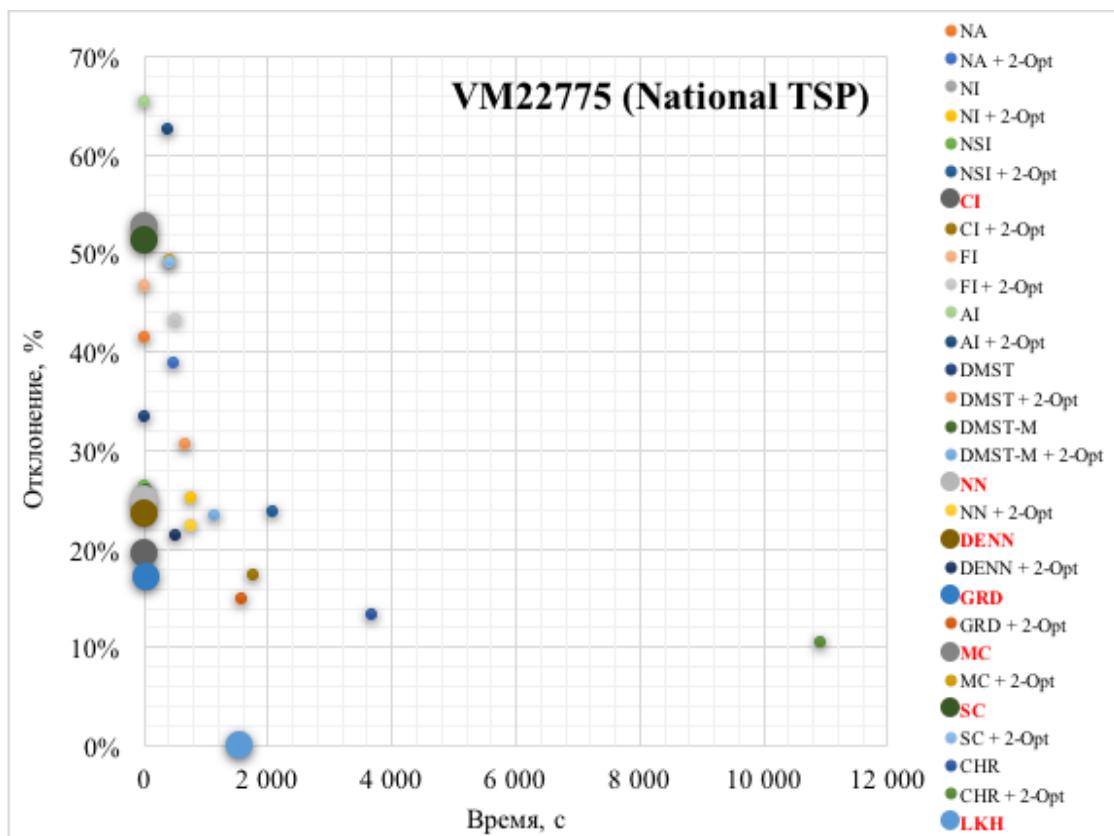


Рисунок 3.17. Парето-оптимальные алгоритмы для набора данных VM22775

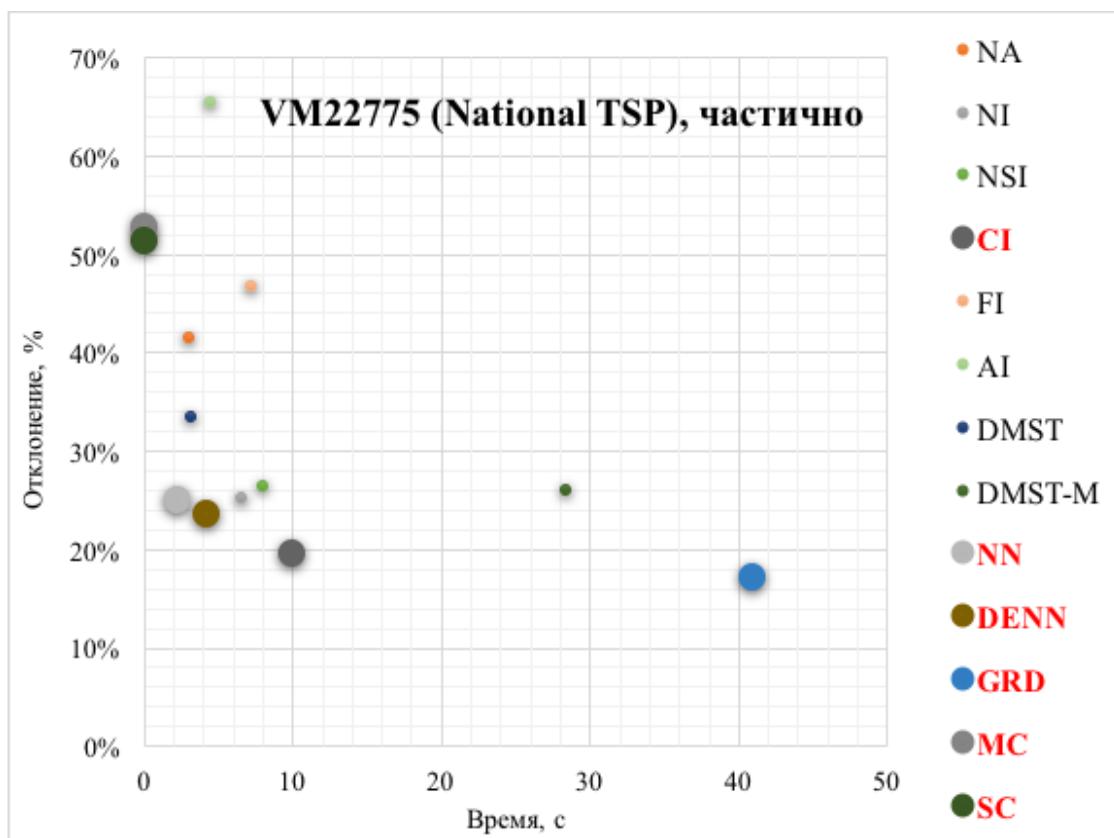


Рисунок 3.18. Парето-оптимальные алгоритмы для набора данных VM22775 (частично)

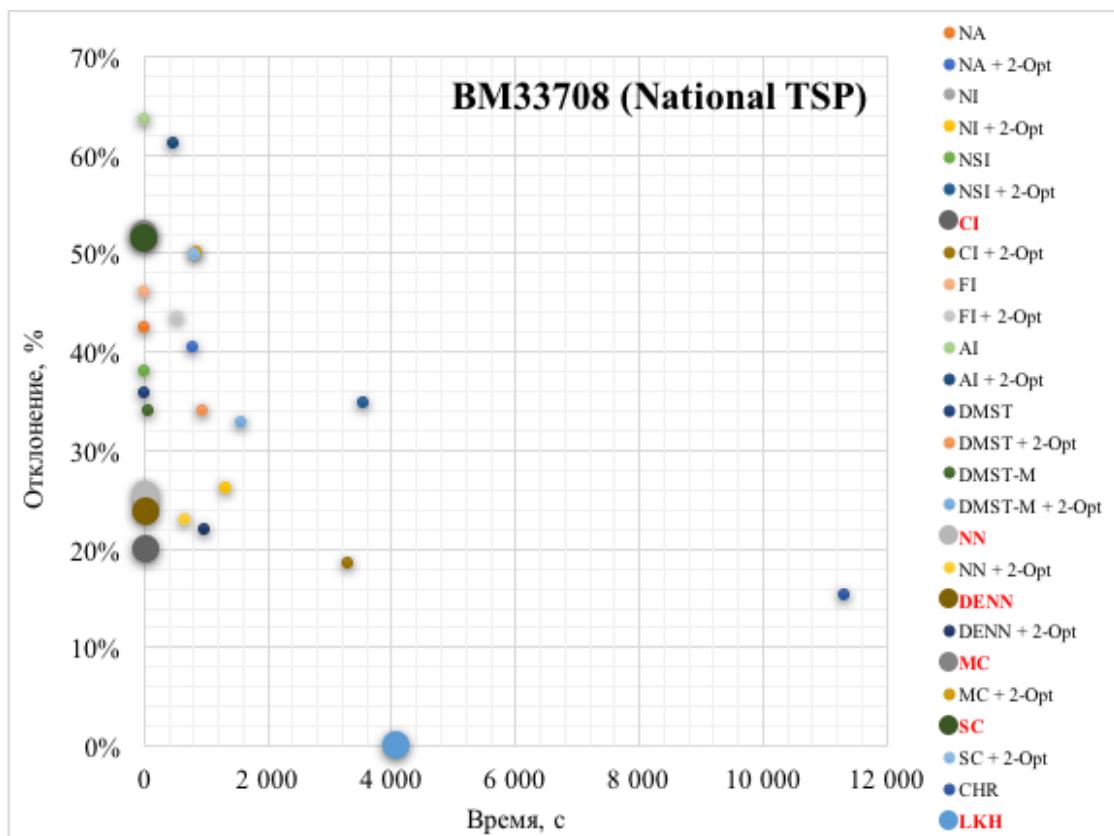


Рисунок 3.19. Парето-оптимальные алгоритмы для набора данных BM33708

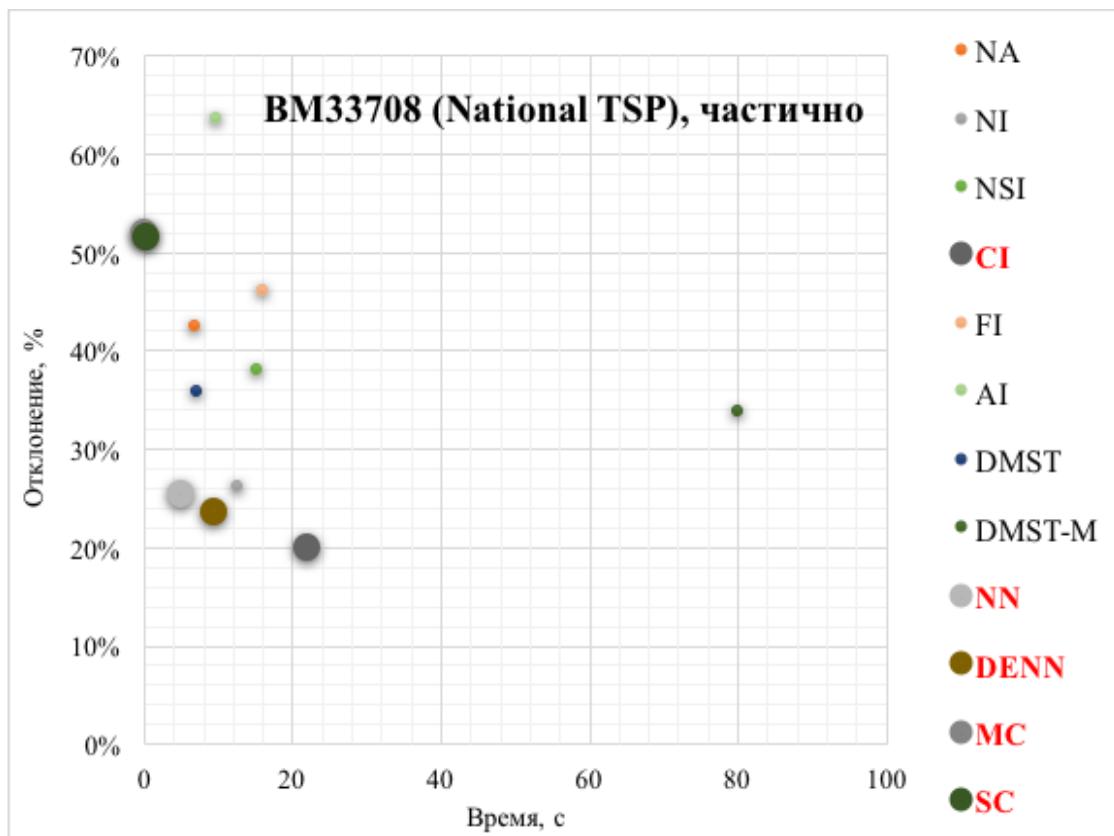


Рисунок 3.20. Парето-оптимальные алгоритмы для набора данных BM33708 (частично)

Приложение И**Сводные таблицы с результатами эксперимента (VLSI Data Sets)**

Таблица И.1.

Определение Парето-оптимальных алгоритмов для каждого набора данных VLSI Data Sets,

$$237 \leq N \leq 5087$$

Название алгоритма	Набор данных						
	XQG237	RBU737	PBD984	XIT1083	DKD1973	DHB3386	FQM5087
NA		+					
NA + 2-Opt							
NI							
NI + 2-Opt							
NSI							
NSI + 2-Opt							
CI		+		+	+	+	+
CI + 2-Opt							
FI							
FI + 2-Opt							
AI							
AI + 2-Opt							
DMST							
DMST + 2-Opt							
DMST-M							
DMST-M + 2-Opt							
NN	+	+	+	+	+	+	+
NN + 2-Opt							
DENN	+	+	+	+	+	+	+
DENN + 2-Opt							
GRD	+	+	+	+		+	
GRD + 2-Opt	+	+					
MC	+	+	+	+	+	+	+
MC + 2-Opt							
SC							
SC + 2-Opt							
CHR	+	+	+	+	+		
CHR + 2-Opt							
LKH	+	+	+	+	+	+	+

Таблица И.2.

Определение Парето-оптимальных алгоритмов для каждого набора данных VLSI Data Sets,

$$7168 \leq N \leq 33708$$

Название алгоритма	Набор данных					
	BND7168	XMC10150	XRB14233	XIA16928	LSB22777	BM33708
NA						
NA + 2-Opt						
NI						
NI + 2-Opt						
NSI						
NSI + 2-Opt						
CI	+	+	+	+	+	+
CI + 2-Opt						
FI						
FI + 2-Opt						
AI						
AI + 2-Opt						
DMST						
DMST + 2-Opt						
DMST-M						
DMST-M + 2-Opt						
NN	+	+	+	+	+	+
NN + 2-Opt						
DENN	+	+	+	+	+	+
DENN + 2-Opt						
GRD	+	+	+	+	+	
GRD + 2-Opt						
MC	+	+	+	+	+	+
MC + 2-Opt						
SC	+			+		
SC + 2-Opt						
CHR						
CHR + 2-Opt						
LKH	+	+	+	+	+	+

Таблица И.3.

Определение Парето-оптимальных алгоритмов для каждого набора данных VLSI Data Sets,

$$43748 \leq N \leq 744710$$

Название алгоритма	Набор данных					
	RBZ43748	FNA52057	SRA104815	ARA238025	LRA498378	LRB744710
NA						
NA + 2-Opt						
NI						
NI + 2-Opt						
NSI						
NSI + 2-Opt						
CI	+	+	+	+	+	
CI + 2-Opt	+	+	+			
FI						
FI + 2-Opt						
AI						
AI + 2-Opt						
DMST						
DMST + 2-Opt						
DMST-M						
DMST-M + 2-Opt						
NN	+	+	+	+	+	+
NN + 2-Opt						
DENN	+	+	+	+	+	+
DENN + 2-Opt			+			
GRD						
GRD + 2-Opt						
MC	+	+	+	+	+	+
MC + 2-Opt						
SC	+	+	+	+	+	+
SC + 2-Opt						
CHR						
CHR + 2-Opt						
LKH	+	+	+			

Приложение К**Сводные таблицы с результатами эксперимента (National TSPs)**

Таблица К.1.

Определение Парето-оптимальных алгоритмов для каждого набора данных National TSPs,

$$734 \leq N \leq 7146$$

Название алгоритма	Набор данных				
	UY734	LU980	MU1979	NU3496	EG7146
NA					
NA + 2-Opt					
NI			+		
NI + 2-Opt					
NSI					
NSI + 2-Opt					
CI	+		+		+
CI + 2-Opt					
FI					
FI + 2-Opt					
AI					
AI + 2-Opt					
DMST					
DMST + 2-Opt					
DMST-M					
DMST-M + 2-Opt					
NN	+	+	+	+	+
NN + 2-Opt					
DENN	+	+	+	+	+
DENN + 2-Opt					
GRD	+	+	+	+	+
GRD + 2-Opt		+			
MC	+	+	+	+	+
MC + 2-Opt					
SC					+
SC + 2-Opt					
CHR	+	+	+		
CHR + 2-Opt	+				
LKH	+	+	+	+	+

Таблица К.2.

Определение Парето-оптимальных алгоритмов для каждого набора данных National TSPs,

$$9976 \leq N \leq 33708$$

Название алгоритма	Набор данных				
	KZ9976	MO14185	IT16862	VM22775	BM33708
NA					
NA + 2-Opt					
NI					
NI + 2-Opt					
NSI					
NSI + 2-Opt					
CI	+	+	+	+	+
CI + 2-Opt					
FI					
FI + 2-Opt					
AI					
AI + 2-Opt					
DMST					
DMST + 2-Opt					
DMST-M					
DMST-M + 2-Opt					
NN	+	+	+	+	+
NN + 2-Opt					
DENN	+	+	+	+	+
DENN + 2-Opt					
GRD	+	+	+	+	
GRD + 2-Opt					
MC	+	+	+	+	+
MC + 2-Opt					
SC	+	+	+	+	+
SC + 2-Opt					
CHR					
CHR + 2-Opt					
LKH	+	+	+	+	+

Приложение Л

Диплом 2 степени за лучшую работу на конференции Е.В. Арменского



Рисунок Л.1. Диплом 2 степени за лучшую работу на конференции Е.В. Арменского.