УДК 519.17

СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ КРАСКАЛА И ПРИМА ДЛЯ ГРАФОВ С РАЗЛИЧНЫМ КОЛИЧЕСТВОМ ВЕРШИН

Сапожников Н.А.

студент,

Российский университет транспорта (МИИТ),

Москва, Россия

Иванова А.П.

к.ф.-м.н., доцент

Российский технологический университет – МИРЭА,

Российский университет транспорта (МИИТ),

Москва, Россия

Аннотация:

В статье проведено исследование алгоритмов Краскала и Прима построения минимального остовного дерева графов разной размерности и разной плотности (под плотностью графа понимается число, равное отношению числа рёбер графа к максимально возможному числу рёбер в графе). Была написана программа на языке Python. В результате получено, что эффективность алгоритма Прима растёт с увеличением числа рёбер исходного графа. Приведены графики зависимости времени работы обоих алгоритмов от количества вершин и график разницы времени работы алгоритмов в зависимости от плотности графов.

Ключевые слова: минимальное остовное дерево, алгоритм Краскала, алгоритм Прима.

COMPARATIVE ANALYSIS OF KRASKAL AND PRIM ALGORITHMS FOR GRAPHS WITH DIFFERENT NUMBER OF VERTICES

Sapozhnikov N.A.

student,

Russian University of Transport (MIIT),

Moscow, Russia

Ivanova A.P.

Ph.D., Associate Professor

MIREA – Russian Technological University,

Russian University of Transport (MIIT),

Moscow, Russia

Abstract: The article investigates the algorithms of Kruskal and Prim for constructing a shortest spanning tree of graphs of different dimensions and densities (graph density is understood as a number equal to the ratio of the number of edges of the graph to the maximum possible number of edges in the graph). A program was written in Python. As a result, it is found that the efficiency of the Prim algorithm increases with an increase in the number of edges of the original graph. Graphs of the dependence of the operating time of both algorithms on the number of vertices and a graph of the difference in the operating time of the algorithms depending on the density of graphs are presented.

Keywords: shortest spanning tree (SST), Kruskal's algorithm, Prim's algorithm.

Графы широко используются для решения задач оптимизации в различных областях, таких как логистика, инженерия, биоинформатика и информационные технологии [1-4]. Одной из важных задач является поиск минимального остовного дерева. Решение этой задачи особенно важно при проектировании сетей, минимизации затрат и моделировании сложных систем.

Цель работы: проведение сравнительного анализа алгоритма Краскала и алгоритма Прима для поиска минимального (кратчайшего) остовного дерева. Эти алгоритмы основаны на различных подходах и имеют свои особенности, влияющие на производительность в зависимости от структуры графа [5-7].

Пусть задан граф $G = (V, E)$ где V – множество вершин, E – множество рёбер.

Приведём описание алгоритма Краскала в общем случае [1,3,4].

На первом шаге считаем, что множество рёбер в минимальном остове пусто. Затем на каждой итерации выбираем ребро минимального веса. Если при добавлении этого ребра не образуется цикл, то мы его включаем в граф, иначе ищем следующее по весу ребро. Когда число рёбер, включенных в остов, будет равно $|E| = |V| - 1$, алгоритм завершается, а полученный граф будет являться минимальным остовным деревом исходного графа. Сложность алгоритма Краскала равна $O(|E|ln|E|)$.

Дадим описание алгоритма Прима [1,3,4] в общем случае. Рассмотрим взвешенный связный неориентированный n-вершинный граф $G = (V, E)$. Обозначим $l(v_i, v_j)$ вес ребра (v_i, v_j).

В алгоритме применяется разметка вершин. Метка вершины v_j имеет вид: $L(v_j) = [\alpha_j, \beta_j]$, здесь α_j – метка предшествования: вершина, предшествующая вершине v_j и определяющая ребро (α_j, v_j), а β_j – метка, соответствующая весу этого ребра.

Шаг 1. Пусть $T = \{v_s\}$, где v_s – произвольно выбранная вершина графа G, и $S = \varnothing$ (Т – множество вершин, входящих в кратчайший остов, S – множество рёбер, входящих в кратчайший остов).

Шаг 2. Для каждой вершины $v_j \notin T$ найти вершину $\alpha_j \in T$ такую, что $l(\alpha_j, v_j) = min\, l(v_i, v_j) = \beta_j$ $(v_i \in T)$, и приписать вершине v_j метку $[\alpha_j, \beta_j]$. Если такой вершины нет (т. е. $\Gamma(v_j) \cap T = \varnothing$) , то приписать метку $[0, \infty]$.

Шаг 3. Выбрать вершину v_k: $\beta_k = min[\beta_j]$ $(v_j \notin T)$. Обновить данные: $T = T \cup \{v_k\}, S = S \cup (\alpha_k, v_k)$. Если $|T| = n$, процесс окончен. В противном случае переход к шагу 4.

Шаг 4. Для всех вершин $v_j \notin T : v_j \in \Gamma(v_k)$ обновить метки следующим образом: Если $\beta_j > l(v_k, v_j)$, то $\beta_j = l(v_k, v_j)$, $\alpha_j = v_k$, переход к шагу 3. Если $\beta_j \leq l(v_k, v_j)$, то переход к шагу 3.

Получившийся граф есть кратчайший остов графа G.

Сложность алгоритма Прима равна $O(|E|ln|V|)$.

Для проведения исследования в рамках данной работы было принято решение использовать случайно сгенерированные графы. Такой подход позволяет объективно оценить эффективность алгоритмов Краскала и Прима на графах различной структуры, не ограничиваясь заранее заданными параметрами.

Однако, чтобы алгоритмы корректно функционировали и могли строить минимальное остовное дерево, необходимо, чтобы исходный граф был связным. В несвязном графе существуют изолированные компоненты, между которыми отсутствуют рёбра, а значит, невозможно объединить все вершины в единое дерево. Алгоритмы Краскала и Прима в таких случаях не смогут обработать весь граф, так как их работа предполагает наличие путей между всеми вершинами для последовательного включения рёбер в кратчайшее остовное дерево.

Для построения случайного связанного графа был разработан алгоритм, представленный в виде функции, генерирующей псевдослучайную матрицу смежности, удовлетворяющую условию связности графа, элементы которой имеют равномерное распределение в пределах от 1 до установленного значения максимального веса рёбер. Код представлен на рисунке 1.

```python
def generate_matrix(size, density=0.6, max_weight=30):

    matrix = np.zeros((size, size), dtype=int)
    nodes = list(range(size))
    random.shuffle(nodes)

    for i in range(size - 1):
        u, v = nodes[i], nodes[i + 1]
        matrix[u][v] = matrix[v][u] = random.randint(1, max_weight)

    required_edges = size * (size - 1) * density // 2
    current_edges = size - 1
    while current_edges < required_edges:
        u, v = random.sample(range(size), 2)
        if matrix[u][v] == 0:
            matrix[u][v] = matrix[v][u] = random.randint(1, max_weight)
            current_edges += 1

    return matrix
```

Рис. 1 – Функция генерации матрицы смежности графа. Авторская разработка

Для корректной работы программы были использованы библиотеки numpy и random языка программирования Python [9,10]. Функция реализует два этапа: построение остовного графа для обеспечения связности и добавление дополнительных рёбер для достижения заданной плотности.

Плотностью графа назовём число, равное отношению числа рёбер графа $|E|$ к максимально возможному числу рёбер в графе $\frac{|V|(|V|-1)}{2}$. Для ориентированного графа плотность вычисляется как $\frac{2|E|}{|V|(|V|-1)}$.

На первом этапе генерируется минимально связный граф, который гарантирует, что каждая вершина соединена хотя бы с одной другой вершиной. Для этого создаётся список всех вершин графа, случайным образом перемешивается их порядок, а затем последовательно добавляются рёбра между соседними вершинами в этом списке. Вес каждого ребра задаётся случайным образом в пределах от 1 до установленного предельного значения веса (в данной работе было взято значение, равное 30): генерируется псевдослучайная равномерно распределённая величина на заданном отрезке [1,30].

На втором этапе алгоритм добавляет дополнительные рёбра для достижения требуемой плотности графа.

Целевое количество рёбер вычисляется как произведение максимально возможного числа рёбер данного графа $\frac{|V|(|V|-1)}{2}$ на заданный коэффициент плотности графа. Далее в цикле добавляются дополнительные рёбра: случайным образом выбираются две вершины, и если между ними ещё нет ребра, то присоединяется новое ребро со случайным весом в пределах от 1 до заданного значения максимального веса. Этот процесс повторяется до тех пор, пока общее количество рёбер графа не достигнет требуемого значения.

В качестве примера приведём сгенерированные графы с 7 вершинами и разной плотностью. На рисунке 2 изображён граф с плотностью 0.1, на рисунке 3 – с плотностью 0.5, на рисунке 4 – с плотностью 0.9.

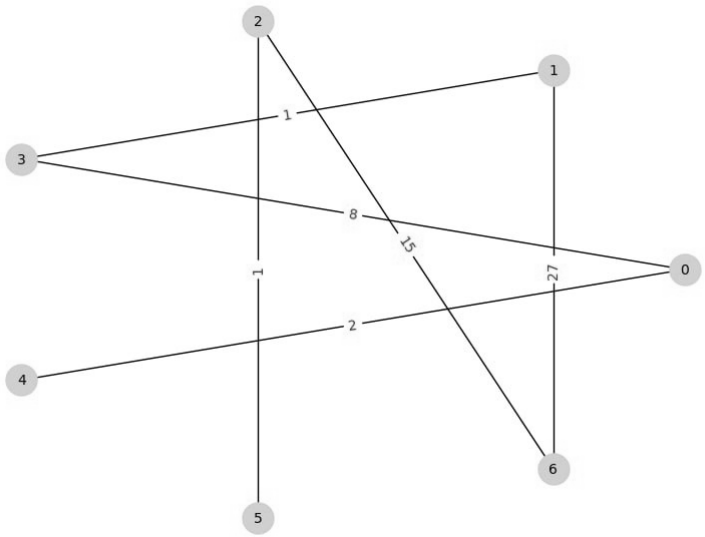

Рис. 2 – Пример графа для плотности 0,1. Авторская разработка

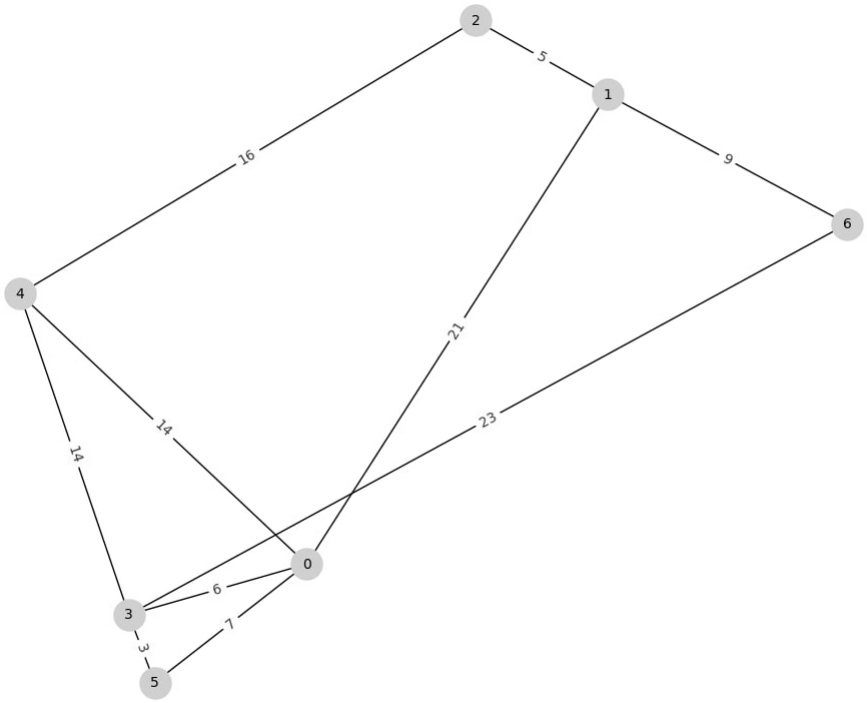

Рис. 3 – Пример графа для плотности 0,5. Авторская разработка

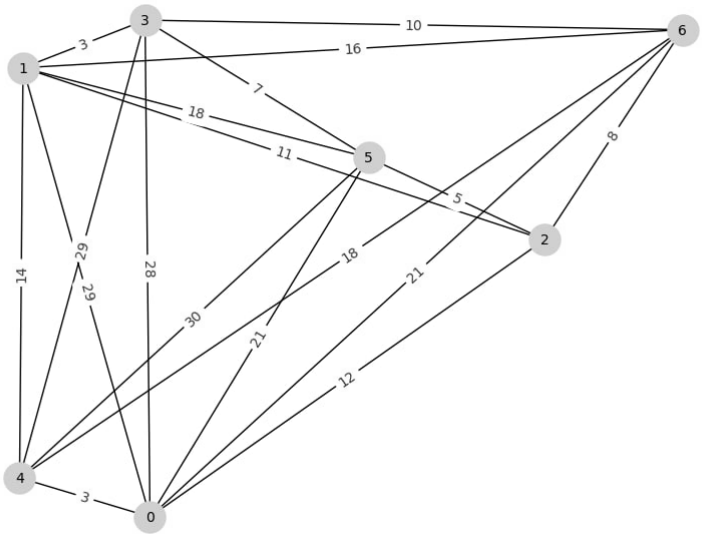

Рис. 4 – Пример графа для плотности 0,9. Авторская разработка

Проверим теоретическую сложность исследуемых алгоритмов. Алгоритм Краскала имеет временную сложность $O(|E|ln|E|)$, а алгоритм Прима – сложность $O(|E|ln|V|)$. Можно предположить, что производительность данных алгоритмов должна быть практически одинаковой для разреженных графов,

когда число рёбер почти равно числу вершин, то есть при маленькой заданной плотности. И наоборот, для плотных графов, когда количество рёбер значительно превышает число вершин, алгоритм Прима должен работать быстрее алгоритма Краскала.

Для визуализации графиков создадим список, в котором будут храниться результаты работы программы в зависимости от заданной плотности и количества вершин случайно сгенерированного связного графа. На этапе инициализации задается фиксированное значение плотности графа. Проведём измерение времени для диапазона количества вершин случайного графа от 10 до 300 вершин с шагом 10. Для сглаживания выбросов процедура для заданного количества вершин выполняется 10 раз, в ходе чего высчитывается среднее время работы алгоритма для заданных числа вершин и плотности. Полученное значение сохраняется в результирующий список, на основе которого в итоге строятся целевые графики.

Исследуем эффективность алгоритмов Краскала и Прима на разреженных графах с малым коэффициентом плотности, заданным значением 0.01. На рисунке 5 показаны графики зависимость времени работы алгоритмов от количества вершин графа при плотности 0.01.

Можем заметить, что время работы исследуемых алгоритмов, как и было предположено, практически одинаково. Разница между ними составляет всего лишь 1-2 миллисекунды, что делает данные алгоритмы равносильными на графах малой плотности.

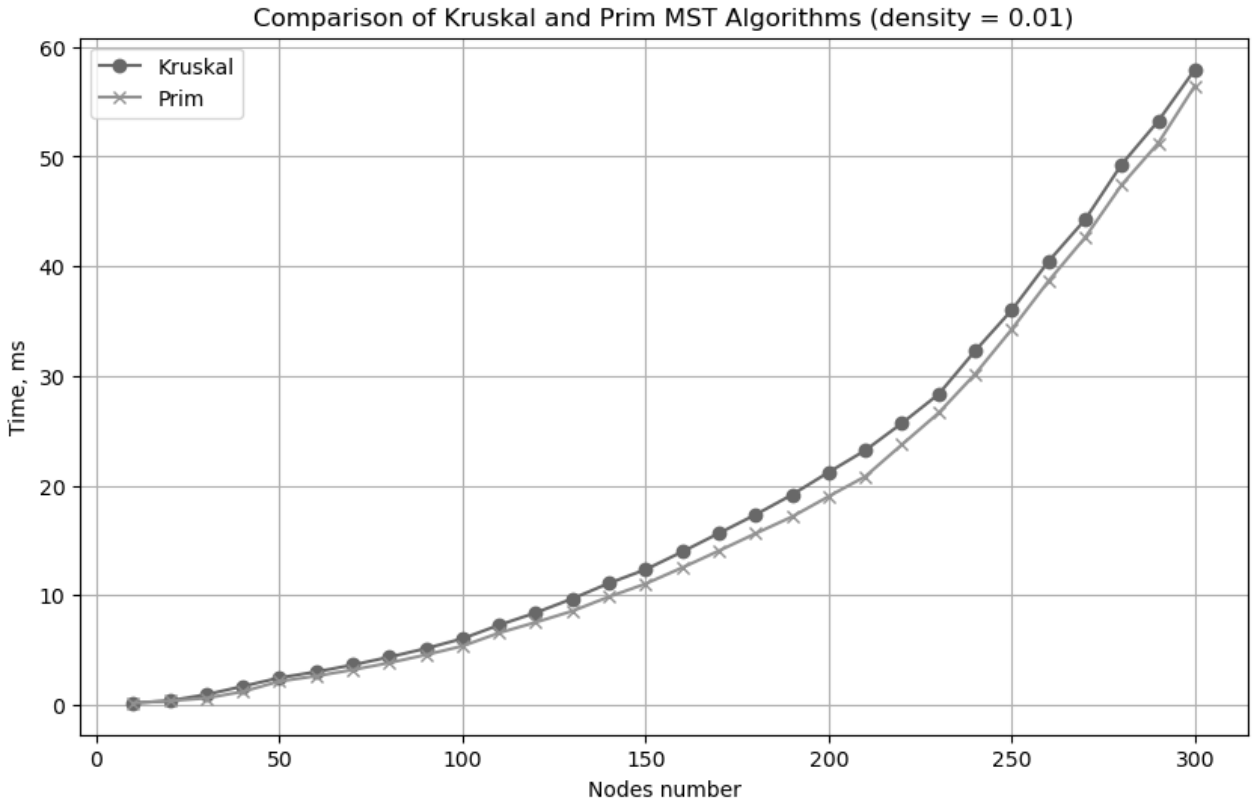

Рис. 5 – Зависимость времени работы алгоритмов от количества вершин графа при плотности 0,01. Авторская разработка

Теперь исследуем временную производительность алгоритмов Краскала и Прима на плотных графах с заданным значением коэффициента плотности 0.95. На рисунке 6 показаны графики зависимость времени работы алгоритмов от количества вершин графа при плотности 0.95.

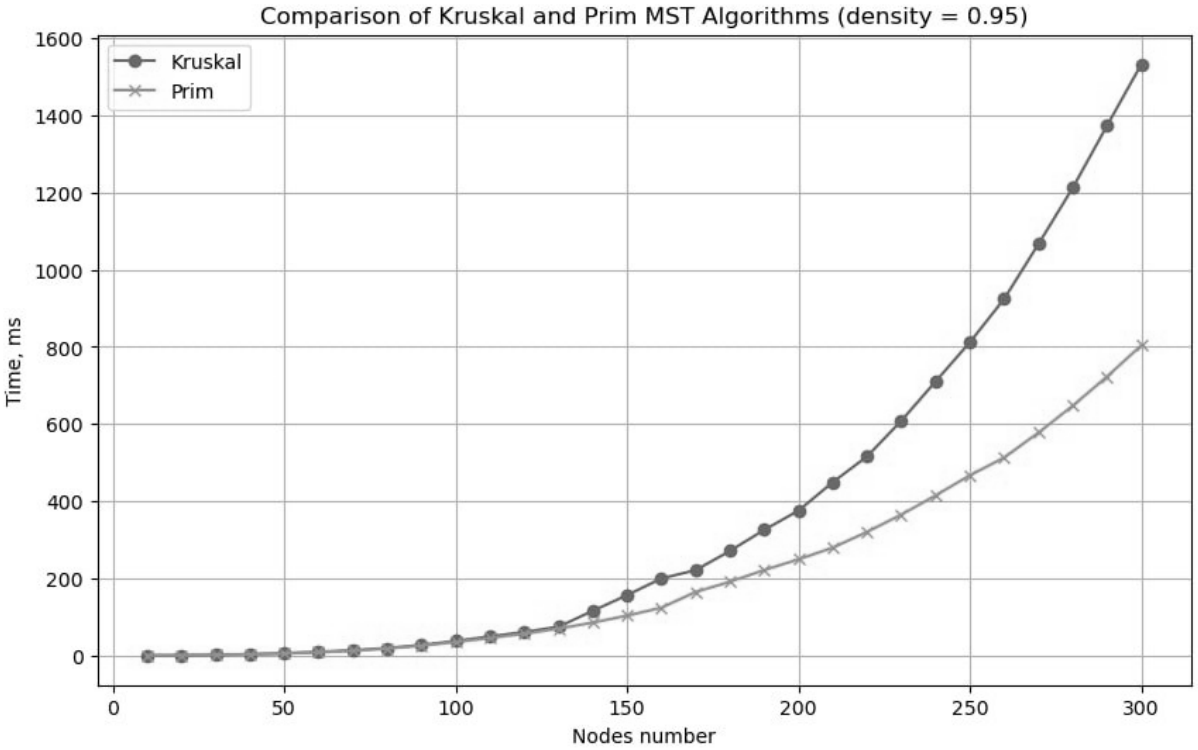

Рис. 6 – Зависимость времени работы алгоритмов от количества вершин графа при плотности 0,95. Авторская разработка

Как и было предположено, алгоритм Прима работает быстрее алгоритма Краскала: разница между алгоритмами составляет около 900 миллисекунд.

Теперь измерим зависимость временной разницы алгоритмов Краскала и Прима на графах размером в 300 вершин от коэффициента плотности случайного графа в диапазоне от 0.05 до 1 с шагом 0.05 (для получения каждой точки будем решать 20 задач и вычислять среднее время работы алгоритмов). На рисунке 7 показан график зависимости разницы времени работы алгоритмов Краскала и Прима от плотности для графов в 300 вершин. Для плотности 0.95 среднее время работы алгоритма Краскала равно 1531.8 мс, алгоритма Прима – 803.4 мс, что в 1,9 раза меньше.

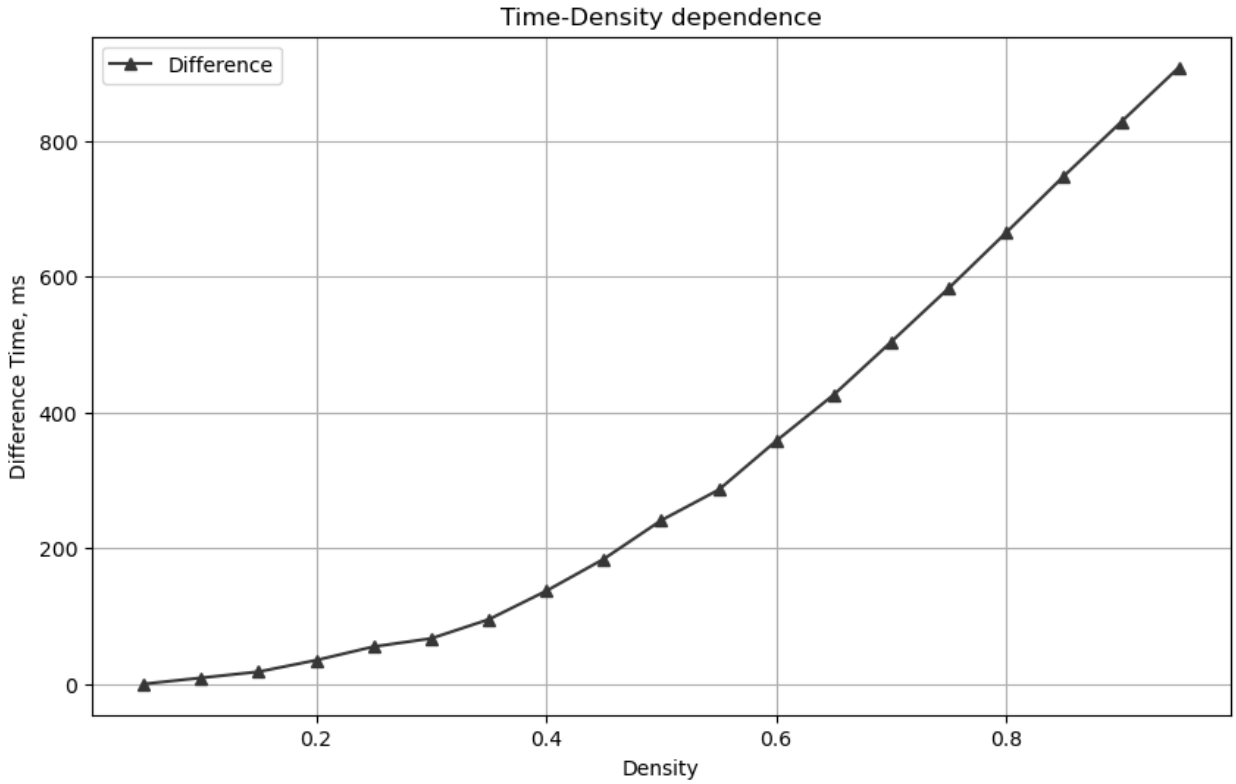

Рис. 7 – Зависимость разницы времени работы алгоритмов Краскала и Прима от плотности для графов в 300 вершин. Авторская разработка

Полученные результаты позволяют сделать вывод, что существует явная зависимость в разнице во времени работы алгоритма Прима и алгоритма Краскала от плотности графа. При этом алгоритм Прима становится эффективнее с увеличением числа рёбер исходного графа: если граф почти полный (плотность 0.95), разница во времени около 2 раз.

Библиографический список:

1. Кристофидес Н. Теория графов. Алгоритмический подход. – М.: Мир, 1978. – 432 с.

2. Лекции по теории графов / Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И. – М.: Наука, 1990. – 384 с.

3. Алексеев В.Е, Захарова Д.В. Теория графов: Учебное пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 119 с.

4. Теория графов. Часть 1: учебное пособие по дисциплине «Теория графов»: Иванова А.П. – Москва: Янус-К, 2024. – 96 с.

5. Сигал, И. Х. Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы: учебное пособие / И. Х. Сигал, А. П. Иванова. – 2-е изд., испр. и доп. – М.: ФИЗМАТЛИТ, 2007. – 304 с.

6. Дискретная математика и математическая логика: учебник / И.Н. Белоусов, В.И. Белоусова; Министерство науки и высшего образования РФ, Уральский федеральный университет. – Екатеринбург: Изд-во Урал. ун-та, 2024. – 178 с.

7. Методические указания и задания к лабораторным работам по курсу «Компьютерная дискретная математика» (для студентов, обучающихся по направлениям подготовки «Программная инженерия») / Сост.: И.А. Назарова, Л.В. Незамова – Донецк: ДонНТУ, 2016. – 100 с.

8. Батчаева, З.Б. Нахождение остовного дерева минимального веса с применением алгоритма Краскала и алгоритма Прима / З.Б. Батчаева, А.Б. Биджиева, Р.А. Турклиев // Мировая наука. – 2018. – № 10(19). – С. 81-85.

9. Озерова, Г.П. Основы программирования на языке Python в примерах и задачах: учебное пособие для вузов / Политехнический институт ДВФУ. – Владивосток: Изд-во Дальневост. федерал. ун-та, 2022. – 128 с.

10. Документация по Matplotlib [Электронный ресурс] – Режим доступа – URL: https://matplotlib.org/stable/index.html.

Оригинальность 84%