МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)
01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)
Системное программирование и компьютерные технологии
(направленность (профиль)/специализация

БАКАЛАВРСКАЯ РАБОТА

на тему **Сравнительный анализ алгоритмов поиска максимального потока**

Студент	А.С. Захаров	_____
	(И.О. Фамилия)	(личная подпись)
Руководитель	Т.Г. Султанов	_____
	(И.О. Фамилия)	(личная подпись)
Консультанты	Н.В. Андрюхина	_____
	(И.О. Фамилия)	(личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«____» _____ 2019 г.

Тольятти 2019

АННОТАЦИЯ

Тема бакалаврской работы: «Сравнительный анализ алгоритмов поиска максимального потока». Актуальность работы заключается в том, что при решении различных практических задач возникает необходимость переслать что-либо из одной точки в другую. Для решения данной проблемы может использоваться модель задачи о максимальном потоке. Поэтому выбор правильного алгоритма поиска максимального потока имеет большое значение, так как это чревато неэффективным или неточным решением исходной проблемы. Также, к задаче поиска максимального потока сводятся и другие важные оптимизационные практические задачи.

Объект исследования: процесс поиска максимального потока.

Предмет исследования: алгоритмы Форда-Фалкерсона, Эдмондса-Карпа, Диница поиска максимального потока.

Цель: выявление преимуществ и недостатков алгоритмов поиска максимального потока для их правильного использования на практике.

Первая глава работы посвящена описанию задачи о максимальном потоке, ее математической модели и применениях.

Во второй главе описываются алгоритмы Форда-Фалкерсона, Эдмондса-Карпа, Диница, а также приведены доказательства корректности и сложности данных алгоритмов.

Третья глава посвящена тестированию алгоритмов и их сравнительному анализу.

Бакалаврская работа выполнена на 52 страницах, состоит из введения, трёх глав, заключения, списка литературы, состоящего из 22 литературных источников, 31 рисунка и 2 таблиц.

ABSTRACT

The title of graduation work is “The comparative analysis of the maximum flow algorithms”.

Network Flow Problems have always been among the best studied combinatorial optimization problems. These problems are central problems in operations research, computer science, and engineering and they arise in many real world applications. Flow networks are very useful to model real world problems like, current flowing through electrical networks, commodity flowing through pipes and so on. Therefore, the choice of the correct algorithm for finding the maximum flow is of great importance, since this is fraught with an ineffective or inaccurate solution to the original problem.

The object of this work is a process of finding the maximum flow.

The subject of this work: Ford-Fulkerson’s, Edmonds-Karp’s and Dinic’s algorithms for finding the maximum flow.

The aim of the graduation work is to find the advantages and disadvantages of the maximum flow algorithms for their proper use in practice.

The work includes three chapters.

The first chapter of the work is devoted to the description of the maximum flow problem, its mathematical model and applications.

The second chapter is about Ford-Fulkerson’s, Edmonds-Karp’s, Dinic’s algorithms. There are also provided the proofs of correctness and complexity of these algorithms.

The third chapter is devoted to testing algorithms and their comparative analysis.

The graduation thesis is on 52 pages, consists of an introduction, three chapters, conclusion, 22 references, 31 figures and 2 tables.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Задача максимального потока была решена еще в середине прошлого века, но несмотря на это, она до сих пор не теряет своей актуальности. Как правило, поток определяет способ передачи чего-либо из одной точки в другую. К примеру, перевозка груза от завода до распределительного склада и от склада до самих магазинов; движение людей от мест проживания к местам работы; доставка писем от отправителей к получателям. Так же, как будет показано дальше, к потокам можно свести и ряд других задач, которые, с первого взгляда, с потоками ничего общего не имеют. [6]

Несмотря на большое разнообразие ситуаций, связанных с потоками, в них возникает ряд определенных проблем. Примерами таких проблем могут быть максимизация объемов перевозки товара и минимизация времени этих перевозок, максимизация прибыли и минимизация суммарных затрат при выполнении каких-либо работ. [1][3]

Задача нахождения максимального потока является одной из наиболее изучаемых задач комбинаторной оптимизации и представляет собой частный случай задачи линейного программирования. Она была впервые сформулирована в 1954 году Харрисом как упрощенная модель советской железнодорожной системы движения. [3][8] Позже, в 1955 году Форд и Фалкерсон создали первый алгоритм нахождения максимального потока. [4][5] Но, несмотря на то, что данный алгоритм был не столь эффективен, он стал трамплином к более производительным и элегантным решениям.

Как уже было сказано, задача нахождения потока имеет широкое применение в реальных задачах, а значит существует необходимость в ее эффективном решении. Отсюда возникает необходимость в алгоритмах поиска потока, имеющих наибольшую производительность. Но, несмотря на широкий ряд существующих алгоритмов, какого-то универсального, подходящего для любой задачи, скорее всего не существует. В каких-то случаях лучше использовать один алгоритм, в каких-то другой.

Данная работа посвящена как теоретическому, так и практическому сравнительному анализу алгоритмов поиска максимального потока.

Целью бакалаврской работы является выявление сильных и слабых сторон алгоритмов поиска максимального потока в некоторых ситуациях для их правильного и эффективного использования на практике.

Объект исследования – процесс поиска максимального потока.

Предметом исследования являются алгоритмы поиска максимального потока Форда-Фалкерсона, Эдмонса-Карпа, Диница.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- рассмотреть задачу о максимальном потоке и ее применения;
- рассмотреть алгоритмы решения задачи о максимальном потоке;
- провести теоретический сравнительный анализ алгоритмов;
- изучить различные модели сетей;
- разработать компьютерную программу генерирования сетей;
- провести генерирование тестовых данных;
- разработать компьютерную программу, реализующую алгоритмы поиска максимального потока;
- провести тестирование алгоритмов на сгенерированных данных;
- провести сравнительный анализ алгоритмов на основе результатов тестирования.

Бакалаврская работа состоит из введения, трех глав и заключения.

В первой главе рассказывается о задаче поиска максимального потока, ее математической модели и применениях.

Во второй главе дано описание алгоритмов поиска максимального потока. Приведены доказательства корректности и асимптотические оценки работы алгоритмов, а также проведен теоретический сравнительный анализ их работы.

Третья глава посвящена генерированию тестовых данных, тестированию алгоритмов и их сравнительному анализу на основе результатов тестирования.

ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ НА ИССЛЕДОВАНИЕ

1.1 Описание задачи о максимальном потоке

В повседневной жизни мы часто сталкиваемся с различными типами сетей: электрическими, телефонными, кабельными, автомобильными, железнодорожными, производственными, компьютерными и другими сетями. Электрические сети приносят освещение и развлечения в наши дома; телефонные сети позволяют общаться на расстоянии без каких-либо усилий; автомобильные и железнодорожные сети предоставляют нам средства для преодоления расстояния для того, чтобы увидеть наших близких, посещать новые места и наслаждаться новыми впечатлениями; производственные сети дают нам доступ к жизненно важным запасам продуктов питания и потребительским товарам; компьютерные сети позволяют нам обмениваться большим количеством информации на расстоянии, экономя наше время и деньги.

В каждой из приведенных сетей, и во многих других, мы хотим переместить некоторую «сущность» (например, электричество, продукт питания, человека, транспортное средство, сообщение) из одной точки в другую, и сделать это настолько эффективно, насколько это возможно, но не превышая ограничений сети.

Рассмотрим что из себя представляет сеть. Сеть состоит из специальных точек, называемых вершинами или узлами, и звеньев, соединяющих пары этих узлов, называемых ребрами или дугами. По ребрам сети, как уже было сказано, может перемещаться некоторая «сущность», называемая потоком. Примеры сетей приведены в таблице 1.

Также, как говорилось ранее, сети могут иметь ограничения. Они могут накладываться как на вершины, так и на ребра сети. Примерами таких ограничений могут служить: для транспортных сетей – максимальная пропускная способность автомобильной дороги, или максимальное количество поездов, которые одновременно могут остановиться на одной станции; для производственных сетей – максимальное количество товара, которое может

быть произведено за единицу времени, или максимальное количество продукции, которую может перевезти одно транспортное средство.

Таблица 1.1 – Примеры сетей

Сети	**Вершины**	**Ребра**	**Поток**
коммуникационные	телефонные станции, компьютеры, спутники	кабели	голосовые сообщения, видео, данные
транспортные	аэропорты, железнодорожные станции, перекрестки дорог	маршруты авиалиний, железнодорожные пути, автомобильные дороги	самолеты, поезда, автомобили
производственные	заводы, склады, магазины	маршруты перевозки	продукты питания, товары для дома

Остается определить понятие максимального потока. Допустим имеется некоторая сеть, в ней существуют вершины и связи между ними, также определено понятие потока в этой сети. Тогда с точки зрения какой-либо предметной области максимальным потоком может быть: максимальное количество информации, которое можно передать от одного компьютера другому; максимальная прибыль компании при реализации каких-либо проектов; необходимое количество транспортных средств для доставки товаров со складов.

Как можно видеть из приведенных примеров, задача о максимальном потоке имеет широкое применение в реальных задачах. Но больший интерес она представляет с математической точки зрения, так как существует необходимость в эффективных алгоритмах ее решения.

1.2 Математическая модель задачи о максимальном потоке

Общая абстракция, которая моделирует сеть, представляет собой ориентированный граф $G = (V, E)$, где V – множество вершин, а E – множество ребер, соединяющих эти вершины. Специальная исходная вершина s,

называемая истоком, производит поток, который поглощается вершиной t – стоком. Предполагается, что может быть произведено и поглощено бесконечное количество потока.

Каждое ребро u, v имеет поток $f_{(u.v)}$, который определяет количество единиц потока, протекающего по этому ребру, а также величину $c_{(u,v)}$ – максимальную пропускную способность ребра, то есть максимальное количество единиц потока, которое может пройти по ребру. Здесь и дальше на рисунках обозначение $f_{(u,v)}/c_{(u,v)}$ на ребрах будет показывать текущий поток и максимальный поток ребра.

Пример описанной модели изображен на рисунке 1.2.

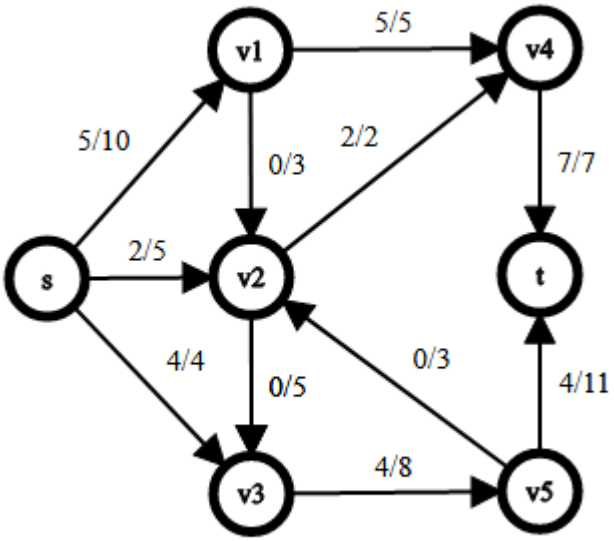

Рисунок 1.2 – Пример сети

Сформулируем понятие потока в графе. Потоком f в графе G называется функция $f : V \times V \rightarrow R$. Для любого возможного потока должны быть выполнены следующие условия.

Первым условием является свойство антисимметричности. Значение $f_{(u,v)}$ представляет собой количество единиц потока, который проходит из вершины u в вершину v, а значит должно выполняться следующее:

$$f_{(u,v)} = -f_{v,u} \quad (1)$$

Данное ограничение вводится лишь для удобства. Дело в том, что существует альтернативное определение задачи (по Асанову [2]), в котором данное свойство не вводится, но из-за этого с этим определением труднее работать.

Второе условие – ограничение пропускной способности. То есть поток через ребро $f_{(u,v)}$ не может превышать пропускную способность ребра $c_{(u,v)}$. Формально должно выполняться следующее:

$$|f_{u,v}| \leq c_{u,v} \quad (2)$$

Если ребра u, v не существует, то считают, что $c_{(u,v)} = 0$.

Последнее условие – это закон сохранения потока. Для каждой вершины, исключая исток и сток, должно выполняться следующее: сумма входящего в вершину потока равна сумме выходящего из вершины потока. То есть:

$$\sum_{v \in V} f_{(u,v)} = 0, \text{для всех } u \in V - \{s, t\} \quad (3)$$

Это свойство гарантирует, что поток не производится и не поглощается нигде, кроме как в истоке и стоке.

Тогда величиной потока f будет являться следующая величина:

$$f = \sum_{v \in V} f_{(s,v)} - \sum_{u \in V} f_{(u,s)} \quad (4)$$

Она интерпретируется как разность суммы выходящего из истока потока и суммы входящего в исток потока.

Таким образом, задача о максимальном потоке заключается в поиске потока максимальной величины.

1.3 Приложения задачи о максимальном потоке

Прежде чем приступить к анализу алгоритмов, решающих задачу о максимальном потоке, рассмотрим примеры других задач, которые можно свести к задача поиске максимального потока:

- нахождение максимального количества непересекающихся путей в ориентированном графе;

- нахождение связности графа – минимального количества ребер (вершин), которые нужно удалить, чтобы граф стал несвязным;
- нахождение максимального паросочетания в двудольных графах – максимального количества ребер, таких что любые два ребра не имеют общей вершины;
- нахождение минимального вершинного покрытия двудольного графа;
- нахождение минимального разреза графа.

Среди реальных задач, сводящихся к поиску максимального потока, можно выделить следующие:

- задача циркуляции; К примеру, существуют некоторые заводы, которые производят товары, и некоторые фабрики, куда товары должны быть доставлены. Они связаны сетью дорог, каждая из которых имеет величину равную максимальному количеству товаров, которое можно по ней провезти. Задача состоит в нахождении такой циркуляции, которая удовлетворит спрос каждой фабрики.
- авиа планирование; В авиации одной из основных проблем является составление расписания для летных экипажей. В различных вариациях данная проблема может формулироваться как составление выполнимого расписания с использованием определенного количества экипажей или нахождения минимального количества экипажей.
- сегментация изображений; К примеру, выделение в изображении переднего и заднего плана.
- задача выбора проектов; Например, имеются различные проекты, которые приносят прибыль, для выполнения которых требуются инструменты, за которые нужно заплатить. Проблема состоит в выборе таких проектов, что суммарная прибыль будет максимальной.

Как можно видеть, модель задачи поиска максимального потока используется во многих не только абстрактных, но и реальных задачах. Отсюда

появляется высокая необходимость в точных и эффективных алгоритмах ее решения.

ГЛАВА 2 АЛГОРИТМЫ ПОИСКА МАКСИМАЛЬНОГО ПОТОКА

2.1 Алгоритм Форда-Фалкерсона

Рассмотрим первый изобретенный алгоритм, специально предназначенный для решения задачи поиска максимального потока – алгоритм Форда-Фалкерсона, придуманный в 1955 году Лестером Фордом и Делбертом Фалкерсоном. [4][5] Также, вместо «алгоритм Форда-Фалкерсона» часто используют «метод Форда-Фалкерсона», так как он имеет множество реализаций с различной асимптотической сложностью.

Алгоритм Форда-Фалкерсона построен на трех важных понятиях: остаточные сети, дополняющие пути и разрезы; а также ключевую роль в доказательстве корректности алгоритма играет теорема Форда-Фалкерсона о максимальном потоке и минимальном разрезе.

Прежде чем приступить к описанию и анализу алгоритма Форда-Фалкерсона, введем определения остаточной пропускной способности, остаточной сети, дополняющего пути.

Остаточной пропускной способностью $c'_{u,v}$ ребра $(u, v) \in E$ является величина потока, который можно направить по данному ребру, не превысив его пропускной способности, то есть $c'_{(u,v)} = c_{(u,v)} - f_{(u,v)}$.

Остаточной (дополняющей) сетью называется такая сеть $G' = (V, E')$, порожденная сетью $G = V, E$ и потоком f, где $E' = u, v \in V \times V \quad c'_{u,v} > 0\}$. Пример остаточной сети приведен на рисунке 2.1.

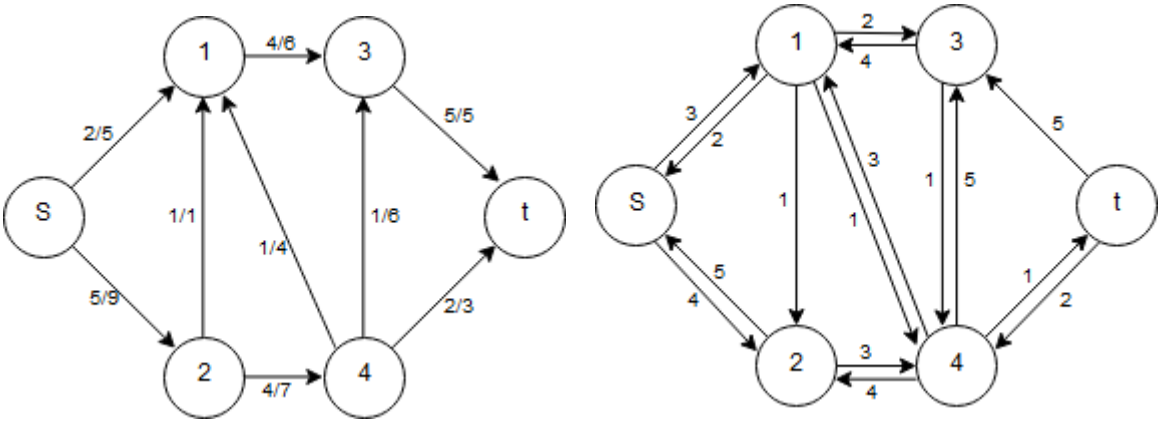

Рисунок 2.1 – Сеть с некоторым потоком (слева)
и остаточная сеть (справа)

Дополняющим путем для сети $G = V, E$ и потока f является простой путь из истока в сток в остаточной сети $G' = (V, E')$ (рисунок 2.2).

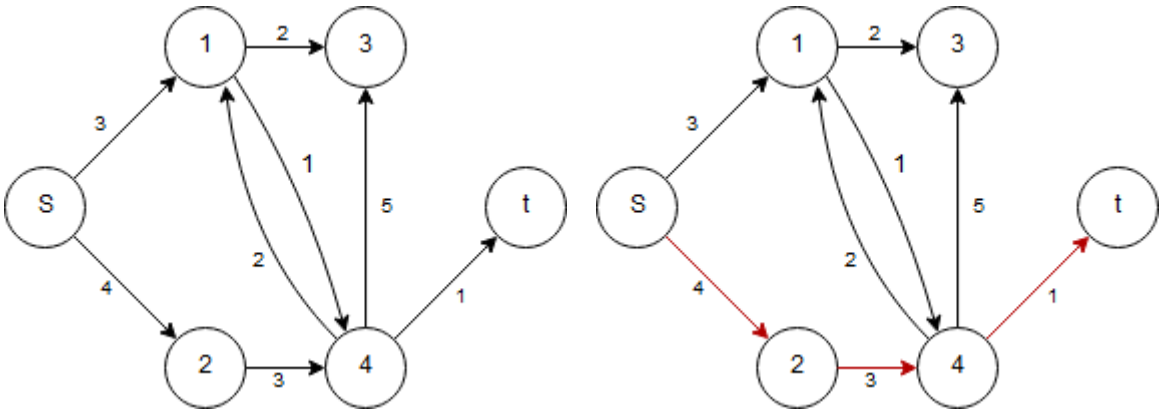

Рисунок 2.2 – Остаточная сеть (слева) и дополняющий путь в этой сети (справа)

Теперь можно описать алгоритм Форда-Фалкерсона. Алгоритм является итеративным. В начале положим для всех ребер поток равным нулю. Затем на каждой итерации будем находить дополняющий путь в остаточной сети. Если такой путь не удается найти, то работа алгоритма завершается, иначе найдем минимальное количество потока, который можно добавить вдоль найденного пути, и добавим его.

Формально алгоритм Форда-Фалкерсона можно описать следующим образом:

1. $f_{u,v} := 0$ для всех ребёр (u, v)
2. Пока есть дополняющий путь p из истока в сток в сети G':
   2.1. Найти $c_{min} = \min\ c'_{u,v} \quad (u, v) \in p\}$
   2.2. Для каждого ребра $u, v \in p$:
      2.2.1. $f_{u,v} := f_{u,v} + c_{\min}$
      2.2.2. $f_{v,u} := f_{v,u} - c_{min}$
3. Вернуть f

Результатом работы алгоритма является поток f максимальной величины. Доказательством того, что на выходе работы алгоритма получается именно максимальным поток служит теорема Форда-Фалкерсона, которая говорит, что

если в остаточной сети невозможно найти дополняющего пути, то поток в данной сети является максимальным.

Рассмотрим пример работы алгоритма Форда-Фалкерсона, изображенный на рисунке 2.3.

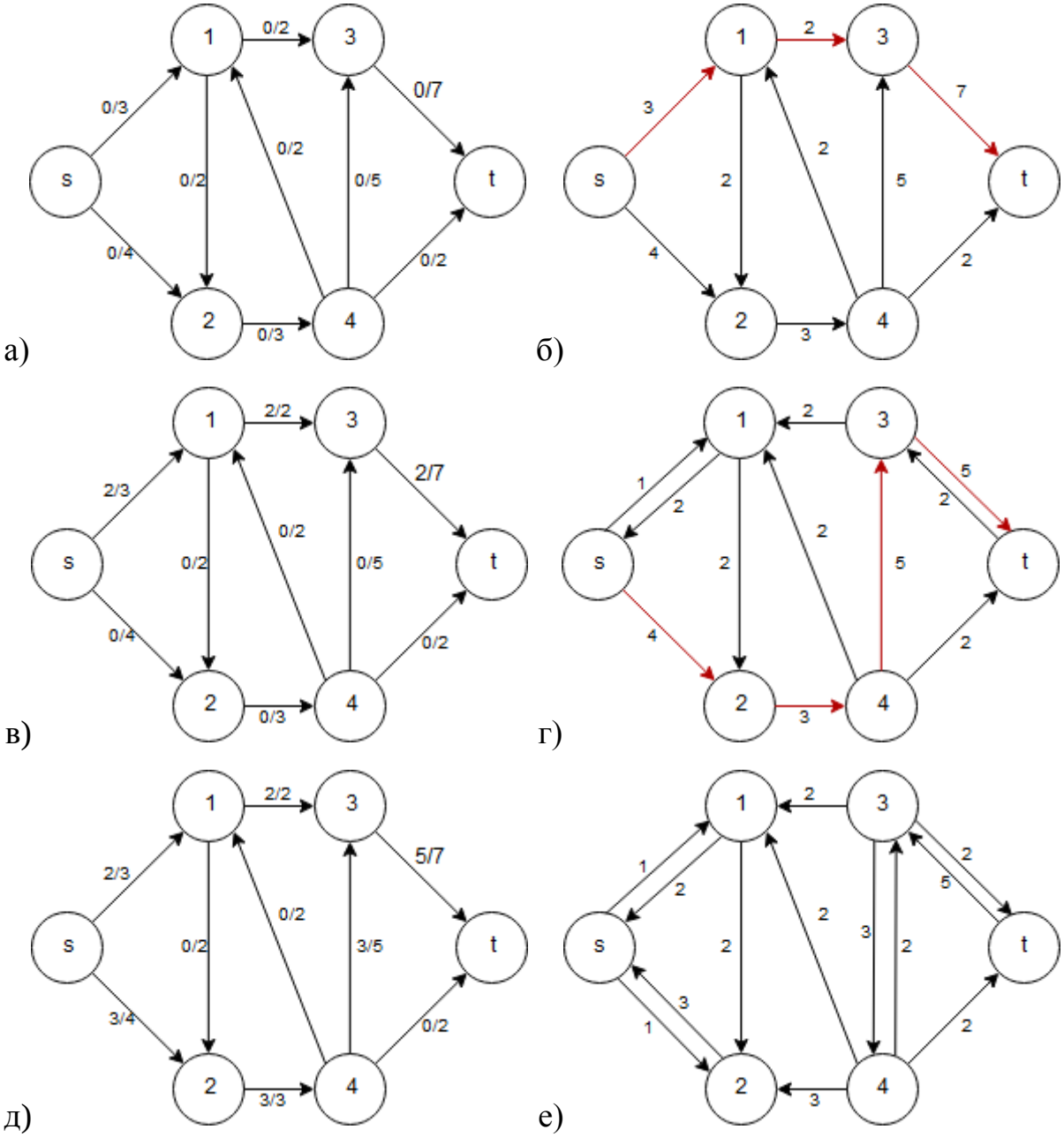

Рисунок 2.3 – Пример работы алгоритма Форда-Фалкерсона

Слева изображено текущее состояние сети в начале каждой итерации, справа изображены остаточные сети и дополняющие пути на соответствующей итерации. Изначально каждому ребру сети положен поток равный нулю (рисунок 2.3 (а)). Затем в остаточной сети находится дополняющий

путь $\{s, 1, 3, t\}$, вдоль которого можно добавить поток равный двум (рисунок 2.3 (б)). На следующей итерации алгоритм находит дополняющий путь $\{s, 2, 4, 3, t\}$ и добавляет три единицы потока вдоль этого пути (рисунок 2.3 (г)). Как можно видеть, в остаточной сети, построенной на последней итерации (рисунок 2.3 (е)) не существует дополняющего пути из истока в сток, а значит поток в данной сети является максимальным и алгоритм завершает свою работу. Величина найденного потока в данном случае равна $f_{s,1} + f_{s,2} = 2 + 3 = 5$.

Проанализируем сложность работы алгоритма Форда-Фалкерсона. Время работы всего алгоритма зависит от способа нахождения дополняющего пути. На самом деле, алгоритм может вовсе не закончить свою работу, так как значение потока будет расти все медленнее и медленнее, так и не достигнув максимума. Однако, такой проблемы не будет возникать, если все пропускные способности будут являться целыми числами (на самом деле, такой случай часто встречается на практике; а если веса все-таки вещественные, то их приводят к целым умножая все пропускные способности на какое-либо число).

Рассмотрим случай, когда величины пропускных способней являются целыми числами. Тогда время работы алгоритма Форда-Фалкерсона $O(E\ f)$, так как время поиска одного дополняющего пути $O(E)$, а всего таких поисков может быть $O(f)$, потому что величина потока на каждой итерации увеличивается как минимум на единицу.

Как можно видеть, время работы алгоритма зависит от величины ответа. Если ответ невелик, то время работы также будет невелико. Однако, если величина ответа является достаточно большой, то даже на сетях с малым количеством вершин и ребер алгоритм может работать долго (рисунок 2.4).

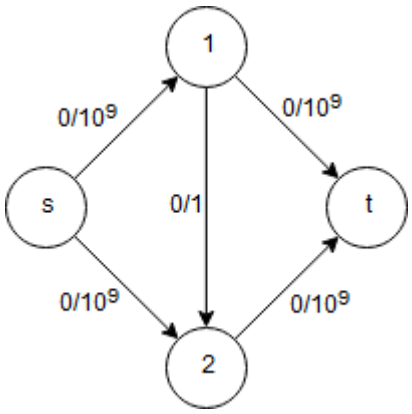

Рисунок 2.4 – Пример сети, на которой алгоритм Форда-Фалкерсона может работать долго

Так как в алгоритме Форда-Фалкерсона не дается никаких ограничений на то, какой из дополняющих путей выбирать, то в данном случае алгоритм может последовательно выбирать следующие пути: $p_1 = s, 1, 2, t$, $p_2 = \{s, 2, 1, t\}$, и добавлять по одной единице потока, а значит суммарно алгоритм выполнит 2×10^9 итераций поиска дополняющего пути.

Таким образом, главным недостатком алгоритма Форда-Фалкерсона является его зависимость от величины ответа, однако, как будет показано дальше, некоторые модернизации алгоритма Форда-Фалкерсона позволят избавиться от этой зависимости.

2.2 Алгоритм Эдмондса-Карпа

Рассмотрим следующий алгоритм, который является модернизацией алгоритма Форда-Фалкерсона – алгоритм Эдмондса-Карпа, который был разработан в 1971 году Джеком Эдмондсом и Ричардом Карпом. [1]

Как уже было сказано, недостатком алгоритма Форда-Фалкерсона является его зависимость от величины ответа, однако Эдмондсу и Карпу удалось справиться с этой проблемой, создав алгоритм, работающий за полиномиальное время.

Главное отличие алгоритма Эдмондса-Карпа от алгоритма Форда-Фалкерсона заключается в поиске дополняющих путей. Как известно в

алгоритме Форда-Фалкерсона на дополняющие пути не накладывается никаких ограничений, тогда как в алгоритме Эдмонса-Карпа обязательным является то, чтобы дополняющий путь был кратчайшим (в данном случае длина пути измеряется в количестве ребер в пути).

Тогда формально алгоритм Эдмондса-Карпа можно записать следующим образом:

1. $f_{u,v} := 0$ для всех ребёр (u, v)
2. Пока есть кратчайший дополняющий путь p из истока в сток в сети G':
   2.1. Найти $c_{min} = \min\ c'_{u,v} \quad (u, v) \in p\}$
   2.2. Для каждого ребра $u, v \in p$:
      2.2.1. $f_{u,v} := f_{u,v} + c_{\min}$
      2.2.2. $f_{v,u} := f_{v,u} - c_{min}$
3. Вернуть f

Как можно видеть, алгоритм Эдмондса-Карпа не сильно отличается от алгоритма Форда-Фалкерсона: он также находит дополняющие пути, пока они существуют и добавляет поток вдоль этих путей. Корректность работы алгоритма Эдмондса-Карпа доказывается также как и корректность алгоритма Форда-Фалкерсона.

Проанализируем временную сложность алгоритма Эдмондса-Карпа. Докажем, что асимптотика работы алгоритма равна $O(VE^2)$.

Пусть δ_u^i – кратчайшее расстояние от истока до вершины u на i итерации алгоритма. Докажем следующую лемму: $\delta_u^i \leq \delta_u^{i+1}$ для всех u. Предположим обратное, пусть существует такая вершина v, для которой кратчайшее расстояние на $i + 1$ итерации алгоритма уменьшилось, то есть $\delta_v^{i+1} < \delta_v^i$, а δ_v^{i+1} – минимально возможное. Обозначим $G_i' = (V, E_i')$ - остаточную сеть на i итерации. Рассмотрим кратчайший путь $p = s \rightsquigarrow u \to v$ на $i + 1$ итерации алгоритма. Тогда для вершины u выполняется следующее: $\delta_v^{i+1} = \delta_u^{i+1} + 1$ и $\delta_u^{i+1} \geq \delta_u^i$. Предположим ребро $u, v \in E_i'$, тогда $\delta_v^i \leq \delta_u^i + 1 \leq \delta_u^{i+1} + 1 =$

δ_v^{i+1}. Но это противоречит $\delta_v^{i+1} < \delta_v^i$. Пусть $u, v \notin E_i'$, но известно, что $u, v \in E_{i+1}'$. Появление ребра в остаточной сети на $i+1$ итерации говорит о том, что на i итерации поток был добавлен вдоль ребра v, u. Так как поток добавляется вдоль кратчайших путей, то путь выглядел следующим образом: $p = s \rightsquigarrow v \rightarrow u \rightsquigarrow t$, из чего получаем $\delta_v^i \leq \delta_u^i - 1 \leq \delta_u^{i+1} - 1 = \delta_v^{i+1} - 2$, что также противоречит начальному утверждению. Таким образом, лемма доказана.

Докажем, что общее число итераций алгоритма (поисков дополняющего пути) равно $O(VE)$.

Назовем ребро (u, v) критическим, если его остаточная пропускная способность минимальна среди всех ребер какого-то дополняющего пути. Очевидно, что путь содержит хотя бы одно критическое ребро, и что после добавления потока вдоль этого пути, все критические ребра пропадут. Но ребро u, v снова может стать критическим, если на какой-то итерации будет добавлен поток вдоль ребра v, u. Докажем, что каждое ребро может становиться критическим не более $O(V)$ раз.

Рассмотрим две вершины u и v, соединенные ребром в исходной сети. Так как добавление потока производится вдоль кратчайших дополняющих путей, то когда ребро в первый раз становится критическим верно, что $\delta_v^i = \delta_u^i + 1$. После этого оно пропадает из остаточной сети. Оно не появится в остаточной сети до тех пор, пока не будет добавлен поток вдоль ребра (v, u). Это произойдет в том случае, если ребро v, u встретиться на некотором увеличивающем пути. Так как увеличивающие пути кратчайшие, то верно, что $\delta_u^{i+j} = \delta_v^{i+j} + 1$, где $j > 0$. Согласно доказанной выше лемме: $\delta_u^{i+j} = \delta_v^{i+1} + 1 \geq \delta_v^i + 1 = \delta_u^i + 2$. Отсюда следует, что от момента, когда ребро становится критическим в первый раз до момента, когда оно становится критическим в следующий раз, кратчайшее расстояние до вершины u увеличивается минимум на 2.

Так как дополняющие пути являются простыми (то есть не содержат циклов), то их длина строго меньше количества вершин. Также известно, что изначально кратчайшие расстояния до каждой вершины больше нуля. Следовательно к тому моменту, когда вершина u станет недостижимой из истока, кратчайшее расстояние до нее не превысит $V - 2$. А значит, что ребро u, v могло стать критическим не более $\frac{V-2}{2}$ раз. Так как в сети $O\ E$ пар вершин, то суммарное количество критических ребер в ходе работы алгоритма Эдмондса-Карпа равно $O(VE)$.

Так как дополняющий путь содержит минимум одно критическое ребро, то общее число дополняющий путей равно $O(VE)$.

Для поиска одного кратчайшего дополняющего пути из истока в сток используется поиск в ширину. Как известно, сложность алгоритма поиска в ширину равна $O(E)$.

Таким образом, общая время алгоритма Эдмондса-Карпа равна $O\ VE^2$.

Рассмотрим работу алгоритма Эдмондса-Карпа на примере (рисунок 2.5).

В левой части рисунка 2.5 изображено состояние сети в начале каждой итерации, в правой части остаточная сеть и дополняющий путь на каждой итерации. Как можно видеть, на каждой итерации выбирается кратчайший дополняющий путь и добавляется поток вдоль этого пути (рисунок 2.5 (б-и)). Алгоритм завершает работу, когда в остаточной сети не остается дополняющих путей (рисунок 2.5 (к)). Также отметим, что в процессе работы алгоритма длины дополняющий путей не убывают. Данное свойство выполняется благодаря тому, что на каждой итерации ищется кратчайший дополняющий путь.

Таким образом, алгоритм Эдмондса-Карпа является первым и самым простым улучшением алгоритма Форда-Фалкерсона, который уже не зависит от величины ответа и является полностью полиномиальным. Однако, как будет рассмотрено далее, существуют и другие более эффективные модернизации алгоритмов Форда-Фалкерсона и Эдмондса-Карпа.

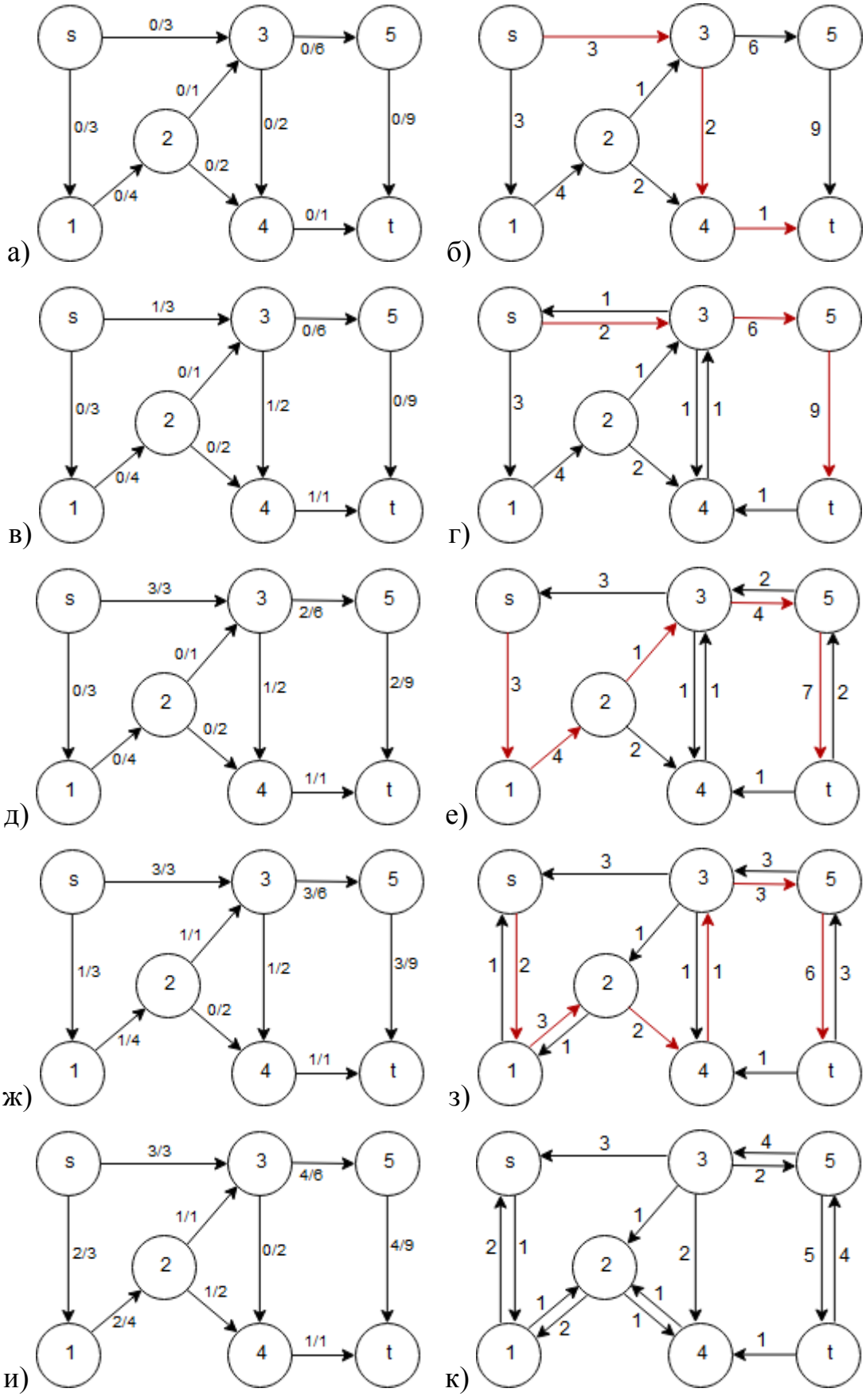

Рисунок 2.5 – Пример работы алгоритма Эдмондса-Карпа

2.3 Алгоритм Диница

Следующим рассматриваемым алгоритмом является алгоритм Диница, разработанный Ефимом Диницем в 1970 году. [10] Он, также как и алгоритмы Форда-Фалкерсона и Эдмондса-Карпа, основан на поиске дополняющих путей и добавления потока вдоль них. Но, несмотря на общую идею, алгоритм Диница имеет серьезное отличие от алгоритмов Форда-Фалкерсона и Эдмондса-Карпа.

Для описания алгоритма Диница введем необходимые определения.

Пусть δ_u – кратчайшее расстояния до вершины u в остаточной сети $G' = (V, E')$. Тогда слоистая сеть остаточной сети G' – это сеть $G_L = (V, E_L)$, где $E_L = \{u, v \quad \delta_v = \delta_u + 1\}$. Пример слоистой сети изображен на рисунке 2.6.

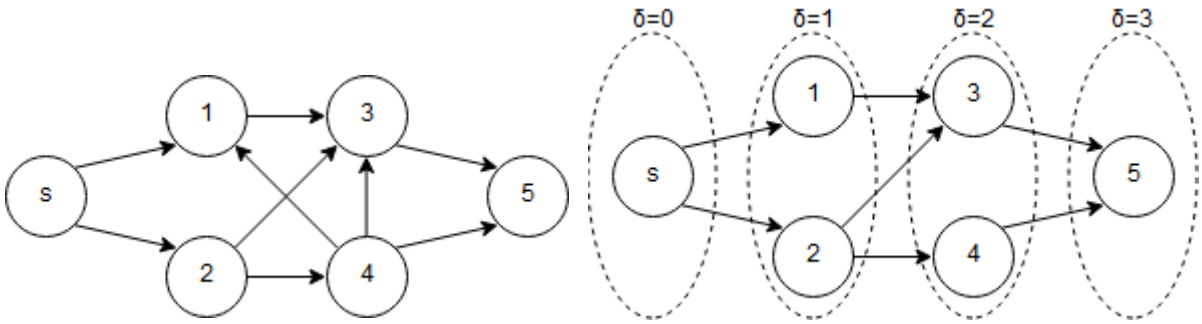

Рисунок 2.6 – Остаточная сеть (слева) и ее слоистая сеть (справа)

Блокирующий поток – это такой поток в сети G, что на любом пути из истока в сток существует такое ребро, остаточная пропускная способность которого равна нулю. Иными словами, в данной сети G не существует пути, вдоль которого можно добавить поток.

Как можно видеть, определение блокирующего потока похоже на определение максимального потока. Однако, по теореме Форда-Фалкерсона поток является максимальным, если в остаточной сети не существует дополняющего пути. В определении же блокирующего потока ничего не говорится о существовании пути из истока в сток, который может появиться в остаточной сети. Поэтому, в общем случае блокирующий поток не равен

максимальному потоку. Рассмотрим следующий пример, демонстрирующий разницу между блокирующим и максимальным потоком (рисунок 2.7).

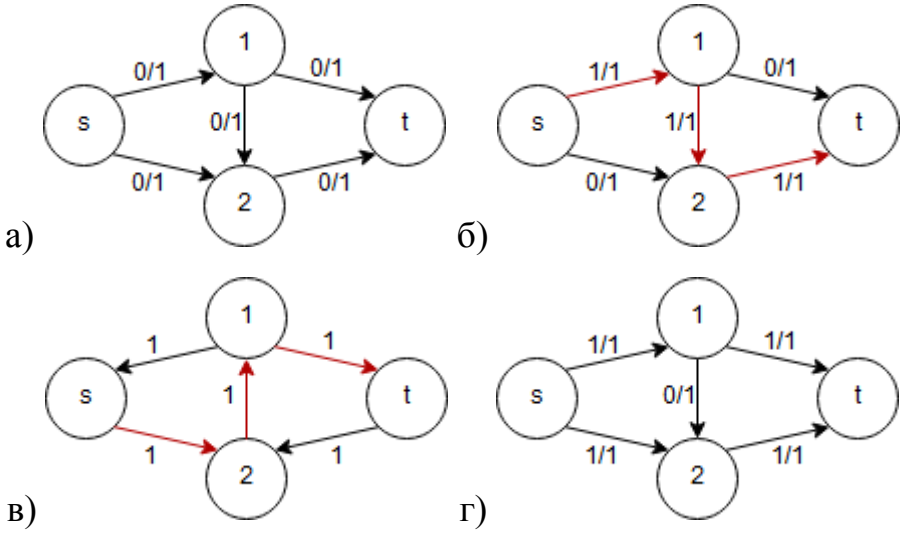

Рисунок 2.7 – Разница между блокирующим и максимальным потоком

На рисунке 2.7 (а) изображена исходная сеть. На рисунке 2.7 (б) изображен один из блокирующих потоков в данной сети. Как можно видеть, в сети на рисунке 2.7 (б) больше нельзя добавить поток вдоль какого-либо пути. Однако, в остаточной сети, изображенной на рисунке 2.7 (в), существует дополняющий путь из истока в сток. Таким образом, блокирующий поток, изображенный на рисунке 2.7 (б) не равен максимальному потоку, изображенному на рисунке 2.7 (г).

Теперь можно дать формальное описание алгоритма Диница:

1. $f_{u,v} := 0$ для всех ребёр u, v
2. Построить слоистую сеть G_L из остаточной сети G'
3. Пока существует путь из истока в сток в слоистой сети:
   3.1. Найти блокирующий поток f' в G_L
   3.2. Дополнить поток f найденным потоком f'
   3.3. Перейти к шагу 2
4. Вернуть f

Докажем корректность алгоритма Диница, а именно то, что если алгоритм завершается, то результатом его работы является поток максимальной

величины. Как известно, алгоритм завершается, когда в слоистой сети не существует пути из истока в сток, но так как слоистая сеть строится из остаточной сети, то это в свою очередь означает, что в остаточной сети также не существует пути из истока в сток. Отсюда, по теореме Форда-Фалкерсона, следует, что поток является максимальным.

Проанализируем временную сложность алгоритма Диница. Как можно видеть, алгоритм состоит из следующих фаз: построение слоистой сети и нахождения блокирующего потока в этой сети. Таким образом, общее время работы алгоритма Диница равно произведению количества фаз на время работы одной фазы.

Докажем, что число фаз алгоритма Диница не превышает $O(V)$. Для этого докажем, что расстояние между истоком и стоком на каждой фазе алгоритма строго увеличивается. Допустим, что это не так, и длина кратчайшего пути из истока в сток после очередной фазы алгоритма не изменилась. Обозначим ее δ_p. Тогда в остаточной сети будут содержаться только ребра, которые были в ней до начала фазы или обратные к ним. Отсюда, получается, что в данной сети найдется путь из истока в сток, по которому можно добавить поток, и он имеет ту же длину δ_p. Но вдоль такого пути должен был быть добавлен блокирующий поток, чего не случилось. Получили противоречие, а значит расстояние между истоком и стоком увеличилось. А так как максимальное расстояние между истоком и стоком не превышает количества вершин, то всего будет выполнено не более $O\ V$ фаз.

Для построения слоистой сети необходимо посчитать кратчайшие расстояния до каждой из вершин. Для этого лучше всего подойдет алгоритм поиска в ширину. Как известно, его временная сложность составляет $O(E)$.

После того как слоистая сеть построена, необходимо найти в ней блокирующий поток. Существует несколько алгоритмов поиска блокирующего потока. Рассмотрим некоторые из них и проанализируем их временную сложность.

Первый, и самый простой, алгоритм поиска блокирующего потока заключается в последовательном поиске путей в слоистой сети, по которым можно добавить ненулевой поток, и добавления потока вдоль этих путей. Таким образом, на поиск одного пути с помощью обхода в глубину потребуется $O\ E$ времени, а всего таких поисков будет $O(E)$, так как после каждого добавления потока вдоль какого-то пути, хотя бы одно ребро этого пути станет насыщенным, то есть его остаточная пропускная способность станет равна нулю. Отсюда следует, что такой алгоритм поиска блокирующего потока работает за $O(E^2)$.

Рассмотрим второй алгоритм поиска блокирующего потока. Он, также как и первый алгоритм, заключается в поиске путей в слоистой сети, однако в процессе поиска в глубину удаляются все ребра, вдоль которых не получается дойти до стока. Пусть p_u – указатель на первое не удаленное ребро в списке смежности вершины u. Если обход в глубину не достигает стока, то как минимум один указатель продвигается вперед. Таким образом, один запуск обхода в глубину работает $O(V + P)$, где P – число продвижений указателей. Пусть всего было сделано T запусков обхода в глубину. Тогда весь алгоритм поиска блокирующего потока отработает за $O(TV + \ _i P_i)$. Так как указатели могут двигаться только вперед, то $\ _i P_i = O(E)$. Учитывая, что в худшем случае может быть $O(E)$ запусков обхода в глубину получаем асимптотику $O\ VE + E\ = O(VE)$.

Отметим, что существуют и другие алгоритмы поиска блокирующего потока. Например, алгоритм, использующий динамические деревья Слетора и Тарьяна [12]; или алгоритм Малхотры — Кумара — Махешвари [7].

Посчитаем общее время работы алгоритма Диница. Если искать блокирующий за $O(VE)$, то суммарное время работы алгоритма будет равно $O\ V\ *\ O\ E\ + O\ VE\ = O(V^2E)$, что уже лучше, чем у алгоритма Эдмондса-Карпа.

Таким образом, алгоритм Диница показал себя асимптотически лучше алгоритмов Форда-Фалкерсона и Эдмондса-Карпа, однако он более сложен в реализации и анализе. Каждый из существующих алгоритмов поиска блокирующего потока также требует тщательного анализа. Но с другой стороны, их разнообразие позволяет выбрать тот, который будет приемлем в конкретной ситуации.

2.4 Метод масштабирования потока

Рассмотрим следующий метод, который позволит улучшить асимптотики рассмотренных выше алгоритмов Форда-Фалкерсона, Эдмондса-Карпа, Диница. Вспомним, что в алгоритме Форда-Фалкерсона поток добавляется вдоль любого дополняющего пути, а в алгоритмах Эдмондса-Карпа и Диница используется идея добавления потока вдоль кратчайших путей. Однако, ни в одном алгоритме не накладывается ограничения на то, сколько именно можно добавить потока вдоль какого-то пути. Оказывается, что добавление потока вдоль путей с высокой пропускной способностью является более эффективным, чем добавление потока вдоль кратчайших путей. На этом основан метод масштабирования потока [11], который можно применить ко всем рассмотренным алгоритмам: алгоритму Форда-Фалкерсона, алгоритму Эдмондса-Карпа, алгоритму Диница. Однако, стоит отметить, что данный метод подходит только для сетей с целочисленными пропускными способностями.

Рассмотрим применение метода масштабирования потока к алгоритму Форда-Фалкерсона. Пусть дана сеть $G = V, E$. Обозначим за C максимальную пропускную способность среди всех ребер, то есть $C = max_{(u,v)\in E} c_{u,v}$. А также введем следующую переменную Δ называемую масштабом. Пусть изначально $\Delta = 2^{\log_2 C}$. На каждой итерации алгоритм Форда-Фалкерсона с масштабированием потока будет искать такой путь, вдоль которого можно добавить хотя бы Δ единиц потока. Когда все такие пути закончатся, уменьшим

масштаб в два раза. Как можно видеть, при $\Delta = 1$ данный алгоритм вырождается в обычный алгоритм Форда-Фалкерсона.

Формально алгоритм Форда-Фалкерсона с масштабированием потока можно записать следующим образом.

1. $f_{u,v} \coloneqq 0$ для всех ребёр (u, v)
2. $C \coloneqq \max_{(u,v)\in E} c_{(u,v)}$
3. $\Delta \coloneqq 2^{\log_2 C}$
4. Пока $\Delta \geq 1$:
   4.1. Пока есть дополняющий путь p с пропускной способностью не меньше Δ в сети G':
      4.1.1. Найти $c_{min} = \min\ c'_{u,v} \quad (u, v) \in p\}$
      4.1.2. Для каждого ребра $u, v \in p$:
         4.1.2.1. $f_{u,v} \coloneqq f_{u,v} + c_{\min}$
         4.1.2.2. $f_{v,u} \coloneqq f_{v,u} - c_{min}$
   4.2. $\Delta \coloneqq \frac{\Delta}{2}$
5. Вернуть f

Данный алгоритм является корректным, так как при величине масштаба равной единице он вырождается в обычный алгоритм Форда-Фалкерсона.

Проанализируем время работы данного алгоритма. Докажем следующую лемму, которая говорит, что величина остаточного потока после итерации с масштабом 2^k не превосходит $2^k E$. Так как после итерации с масштабом 2^k остаточную сеть можно разбить на два непересекающихся множества, так что остаточные пропускные способности всех ребер, идущих из вершин первого множества в вершины второго множества, не превышают масштаба, то образуется разрез. Так как количество ребер в разрезе не превосходит E, то значение остаточного потока не превосходит $2^k E$.

Далее докажем, что на каждой итерации алгоритма Форда-Фалкерсона с масштабированием потока выполняется не более E поисков дополняющих путей. Пусть на некоторой итерации все дополняющие пути имеют пропускную

способность не меньше 2^k. Тогда по доказанной выше лемме, величина остаточного потока на данной итерации ограничена $2^{k+1}E$, а так как вдоль каждого пути добавляется как минимум 2^k единиц потока, то всего таких путей на данной итерации будет не больше E. Отсюда следует, что общее количество дополняющих путей равно $O(ElogC)$.

Таким образом, общее время работы алгоритма Форда-Фалкерсона с масштабированием потока равно $O(E^2logC)$, так как поиск одного дополняющего пути работает за $O(E)$.

Как можно видеть, данный метод избавляет классический алгоритм Форда-Фалкерсона от его самого главного недостатка – зависимости от величины ответа. Далее применим метод масштабирования потока к алгоритмам Эдмондса-Карпа и Диница.

Вспомним, что ключевое отличие алгоритма Эдмондса-Карпа от алгоритма Форда-Фалкерсона заключается в том, что поток добавляется вдоль кратчайших путей. Однако, при применении метода масштабирования потока к алгоритму Эдмондса-Карпа данное различие не имеет особого значения. Как уже было сказано, общее количество дополняющих путей при использовании масштабирования потока равно $O(ElogC)$. А так как в алгоритме Эдмондса-Карпа для поиска дополняющих путей используется алгоритм поиска в ширину, который работает за $O\ E$ для одного пути, то суммарная асимптотика такая же, как и у алгоритма Форда-Фалкерсона, то есть $O(E^2logC)$.

Как можно видеть, асимптотика алгоритма Эдмондса-Карпа улучшилась, однако применение метода масштабирования потока не избавило данный алгоритм от квадратичной зависимости от числа ребер.

Использование идеи масштабирования для алгоритма Диница ничем не отличается от применения данного метода к алгоритмам Эдмондса-Карпа и Форда-Фалкерсона, однако требует более тщательного анализа, ввиду того, что сам алгоритм Диница существенно отличается от алгоритмов Эдмондса-Карпа и Форда-Фалкерсона.

Как говорилось ранее, время работы алгоритма поиска одного блокирующего потока в алгоритме Диница равно $O(TV + \sum_i P_i)$, где T – количество путей, вдоль которых добавлялся поток, $\sum_i P_i$ – число продвижений указателей, что в худшем случае дает $O(VE + E)$. Заметим, что первое слагаемое в асимптотике $O(TV + \sum_i P_i)$ равно сумме длин путей, найденных алгоритмом поиска блокирующего потока на какой-то фазе. Обозначим данную сумму длин путей, найденных на j фазе, как S_j. Тогда общее время работы алгоритма Диница можно записать как $O(\sum_j S_j + VE)$. Здесь $\sum_j S_j$ является суммой длин всех путей, найденных алгоритмом Диница. Но как было доказано выше, при заданном масштабе 2^k общее число путей не превышает E, а значит $\sum_j S_j = O(VE)$. Таким образом, суммарное время работы алгоритма Диница с масштабированием потока равно $O(VElogC)$, что избавляет классический алгоритм Диница от его квадратичной зависимости от числа вершин.

Исходя из вышесказанного видно, что метод масштабирования потока может существенно ускорить время работы алгоритмов поиска максимального потока, а именно алгоритмов Форда-Фалкерсона, Эдмондса-Карпа, Диница. Однако, несмотря на свою эффективность, он имеет достаточно большой недостаток – данный метод можно применять только при целочисленных величинах пропускных способностей.

2.5 Сравнение алгоритмов поиска максимального потока

Были рассмотрены основные алгоритмы поиска максимального потока в сети, а именно алгоритмы Форда-Фалкерсона, Эдмондса-Карпа, Диница, а также метод масштабирования потока, применимый ко всем рассмотренным алгоритмам. Временные сложности данных алгоритмов различные, а значит при решении какой-либо практической задачи может возникнуть вопрос: какой алгоритм поиска максимального потока выбрать, чтобы решить задачу наиболее эффективно.

В таблице 2.1 приведены временные сложности работы алгоритмов поиска максимального потока.

При выборе алгоритма поиска максимального потока важно оценить сеть, в которой будет искаться поток. Вначале необходимо оценить размеры сети, то есть выяснить какое максимальное количество вершин и ребер в ней может быть. Затем нужно узнать какие пропускные способности используются: целочисленные или вещественные. При возможности преобразовать сеть к целочисленным пропускным способностям. Далее требуется узнать максимальную пропускную способность. Зная все эти величины можно подобрать наилучший алгоритм поиска максимального потока, подходящий в конкретной ситуации. К примеру, на малых сетях, где число вершин, ребер и величина пропускной способности невелики, особой разницы между алгоритмами не будет видно; однако, на больших сетях ситуация может сильно измениться. Рассмотрим несколько примеров.

Таблица 2.1 – Асимптотики работы алгоритмов поиска максимального потока

Алгоритм	**Асимптотика**
Форд-Фалкерсон	$O(E\ f\ )$
Эдмондс-Карп	$O\ VE^2$
Диниц	$O(V^2E)$
Форд-Фалкерсон с масштабированием	$O(E^2logC)$
Эдмондс-Карп с масштабированием	$O(E^2logC)$
Диниц с масштабированием	$O(VElogC)$

Пусть величина максимальной пропускной способности много больше числа вершин и ребер. Тогда выбор классического алгоритма Форда-Фалкерсона будет нецелесообразным, так как он зависит от величины ответа, которая может быть очень большой.

Далее рассмотрим случай, когда сеть является разреженной, то есть число ребер немногим больше числа вершин ($O\ E\ = O(V)$). Тогда алгоритмы Эдмондса-Карпа и Диница асимптотически становятся равными ($O(V^3)$), а также становятся равными все алгоритмы с использованием метода масштабирования потока ($O(V^2 logC)$). Однако алгоритмы, использующие масштабирование имеют меньшую зависимость от числа вершин, а значит, скорее всего, окажутся более эффективными.

Также рассмотрим вариант плотной сети, в которой $O\ E\ = O(V^2)$. Тогда лучшим выбором, согласно асимптотикам работ алгоритмов, будет алгоритм Диница с масштабированием ($O(V^3 logC)$), тогда как у классического алгоритма Диница время работы будет $O(V^4)$, а у алгоритма Эдмондса-Карпа без масштабирования и с масштабированием $O(V^5)$ и $O(V^4 logC)$, соответственно.

Таким образом, для выбора эффективного алгоритма необходимо вначале вычислить максимальное время его работы. Однако, если необходимо выбрать из нескольких алгоритмов, то возникает вопрос: по какому критерию выбрать лучший.

Как известно, сеть можно охарактеризовать не только ее размерами или величинами пропускных способностей. Различные сети могут иметь различную архитектуру. Одни могут быть сильно вытянуты в длину, другие наоборот иметь большое количество коротких путей, однако их размеры могут быть равны. В таком случае, непонятно, какой алгоритм поиска максимального потока выбрать. Возможно, на разных архитектурах сетей алгоритмы будут показывать различную эффективность. Поэтому, для выбора алгоритма поиска максимального потока, лучше всего подходящего для конкретной задачи, необходимо провести их тестирование на приближенных к реальным данным и осуществить выбор, согласно результатам этого тестирования.

ГЛАВА 3 ТЕСТИРОВАНИЕ АЛГОРИТМОВ ПОИСКА МАКСИМАЛЬНОГО ПОТОКА

3.1 Исходные данные для тестирования

Для проведения тестирования рассмотренных выше алгоритмов поиска максимального потока необходимо подготовить тестовые данные, а также разработать компьютерную программу, реализующую данные алгоритмы.

Для подготовки тестовых данных реализована компьютерная программа, генерирующая следующие графы:

- случайные графы;
- ациклические разреженные графы;
- полные графы;
- двудольные графы;
- графы решётки (двухмерные и трехмерные).

Рассмотрим процесс генерирования каждого графа подробнее.

Генерирование случайных графов происходит следующим образом. Фиксируется количество вершин V и ребер E, затем с помощью генератора случайных чисел E раз генерируются пары чисел – ориентированные ребра графа. Сами значения V и E лежат на отрезке $[5 \times 10^3; 10^5]$ и выбираются случайно.

Рассмотрим процесс генерирования ациклических разреженных графов. Как известно, количество ребер в них немногим больше количества вершин, поэтому фиксируется два значения V – количество вершин и T – максимальное количество ребер, которое может выходить из одной вершины. Значение V лежит на отрезке $[10^2; 10^5]$, а T – на отрезке $[10; 10^2]$, они выбираются случайно. Также граф должен быть ациклическим. Для этого для каждой вершины u не более T раз генерируются числа $v > u$, и в граф добавляются ориентированные ребра (u, v). Таким образом, получается граф с V вершинами и VT ребрами. Пример ациклического разреженного графа приведен на рисунке 3.1.

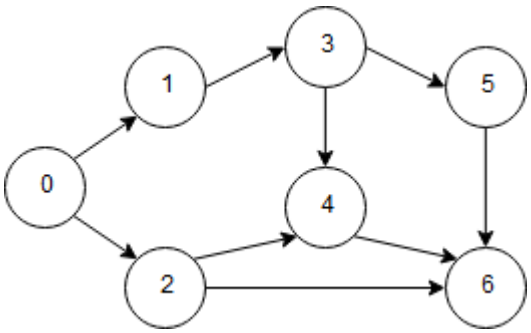

Рисунок 3.1 – Пример ациклического разреженного графа

Для генерирования полных графов необходимо зафиксировать только количество вершин V. Оно выбирается случайно на отрезке $[10^2; 5 \times 10^3]$. После этого будет сгенерирован граф, в котором у каждой вершины есть ориентированные ребра во все вершины. Пример полного графа приведен на рисунке 3.2.

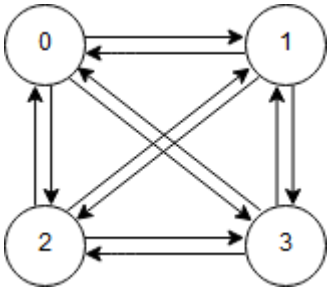

Рисунок 3.2 – Пример полного графа с 4 вершинами

Двудольные графы представляют собой графы, множество вершин которых можно разбить на два непересекающихся подмножества, таких, что не существует ребер между вершинами находящихся в одном подмножестве. Пример такого графа изображен на рисунке 3.3.

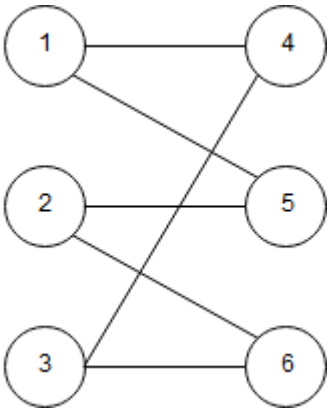

Рисунок 3.3 – Пример двудольного графа

Таким образом, на рисунке 3.3 есть два подмножества $1,2,3$ и $\{4,5,6\}$ – левая и правая доля, соответственно.

Для генерации двудольных графов фиксируется число вершин V, равное для каждой доли, а также число ребер E между вершинами из разных долей. Далее E раз генерируются пары чисел (u, v) и добавляется два ориентированных ребра: из u-й вершины левой доли в v-ю вершину правой доли, и обратное к нему. Однако для поиска максимального потока необходимы исток и сток. Для этого соединим ориентированными ребрами исток со всеми вершинами левой доли и вершины правой доли со стоком. Значение V лежит на отрезке $[5 \times 10^3; 10^5]$, а значение E – на отрезке $[5 \times 10^4; 10^6]$. Оба выбираются случайно.

Граф решётка – это граф, рисунок которого, вложенный в некоторое евклидово пространство R^n образует регулярную мозаику. В данной работе используются графы квадратной решётки – графы, вершины которых соответствуют точкам на плоскости с различными координатами, а ребрами соединены вершины, если их соответствующие точки находятся на расстоянии 1. Пример графа двухмерной и трехмерной квадратной решётки приведен на рисунке 3.4.

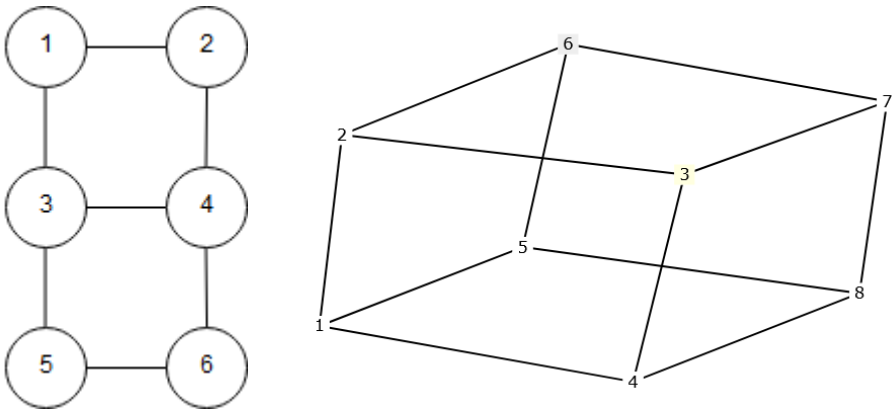

Рисунок 3.4 – Пример графа двухмерной (слева) и трехмерной (справа) квадратной решётки

Рассмотрим процесс генерации графа двухмерной квадратной решётки. Для этого зафиксируем две величины H – «высоту» и W – «длину» графа.

После этого создадим вершины решётки $H \times W$ и соединим ориентированными ребрами те, которые являются «соседними». Однако, также как и в случае с двудольными графами необходим исток и сток. Для этого добавим исток и соединим его ориентированными ребрами с самыми «левыми» вершинами решётки. Также добавим сток и соединим ориентированными ребрами самые «правые» вершины с ним (рисунок 3.5). Величины H и W лежат на отрезке $[10; 10^3]$ и выбираются случайно.

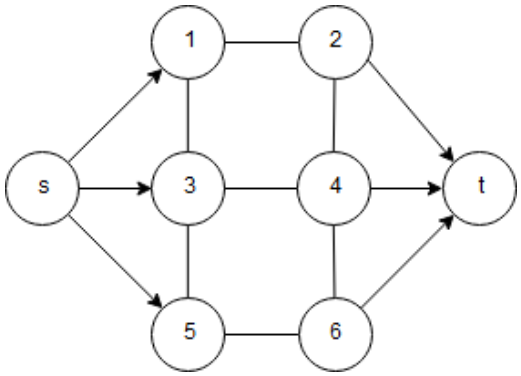

Рисунок 3.5 – Пример графа двухмерной квадратной решётки с истоком и стоком

Последним рассматриваем видом графа является граф трехмерной квадратной решётки. Ввиду того, что анализировать функцию от трех и более переменных достаточно сложно, для генерации данного графа будет фиксироваться два значения H – «высота» графа и величина A^2 – размер «горизонтальной» сетки. Добавление вершин и ребер происходит аналогично графу двухмерной квадратной решётки, однако исток соединяется со всеми вершинами «верхней» горизонтальной сетки, а со стоком соединяются все вершины «нижней» горизонтальной сетки (рисунок 3.6). Значения H и A лежат на отрезке $10; 10^2$ и выбираются случайно.

После того, как исходные данные были подготовлены, была разработана компьютерная программа, реализующая рассмотренные алгоритмы поиска максимального потока.

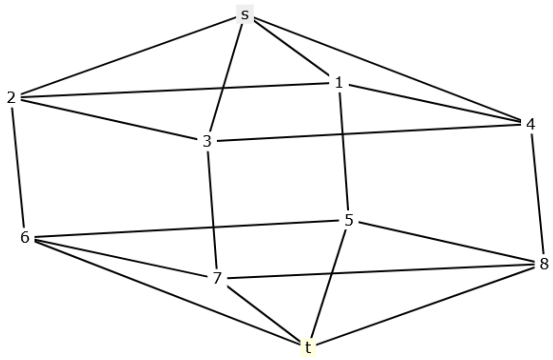

Рисунок 3.6 – Пример графа трехмерной квадратной решетки с истоком и стоком

3.2 Тестирование алгоритмов

Тестирование рассмотренных выше алгоритмов поиска максимального потока проводилось следующим образом. Вначале фиксируется один из сгенерированных графов G. После фиксируется значение максимальной пропускной способности C_{max} из набора $\{10^3, 10^6, 10^9\}$. Далее каждому ребру u, v графа G присваивается случайное целое число из диапазона $[1; C_{max}]$. Если в данном графе нет истока и стока, то ими являются вершины 0 и $V - 1$, соответственно. На получившейся сети производится несколько запусков каждого алгоритма и вычисляется среднее время работы.

Ниже представлены результаты проведенного тестирования.

3.2.1 Тестирование на случайных графах

Рассмотрим результаты тестирования рассмотренных алгоритмов поиска максимального потока на случайных графах с различными ограничениями максимальной пропускной способности (C_{max}) на рисунках 3.7-3.9.

Как можно видеть по рисунку 3.7, все алгоритмы, кроме алгоритма Форда-Фалкерсона (рисунок 3.7 (д)), отработали достаточно быстро. Большое время работы алгоритма Форда-Фалкерсона подтверждается его зависимостью от величины ответа, и даже на случайных графах с достаточно малым

ограничением максимальной пропускной способности данный алгоритм может быть крайне неэффективен.

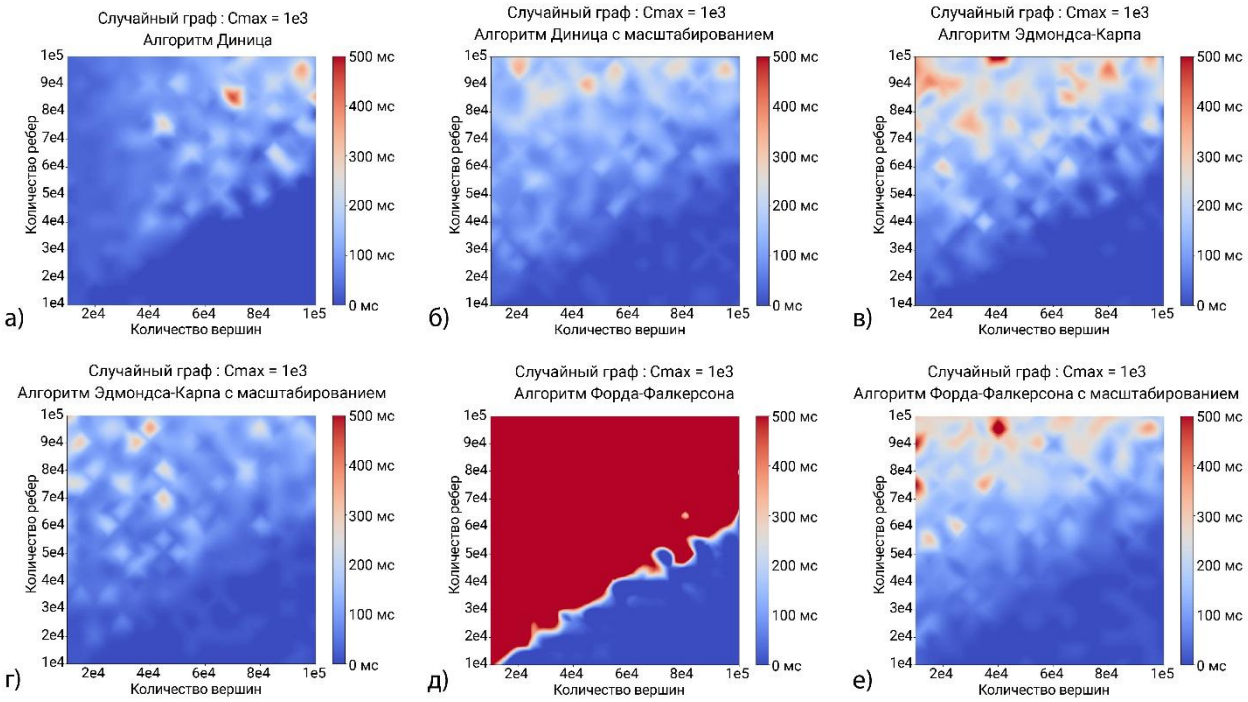

Рисунок 3.7 – Результаты тестирования на случайных графах с $C_{max} = 10^3$

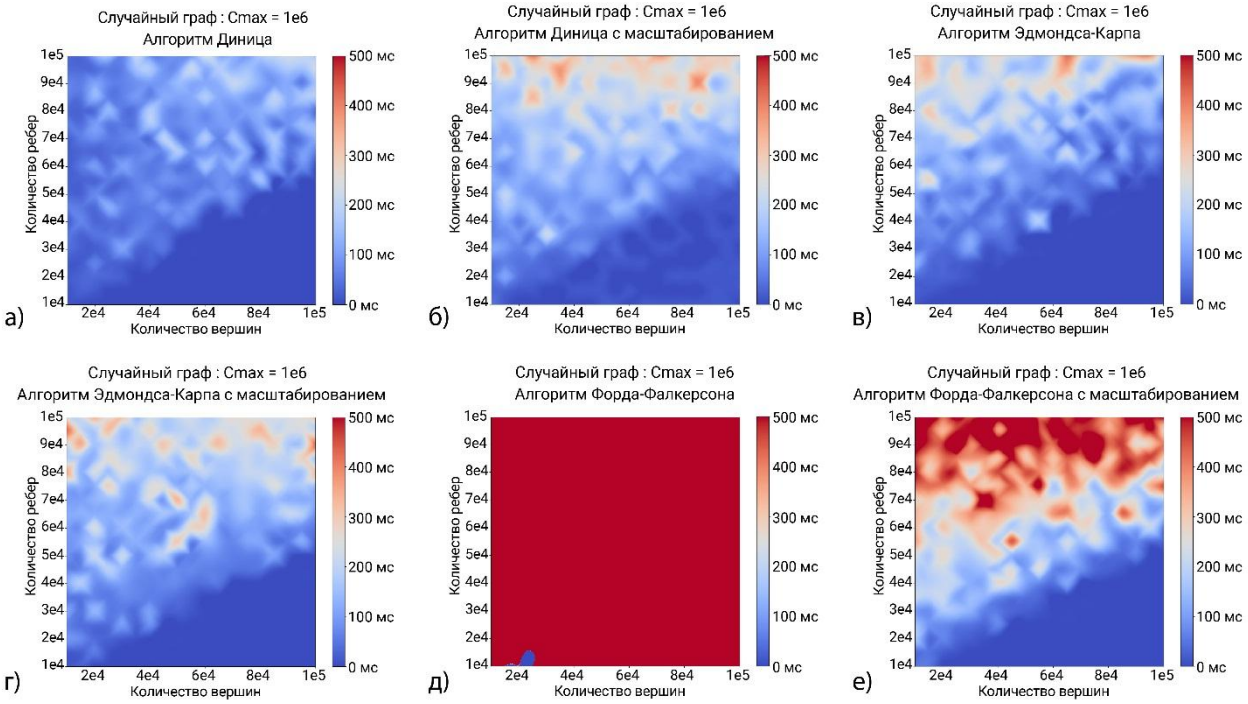

Рисунок 3.8 – Результаты тестирования на случайных графах с $C_{max} = 10^6$

При увеличении максимальной пропускной способности ребер видно, что алгоритм Форда-Фалкерсона стал работать еще медленнее (рисунок 3.8 (д)), что вполне ожидаемо, но также и увеличилось время работы алгоритма Форда-Фалкерсона с масштабированием потока (рисунок 3.8 (е)).

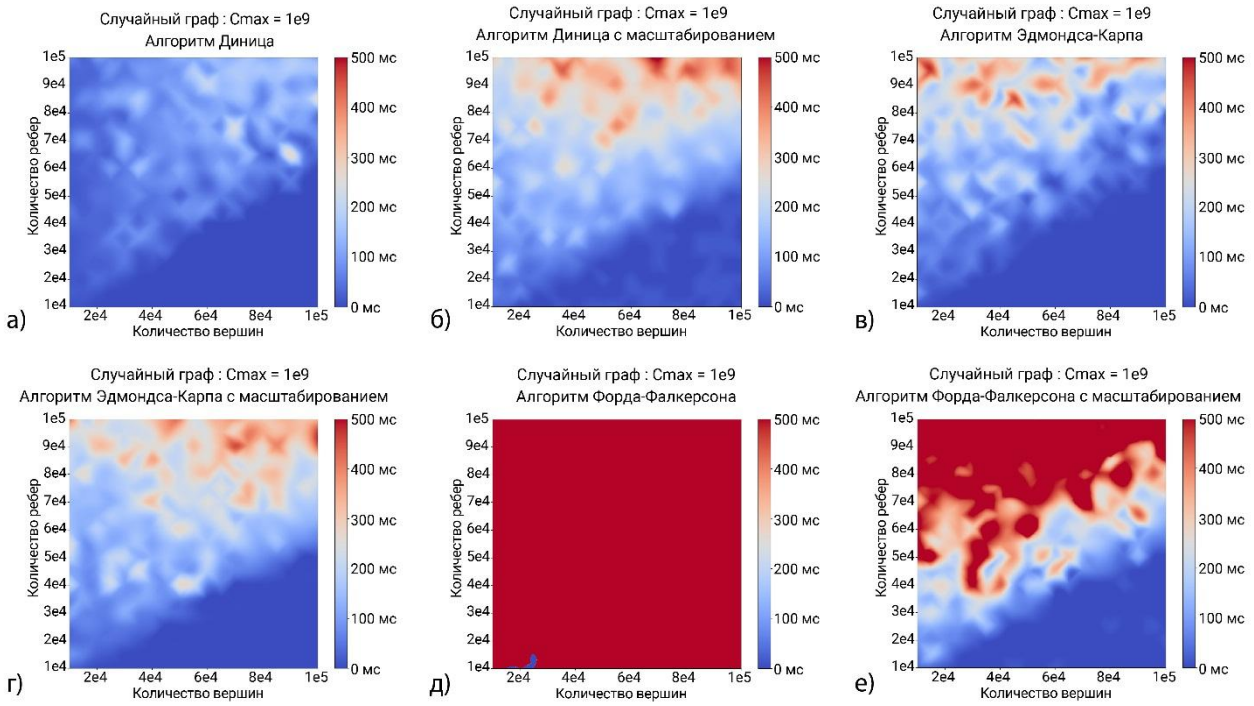

Рисунок 3.9 – Результаты тестирования на случайных графах с $C_{max} = 10^9$

По данным тестирования на случайных графах видно, что алгоритмы Диница и Эдмондса-Карпа с масштабированием потока и без (рисунки 3.7-3.9 (а-г)) сохраняют свою эффективность, несмотря на увеличение максимальной пропускной способности ребер. А вот алгоритм Форда-Фалкерсона с масштабированием потока и без с увеличением пропускной способности теряет свою эффективность. Снижение эффективности объясняется зависимостью от максимальной пропускной способности ребра. В варианте с масштабированием – логарифмической, без масштабирования – линейной. Однако алгоритмы Диница и Эдмондса-Карпа с масштабированием также имеют логарифмическую зависимость от пропускной способности ребер, но на случайных графах, как можно видеть, она не играет существенной роли.

3.2.2 Тестирование на ациклических разреженных графах

Далее рассмотрим результаты тестирования алгоритмов поиска максимального потока на ациклических разреженных графах (рисунки 3.10-3.12).

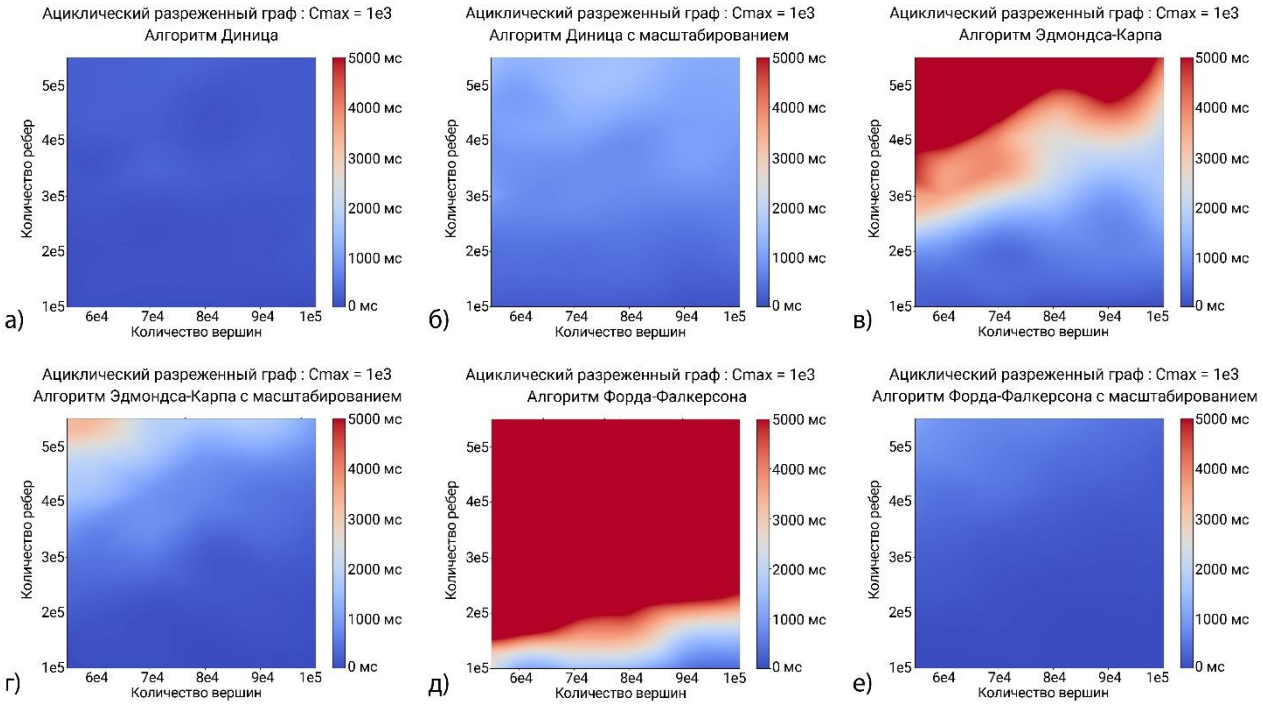

Рисунок 3.10 – Результаты тестирования
на ациклических разреженных графах с $C_{max} = 10^3$

Как можно видеть по рисунку 3.10, время работы алгоритма Эдмондса-Карпа без масштабирования потока с увеличением количества ребер возрастает, однако при использовании масштабирования эффективность данного алгоритма увеличивается. То же самое можно сказать и про алгоритм Форда-Фалкерсона.

При увеличении максимальной пропускной способности ребра время работы алгоритмов, кроме алгоритма Форда-Фалкерсона, осталось почти неизменным. Однако судя по графикам на рисунках 3.10 (е) и 3.11 (е), эффективность работы алгоритма Форда-Фалкерсона с масштабированием потока начала снижаться.

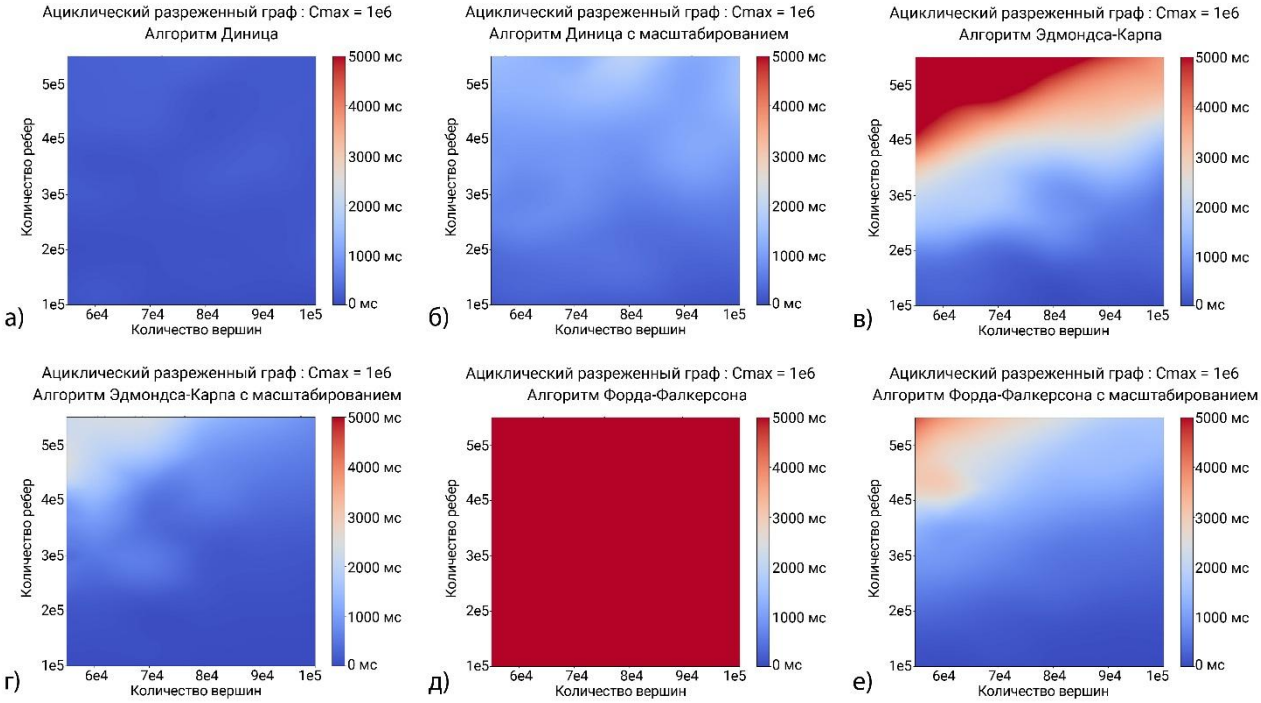

Рисунок 3.11 – Результаты тестирования

на ациклических разреженных графах с $C_{max} = 10^6$

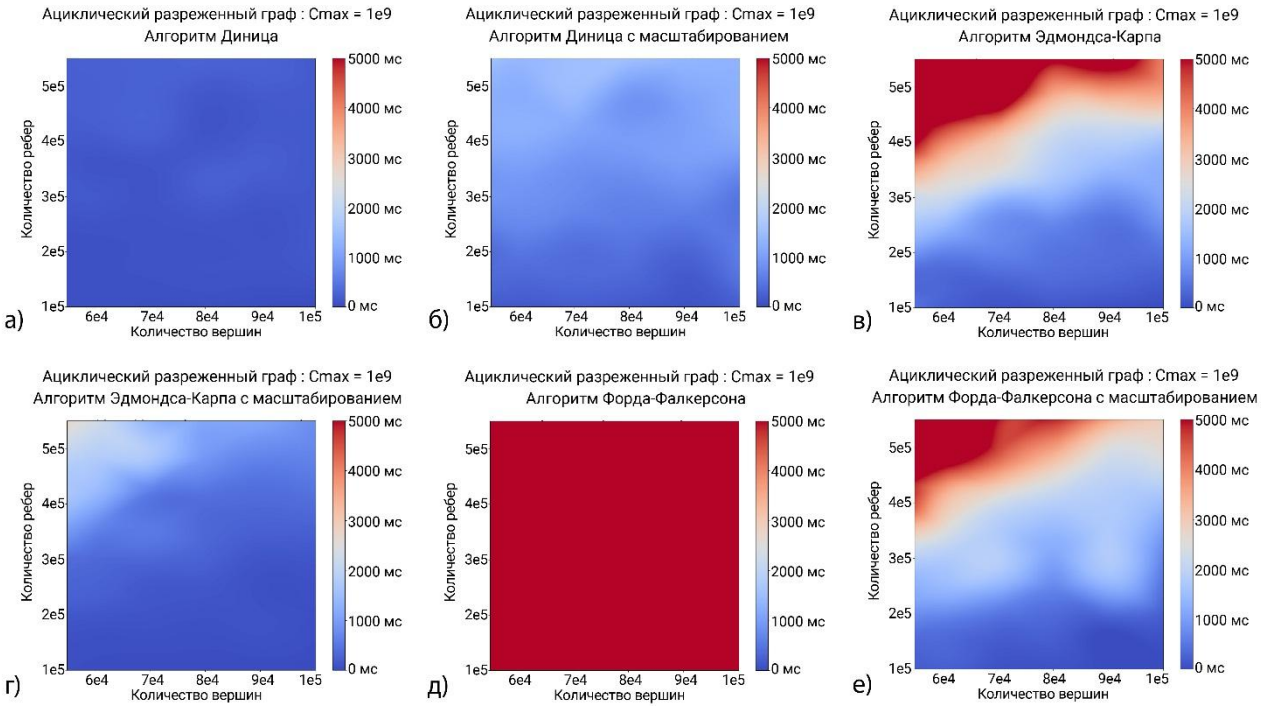

Рисунок 3.12 – Результаты тестирования

на ациклических разреженных графах с $C_{max} = 10^9$

Таким образом, результаты тестирования на ациклических разреженных графах не показали ничего необычного. Алгоритм Форда-Фалкерсона, из-за

зависимости от величины ответа, является неэффективным (рисунки 3.10-3.12 (д)). Однако, применение метода масштабирования потока позволяет использовать алгоритм Форда-Фалкерсона, но с увеличением максимальной пропускной способности ребер скорость его работы также снижается (рисунки 3.10-3.12 (е)). Алгоритм Эдмондса-Карпа зависит квадратично от числа ребер в графе, что влияет на большое время работы на графах с большим количеством ребер. Это подтверждается результатами тестирования (рисунки 3.10-3.12 (в)). Алгоритм Диница с масштабированием и без на случайных и ациклических разреженных графах, как и предполагалось, показывает лучший результат.

3.2.3 Тестирование на полных графах

Проведем тестирование алгоритмов поиска максимального потока на полных графах. Как было показано выше, алгоритм Диница с масштабированием потока является наилучшим выбором среди рассматриваемых алгоритмов для поиска максимального потока на полных графах. Проверим данное предположение.

Рассмотрим результаты тестирования алгоритмов поиска максимального потока на полных графах (рисунок 3.13-3.15).

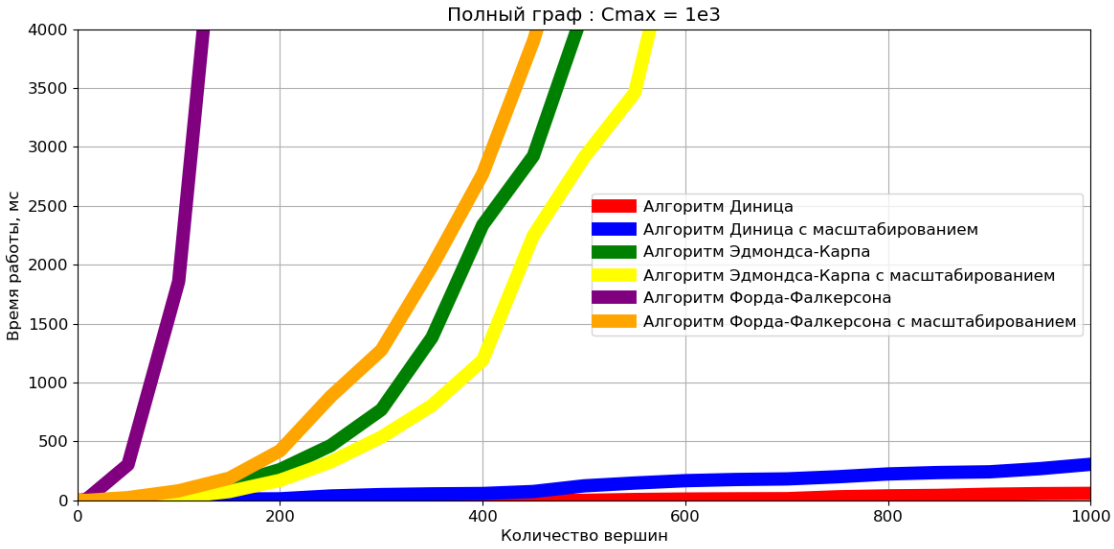

Рисунок 3.13 – Результаты тестирования на полных графах с $C_{max} = 10^3$

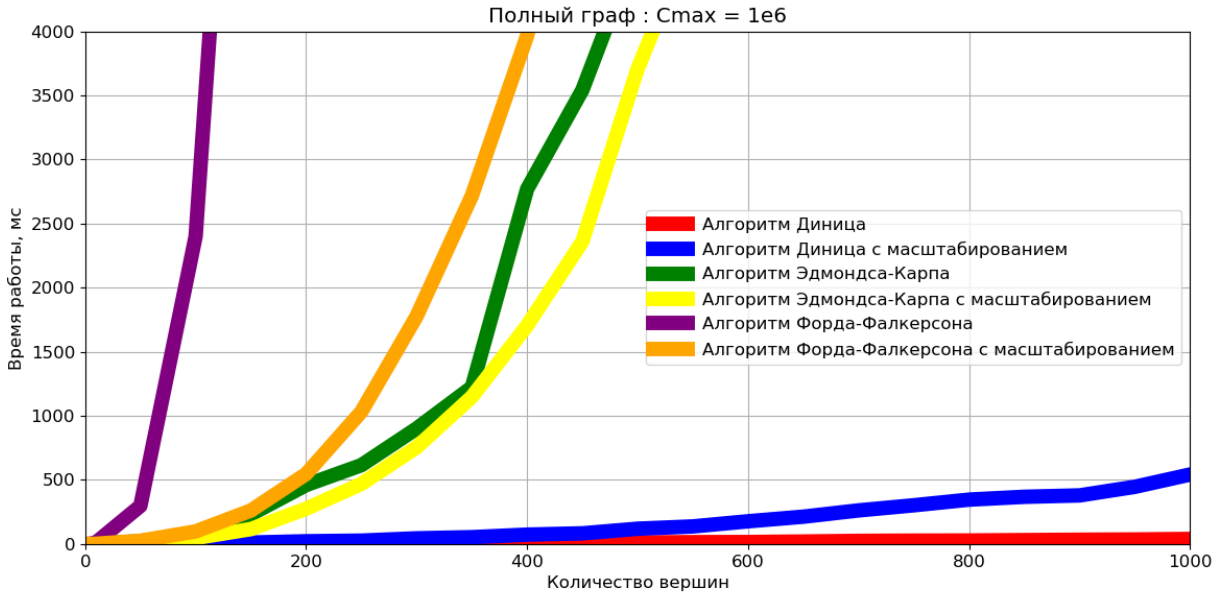

Рисунок 3.14 – Результаты тестирования на полных графах с $C_{max} = 10^6$

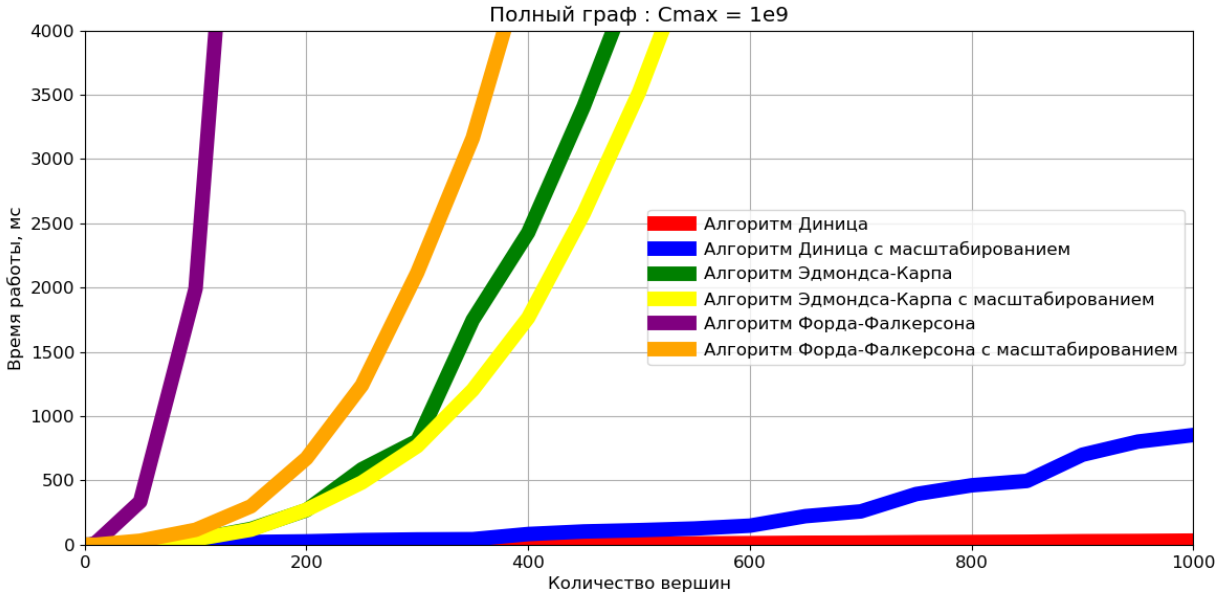

Рисунок 3.15 – Результаты тестирования на полных графах с $C_{max} = 10^9$

На основе результатов тестирования на полных графах можно сделать вывод, что алгоритм Диница с масштабированием потока не является самым эффективным. Как оказалось, на практике классический алгоритм Диница работает очень быстро на полных графах, чему довольно сложно дать объяснение. Небольшая разница во времени работы алгоритма Диница с масштабированием и без скорее всего обусловлена тем, что при использовании масштабирования алгоритм совершает большее количество фаз, ввиду

фиксирования масштаба. Далее по эффективности следует алгоритм Эдмондса-Карпа с масштабированием потока и без, после алгоритм Форда-Фалкерсона с масштабированием и без. Согласно асимптотикам работы, алгоритмы Эдмондса-Карпа с масштабированием и Форда-Фалкерсона с масштабированием на полных графах должны работать примерно одинаково, однако разница во времени их работы во время тестирования достаточно существенна. Также видно, что даже классический алгоритм Эдмондса-Карпа оказался более эффективным, чем алгоритм Форда-Фалкерсона с масштабированием, и почти таким же эффективным, как и вариант с примененим масштабирования.

Отсюда можно сделать вывод, что при выборе алгоритма поиска максимального потока для решения какой-либо задачи теоретическая оценка сложности алгоритмов может не отражать реальной скорости их работы.

3.2.4 Тестирование на двудольных графах

Рассмотрим результаты тестирования на двудольных графах, представленные на рисунках 3.16-3.17.

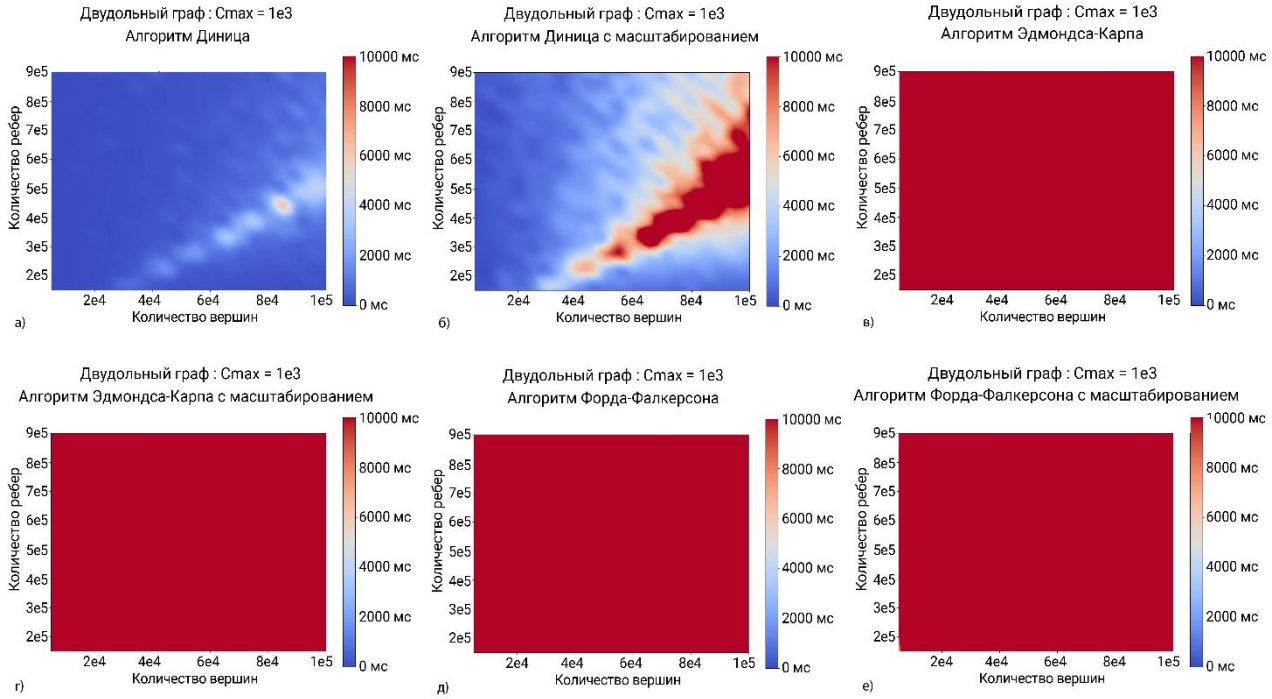

Рисунок 3.16 – Результаты тестирования на двудольных графах с $C_{max} = 10^3$

Как можно видеть по рисунку 3.16, алгоритм Диница как с масштабированием так и без работает очень быстро относительно алгоритмов Эдмондса-Карпа и Форда-Фалкерсона. Также стоит отметить, что классический алгоритм Диница снова показал себя более эффективным (рисунок 3.16 (а)), чем вариант с применением метода масштабирования потока (рисунок 3.16 (б)).

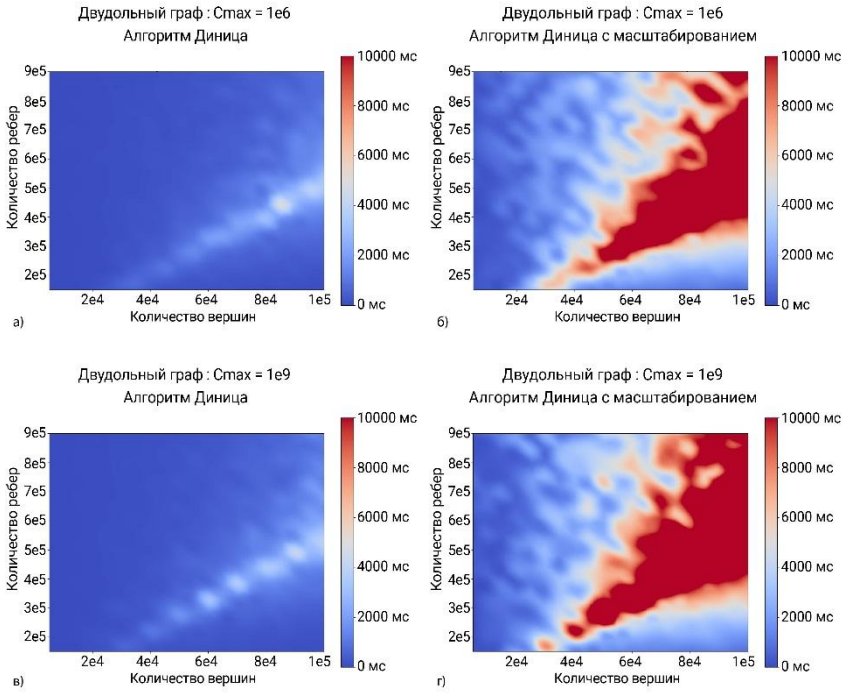

Рисунок 3.17 – Результаты тестирования на двудольных графах с $C_{max} = 10^6$ и $C_{max} = 10^9$

С увеличением максимальной пропускной способности ребра скорость работы классического алгоритма Диница осталась прежней (рисунок 3.17 (а, в)), а у алгоритма Диница с масштабированием эффективность снизилась (рисунок 3.17 (б, г)), однако это можно объяснить тем, что алгоритм совершает больше фаз. Алгоритмы Эдмондса-Карпа и Форда-Фалкерсона даже при малой пропускной способности ребер показали очень медленное время работы, поэтому тестирование данных алгоритмов с большими пропускными способностями не проводилось.

3.2.5 Тестирование на графах решётках

Далее проведем тестирование рассмотренных алгоритмов поиска максимального потока на графах решётках. Данные графы особенны тем, что количество ребер в них немногим превышает количество вершин. Для отображения результатов тестирования будет использоваться зависимость от двух переменных: для графов двухмерной решётки – «длина» и «ширина», для графов трехмерной решётки – «высота» и размер «горизонтальной» сетки.

Рассмотрим результаты тестирования на графах двухмерной квадратной решётки, изображенные на рисунка 3.18-3.20.

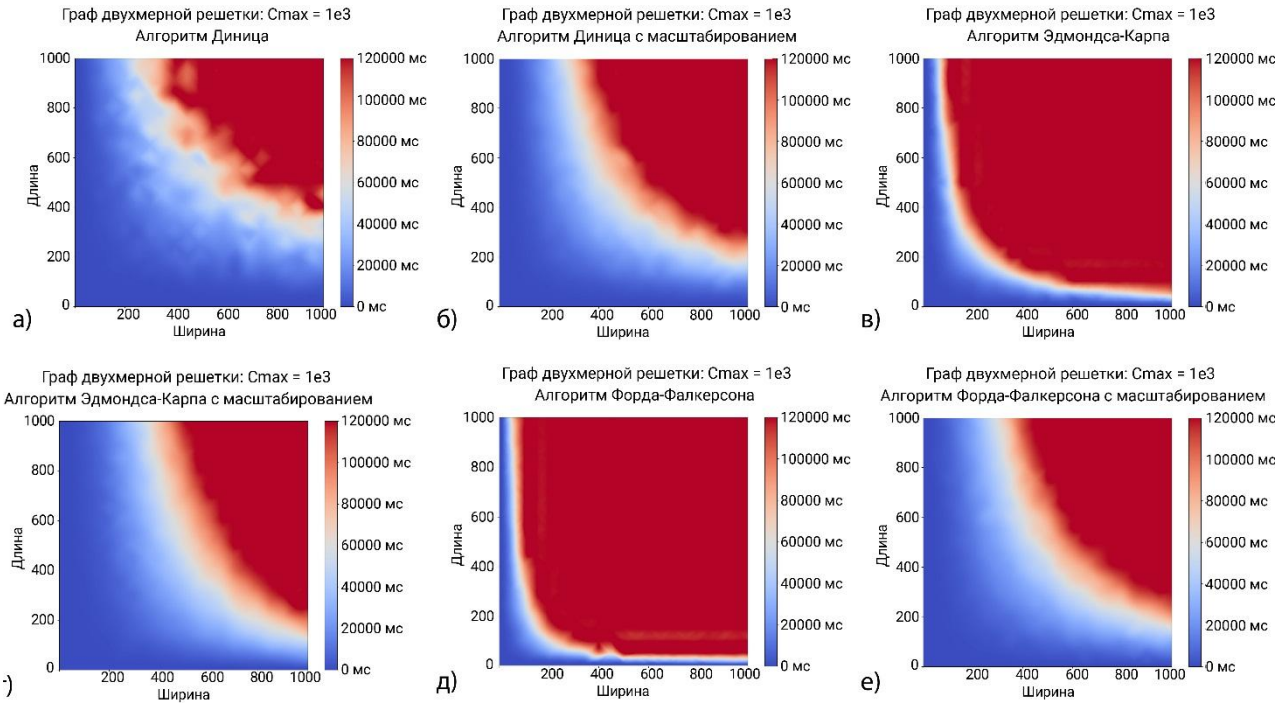

Рисунок 3.18 – Результаты тестирования
на графе двухмерной решётки с $C_{max} = 10^3$

Согласно асимптотикам работ алгоритмов, на данном виде графа самыми эффективными должны быть все алгоритмы с применением масштабирования потока, что и подтверждается результатами тестирования. Алгоритмы Диница и Эдмондса-Карпа без использования масштабирования асимптотически на графе двухмерной решётки тоже равны. Однако, алгоритм Диница показывает более высокий результат, сравнимый с результатами алгоритмов, использующих

масштабирование, а алгоритм Эдмондса-Карпа, наоборот, по скорости работы близок к классическому алгоритму Форда-Фалкерсона.

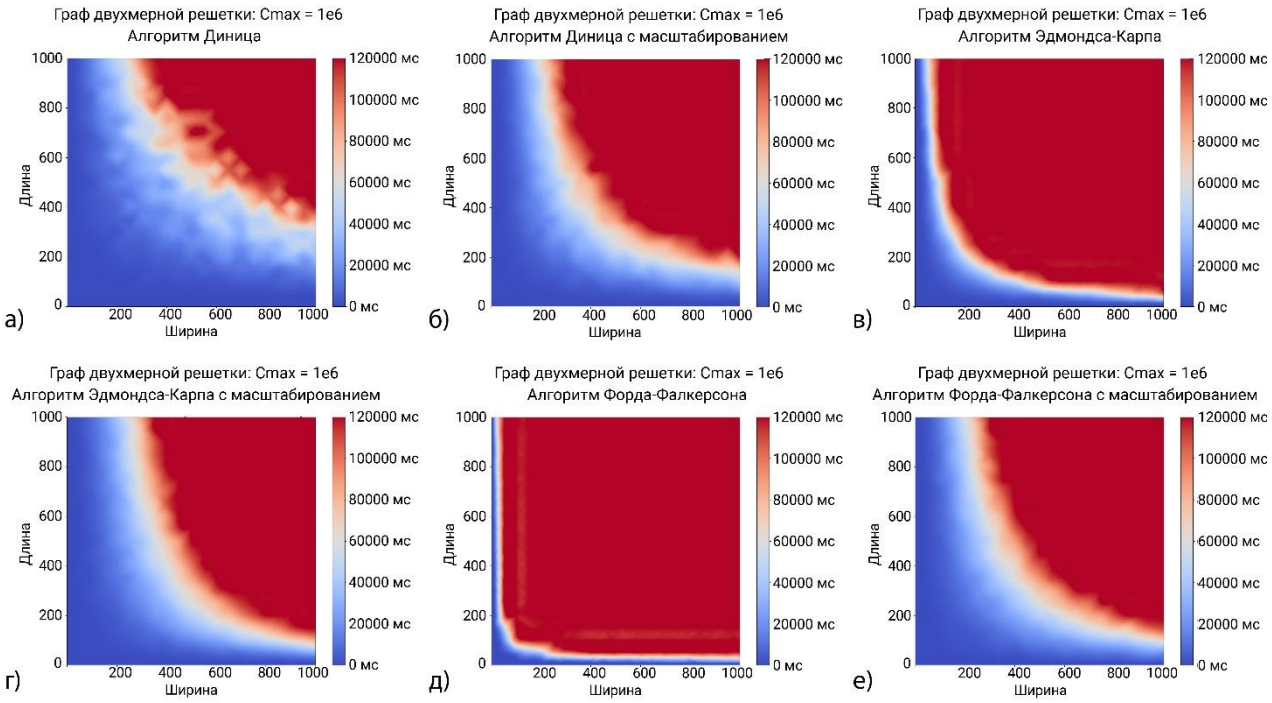

Рисунок 3.18 – Результаты тестирования
на графе двухмерной решётки с $C_{max} = 10^6$

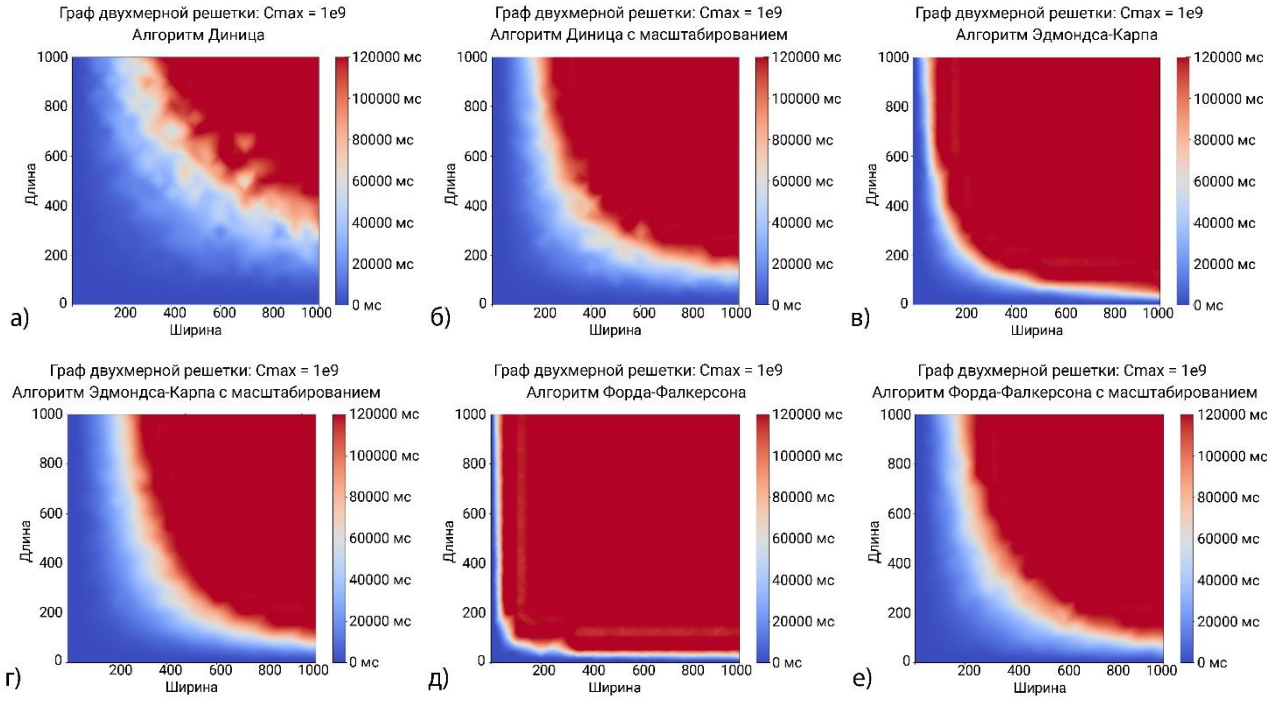

Рисунок 3.19 – Результаты тестирования
на графе двухмерной решётки с $C_{max} = 10^9$

При увеличении максимальной пропускной способности видно, что эффективность классического алгоритма Диница осталась прежней (рисунки 3.18-3.19 (а)), а у всех алгоритмов, использующих масштабирование потока, скорость работы снизилась (рисунки 3.18-3.19 (б, г, е)). Алгоритм Эдмондса-Карпа хоть работал не очень быстро, но также как и алгоритм Диница, показывает стабильный результат (рисунки 3.18-3.19 (в)).

Проведем тестирование алгоритмов поиска максимального потока на следующем виде графа: графе трехмерной решётки. В нем количество ребер также не сильно превышает количество вершин, а значит все предположения, сделанные относительно разреженных графов верны и тут. Результаты тестирования представлены на рисунках 3.20-3.22.

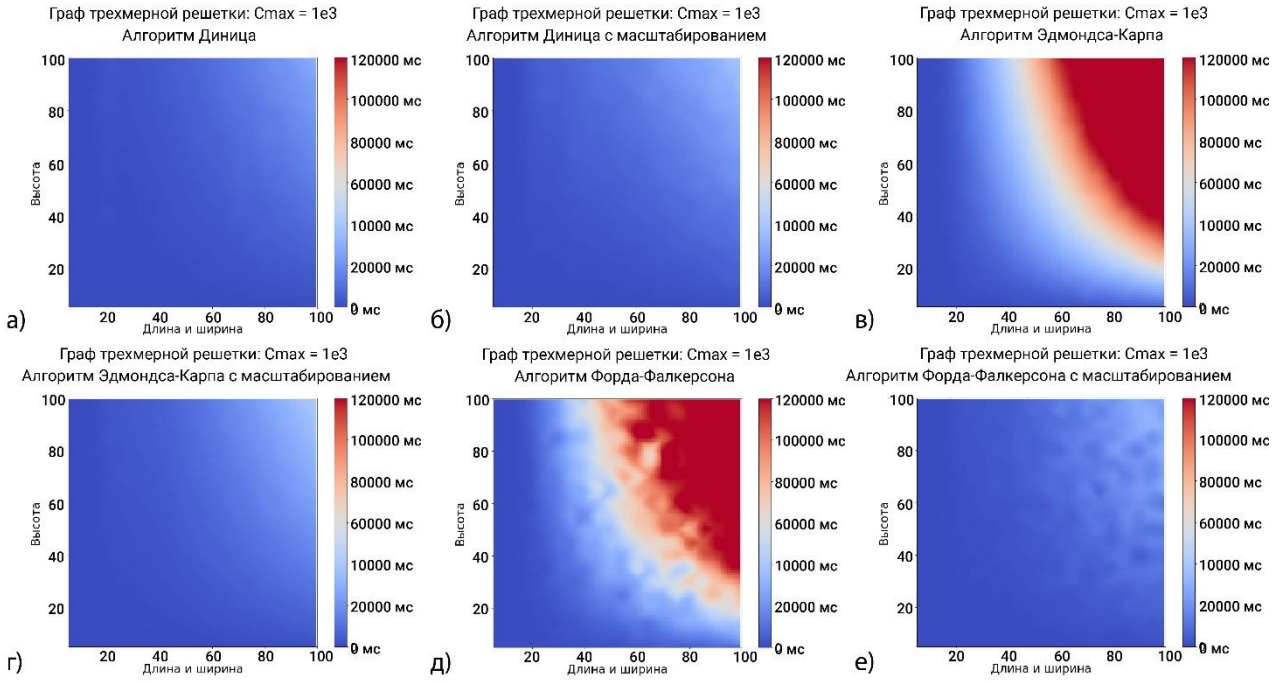

Рисунок 3.20 – Результаты тестирования
на графе трехмерной решётки с $C_{max} = 10^3$

Как можно видеть по рисунку 3.20, при малой максимальной пропускной способности ребра классический алгоритм Диница и все варианты алгоритмов с масштабированием потока показали достаточно быстрое время работы (рисунок 3.20 (а, б, г, е)). Также в некоторых случаях скорость работы

алгоритмов Эдмондса-Карпа и Форда-Фалкерсона (рисунок 3.20 (в, д)) не уступала более «быстрым» алгоритмам.

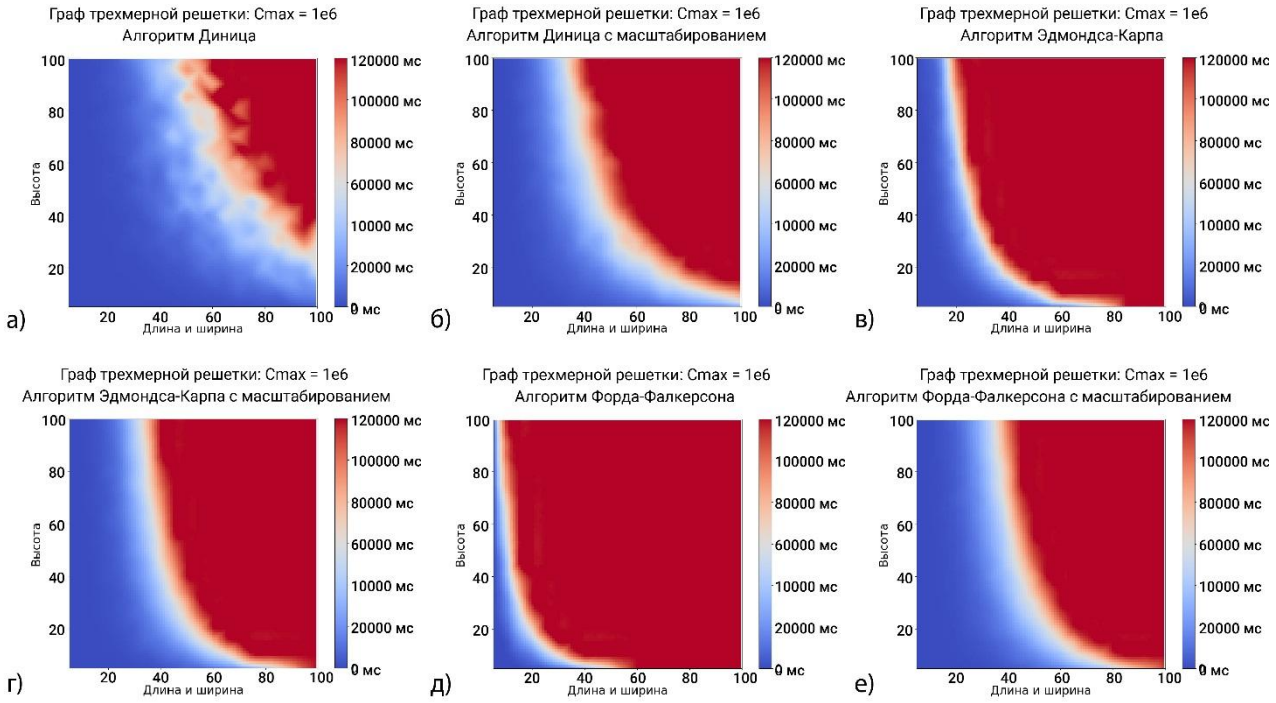

Рисунок 3.21 – Результаты тестирования
на графе трехмерной решётки с $C_{max} = 10^6$

Однако, при увеличении максимальной пропускной способности ситуация сильно изменилась. На больших графах все алгоритмы начали терять свою эффективность. Но все-таки, среди всех рассматриваемых алгоритмов алгоритм Диница показывает наилучший результат (рисунок 3.21 (а)). Далее следуют алгоритмы с масштабированием потока (рисунок 3.21 (б, г, е)). После алгоритм Эдмондса-Карпа и Форда-Фалкерсона (рисунок 3.21 (в, д)). Также стоит отметить, что из-за того, что представить график функции трех переменных достаточно сложно, длина и ширина графа решётки равны. Отсюда следует, что зависимость от параметра длины и ширины выше, чем от высоты, что и подтверждают результаты тестирования.

При еще большем увеличении максимальной пропускной способности ребра эффективность алгоритмов осталась примерно такой же.

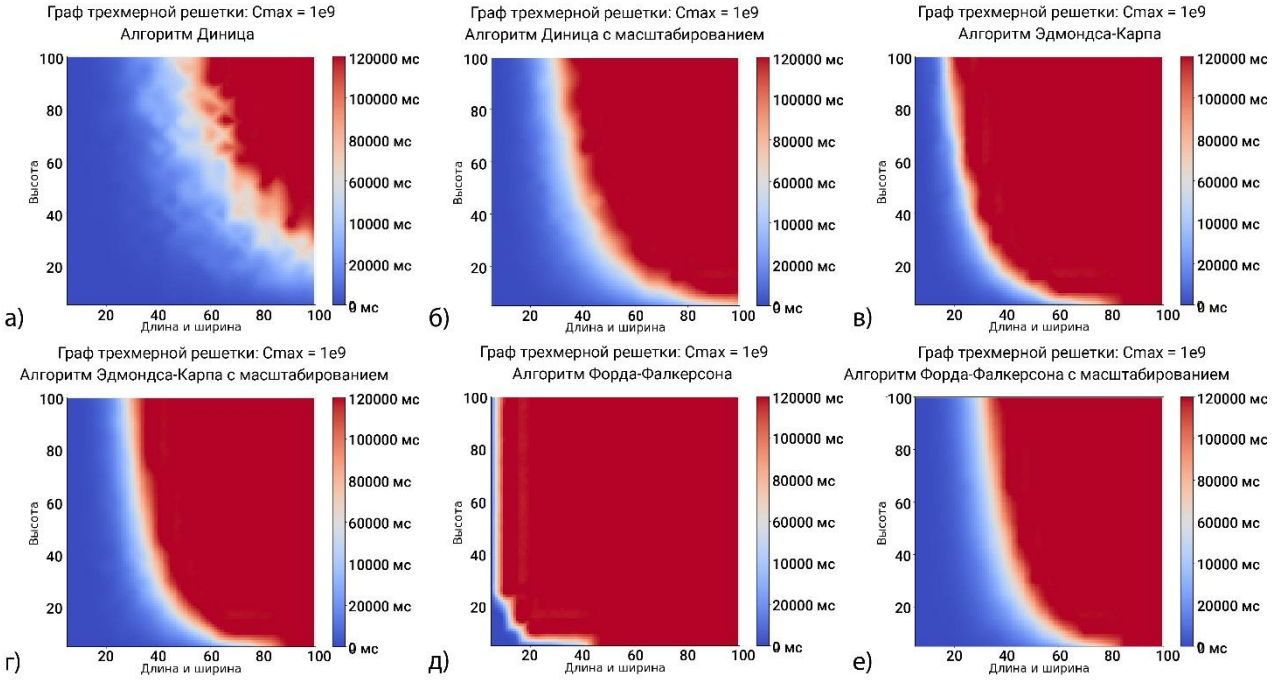

Рисунок 3.22 – Результаты тестирования
на графе трехмерной решётки с $C_{max} = 10^9$

Таким образом, что на графе двухмерной решётки, что на графе трехмерной решётки на практике показал классический алгоритм Диница. Однако, асимптотически он не является самым быстрым для данного вида графа. Далее по эффективности следуют все варианты алгоритмов с масштабированием потока. Они показали примерно одинаковый результат, что и ожидалось. Но предполагалось, что этот результат будет лучшим. Алгоритмы Эдмондса-Карпа и Форда-Фалкерсона показывают неплохой результат только на малых размерах графов, что еще раз подтверждает их сильную зависимость от размера графа.

3.3 Выводы по тестированию

По результатам тестирования на рассматриваемых видах графов классический алгоритм Форда-Фалкерсона, как и ожидалось, показал самое медленное время работы среди рассматриваемых алгоритмов. Прежде всего это подтверждает его сильную зависимость от величины ответа. Поэтому

использовать данный алгоритм для поиска максимального потока на больших сетях или на сетях с большими пропускными способностями не рекомендуется.

Следующим по эффективности следует алгоритм Эдмондса-Карпа. По результатам тестирования он показал достаточно эффективное время работы на случайных графах и ациклических разреженных графах. Видимо, алгоритм достаточно быстро работает на графах, которые имеют случайную структуру. Однако, на всех остальных рассматриваемых видах графов алгоритм Эдмондса-Карпа уже не был столь эффективен.

Далее следуют алгоритмы Форда-Фалкерсона и Эдмондса-Карпа с использованием масштабирования потока. Несмотря на то, что они имеют одинаковую асимптотическую сложность, алгоритм Эдмондса-Карпа оказался чуть более эффективнее. Это, скорее всего, связано с тем, что классический алгоритм Эдмондса-Карпа работает быстрее классического алгоритма Форда-Фалкерсона, и метод масштабирования потока, хоть и ускоряет данные алгоритмы, но не делает их одинаково эффективными.

Вариант алгоритма Диница с масштабированием потока на двудольных и полных графах оказался более эффективным, чем алгоритмы Форда-Фалкерсона и Эдмондса-Карпа с масштабированием. Это объясняется тем, что в данных видах графов число ребер много больше числа вершин. Но на всех остальных рассматриваемых видах графов все варианты алгоритмов, использующие масштабирование потока показали примерно одинаковый результат, что подтверждается их одинаковой асимптотикой на этих графах.

Самым эффективным среди рассматриваемых алгоритмов поиска максимального потока оказался классический алгоритм Диница. На всех рассматриваемых видах графов он показал самое быстрое время работы. И хотя асимптотически он не является самым эффективным, на практике он работает достаточно хорошо. Однако, не стоит забывать, что пропускные способности для всех графов генерировались случайным образом. Возможно, алгоритм Диница имеет высокую устойчивость к такому виду генерирования сетей.

Также, возможно существуют другие виды графов, на которых классический алгоритм Диница не будет показывать самый лучший результат.

ЗАКЛЮЧЕНИЕ

В ходе выполнения бакалаврской работы была изучена задача о максимальном потоке и ее применения. Она актуальна как в классических проблемах, в которых необходимо переслать что-то из одной точки в другую, так и в весьма необычных, таких как сегментация изображений или нахождение циркуляции. Поэтому эффективное решение задачи о максимальном потоке имеет большое значение.

Были изучены алгоритмы Форда-Фалкерсона, Эдмондса-Карпа, Диница, позволяющие решать задачу поиска максимального потока; а также метод масштабирования потока, улучшающий асимптотическую сложность всех рассматриваемых алгоритмов. Был проведен анализ корректности и сложности каждого алгоритма.

Далее были сгенерированы исходные данные для тестирования алгоритмов и произведено само тестирование.

По результатам тестирования и теоретического анализа алгоритмов был проведен их сравнительный анализ. На практике самым эффективным алгоритмом оказался классический алгоритм Диница, хотя асимптотически наиболее быстрым является вариант данного алгоритма с использованием масштабирования потока. Все же остальные алгоритмы относительно друг друга сработали так, как и ожидалось.

Данный результат показывает, что при решении какой-либо реальной задачи, которая решается с помощью поиска максимального потока, недостаточно выбрать алгоритм, который в теории является наиболее эффективным, так как на практике он может показать не тот результат, который ожидался. Лучше провести тестирование алгоритмов на данных максимально приближенным к реальным и выбрать нужный алгоритм по результатам данного тестирования.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ. — М.: «Вильямс», 2017. — 1328 с.
2. Асанов, М.О. Дискретная математика: Графы, матроиды, алгоритмы / М.О. Асанов, В.А. Баранский, В.В. Расин. – СПб. : Лань, 2017. – 368 с.
3. Harris, T.E. Fundamentals of a Method for Evaluating Rail Net Capacities / T.E. Harris, F.S. Ross // 1955.
4. Ford, L.R. Maximal flow through a network / L.R. Ford, D.R. Fulkerson // Canadian Journal of Mathematics. – 1956.
5. Ford, L.R. Flows in Networks / L.R. Ford, D.R. Fulkerson. – Princeton : Princeton University Press, 1962. – 332 с.
6. Майника, Э. Алгоритмы оптимизации на сетях и графах / Э. Майника. – М. : Мир, 2013. – 324 с.
7. Malhotra, V.M. An O(|V|3) algorithm for finding maximum flows in networks / V.M. Malhotra, M. Pramodh Kumar, S.N. Maheshwari // Information Processing Letters. – 1978.
8. Schrijver, A. On the history of the transportation and maximum flow problems / A. Schrijver // Mathematical Programming. – 2014. – С. 437-445
9. Пападимитриу Х. Комбинаторная оптимизация. Алгоритмы и сложность. / Х.Пападимитриу, К. Стайглиц, пер. с англ. – М.: Мир, 2017 – 512 с.
10. Dinitz, E.A. Algorithm for solution of a problem of maximum flow in a network with power estimation / E.A. Dinitz // Doklady Akademii Nauk SSSR. – 1970. – Т. 194, № 4. – С. 1277–1280
11. Juan, Olivier Capacity Scaling for Graph Cuts in Vision / Olivier Juan, Yuri Boykov // IEEE 11th International Conference on Computer Vision. – 2007.
12. Sleator, D.D. A data structure for dynamic trees / D.D. Sleator, R.E. Tarjan // Journal of Computer and System Sciences. – 2014. – . – Т. 26, № 3. – С. 362-391

13. Акулич, Н.А. Математическое программирование в примерах и задачах / Н.А. Акулич. – М. : Высшая школа, 2015.

14. Ахо А., Структуры данных и алгоритмы. / А. Ахо, Дж. Ульман, Дж. Хоп-крофт, пер. с англ. – М: Вильямс, 2015 г – 384 с.

15. Павловская Т. А., Щупак Ю. А. "C/C++. Структурное и объектно-ориентированное программирование". — СПб: 2017. — 352 с.

16. Скиена С. С., Олимпиадные задачи по программированию. Руководство поподготовке к соревнованиям. / С.С. Скиена, М.А. Ревилла, пер. с англ. – М.:Кудиц-Образ, 2015 – 416 с.

17. Роберт Седжвик. Фундаментальные алгоритмы на C. Алгоритмы на графах = Algorithms in C. Graph Algorithms. — СПб.: ДиаСофтЮП, 2018. — 480 с.

18. Ahuja R.K. Network Flows: Theory, Algorithms, and Applications / R.K. Ahuja,T.L. Magnanti, J.B. Orlin – USA: Prentice Hall, 2017 – 864p.

19. Лутц, М. Изучаем Python / М. Лутц. – СПб. : Символ-Плюс, 2014. – 1280 с.

20. Шабанов П.А. Научная графика в python [Электронный ресурс]. – Режим доступа: https://github.com/whitehorn/Scientific_graphics_in_python. – Дата доступа: 25.04.2019.

21. Задача о максимальном потоке [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Задача_о_максимальном_потоке. – Дата доступа: 19.03.2019.

22. Теория графов - Викиконспекты [Электронный ресурс]. – Режим доступа: http://neerc.ifmo.ru/wiki/index.php?title=Теория_графов. – Дата доступа: 23.03.2019.