Санкт–Петербургский государственный университет

ФОМИН Никита Павлович

Выпускная квалификационная работа

Децентрализованная многокритериальная маршрутизация

Уровень образования: бакалавриат
Направление 02.03.01 «Математика и компьютерные науки»
Основная образовательная программа СВ.5152.2020
«Математика, алгоритмы и анализ данных»

Научный руководитель:
Доцент, Факультет математики и компьютерных наук, СПбГУ
к.ф.-м.н., Яковлев Константин Сергеевич

Рецензент:
Главный научный сотрудник, Федеральный исследовательский центр «Информатика и управление» Российской Академии Наук
д.т.н, Хачумов Вячеслав Михайлович

Санкт-Петербург
2024 г.

Содержание

Введение

Компьютерные сети стали неотъемлемой частью современного мира. Пользователи ожидают от них высоких пропускных способностей, стабильности соединения и предсказуемости поведения. Эти параметры сети зависят от множества факторов. Одним из ключевых является выбор алгоритма маршрутизации пакетов. Алгоритм работает на устройствах (маршрутизаторах), составляющих инфраструктуру сети и именно он определяет по какому маршруту данные попадут от одного пользователя к другому.

Пакет данных, отправленный по сети Интернет, в среднем совершает 8 переходов, чтобы добраться до конечной точки (см. рисунок №1). Каждый такой переход добавляет задержку и увеличивает риск возникновения ошибок в передаваемых данных. Поэтому важно, чтобы из всех доступных маршрутов передачи был выбран оптимальный.

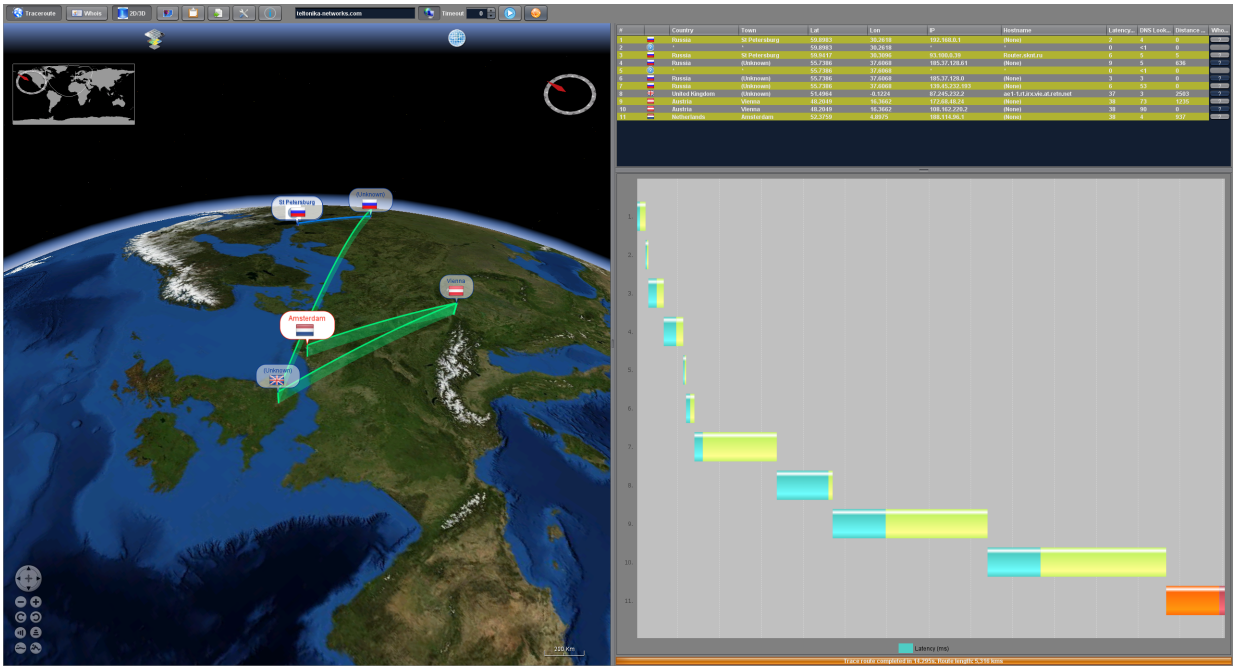

Рис. 1: Интерфейс программы Open Visual Traceroute. Запрос к teltonica-networks.com прошёл 11 узлов сети, прежде чем попал на нужный сервер

Протоколы маршрутизации используют различные показатели соединения для определения наилучшего маршрута передачи данных. Но даже современные протоколы, способные учитывать несколько характеристик сети, ограничены алгоритмами поиска кратчайших путей, применяемыми в них. В современных решениях используются алгоритмы Дейкстры [2] или Беллмана-Форда [3], которые работают с графами, где каждое

ребро описывается лишь одной величиной. В результате метрика соединений представляется как взвешенная сумма учитываемых параметров. Такая упрощенная модель может снижать точность выбора оптимального маршрута в сложных сетевых сценариях.

Появление оптимизированных версий алгоритмов многокритериального поиска кратчайшего пути открывает перспективы для их использования в протоколах маршрутизации. Целью данной ВКР является рассмотрение применимости современных алгоритмов MOSP (Multiobjective Shortest Path) для решения задачи децентрализованной маршрутизации.

Для достижения данной цели необходимо выполнить следующие задачи:

1. Провести анализ актуальных научных исследований по темам многокритериальных алгоритмов поиска кратчайшего пути, а также маршрутизации в компьютерных сетях.
2. Реализовать State of the Art MOSP алгоритмы.
3. Разработать методы решения задачи децентрализованной многокритериальной маршрутизации на основе исследованных алгоритмов.
4. Подготовить среду для тестирования и провести бенчмаркинг полученных методов.

Реализация алгоритмов в рамках данной работы предполагает создание открытого репозитория, который будет доступен другим исследователям для анализа SOTA алгоритмов. Кроме того, мы предоставляем среду для создания наборов данных для тестирования алгоритмов многокритериальной оптимизации.

1. Обзор

Данная глава состоит из двух смысловых частей. В первой даётся краткий обзор важных для понимания задачи принципов работы маршрутизаторов. Вторая часть является основной и посвящена неформальной постановке задачи, а также методам, применяемым для решения задачи поиска кратчайших путей.

1.1. Представление сети в маршрутизаторе

Каждый маршрутизатор основывает свою работу на заложенном в нём протоколе — наборе правил, определяющих, как маршрутизаторы идентифицируют и пересылают пакеты данных по сети. Существует множество протоколов маршрутизации, каждый из которых имеет свои преимущества и недостатки. Примерную иерархию протоколов можно увидеть на рисунке №2.

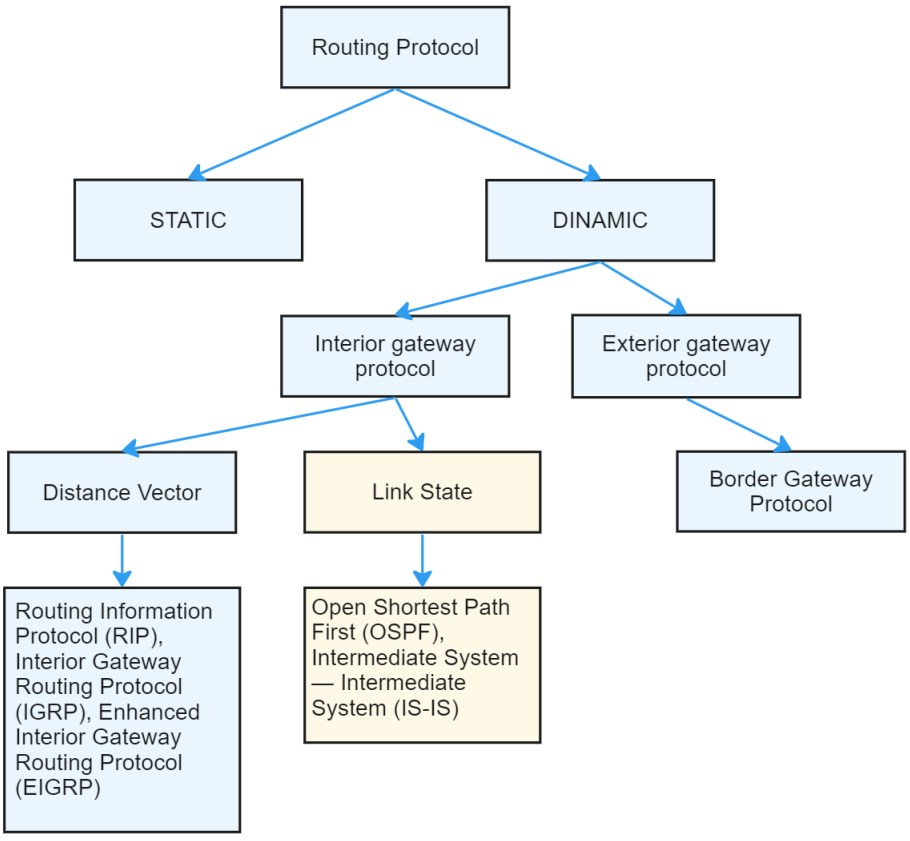

Рис. 2: Классификация алгоритмов маршрутизации

В данной работе мы будем рассматривать link-state протоколы маршрутизации [6]. Примерами таких протоколов служат OSPF [4] и IS-IS [5]. Link-state решения основываются на обмене информацией о состоянии связей между всеми узлами в сети. Каждый маршрутизатор отслеживает соединения со своими непосредственными соседями и распространяет эту информацию всем остальным участникам сети. Маршрутизаторы используют эти данные для формирования в своей внутренней памяти полной карты сети, которая затем используется для вычисления наилучших маршрутов передачи пакетов информации.

Type	*Dest*	*Area*	*Path Type*	*Cost*	*Next Hop(s)*	*Advertising Router(s)*
N	N1	0	1	10	RT3	*
N	N2	0	1	10	RT3	*
N	N3	0	1	7	RT3	*
N	N4	0	1	8	RT3	*
N	Ib	0	1	7	*	*
N	Ia	0	1	12	RT10	*
N	N6	0	1	8	RT10	*
N	N7	0	1	12	RT10	*
N	N8	0	1	10	RT10	*
N	N9	0	1	11	RT10	*
N	N10	0	1	13	RT10	*
N	N11	0	1	14	RT10	*
N	H1	0	1	21	RT10	*
ASBR	RT5	0	1	6	RT5	*
ASBR	RT7	0	1	8	RT10	*
N	N12	*	3	10	RT10	RT7
N	N13	*	3	14	RT5	RT5
N	N14	*	3	14	RT5	RT5
N	N15	*	3	17	RT10	RT7

Рис. 3: Пример таблицы маршрутизации, построенной в ходе работы маршрутизатора, использующего протокол OSPF

После получения полной карты сети маршрутизатор запускает алгоритм поиска кратчайшего пути до всех известных узлов. Результаты этого поиска используются для построения таблицы маршрутизации, которая определяет, какому из соседних узлов передавать входящие пакеты данных. Формат таблицы может варьироваться, но она всегда содержит информацию, необходимую для установления связей вида "адресат пакета: следующий узел передачи". Примером такой таблицы маршрутизации может служить результат работы маршрутизатора, использующего протокол OSPF (рисунок №3).

1.2. Неформальная постановка задачи

Задачей является сопоставление каждому маршрутизатору в сети алгоритма, следуя которому устройства смогут на основании имеющейся карты сети построить собственную таблицу маршрутизации. Также в нашей задаче действуют следующие ограничения:

1. **Децентрализованность:** маршрутизаторы не могут обмениваться информацией о выбранных маршрутах передачи пакетов
2. **Ограниченность:** для любого пакета, отправленного в сети, суммарная стоимость пути от отправителя до адресата не должна превышать заданной константы
3. **Детерминированность:** маршрутизатор-отправитель пакета может однозначно установить по какому маршруту пройдет пакет

1.3. Алгоритмы, использующиеся современными маршрутизаторами

Современные маршрутизаторы в качестве алгоритма поиска кратчайшего пути используют алгоритм Дейкстры. Сеть представляется как взвешенный граф, в котором вершины — это узлы, а рёбра — соединения между ними.

Алгоритм Дейкстры является методом поиска кратчайшего пути от одной из вершин взвешенного графа ко всем остальным. В отличие от того, чтобы искать самый короткий путь перебором всех возможных вариантов, данный алгоритм использует пошаговое построение маршрута. Его основная идея заключается в поддержании двух множеств вершин: тех, которые уже рассмотрены, и тех, которые еще предстоит посетить. На каждом шаге алгоритм выбирает вершину из множества еще не рассмотренных, причём такую, расстояние до которой минимально. Затем происходит перерасчет кратчайших путей до соседей данной вершины, и сама вершина добавляется в множество уже рассмотренных.

Данный алгоритм прост в реализации и хорошо подходит для компьютерных сетей, ведь в них рёбра не могут иметь отрицательного веса. Руководства по реализации протоколов OSPF и IS-IS предполагают использование именно алгоритма Дейкстры [5].

1.4. Алгоритмы поиска кратчайшего пути для многокритериального случая

Алгоритм Дейкстры, применяемый в текущих реализациях маршрутизаторов, не предусматривает наличие рёбер с несколькими стоимостями. Это означает, что в случаях, когда необходимо учитывать несколько характеристик соединения, приходится выбирать между двумя подходами: либо объединять все характеристики в единую метрику, либо запускать алгоритм Дейкстры для каждого отдельного веса ребра.

Однако возникает вопрос: как сохранить точность оптимизации маршрута по всем используемым критериям? Этот вопрос приводит нас к задаче многокритериального поиска кратчайшего пути. Последние исследования в области алгоритмов, решающих данную задачу, предлагают оптимизированные методы, которые могут быть полезны для маршрутизации в сети. Рассмотрим последние достижения в этой области.

Базовый алгоритм A*

Все алгоритмы, решающие задачу многокритериального поиска кратчайшего пути, берут своё начало в алгоритме A* [6]. Поэтому для лучшего понимания внутреннего устройства и оптимизаций, применяемых в State of the Art алгоритмах, важно начать с основ.

A* - это модифицированный алгоритм Дейкстры, специально адаптированный для эффективного поиска оптимального маршрута между двумя вершинами. Этот алгоритм использует дерево поиска для хранения множества известных путей, а также эвристическую функцию оценки, которая используется для прогнозирования стоимости достижения цели из каждой вершины. A* рассматривает вершины с наименьшей суммой стоимости пути от начальной вершины до текущей, добавляя к этой сумме эвристическую оценку стоимости до конечной вершины.

Как видно на рисунке №4, такая простая модификация существенно снижает количество вершин, рассмотренных в ходе работы алгоритма поиска. А значит снижается время выполнения алгоритма и его требования к используемой памяти.

MOA*, NAMOA*

Задача многокритериального поиска отличается от стандартной скалярной версии тем, что теперь результат вычислений не является единственным ответом, представляющим самый короткий путь, а целым мно-

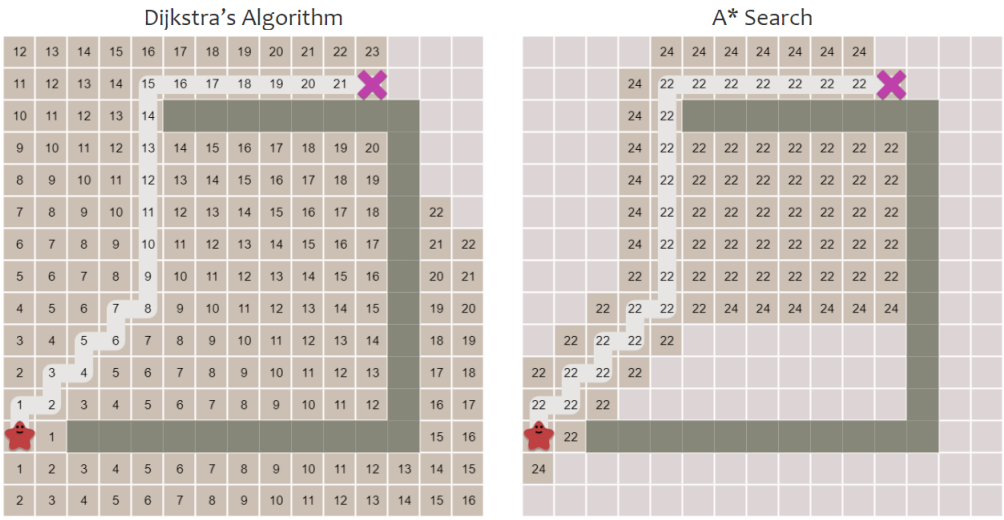

Рис. 4: Сравнение количества вершин, рассмотренных в ходе работы алгоритма Дейкстры и A*

жеством Парето-оптимальных решений — набором всех недоминируемых вариантов. Решение A доминирует над решением B, если (1) стоимость решения A для каждого из критериев не хуже соответствующих стоимостей решения B, и (2) решение A превосходит по крайней мере по одному критерию решение B.

Алгоритм Multiobjective A* (MOA*) [7] и его улучшенная версия New Approach Multiobjective A* (NAMOA*) [8] являются базовыми для решения задачи многокритериального поиска. Их основные отличия от A* заключаются в принципах хранения и раскрытия состояний поиска. Каждой вершине s исходного графа сопоставляется два множества состояний - $G_{cl}(s)$ и $G_{op}(s)$. Так как каждое состояние поиска отражает пройденный путь и его стоимость, то данные множества хранят информацию о стоимости уже рассмотренных путей, ведущих в вершину, а также путей, которые были обнаружены, но ещё не были раскрыты алгоритмом. При раскрытии состояния поиска происходит проверка недоминируемости найденного пути. Время выполнения операции проверки доминирования линейно зависит от размера множеств, которые используются алгоритмом, что значительно замедляет общее время выполнения.

Результатом работы алгоритма является Парето-множество решений — множество, содержащее все недоминируемые решения (рисунок №5). Под решением в данном случае предполагается стоимость найденного пути и указание на состояние дерева поиска. По состоянию в дальнейшем возможно восстановить весь путь.

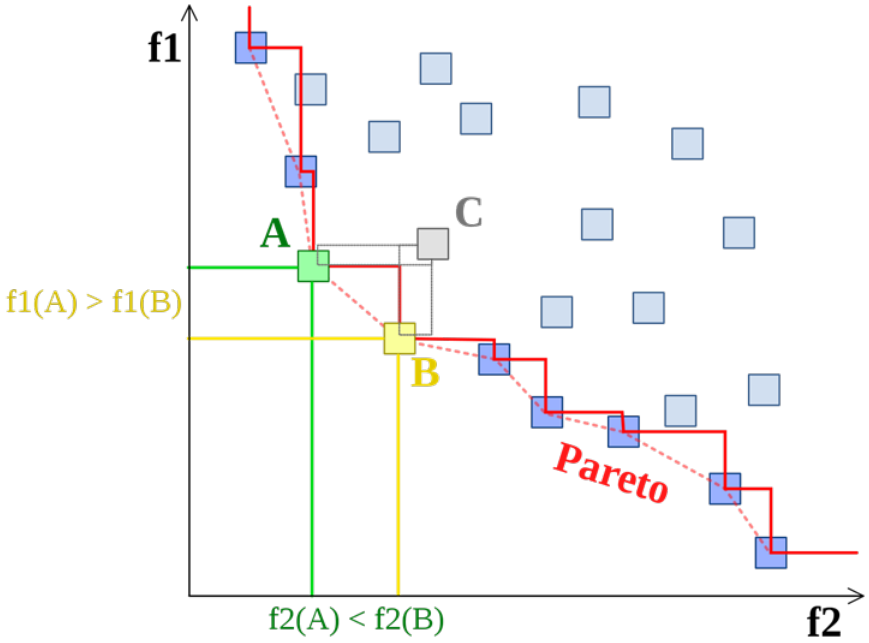

Рис. 5: Парето-множество. Яркими цветами обозначены недоминируемые решения, а менее насыщенными - решения, не включенные в конечное множество.

BOA*, BOD

Алгоритм MOA* подвергался множественным оптимизациям, основная задача которых заключалась в улучшении асимптотики проверки доминируемости при нахождении нового пути. Больших результатов удалось добиться при рассмотрении частного случая задачи, где количество критериев поиска ограничивается двумя. Так, Carlos Hernández и другие в своей статье [9] предложили модифицированную версию алгоритма NAMOA*, в которой значительно улучшена асимптотика проверки доминируемости при раскрытии состояний поиска. Линейное время выполнения проверок в новом методе сведено к константному. Предложенный алгоритм Bi-objective A* (BOA*) стал новым State of the Art решением частного случая задачи многокритериального поиска кратчайшего пути.

Также авторами статьи о BOA* был предложен Bi-Objective Dijkstra's (BOD) алгоритм [10]. BOD использует идеи BOA* для решения задачи поиска кратчайшего пути от выбранной начальной вершины графа до всех остальных. В BOD все эвристические оценки заменены на нулевые, а также отсутствует целевая вершина поиска.

2. Формальная постановка задачи

В данной главе представлена формальная постановка задачи поиска кратчайшего пути. Также вводятся определения задачи децентрализованной маршрутизации и критериев оценки её решения — ограниченности и детерминированности.

2.1. Задача поиска кратчайшего пути

Один критерий поиска

Пусть задан направленный взвешенный граф $G = (S, E, c)$, где S — конечное множество вершин, $E \subseteq S \times S$ — конечное множество рёбер, и $c : E \to \mathbb{R}^q_{>0}$ — функция стоимости, сопоставляющая каждому ребру положительное число. Путь между двумя вершинами s_k и s_m — это последовательность вершин $\pi = (s_1, s_2, \ldots, s_n)$, такая что $s_1 = s_k$, $s_n = s_m$ и для любого $i \in (1, \ldots, n-1)$, $(s_i, s_{i+1}) \in E$. Стоимость пути определяется как $c(\pi) = \sum_{i=1}^{n-1} c(s_i, s_{i+1})$. Задачей поиска кратчайшего пути является нахождение пути с минимальной стоимостью из заданной начальной вершины $s_{start} \in S$ в целевую вершину поиска $s_{goal} \in S$. Таким образом, задача описывается множеством $(S, E, c, s_{start}, s_{goal})$.

Многокритериальный случай

Введём определение отношения доминирования. Отношение доминирования вектора $\vec{p}$ над вектором $\vec{q}$ записывается как $\vec{p} \prec \vec{q}$ и обозначает, что:

$$\forall \vec{p},\ \vec{q} \in \mathbb{R}^q :\ \vec{p} \prec \vec{q} \Leftrightarrow \forall i\ \ p_i \leq q_i \wedge \vec{p} \neq \vec{q}$$

В данной записи p_i обозначает i-ый элемент вектора стоимости $\vec{p}$.

В многокритериальном случае функция стоимости $c : E \to \mathbb{R}^q_{>0}$ заменяется на $\vec{c} : E \to \mathbb{R}^q_{>0}$, то есть теперь каждому ребру графа сопоставляется вектор положительных значений. Меняется и само определение задачи — теперь необходимо найти множество **недоминируемых путей** (так называемого Парето-оптимального множества) из заданной начальной вершины $s_{start} \in S$ в целевую вершину поиска $s_{goal} \in S$.

Парето-оптимальное множество решений для состояния $s \in S$, обозначаемое $sols(s)$, содержит все пути π, начинающиеся в s_{start} и идущие в s, причём такие, что выполняется условие: для любого другого пути π‘ из s_{start} в s: π‘ $\not\prec \pi$. Таким образом, множество содержит все недоминируемые пути из s_{start} в s.

One to Many

Обе формулировки задачи поиска кратчайшего пути могут быть расширены для случая, когда необходимо найти кратчайший путь (недоминируемые пути) от начальной вершины до всех остальных. Так, многокритериальный случай задачи в данной постановке задаётся множеством $(\underbrace{S, E, \vec{c}}_{\text{G}}, s_{start})$ - без указания целевой вершины поиска.

2.2. Частная задача маршрутизации

Задан направленный взвешенный граф $G = (S, E, \vec{c})$, вершины которого будем называть узлами, а сам граф - сетью. На нём выбрана вершина s. Множество всех вершин, доступных из вершины s обозначается как $Succ(s) = \{t \in S \mid (s, t) \in E\}$. Пакетом данных будем называть пару из узла-отправителя данных и узла-адресата: $p = (s_{start}, s_{target})$. Задача маршрутизации заключается в нахождении функции $nextHop : S \times S \to Succ(s) \subseteq S$, которая сопоставляет любому входному пакету данных p допустимого соседа узла s.

2.3. Общая задача маршрутизации

Задана сеть $G = (S, E, \vec{c})$, $|S| = N$. Общей задачей маршрутизации является сопоставление каждому узлу сети s алгоритма решения частной задачи маршрутизации (нахождения функции $nextHop$). Множество, полученное в результате выполнения сопоставленных вершинам алгоритмов, будем называть **полным множеством правил маршрутизации**. $rules = \{nextHop_{s_1}, nextHop_{s_2}, ..., nextHop_{s_N}\}$, где $nextHop_{s_i}$ — решение частной задачи маршрутизации для узла s_i.

Следом пакета будем называть путь π, пройденный пакетом данных $p = (s_m, s_k)$ с учётом полного множества правил маршрутизации. Первой вершиной следа обязательно является узел-отправитель пакета и каждый следующий узел определяется как $s_{i+1} = nextHop_{s_i}(p)$. Операцию нахождения следа пакета будем называть **трассировкой**: $trace(p, rules)$

Введём обозначения критериев оценки решения общей задачи маршрутизации:

1. **Ограниченность.** Для любой пары вершин (s_{start}, s_{target}), если в исходном графе существует путь из s_{start} в s_{target}, стоимость которого не превышает заданной константы $\vec{C}$, то след π пакета $p = (s_{start}, s_{target})$:

- Начинается в s_{start} и заканчивается в s_{target}.
- По стоимости не превышает $\vec{C}$. Т.е $\vec{c}(\pi) = \sum_{i=1}^{n-1} \vec{c}(s_i, s_{i+1}) <= \vec{C}$

2. **Детерминированность.** Для любого пакета $p = (s_{start}, s_{target})$ трассировка пакета $trace(p, rules)$ гарантированно выдаёт один и тот же результат.

Экземпляром задачи общей маршрутизации будем называть множество (G, C_1, C_2), где G - граф (сеть), C_1, C_2 - константы, ограничивающие суммарную стоимость пути, по которому может пройти пакет.

Также стоит отметить, что в работе рассматриваются две постановки общей задачи маршрутизации:

1. $nextHop$ может зависеть как от отправителя, так и от адресата пакета
2. $nextHop$ должна зависеть только от адресата пакета

3. Методы

В данной главе подробно описываются методы решения задачи многокритериального поиска и маршрутизации, которые были использованы в ВКР.

В начале рассматриваются алгоритмы, решающие задачу многокритериального поиска, где Парето-оптимальное множество решений необходимо найти только для единственной вершины. Далее разбирается версия задачи, в которой множества решений должны быть построены для всех вершин. Данные алгоритмы описаны для частного случая задачи, когда количество критериев равно двум, так как именно в такой постановке возможны наиболее существенные оптимизации.

Методы решения задачи маршрутизации, не являясь привязанными к конкретной задаче поиска, описываются в конце главы.

3.1. Базовые принципы работы алгоритмов поиска кратчайшего пути

Алгоритмы, рассмотренные в данной главе, берут своё начало в базовом алгоритме A*. Этот алгоритм представляет собой эффективный метод решения задачи поиска кратчайшего пути между двумя вершинами во взвешенном графе. Исходный A* применяется для графов, где веса представлены скалярными значениями. Однако принципы, на которых строится его работа, послужили основой для многокритериальных решений.

Алгоритм A* в ходе выполнения строит дерево поиска. Каждая вершина этого дерева, называемая **состоянием**, ассоциирована с конкретной вершиной s исходного графа и хранит в себе следующую информацию:

- Ссылка на саму вершину s
- g-значение: стоимость пути от начальной вершины до текущей.
- h-значние: эвристическая оценка стоимости пути от текущей вершины до конечной.
- f-значение: сумма g и h, представляющая оценку общей стоимости пути через текущую вершину.
- Ссылка на родительскую вершину: для восстановления кратчайшего пути после завершения поиска.

Algorithm 1: Стандартный алгоритм A*

Input: A search problem$(S, E, c, s_{start}, s_{goal})$ and a consistent heuristic function h
Output: Path $\pi = (s_{start}, ..., s_{goal})$ if it exists

```
1  Initialize Open and add (s_start, 0, h(s_start)) to it
2  Closed := ∅
3  while Open ≠ ∅ do
4      Remove a node (s, g_s, f_s) from Open with the lexicographically smallest f-value of
         all nodes in Open
5      if s == s_goal then
6          return makePath(s)
7      add s to Closed
8      for t ∈ Succ(s) do
9          if node corresponding to t have not been generated then
10             g_t ← g_s + c(s, t)
11             f_t ← g_t + h(t)
12             parent(t) = s
13             add (t, g_t, f_t) to Open
14         else
15             if g_t > g_s + c(s, t) then
16                 g_t ← g_s + c(s, t)
17                 update t node information in Open
18 return Path not found
```

В ходе выполнения алгоритм последовательно заполняет дерево поиска, поддерживая два списка: $Open$ и $Closed$. На каждом этапе основного цикла выполняется операция **раскрытия** состояния из списка $Open$: генерируются соседи текущей вершины, рассчитываются их g, h, f значения и затем соседи добавляются в $Open$, а раскрытое состояние попадает в список $Closed$.

Сгенерированное состояние может попасть в список $Open$ только в том случае, если оно еще не было рассмотрено, то есть не находится ни в списке $Open$, ни в списке $Closed$. Если новый найденный путь до вершины оказался более выгодным, чем уже рассчитанный, то значения g, h и f состояния в списке $Open$ обновляются для дальнейшего рассмотрения.

Алгоритм начинает свою работу с добавления стартового состояния в $Open$ и продолжает выполнение до того момента, пока $Open$ не опустеет или не будет найден путь до целевой вершины.

Выбирая очередное состояние для раскрытия из $Open$, алгоритм руководствуется f-значением. В обработку попадает состояние, имеющее минимальное f-значение среди всех остальных.

Важную роль играет выбор эвристической функции. Пусть $h^*(s)$ — реальная оптимальная стоимость пути от вершины s до s_{goal}. **Оптимистич-**

ной эвристикой будем называть ту, для которой верно: $h(s) \leq h^*(s)\ \forall s$. При использовании такой эвристики алгоритм гарантированно найдёт оптимальное решение, если такое существует.

Также будем называть эвристику **монотонной** (последовательной), если $h(n) \leq h(n') + c(n, n')\ \forall n, n' \in S$. Запуск алгоритма с такой эвристикой гарантирует, что никакое состояние не будет рассмотрено более одного раза.

3.2. Методы решения задачи многокритериального поиска кратчайшего пути

3.2.1 NAMOA*

NAMOA* является базовым алгоритмом, решающим задачу многокритериального поиска. Полное описание его работы приведено в псевдокоде №2. Алгоритм всё так же использует список $Open$, но состояния, хранящиеся в нём, теперь описываются как $(s, \mathbf{g}_s, \mathbf{f}_s)$. Здесь и далее жирный шрифт будет использоваться для обозначения пар. К примеру, для $\mathbf{p} = (p_1, p_2)$, p_1 обозначает первый компонент пары $\mathbf{p}$, а p_2 - второй компонент.

Для каждой вершины $s \in S$ поддерживается два новых множества, содержащих **g**-значения. $\mathbf{G}_{op}(s)$ содержит **g**-значения состояний поиска, ассоциированных с s, которые уже были сгенерированы, но не были раскрыты. $G_{cl}(s)$ содержит **g**-значения уже раскрытых состояний.

Вместо единственного предка вершины s теперь хранится множество **g**-значений некоторых предшественников вершины s. Также список предков теперь ассоциирован с g-значениями g_s, а не с самой вершиной.

NAMOA* на каждом шаге выполнения основного цикла извлекает из списка $Open$ состояние, чьё **f**-значение не доминируется никаким другим состоянием в $Open$ (Т.е вершины подаются на раскрытие в лексикографическом порядке).

В процессе раскрытия состояния $(s, \mathbf{g}_s, \mathbf{f}_s)$ и рассмотрения соседей, достижимых из него, происходит несколько проверок доминируемости:

1. Если вершина s рассматриваемого состояния является целевой, то из $Open$ удаляются все состояния, **f**-значения которых доминируются $\mathbf{f}_s$. Т.е если где-то есть пути, которые на данный момент ещё не привели к целевой вершине и они уже хуже найденного, то лучше они не станут, ведь все рёбра имеют неотрицательный вес.

2. В случае, если $\mathbf{g}_s$ рассматриваемого соседа t вершины s доминируется кем-то из $\mathbf{G}_{op}(t) \cup \mathbf{G}_{cl}(t)$, то состояние для этого соседа не гене-

рируется и данный путь далее не рассматривается. Также состояние не добавляется в обработку, если его $\mathbf{f}_t$ значение доминирует хотя бы одно из уже полученным решений.

3. В противном случае происходит удаление всех путей из $\mathbf{G}_{op}(t) \cup \mathbf{G}_{cl}(t)$, которые доминируют $\mathbf{g}_t$

Проверки доминируемости требуют каждый раз итерироваться по большому числу элементов: $|sols| + |open| + |\mathbf{G}_{op}(t)| + |\mathbf{G}_{cl}(t)|$. Именно оптимизация этих проверок станет целью последующих рассматриваемых нами алгоритмов.

3.2.2 NAMOA*dr

NAMOA*dr - модифицированная версия NAMOA*, оптимизации которой направлены на решение проблемы линейного времени проверок доминирования.

Оптимизации получается добиться за счёт выполнения двух условий:

1. Должна быть использована **последовательная эвристика** ($\mathbf{h}(s) \leq \mathbf{c}(s,t) + \mathbf{h}(t)$). Т.е для каждой вершины s и каждого её преемника t, оценка стоимости достижения s_{goal} из s не должна превосходить стоимость шага до p плюс оценка стоимости достижения s_{goal} из t.

2. Изъятие состояний из списка $Open$ должно происходить в лексикографическом порядке и выполняться за $O(1)$. Это подразумевает, что список $Open$ будет использовать специфичную структуру для упорядочивания состояний внутри себя.

Соблюдение этих условий позволяет изменить процесс раскрытия состояний. Так как теперь, извлекая состояние с вершиной s мы можем быть уверены, что его g_1 значение как минимум не меньше g_1 значений всех уже найденных путей в s, то нам остаётся проверить только значение g_2.

В псевдокоде №2 показан алгоритм NAMOA*dr, в котором синим цветом выделены фрагменты, отличающие его от стандартного NAMOA*. Алгоритм запоминает минимальные значения g_2 для каждой вершины s и записывает их в $g_2^{min}(s)$.

Algorithm 2: NAMOA*

Input: A search problem$(S, E, \mathbf{c}, s_{start}, s_{goal})$ and a consistent heuristic function
Output: The Pareto-optimal solution set

$sols \leftarrow \emptyset$
for ***each*** $s \in S$ **do**
    $\mathbf{G}_{op}(s) \leftarrow \emptyset$
    $\mathbf{G}_{cl}(s) \leftarrow \emptyset$
$\mathbf{G}_{op}(s) \leftarrow (0, 0$
$parent((0,0)) \leftarrow \emptyset$
Initialize $Open$ and add $(s_{start}, (0,0), \mathbf{h}(s_{start}))$ to it
while $Open \neq \emptyset$ **do**
    Remove a node $(s, \mathbf{g}_s, \mathbf{f}_s)$ from $Open$ with the lexicographically smallest f-value of all nodes in $Open$
    Remove $\mathbf{g}_s$ from $\mathbf{G}_{op}(s)$ and add it to $G_{cl}(s)$
    if $s == s_{goal}$ **then**
        Add $\mathbf{g}_s$ to $sols$
        Remove all nodes $(u, \mathbf{g}_u, \mathbf{f}_u)$ with $\mathbf{f}_s \prec \mathbf{f}_u$ from $Open$
        continue
    for ***each*** $t \in Succ(s)$ **do**
        $\mathbf{g}_t \leftarrow \mathbf{g}_s + \mathbf{c}(s, t)$
        if $\boldsymbol{g}_t \in \boldsymbol{G}_{op}(t) \cup \boldsymbol{G}_{cl}(t)$ **then**
            Add $\mathbf{g}_s$ to $parent(\mathbf{g}_t)$
            continue
        if $\boldsymbol{G}_{op}(t) \cup \boldsymbol{G}_{cl}(t) \prec \boldsymbol{g}_t$ **then**
            continue
        $\mathbf{f}_t \leftarrow \mathbf{g}_t + \mathbf{h}(t)$
        if $sols \prec \boldsymbol{f}_t$ **then**
            continue
        Remove all g-values $\mathbf{g'}_t$ from $\mathbf{G}_{op}(t)$ that are dominated by $\mathbf{g}_t$ and remove their corresponding nodes $(t, \mathbf{g'}_t, \mathbf{f'}_t)$ from $Open$
        Remove all g-values from $\mathbf{G}_{cl}(t)$ that are dominated by $\mathbf{g}_t$
        $parent(\mathbf{g}_t) \leftarrow \mathbf{g}_s$
        Add $\mathbf{g}_t$ to $\mathbf{G}_{op}(t)$
        Add $(t, \mathbf{g}_t, \mathbf{f}_t)$ to $Open$
return $sols$

Algorithm 3: NAMOA*dr

Input: A search problem$(S, E, \mathbf{c}, s_{start}, s_{goal})$ and a consistent heuristic function
Output: The Pareto-optimal solution set

$sols \leftarrow \emptyset$
for ***each*** $s \in S$ **do**
    $\mathbf{G}_{op}(s) \leftarrow \emptyset$
    $\mathbf{G}_{cl}(s) \leftarrow \emptyset$
    $g_2^{min}(s) \leftarrow \infty$
$\mathbf{G}_{op}(s_{start}) \leftarrow \{(0,0)\}$
$parent((0,0)) \leftarrow \emptyset$
Initialize $Open$ and add $(s_{start}, (0,0), \mathbf{h}(s_{start}))$ to it
while $Open \neq \emptyset$ **do**
    Remove a node $(s, \mathbf{g}_s, \mathbf{f}_s)$ from $Open$ with the lexicographically smallest f-value of all nodes in $Open$
    Remove $\mathbf{g}_s$ from $\mathbf{G}_{op}(s)$ and add it to $\mathbf{G}_{cl}(s)$
    if $g_2^{min}(s_{goal}) \leq g_{s,2}$ **then**
        continue
    if $s == s_{goal}$ **then**
        Add $\mathbf{g}_s$ to $sols$
        continue
    for ***each*** $t \in Succ(s)$ **do**
        $\mathbf{g}_t \leftarrow \mathbf{g}_s + \mathbf{c}(s,t)$
        if $\mathbf{g}_t \in \boldsymbol{G}_{op}(t) \cup \boldsymbol{G}_{cl}(t)$ **then**
            Add $\mathbf{g}_s$ to $parent(\mathbf{g}_t)$
            continue
        if $g_2^{min}(t) \leq g_{t,2}$ *or* $\boldsymbol{G}_{op}(t) \prec \boldsymbol{g}_t$ **then**
            continue
        $\mathbf{f}_t \leftarrow \mathbf{g}_t + \mathbf{h}(t)$
        if $g_2^{min} \leq f_{t,2}$ **then**
            continue
    Remove all **g**-values $\mathbf{g'}_t$ from $\mathbf{G}_{op}(t)$ that are dominated by $\mathbf{g}_t$ and remove their corresponding nodes $(t, \mathbf{g'}_t, \mathbf{f'}_t)$ from $Open$
    $parent(\mathbf{g}_t) \leftarrow \{\mathbf{g}_s\}$
    Add $\mathbf{g}_t$ to $\mathbf{G}_{op}(t)$
    Add $(t, \mathbf{g}_t, \mathbf{f}_t)$ to $Open$
return $sols$

3.2.3 BOA*

BOA* является State of the Art алгоритмом решения задачи поиска кратчайшего пути по двум критериям. Метод развивает идеи NAMOA*dr, полностью избавляясь от проверок доминируемости, производимых за линейное время (Алгоритм №3, строки 21, 27).

BOA* всё так же использует минимальные значения g_2 каждой вершины для проверок доминируемости за O(1), но при этом больше не хранит множества $\mathbf{G}_{cl}(s)$ и $\mathbf{G}_{op}(s)$. Такие оптимизации становятся доступными за счёт того, что BOA* использует отложенные (ленивые) проверки доминируемости (Алгоритм №4, строка 12).

Algorithm 4: BOA*

Input: A search problem$(S, E, \mathbf{c}, s_{start}, s_{goal})$ and a consistent heuristic function
Output: A cost-unique Pareto-optimal solution set

1 $sols \leftarrow \emptyset$
2 **for** ***each*** $s \in S$ **do**
3 $\quad sols(s) \leftarrow \emptyset$
4 $\quad g_2^{min}(s) \leftarrow \inf$
5 $x \leftarrow$ new node with $s(x) = s_{start}$
6 $\mathbf{g}(x) \leftarrow (0, 0)$
7 $parent(x) \leftarrow$ null
8 $\mathbf{f}(x) \leftarrow (h_1(s_{start}), h_2(s_{start}))$
9 Initialize $Open$ and add x to it
10 **while** $Open \neq \emptyset$ **do**
11 $\quad$ Remove a node x from $Open$ with the lexicographically smallest f-value of all nodes in $Open$
12 $\quad$ **if** $g_2(x) \geq g^2_{min}(s(x)) \vee f_2(x) \geq g_2^{min}(s_{goal})$ **then**
13 $\quad\quad$ **continue**
14 $\quad g_2^{min}(s(x)) \leftarrow g_2(x)$
15 $\quad$ Add x to $sols(s(x))$
16 $\quad$ **if** $s(x) == s_{goal}$ **then**
17 $\quad\quad$ **continue**
18 $\quad$ **for** ***each*** $t \in Succ(s(x))$ **do**
19 $\quad\quad y \leftarrow$ new node with $s(y) = t$
20 $\quad\quad \mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{c}(s(x), t)$
21 $\quad\quad parent(y) \leftarrow x$
22 $\quad\quad \mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}(t)$
23 $\quad\quad$ **if** $g_2(y) \geq g^2_{min}(t) \vee f_2(y) \geq g_2^{min}(s_{goal})$ **then**
24 $\quad\quad\quad$ **continue**
25 $\quad\quad$ Add y to $Open$
26 **return** $sols(s_{goal})$

3.2.4 BOD

Авторы алгоритма BOA* также предлагают модификацию своего алгоритма для решения задачи поиска кратчайших путей ко *всем* вершинам заданного графа. BOD представляет из себя алгоритм поиска с равномерной стоимостью. Для этого используется нулевая эвристика **h**.

Algorithm 5: BOD

Input: A bi-objective weighted graph $(S, E, \mathbf{c}, s_{start})$
Output: A cost-unique Pareto-optimal solution set for each state in S

for ***each*** $s \in S$ **do**
  $sols(s) \leftarrow \emptyset$
  $g_2^{min}(s) \leftarrow \infty$
$x \leftarrow$ new node with $s(x) = s_{start}$
$\mathbf{g}(x) \leftarrow (0, 0)$
$parent(x) \leftarrow$ null
Initialize $Open$ and add x to it
while $Open \neq \emptyset$ **do**
  Remove a node x from $Open$ with the lexicographically smallest $\mathbf{g}$-value of all nodes in $Open$
  if $g_2(x) \geq g_2^{min}(s(x))$ **then**
    continue
  $g_2^{min}(s(x)) \leftarrow g_2(x)$
  Add x to $sols(s(x))$
  for ***each*** $t \in Succ(s(x))$ **do**
    $y \leftarrow$ new node with $s(y) = t$
    $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{c}(s(x), t)$
    $parent(y) \leftarrow x$
    if $g_2(y) \geq g_2^{min}(t)$ **then**
      continue
    Add y to $Open$
return $sols$

Получаемое на выходе множество $sols$ содержит Парето-множества решений для каждой вершины входного графа.

3.3. Маршрутизация

3.3.1 Случаи, когда ограниченность решения не может быть достигнута

Перед переходом к рассмотрению методов решения задачи общей децентрализованной маршрутизации, докажем следующее утверждение:

Утверждение 1. Существуют такие экземпляры задачи общей маршрутизации (G, C_1, C_2), для которых ограниченное решение не может быть

найдено при условии, что функция $nextHop$ должна зависеть только от адресата пакета.

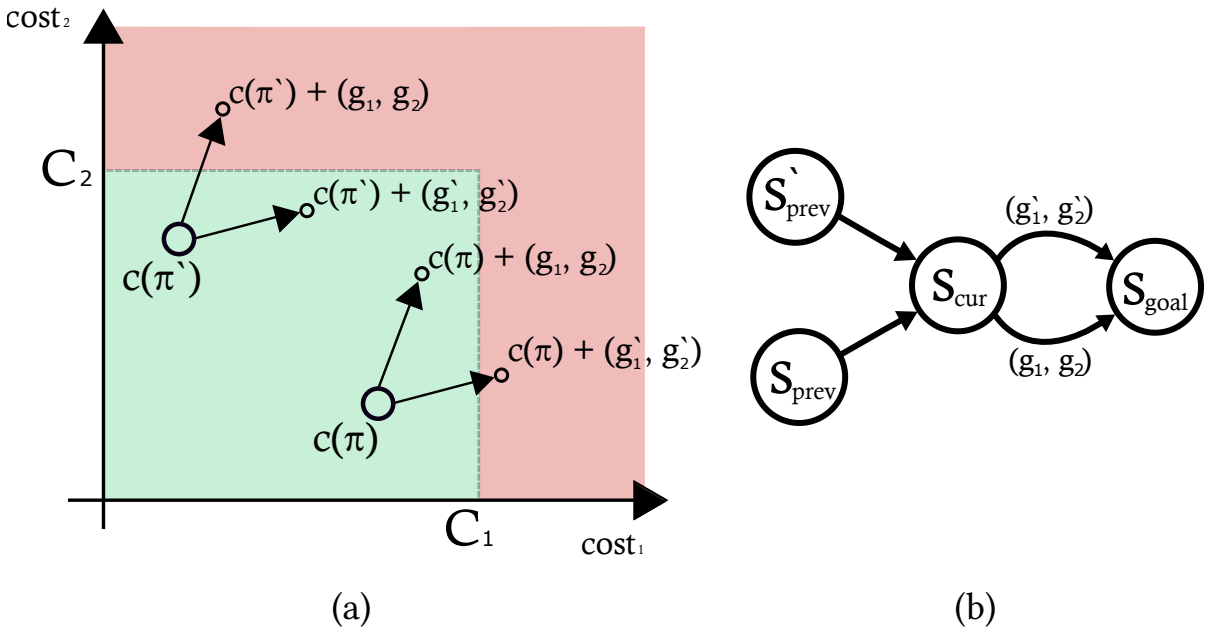

Рис. 6: (a) Пример возможных исходов при продолжении некоторых путей π и π' из вершины s_{cur} и (b) рисунок подграфа, в котором содержится эта вершина.

Доказательство. Пусть G — граф содержащий в себе подграф, изображённый на рисунке 6 (b). Вершины s_{prev} и s'_{prev} имеют некоторые соединения с вершинами графа G. Рассмотрим два пакета $p = (s_{start}, s_{goal})$ и $p' = (s'_{start}, s_{goal})$, $s_{start}, s'_{start} \in S$. Пусть пакет p попадает в узел s_{cur} , пройдя путь $\pi = (s_{start}, \ldots, s_{prev})$. Аналогично, пакет p' проходит путь $\pi' = (s'_{start}, \ldots, s'_{prev})$. Пусть суммарные стоимости путей $c(\pi)$ и $c(\pi')$ не выходят за пределы заданных ограничений C_1 и C_2. При этом $c(\pi) + (g'_1, g'_2)$ и $c(\pi') + (g_1, g_2)$ уже превышают ограниения, как это показано на рисунке 6 (a). Пакетам p и p' остаётся совершить только один переход, который зависит от того, как устроена функция $nextHop_{s_{cur}}$. Рассмотрим все варианты работы функции $nextHop_{s_{cur}}$, учитывая, что функция должна зависеть только адресата пакета:

1. Отправить пакет с адресатом s_{goal} по пути со стоимостью (g_1, g_2). Но тогда суммарная стоимость доставки пакета **p'** превысит заданные ограничения

2. Отправить пакет с адресатом s_{goal} по пути со стоимостью (g'_1, g'_2). Но тогда суммарная стоимость доставки пакета **p** превысит заданные ограничения

Как видим, в обоих случаях нарушается ограниченность итогового решения задачи. □

3.3.2 Метод моделирования поведения узлов

Заметим, что в рассмотренном примере из пункта 3.3.1 узлу s_{cur} для принятия решения о передаче, при котором ограниченность будет соблюдена, достаточно информации о стоимости пути, пройденного пакетом. Но по условию децентрализованности узлы сети не могут обмениваться информацией о выбранных маршрутах передачи пакетов, а значит и информацию о суммарной стоимости передать вместе с пакетом не получится. При этом ограниченное решение для данного экземпляра задачи общей маршрутизации существует и далее будет предложен метод, который позволяет добиться этого при условии, что функция $nextHop_{s_{cur}}$, должна зависеть от отправителя и адресата пакета.

Суть метода состоит в том, что мы задаём единый алгоритм решения задачи частной маршрутизации, который используется каждым узлом сети. Это позволяет узлам моделировать поведение остальных участников сети и за счёт этого строить собственную таблицу маршрутизации[1]. Также для всех узлов сети фиксируется метод выбора конкретного решения из Парето-множества найденных путей в задаче многокритериального поиска кратчайшего пути. Операцию выбора решения из Парето-множества для вершины s будем обозначать $choose(sols(s))$, $choose : (R \times R)^n \to (R \times R)$, где $sols$ — множество решений для каждой вершины графа, как это описано в алгоритмах 2-5. Для примера будем использовать функцию $choose$, которая работает следующим образом:

1. Если среди Парето-множества есть решения, стоимости которых не превышают C_1 и C_2, то выбрать решение с минимальной ценой по первому критерию

2. Если таких решений нет, то также выбирать решение с минимальной ценой первого критерия.

Пусть сеть задаётся графом $G = (S, E, \vec{c})$. Рассмотрим этапы работы алгоритма построения таблицы маршрутизации для некоторого узла cur_node в данной сети.

[1]Здесь и далее функция $nextHop$ для удобства будет называться таблицей маршрутизациии обозначаться $route\_table$. Таблица имеет три столбца - $sender$, $target$ и $next_hop$. Столбцы $sender$ и $target$ отвечают за отправителя и адреса пакета соответственно и служат для установления узла-соседа, которому нужно предать пакет ($next_hop$).

Шаг 1: Нахождение множества reacheble

Множеством $reacheble$ будем называть множество, состоящее из вершин $t \in S$ в которые существует путь $\pi = (cur\_node, \ldots, t)$ из cur_node, стоимость которого не превышает заданные ограничения C_1, C_2. Для нахождения этого множества запустим алгоритм BOD на задаче (G, cur_node) с небольшим изменением (*): теперь состояние поиска не будет добавляться в очередь на рассмотрение, если его g-значения превышают заданные C_1, C_2. В дальнейших шагах это изменение продолжает действовать. Также на этом запуске будем хранить в каждом состоянии поиска не предка ($parent$), а первую из пройденных вершин пути ($first\_step$). После нахождения множества решений $sols$, необходимо пройтись по всем вершинам $s \in S$ и в случае, если $sols(s)$ содержит решения, то записать в таблицу маршрутизации строку cur_node, s, $first\_step(choose(sols(s)))$.

Шаг 2: Нахождение множества possible senders

Множеством $possible\_senders$ будем называть множество, состоящее из вершин $t \in S$ для которых существует путь $\pi = (cur\_node, \ldots, t)$, стоимость которого не превышает заданные ограничения C_1, C_2. Это множество так же будем находить с помощью алгоритма BOD, но запущенного уже на графе G, в котором изменены все направления рёбер. В данном запуске BOD можно отказаться от хранения записей о предках в состояниях поиска, т.к на данном шаге нам необходима информация только о достижимости вершин.

Шаг 3: Моделирование поведения узлов

Для каждого узла $s_i \in possible\_sender$ поочерёдно запустим алгоритм BOD. При этом будем хранить в каждом состоянии поиска не предка, а следующую за cur_node вершину ($next\_node$) если cur_node встретилась на пути. Также запоминаем решения только для узлов из $reacheable$. После запуска алгоритма BOD для вершины s_i для каждой вершины $s_j \in reacheble$ выбираем решение, если оно существует. Если в состоянии, которое соответствует выбранному решению, указано, что на пути был встречен узел cur_node, то записываем в таблицу маршрутизации строку s_i, s_j, $next\_node(choose(sols(s_j)))$.

Шаг 4: Оптимизация полученной таблицы маршрутизации

Для каждой вершины $t \in reacheble$ проверим все строки таблицы, с полем $target = t$. Если все просмотренные строки имеют одинаковое значение поля $next\_hop$, равное p, то мы можем заменить эти записи на строку $t, *, p$. Такая строка будет обозначать, что все пакеты с адресатом t необходимо передавать на узел p вне зависимости от узла-отправителя.

Стоит отметить, что если необходимо найти таблицу маршрутизации, которая будет учитывать в том числе пакеты, для которых маршруты доставки существуют, но стоимость каждого из них превышает C_1, C_2, то необходимо запускать алгоритм BOD без введённого ограничения на стоимость пути (*).

Псевдокод данного метода описан в алгоритме №6. В записи псевдокода верхние индексы алгоритма BOD (строки 1, 6, 11) обозначают, что используется BOD с модификациями, описанными в шаге, номер которого совпадает с индексом.

Algorithm 6: Метод моделирования узлов

Input: A single routing problem $(S, E, \vec{c}, C_1, C_2, s_{cur\_node})$
Output: $route\_table$

1 $sols \leftarrow BOD^1(S, E, \vec{c}, s_{cur\_node})$
2 **for** ***each*** $t \in S$ **do**
3 **if** $sols(t) \neq \emptyset$ **then**
4 $route\_table.write(s_{cur\_node}, t, first\_step(choose(sols(t))))$
5 $reacheble \leftarrow t$
6 $sols \leftarrow BOD^2(S, reverse(E), \vec{c}, s_{cur\_node})$
7 **for** ***each*** $t \in S$ **do**
8 **if** $sols(t) \neq \emptyset$ **then**
9 $possible\_senders \leftarrow p$
10 **for** ***each*** $t \in possible\_senders$ **do**
11 $sols \leftarrow BOD^3(S, E, \vec{c}, t)$
12 **for** ***each*** $p \in reacheble$ **do**
13 **if** $next\_node(sols(p)) \neq \emptyset$ **then**
14 $route\_table.write(t, p, next\_node(choose(sols(p))))$
15 **for** ***each*** $target \in reacheble$ **do**
16 $senders_{target} \leftarrow \emptyset$
17 **for** ***each*** $sender \in possible\_senders$ **do**
18 Add $route\_table.get\_next\_hop(sender, target)$ to $senders_{target}$
19 **if** $|senders_{target}| \leq 1$ **then**
20 // Заменяем все записи, содержащие $target$ в поле адресата пакета на строку с универсальным переходом $(target, *, next\_hop)$
21 $route\_table.prune(target)$
22 **return** *route_table*

Решение, полученное методом моделирования узлов, является детерминированным. Т.е каждый узел сети, отправляя пакет данных, может точно определить по какому маршруту данный пакет будет доставлен.

3.3.3 Метод жадной маршрутизации

Рассмотрим метод, который не гарантирует нахождения ограниченного решения в общем случае, но при этом не использует информацию об узле-отправителе пакета для принятия решения о пересылке.

Аналогично первому рассмотренному методу, каждому узлу сети задаётся единый алгоритм решения задачи частной маршрутизации. Но алгоритм не включает в себя модуляцию работы других узлов сети, и работает по принципу жадного подхода: для каждого пакета выбирается наилучший путь передачи без учета поведения узлов сети, которым будет передан пакет. Наилучший путь определяется тем, как работает выбор $choose()$ решения из Парето-множества. Стоит отметить, что метод жадной маршрутизации в общем случае не обладает детерминированностью.

На вход алгоритму подаётся граф $G = (S, E, \vec{c})$, ограничения по суммарной стоимости C_1, C_2 пути и узел s_{cur_node} для которого нужно построить таблицу маршрутизации. Алгоритм состоит из одного запуска BOD на задаче $(S, E, \vec{c}, s_{cur\_node})$ и последующего заполнения таблицы маршрутизации на основе полученных Парето-множеств решений. BOD запускается с модификацией, описанной в шаге №1 метода моделирования: в каждом состоянии поиска хранится не вершина-предок, а первая из пройденных вершин пути, т.е сосед вершины s_{cur_node}, с которого начался путь.

Algorithm 7: Метод жадной маршрутизации

Input: A single routing problem $(S, E, \vec{c}, C_1, C_2, s_{start})$
Output: $route\_table$

$sols \leftarrow BOD^1(S, E, \vec{c}, s_{start})$
for ***each*** $t \in S$ **do**
  if $sols(t) \neq \emptyset$ **then**
    $route\_table.write(s_{start}, t, first\_step(choose(sols(t))))$
return $route\_table$

Важную роль играет выбор функции $choose$. Рассмотрим три варианта поведения данной функции:

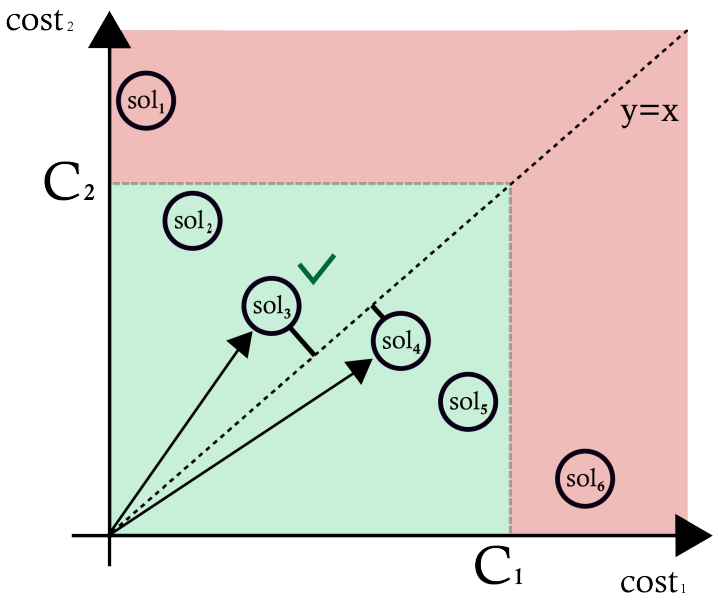

Рис. 7: Демонстрация работы варианта №1 функции $choose$ на Парето-множестве из 6 решений: выбрано решение sol_3.

Вариант №1 выбор ближайшего

Каждое решение в Парето-множестве описывается парой из g-значений (g_1, g_2). Если в Парето-множестве есть решения, которые превосходят заданные константы C_1 и C_2, то далее их рассматривать не будем. Если все решения не подходят под ограничения, то продолжаем рассматривать полное Парето-множество. Представим решения как точки на двумерной координатной плоскости, где g_1 соответствует первой координате и g_2 второй. Сопоставим каждому решению $(g_1, g_2) \in sols$ пару из двух значений:

- Длина вектора [(0,0); (g_1, g_2)]
- Расстояние от (g_1, g_2) до прямой $y = x$

Отсортируем решения на основе данных значений в лексикографическом порядке и выберем первое из них. Пример работы данного варианта показан на рисунке №7.

Вариант №2 и №3: минимальная стоимость по первому или второму критерию

В случае если Парето-множество полностью состоит из решений, которые превосходят установленные ограничения C_1 и C_2, то воспользуемся

вариантом функции $choose$ №1. Иначе будем рассматривать Парето-множество без данных решений. В данном варианте будем выбирать решение с минимальным значением g_1 (вариант №2) или g_2 (вариант №3).

4. Эксперименты

Для экспериментов использовался направленный граф, представляющий собой сеть, состоящую из 1024 узлов, где каждый узел имел в среднем четыре соединения. Рёбра графа описывались двумя величинами: $delay$ и $loss$. 0.00497649 <= $delay$ <= 0.0295323, 0.00869565 <= $loss$ <= 0.0434783. Был выбран план проведения эксперимента, который состоял из двух частей — построения таблиц маршрутизации и трассировки пакетов. Эксперимент был запущен для следующих пар значений C_1, C_2: (0.6, 0.6), (0.7, 0.7), (0.8, 0.8). Разберём оба этапа эксперимента:

Этап №1 Построение таблиц маршрутизации

Первым этапом было построение полного множества правил маршрутизации (таблицы маршрутизации всех узлов сети) для каждого исследуемого метода.

В начале на каждой вершине исходного графа запускался алгоритм решения частной задачи маршрутизации методом моделирования узлов. В качестве функции $choose$ использовалась функция выбора ближайшего (Вариант №1, описанный на странице 27). Во время запуска собирались следующие параметры:

- Время выполнения алгоритма
- Максимальный размер множества $OPEN$, которое использовалось алгоритмом BOD.
- Суммарное количество решений в множестве $sols$, которое получается на выходе алгоритма BOD.

Полученная таблица маршрутизации сохранялась на жёстком диске компьютера в директории, соответствующей методу моделирования.

Далее снова строились таблицы маршрутизации, но уже с использованием метода жадной маршрутизации. Для каждой вершины проводилось три запуска, за которые использовались все три варианта функции выбора $choose$. Таблицы маршрутизации сохранялись в директориях соответствующих использованному методу.

Этап №2 Трассировка

Для каждой вершины s исходного графа запускался алгоритм BOD и из тех вершин, до которых был найден путь, длина которого не превышает

заданные C_1, C_2, выбиралось десять случайных вершины t. Полученные пары (s, t) записывалась в датасет. Таким образом генерировался датасет, состоящий из 10000 пакетов.

Далее проводилась трассировка сгенерированных пакетов. Трассировка выполнялась отдельно для каждого множества таблиц маршрутизации, которые были получены на предыдущем этапе. После трассировки пакета вычислялась стоимость пройденного пакетом пути. Значениями стоимости заполнялась итоговая таблица эксперимента, которая состояла из полей 'Пакет', 'Метод моделирования', 'Жадный метод вариант 1', 'Жадный метод вариант 2', 'Жадный метод вариант 3'. Таким образом каждая строка таблицы содержала информацию о стоимости пройденного пакетом пути в зависимости от выбранного метода решения общей задачи маршрутизации.

Дополнительно было проведено повторное построение таблиц маршрутизации (Этап №1), в котором использовался алгоритм решения частной задачи маршрутизации методом моделирования узлов, но без модифицированных версий алгоритма BOD (Использовалось только ограничение на суммарную стоимость пути). Данный запуск был проведён для исследования потребляемых алгоритмом ресурсов.

Все эксперименты проводились на персональном стационарном компьютере, в котором использовался процессор AMD Ryzen 5 7600X и 64 гигабайта оперативной памяти.

5. Результаты экспериментов

Таблицы №1 и №2 содержат результаты эксперимента с трассировкой пакетов. Greedy_1,2,3 обозначает результаты эксперимента для жадного метода с использованием вариантов функции выбора $choose$ (Вариант 1,2 и 3 соответственно). Modeling - метод моделирования узлов с измененными версиями BOD.

Из результатов видно, что использование вариантов 2 и 3 функции выбора $choose$ в жадном методе наиболее выгодно. Данные варианты выбирают решения с минимальной стоимостью по первому или второму критерию соответственно. Первый вариант функции выдал результаты хуже остальных. Метод моделирования узлов справился с передачей *всех* пакетов, но в своей работе он учитывал также информацию об узле-отправителе. Основываясь на результатах можно сказать, что уровень несоответствия стоимости путей при использовании метода жадной маршрутизации не слишком высок, но если требуется поддержание 100% ограниченности, то только способ моделирования справляется с задачей.

Таблица 1: Процент пакетов для которых не найден путь, стоимость которого не превышает ограничения

Algorithm	**Experiments**		
	C=(0.6, 0.6)	**C=(0.7, 0.7)**	**C=(0.8, 0.8)**
	bad pockets (%)	**bad pockets (%)**	**bad pockets (%)**
Greedy	18	14	9
	14	11	6
	15	10	7
Modeling	**0**	**0**	**0**

Также из таблицы №2 видно, что метод жадной маршрутизации по затраченному времени превосходит метод моделирования ($Greedy$ дано без указания варианта функции $choose$, т.к. функция не влияет на время выполнения). Разрыв в несколько порядков вполне понятен — жадный метод вызывает внутри себя BOD всего один раз, тогда как метод моделирования делает это для каждой вершины графа.

В таблице №3 содержатся результаты эксперимента по сравнению потребляемых ресурсов при использовании метода моделирования узлов. Сравниваются версии с оптимизированными алгоритмами BOD и без. Параметр сравнения $max\ OPEN\ len$ обозначает максимальное количество состояний поиска, которые находились в очереди на раскрытие при запуске алгоритма BOD (Строки 1, 6, 11 алгоритма 6). $Solutions\ count$ - общее число решений во всех вершинах графа при тех же запусках BOD.

Таблица 2: Среднее время выполнения алгоритма

Algorithm	**Experiments**		
	C=(0.6, 0.6)	**C=(0.7, 0.7)**	**C=(0.8, 0.8)**
	avg time (s.)	**avg time (s.)**	**avg time (s.)**
Greedy	**0.3**	**0.42**	**0.51**
Modeling	95	186	305

Таблица 3: Сравнение потребления ресурсов у методов моделирования поведения узлов

$\vec{C}$	Algorithm	average time(s)	max OPENlen	solutions count
(0.6, 0.6)	Modeling	**95**	3406	**7280**
	Modeling_no_opt	81	3406	8032
(0.7, 0.7)	Modeling	**186**	3630	**8290**
	Modeling_no_opt	200	3630	8600
(0.8, 0.8)	Modeling	**305**	3910	**9028**
	Modeling_no_opt	285	3910	9240

Изменения, привнесённые в алгоритм BOD действительно помогли улучшить показатели используемых алгоритмом ресурсов. Полученные данные можно использовать для дальнейшего анализа возможности применения данного метода для конкретного оборудования.

6. Заключение

6.1. Результаты работы

В ходе выполнения работы были исследованы методы маршрутизации в компьютерных сетях, а также методы решения задачи поиска кратчайшего пути для случая одного и многих критериев поиска. Была сформулирована постановка задачи маршрутизации и критерии оценки её решения. Также было приведено доказательство невозможности нахождения ограниченного решения задачи маршрутизации при условии, что функция перехода должна зависеть только от адресата пакета.

На основе научных статей было разработано два метода маршрутизации, которые используют State of the Art алгоритмы многокритериального поиска. Первый метод основывается на моделировании поведения узлов и позволяет найти ограниченное и детерминированное решение для любой сети, но при этом требует учёта узла-отправителя при принятии решения о передаче пакета. Второй метод использует информацию только об адресате пакета, но при этом не гарантирует ограниченности и детерминированности решения в общем случае. Был подготовлен репозиторий с кодом данных методов: https://github.com/heartmarshall/decentralized-biobjective-routing. Также в репозитории были размещены разработанные для тестирования инструменты генерации тестовых карт для задачи многокритериального поиска.

Были проведены эксперименты, в результате которых были получены данные об ограниченности решений при использовании второго метода маршрутизации, а также о ресурсозатратности первого метода. Эксперименты подтвердили эффективность оптимизаций, использованных в первом методе и позволили определить оптимальную функцию выбора решения для второго метода.

6.2. Дальнейшие направления исследования

Можно выделить несколько направлений дальнейших исследований по теме многокритериальной маршрутизации:

Адаптация методов к изменяющейся сети

Предложенные в данной ВКР методы предполагают статическую сеть и требуют перестраивания всей таблицы маршрутизации при каждом изменении в сети. Нахождение способа динамически обновлять правила марш-

рутизации с учётом изменений позволит снизить нагрузки на оборудование.

Вероятностная маршрутизация

Одним из возможных направлений развития является использование вероятностных подходов. В таких методах маршрутизация основывается на вероятностных моделях, которые учитывают различные факторы, такие как загруженность сети или вероятность отказа узлов. Эти модели могут быть адаптированы для работы в реальном времени, что позволяет принимать решения о маршрутизации пакетов на основе текущего состояния сети и прогнозов её будущего состояния.

Маршрутизация с учётом статистики пакетов

Ещё одним направлением является разработка методов маршрутизации, которые учитывают статистику передаваемых пакетов. Это может включать анализ частоты передачи пакетов конкретному адресату, размера пакетов, и т.д. Например, адаптировать маршрутизацию для более равномерного распределения нагрузки между узлами.

Список литературы

[1] P.E. Hart, N. Nilsson, B. Raphael, "A formal basis for the heuristic determination of minimal cost paths", IEEE Trans. Syst. Sci. Cybern. 4 (1968) 100–107

[2] Dijkstra, E. W., “A note on two problems in connexion with graphs”. Numerische Mathematik, 1(1), (1959) 269–271.

[3] Richard Bellman, "On a routing problem", Quart. Appl. Math. 16 (1958), 87-90

[4] J. Moy Request, Proteon, Inc., Network Working Group, Request for Comments: 1131, October 1989

[5] David Oran, Digital Equipment Corp, Network Working Group, Request for Comments: 1142, February 1990

[6] Information Sciences Institute, University of Southern California, Request for Comments: 791, September 1981

[7] B.S. Stewart, C.C. White III, Multiobjective A*, J. ACM 38 (1991) 775–814.

[8] L. Mandow, J.L.P. De La Cruz, Multiobjective A* search with consistent heuristics, J. ACM 57 (2010) 27:1–27:25.

[9] Carlos Hernández Ulloa, William Yeoh, Jorge A. Baier, Han Zhang, Luis Suazo, Sven Koenig, "A Simple and Fast Bi-Objective Search Algorithm", 30th International Conference on Automated Planning and Scheduling, ICAPS 2020, 26 oct. 2020

[10] Carlos Hernández and William Yeoh and Jorge A. Baier and Han Zhang and Luis Suazo and Sven Koenig and Oren Salzman, "Simple and efficient bi-objective search algorithms via fast dominance checks", Artificial Intelligence 314 (2023) 103807

[11] Yetgin, Halil and Cheung, Kent Tsz Kan and Hanzo, Lajos, "Multi-objective routing optimization using evolutionary algorithms", IEEE Wireless Communications and Networking Conference (WCNC) 01-04 April 2012, 3030-3034