Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

«Глубокие самообучающиеся агенты для мультиагентной системы маршрутизации»

Автор: Мухутдинов Дмитрий Вадимович _____
Направление подготовки (специальность): 01.03.02 Прикладная математика и информатика
Квалификация: Бакалавр
Руководитель: Фильченков А.А., к.ф.-м.н _____

К защите допустить
Зав. кафедрой Васильев В.Н., докт. техн. наук, проф. _____

«____» _____ 20_____ г.

Санкт-Петербург, 2017 г.

Студент Мухутдинов Д.В. **Группа** М3438 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования

Направленность (профиль), специализация Математические модели и алгоритмы разработки программного обеспечения

Консультанты:

а) Вяткин В.В., PhD, Luleå University of Technology _____

Квалификационная работа выполнена с оценкой _____

Дата защиты «____» _____ 20_____ г.

Секретарь ГЭК _____

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Задача пакетной маршрутизации - это задача поиска кратчайшего пути в графе в условиях, когда за каждый узел графа отвечает отдельный вычислительный процесс. Это означает, что каждый отдельный узел графа должен принять решение о том, какому из соседей следует отправить очередной пакет, чтобы тот достиг пункта назначения как можно быстрее.

Задача пакетной маршрутизации впервые обрела актуальность с появлением компьютерных сетей. Первые алгоритмы сетевой маршрутизации появились в процессе разработки сети ARPANet. Именно тогда были изобретены такие подходы к пакетной маршрутизации, как distance-vector[1] и link-state[2], которые и по сей день лежат в основе таких стандартных и широко применяемых алгоритмов сетевой маршрутизации, как Routing Information Protocol (RIP)[3] и Open Shortest Path First (OSPF)[4].

Оба этих подхода основаны на идее вычисления кратчайшего пути между текущим узлом сети и пунктом назначения пакета. Однако существуют и другие подходы к решению задачи маршрутизации, основанные на идее обучения с подкреплением (reinforcement learing). Первым таким алгоритмом стал алгоритм Q-routing[5], основанный на методе Q-learning[6]. Этот алгоритм, как и его модификации[7, 8], благодаря обучению с подкреплением оказался способен лучше адаптироваться к изменениям в интенсивности сетевого трафика, чем алгоритмы, основанные на вычислении кратчайшего пути, но из-за использования большего количества служебных сообщений применение таких алгоритмов в реальных сетях ограничено.

Но стоит отметить, что задача маршрутизации встречается не только в компьютерных сетях, но также и в более сложных средах, таких как киберфизические системы. Примером такой задачи, частично решаемой с помощью алгоритмов маршрутизации, является управление конвейерной системой (в частности, системой распределения багажа в аэропорту). Задача управления такой системой в большинстве случаев решается с помощью централизованных алгоритмов, и децентрализованное решение (основанное на алгоритма Беллмана-Форда) было пред-

ложено совсем недавно[9]. Такая задача отличается от задачи сетевого роутинга, с одной стороны, тем, что служебные сообщения и целевые сообщения являются разными сущностями (“целевое сообщение” — это чемодан, а служебные сообщения являются цифровыми), и служебные сообщения передаются мгновенно по сравнению с целевыми. С другой стороны, состояние каждого конвейера и всей системы в целом задается большим количеством параметров — такими как скорости конвейеров, положение, количество и масса чемоданов на каждом конвейере, и так далее. С третьей стороны, желательно, чтобы система оптимизировала не только скорость доставки чемоданов до точки назначения, но и, к примеру, собственное энергопотребление.

Первое обстоятельство снимает технические ограничения на количество служебных сообщений, что позволяет в полной мере применять алгоритмы на основе обучения с подкреплением. Второе и третьи обстоятельства усложняют написание оптимального детерминированного алгоритма и наталкивают на идею реализации приближенного решения, например, с использованием нейронных сетей.

В настоящее время в области обучения с подкреплением с применением нейронных сетей достигнуты впечатляющие успехи. Всплеск активности в этой области произошел после выхода статьи команды Google DeepMind об обучении глубокой сверточной нейронной сети игре на консоли Atari 2600[10]. Применению полученной модели к различным задачам в различных условиях посвящено множество исследований, часть из них посвящена проблеме мультиагентного обучения с подкреплением (multi-agent reinforcement learning). Так как задачу маршрутизации можно сформулировать как задачу мультиагентного обучения с подкреплением, имеет смысл применить данные наработки для ее решения.

В данной работе будет предложен алгоритм маршрутизации, основанный на методе Q-routing, но использующий нейронную сеть в качестве обучающегося агента. На данный момент не существует алгоритма маршрутизации, построенного по такому принципу.

В главе 1 будет сформулирована обобщенная постановка задачи маршрутизации в терминах мультиагентного обучения с подкреплени-

ем. Будут рассмотрены существующие алгоритмы маршрутизации, их сильные и слабые стороны. Также будут рассмотрены существующие методы обучения нейронных сетей с подкреплением, в том числе в мультиагентном случае.

В главе 2 будет рассмотрен предложенный алгоритм и обоснованы решения, принятые в ходе его разработки.

В главе 3 будут приведены экспериментальные результаты работы алгоритма для задач пакетной маршрутизации в компьютерной сети и управления системой багажных конвейеров. Будут приведены результаты работы в условиях неравномерной нагрузки на сеть (или конвейерную систему) и изменения топологии сети. Также будет проведено сравнение с существующими алгоритмами маршрутизации.

ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Постановка задачи

Пусть задана сеть в виде графа: $G = (V, E)$. Каждому узлу и каждому ребру в сети приписывается некоторое *состояние*: узел v имеет состояние $s_v \in S_V$, ребро e имеет состояние $s_e \in S_E$. Состояние всей сети, таким образом, описывается как $s = (s_{v_1}, ..., s_{v_n}, s_{e_1}, ..., s_{e_m}) \in S$.

На узел сети $s \in V$ приходят пакеты $p = (d, prop)$, где $d \in V$ - узел назначения пакета, а $prop$ - произвольная дополнительная информация о нем. Обозначим множество всевозможных свойств пакета как $Prop$ ($prop \in Prop$), а множество пакетов в целом, таким образом, как $P = (V, Prop)$. Также заданы некоторые функции *стоимости* прохождения данного пакета через ребра и узлы сети, зависящие от их состояний: $R_v : V \times S_V \times P \to \mathbb{R}^+$ - стоимость прохождения пакета через узел, $R_e : E \times S_E \times P \to \mathbb{R}^+$. Мы явно указываем, что стоимости неотрицательны, чтобы избежать таких постановок задач, в которых существуют пути стоимости $-\infty$ (циклы отрицательной стоимости).

Задача пакетной маршрутизации заключается в том, чтобы определить, какому из соседей $n \in \{v|(s, v) \in E\}$ узел s должен перенаправить пакет $p = (d, prop)$, чтобы ожидаемая стоимость пути от s до d была минимальной.

1.2. Существующие алгоритмы маршрутизации

Почти все существующие алгоритмы маршрутизации были разработаны для маршрутизации пакетов в компьютерных сетях. Большинство алгоритмов маршрутизации в компьютерных сетях, применяемых на практике, относятся к одному из двух семейств алгоритмов — дистанционно-векторные (distance-vector)[1] или состояния каналов связи (link-state)[2]. Концептуально все алгоритмы внутри каждого из этих семейств одинаковы, и различаются только техническими деталями реализации, обусловленными спецификой конкретной узкой сферы применения. Поэтому мы не будем рассматривать алгоритмы каждого семейства по отдельности, а рассмотрим только концепции в целом.

1.2.1. Дистанционно-векторные алгоритмы

Идея дистанционно-векторных алгоритмов (distance-vector algoritms) заключается в следующем:

— Каждый маршрутизатор s в сети хранит таблицу, в которой для каждого другого узла сети d хранится следующая информация:
  – Предполагаемое кратчайшее расстояние от s до d
  – Сосед n, которому нужно отправить пакет, чтобы пакет прошел по кратчайшему пути до узла d.
— Периодически каждый маршрутизатор рассылает свою версию таблицы кратчайших расстояний всем своим соседям
— При получении вектора кратчайших расстояний от соседа маршрутизатор s сравнивает его поэлементно с текущей версией. Если оказывается, что наименьшая стоимость пути от соседа n до узла d, сложенная с оценкой стоимости ребра (s, d) меньше, чем наименьшая стоимость пути от s до d в текущей таблице, то значение в текущей таблице обновляется, и наилучшим соседом для отправки пакета в узел d становится сосед n.

Как можно видеть, дистанционно-векторный алгоритм является, в сущности, распределенной версией алгоритма Беллмана-Форда поиска кратчайшего пути в графе[11].

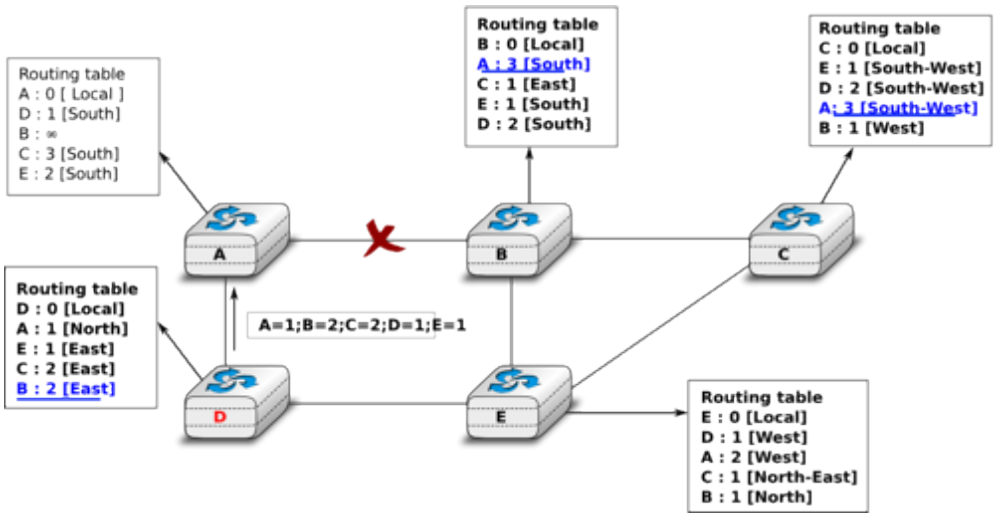

Рисунок 1 – Иллюстрация работы distance-vector алгоритма

Различные реализации дистанционно-векторного метода различаются, в частности, оценками стоимости соединений в сети. Так, например, протокол RIP[3] просто оценивает стоимость каждого соединения в 1, а IGRP[12] оценивает стоимость соединений исходя из оценок задержки и пропускной способности.

Преимуществами дистанционно-векторных алгоритмов являются простота реализации и низкие требования к памяти и вычислительной мощности. Недостатками же являются низкая скорость распространения информации по сети и сложности с приспособлением под изменяющуюся топологию (проблема count-to-infinity). Этих проблем удается избежать при применении другого распространенного подхода - алгоритмов на основе состояния канала связи.

1.2.2. Алгоритмы состояния канала связи

В отличие от дистанционно-векторных алгоритмов, в алгоритмах состояния канала связи (link-state) каждый узел сети хранит у себя модель всей сети в виде графа. Рассмотрим шаги алгоритма подробнее:

— Каждый маршрутизатор периодически проверяет состояние соединений до соседей
— При обнаружении обрыва какого-либо соединения алгоритм удаляет это соединение из собственного графа и рассылает соседям новую версию состояния соединений до них
— Соседи обновляют собственные версии графов в соответствии с полученной информацией и пересылают сообщение дальше
— Чтобы избежать зацикливания сообщений об обновлении состояния, каждое сообщение снабжается *номером версии*. Маршрутизатор n игнорирует сообщение от маршрутизатора n, если номер версии этого сообщения меньше или равен предыдущему.

Имея информацию обо всей сети в целом, маршрутизатор может рассчитать кратчайшие пути до всех остальных узлов. Обычно для этого используется алгоритм Дейкстры[13].

Link-state алгоритмы обладают способностью адаптироваться под изменения топологии сети гораздо быстрее, чем distance-vector алгоритмы за счет несколько более сложной реализации и чуть больших затрат по памяти и вычислительной мощности. Это обуславливает то, что на данный момент именно link-state протоколы, такие как OSPF[4], доминируют в сетевой маршрутизации. Однако даже в решении задачи сетевой маршрутизации link-state алгоритмы в чистом виде не лучшим образом адаптируются к повышению нагрузки в сети. Рассматриваемые в дальнейшем другие алгоритмы, основанные на принципе обучения

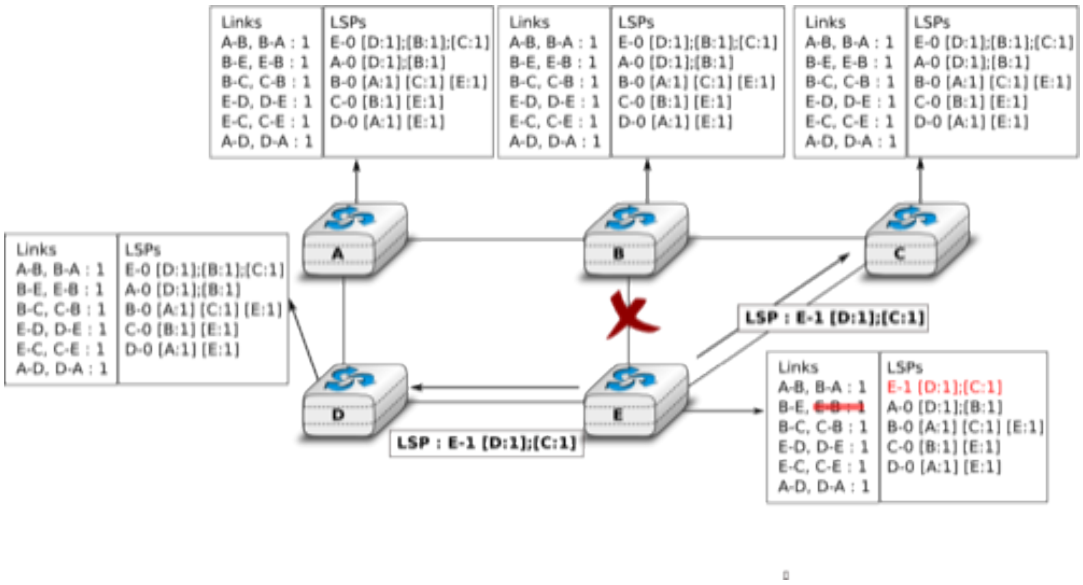

Рисунок 2 – Иллюстрация работы link-state алгоритма

с подкреплением, справляются с задачей адаптации к изменчивой нагрузке лучше.

1.2.3. Q-routing

Среди других подходов особый интерес представляют подходы на основе обучения с подкреплением. Первым алгоритмом маршрутизации, основанным на этой идее, стал алгоритм Q-routing[5]. Принцип его работы таков:

— Каждый маршрутизатор x хранит $Q_x(d, y)$ — оценку минимального времени в пути до узла d, если следующим узлом на пути является сосед y. Очевидно, что $\forall y : Q_x(x, y) = 0$

— Пакет, который необходимо доставить в узел d, отправляется соседу $y = \operatorname{argmin}_{(x,y)\in E} Q_x(d, y)$

— При получении пакета узел y отправляет узлу x время получения t_r и собственную оценку оставшегося времени в пути $t = \min\limits_{(y,z)\in E} Q_y(d, z)$

— Зная время отправления пакета t_s и получив t_r и t, узел x обновляет собственную оценку по формуле: $Q_x(d, y) = \alpha((t_r - t_s) + t - Q_x(d, y)) + Q_x(d, y)$, где α — это learning rate, параметр алгоритма.

Было показано, что этот алгоритм способен хорошо адаптироваться к изменениям в топологии сети и интенсивности трафика. Такие его

модификации, как dual Q-routing[8] и predictive Q-routing[7] демонстрируют еще более высокое качество маршрутизации. Однако по сравнению с distance-vector или link-state методами данные алгоритмы используют гораздо больше служебных сообщений (служебный пакет на каждую пересылку целевого пакета), что ограничивает их применение в реальных высоконагруженных компьютерных сетях.

Однако в задачах маршрутизации вне контекста компьютерных сетей это перестает быть проблемой, так как целевые “пакеты” (чемоданы на конвейере, автомобили на автостраде, etc.) и служебные сообщения в таких задачах являются объектами разной природы и передаются по разным каналам, причем служебные сообщения по сравнению с целевыми “пакетами” доставляются мгновенно. Эти обстоятельства делают применение алгоритмов обучения с подкреплением в таких задачах более привлекательным.

1.2.4. Другие подходы

Для полноты обзора приведем еще несколько примеров.

Идея использования нейросетей для решения задачи маршрутизации не нова. В работах [14, 15] для решения задачи поиска кратчайшего пути в графе используются нейронные сети Хопфилда. Однако, эти исследования преследовали цель ускорения вычисления кратчайшего пути за счет аппаратной реализации нейросети, что кардинально отличается от цели текущей работы.

Еще одним интересным подходом является AntNet[16]. Это алгоритм, построенный на идее исследования состояния сети с помощью специальных пакетов-“агентов”. Алгоритм показал хорошие результаты в ходе исследований, но не получил широкого применения, вероятно, в силу уже массового к тому времени распространения link-state и distance-vector протоколов.

1.2.5. Выводы по обзору существующих решений

Все алгоритмы маршрутизации, рассмотренные в статье, были разработаны для решения задачи маршрутизации именно в компьютерных сетях, но не для решения задачи маршрутизации в общей формулировке. Таким образом, есть потребность в разработке алгоритма, способного решать более общую задачу, что и является целью данной работы.

1.3. Применение обучения с подкреплением к задаче маршрутизации

1.3.1. Термины и понятия

Обучение с подкреплением (reinforcement learning) — вид машинного обучения, в котором *агент* (agent) каждый момент времени t взаимодействует со *средой* (environment), находящейся в *состоянии* (state) $s_t \in \mathcal{S}$ путем выбора *действия* (action) $a \in \mathcal{A}_{s_t}$ и получения *вознаграждения* (reward) $r_{t+1} \in \mathbb{R}$ с переходом в новое *состояние* s_{t+1}.

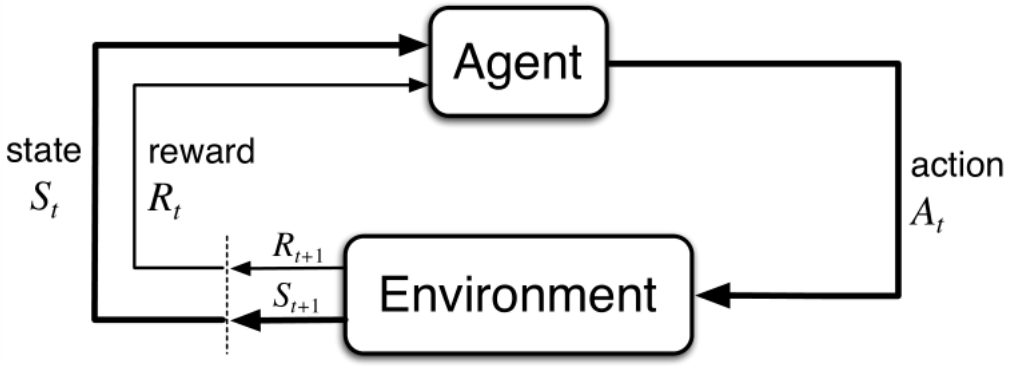

Рисунок 3 – Схема взаимодействия агента и среды в обучении с подкреплением

Марковский процесс принятия решений (Markov decision process, MDP) — это кортеж $(\mathcal{S}, \mathcal{A}_s, P, R, \gamma)$, где

- $\mathcal{S}$ — конечное множество состояний
- $\mathcal{A}_s$ — конечное множество действий, доступных из состояния s
- $P(s'|s,a)$ — вероятность того, что действие a в состоянии s приведет к переходу в состояние s' в следующий момент времени.
- $R : \mathcal{S} \times \mathcal{A}_s \to \mathbb{R}$ — вознаграждение за действия a в состоянии s
- $\gamma \in [0,1]$ — *скидочный коэффициент* (discount factor), управляющий соотношением между важностью текущих вознаграждений и будущих вознаграждений.

Оптимальной стратегией для данного Марковского процесса принятия решений называется такая функция выбора действий $\pi : \mathcal{S} \to \mathcal{A}_s$, что взвешенная сумма вознаграждений $\sum_{t=0}^{\infty} \gamma^t R_{\pi(s_t)}(s_t, s_{t+1})$ максимальна.

Частично наблюдаемый Марковский процесс принятия решений (Partially observed Markov decision process, POMDP) — это кортеж $(\mathcal{S}, \mathcal{A}_s, P, R, \Omega, O, \gamma)$, где

- $\mathcal{S}$ — конечное множество состояний
- $\mathcal{A}_s$ — конечное множество действий, доступных из состояния s
- $P(s'|s,a)$ — вероятность перехода из s в s' при выполнении действия a
- $R : \mathcal{S} \times \mathcal{A}_s \to \mathbb{R}$ — вознаграждение за действие a в состоянии s.
- Ω — множество *наблюдений*
- $O(o|s',a)$ — вероятность получения наблюдения o при переходе в истинное состояние s' в результате действия a.

Определение оптимальной стратегии для частично наблюдаемого Марковского процесса аналогично таковому для обычного.

Для Марковских процессов в условиях известности всех компонентов, включая P, R и O существуют детерминированные методы нахождения оптимальной стратегии. Однако найти оптимальную стратегию можно и в условиях неизвестности P, R и O.

Q-обучение (Q-learning)[6] — это метод нахождения оптимальной стратегии для Марковского процесса принятия решений, не требующий информации о функции R и распределении P. Метод заключается в оценке *функции полезности* (action-value function) $Q(s,a)$. Функция полезности изменяется при каждом предпринятом действии a с переходом из состояния s в s' по следующей формуле:

$$Q(s,a) = Q(s,a) + \alpha \left(r + \gamma \cdot \max_{a \in \mathcal{A}_{s'}} - Q(s',a) \right)$$

Известно, что для любого конечного Марковского процесса принятия решений Q-обучение находит оптимальную стратегию, т. е. $Q(s,a) \xrightarrow{t \to \infty} Q^*(s,a)$, и $\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}_s} Q^*(s,a)$ — оптимальная стратегия.

1.3.2. Формулировка задачи в терминах обучения с подкреплением

К сожалению, сформулировать задачу маршрутизации в виде Марковского процесса принятия решений довольно затруднительно. Попы-

таемся сформулировать ее глобально. Пусть множество состояний $\mathcal{S}$ — это Декартово произведение состояний всех ребер и вершин в графе. Пусть также в состояние каждого ребра (вершины) входит информация о всех пакетах, проходящих через это ребро (обрабатывающихся в этой вершине). Пусть множество действий, доступных из состояния $s \in \mathcal{S}$ $\mathcal{A}_s$ — это множетво пар вида (x,y), где $x \in V$ — это узел сети, имеющий в очереди пакет на обработку, а $y \in \{V|(x,y) \in E\}$ — один из его соседей. Таким образом, действие интерпретируется как "узел x пересылает свой текущий пакет соседу y".

К сожалению, у такой постановки задачи есть следующие проблемы:

а) Некоторые действия должны выполняться одновременно, а не одно за другим. Это можно формально обойти, сказав, что одновременно выполняющиеся действия $a_k, ..., a_{k+n}$ выполняются в последовательные моменты времени $t_k, ..., t_{k+n}$, либо изменив структуру множества действий $\mathcal{A}_s$: вместо пар (x,y) рассматривать списки таких пар.

б) В любом случае невозможно узнать вознаграждение за действие a_t сразу после перехода из состояния s_t в s_{t+1} — вознаграждение за посылку пакета p по ребру e в вершину v вычисляется как

$$-\sum_{t=t_k}^{t_{k+n}} R_e(e, s_e^t, p) - \sum_{t=t_{k+n+1}}^{t_{k+m}} R_v(v, s_v^t, p)$$

, где $t_k ... t_{k+n}$ — моменты времени, в которые пакет проходил через ребро e, а $t_{k+n+1} ... t_{k+m}$ — моменты времени, в которые пакет обрабатывался в узле v. (Суммы взяты с минусом, чтобы максимизация суммарного вознаграждения минимизировала суммарную стоимость путей пакетов). Это можно формально обойти, добавив в множество действий $\mathcal{A}_s$ действия вида a_e^p — "продолжить движение пакета p по ребру e" и a_v^p — "продолжить продвижение пакета p в очереди на обработку в узле v". Однако такую формальную модель будет крайне сложно как-то применить на практике — непонятно, как в какой-либо реальной задаче получить по отдельности стои-

мости вида $R_e(e, s_e^t, p)$ — стоимости "пребывания пакета p на ребре e, которое находится в состоянии s."

Как можно видеть, несмотря на то, что формально процесс маршрутизации в сети можно смоделировать как Марковский решающий процесс, оптимизация такой модели представляет собой неподъемную задачу.

Гораздо более логичным кажется рассмотреть задачу с точки зрения отдельного маршрутизатора как обучающегося агента. Используя подход *независимого Q-обучения* (independent Q-learning, IQL)[17], скажем, что маршрутизатор считает всю остальную часть сети как среду, с которой он взаимодействует. Безусловно, маршрутизатору недоступна исчерпывающая информация о состоянии всей сети, а только небольшая ее часть — свое собственное состояние, и, возможно, какая-то еще информация (о соединениях с соседями, о самих соседях, возможно, что-то еще). Таким образом, кажется, что работу такого агента можно смоделировать как частично наблюдаемый Марковский процесс. Однако и здесь мы встречаемся с проблемой не мгновенного получения вознаграждения — к тому времени, как агент узнает от среды (а именно — от соседа, которому он послал пакет) финальную стоимость прохождения пакета до соседа, он успеет сменить множество состояний. Более того, в такой постановке задачи состояние s_{t+1} почти никак не зависит от действия a_t. К тому же такая постановка мотивирует маршрутизатор все время посылать пакеты по ребру наименьшей стоимости, максимизируя таким образом свой собственный выигрыш, что явно не оптимизирует суммарную стоимость пути пакета.

Однако все встает на свои места, если мы сделаем формальный трюк и в качестве агента, взаимодействующего со средой, рассмотрим *пакет*, двигающийся в сети и стремящийся минимизировать суммарную стоимость своего пути. Пакет рассматривает всю сеть как среду, а наблюдаемое состояние маршрутизатора, в котором он находится, включая идентификатор этого маршрутизатора, как собственное *наблюдение* $o_t \in \Omega$. С точки зрения пакета, во время его движения по ребру ничего не происходит — внутри маршрутизатора пакет выбирает действие $a_t \in \mathcal{A}_o$ — одного из соседей, в которого нужно перейти, "мгновен-

но” перемещается в этого соседа с получением вознаграждения r_t, равного суммарной стоимости этого перемещения, взятого с отрицательным знаком, и получает новое наблюдение $o_{t+1} \in \Omega$ — состояние нового маршрутизатора.

Как можно видеть, в такой формулировке задача хорошо моделируется как частично наблюдаемый Марковский процесс. Для нахождения оптимальной стратегии в частично наблюдаемом Марковском процессе также можно использовать принцип Q-обучения. Несмотря на то, что в общем случае в POMDP Q-обучение не сходится к оптимальной стратегии, на практике это дает хорошие результаты.

Запишем формулу Q-обучения для поставленной задачи:

$$Q(o_t, a_t) = Q(o_t, a_t) + \alpha \left(r_t + \gamma \cdot \max_{a \in \mathcal{A}_{o_{t+1}}} Q(o_{t+1}, a) \right)$$

Заметим, что путь пакета конечен, и нас интересует оптимизация его полной стоимости. Поэтому в данной задаче будем считать, что $\gamma = 1$.

Как можно видеть, если наблюдение состоит только из идентификатора текущего узла, а вознаграждение является просто временем, потраченным пакетом на перемещение, эта формула вырождается в формулу, используемую алгоритмом Q-routing (1.2.3). Безусловно, на практике, как и в оригинальном алгоритме Q-routing, сам пакет не является агентом, вычисляющим собственную стратегию. Вместо этого на место каждого очередного пакета себя ставит маршрутизатор.

Как уже было замечено, в общем случае множество наблюдений Ω может быть очень большим или даже бесконечным, что обуславливает необходимость аппроксимации функции $Q(o, a)$. Для этого можно использовать нейросети. В следующей главе будут рассмотрены примеры применения нейросетей в задачах обучения с подкреплением.

1.4. Обзор методов обучения нейросетей с подкреплением

Активные исследования в области обучения с подкреплением с использованием нейросетей начались с публикации командой DeepMind алгоритма DQN[10]. Алгоритм показал способность эффективно обучаться игре в классические видеоигры на эмуляторе Atari 2600.

Алгоритм DQN базируется на методе Q-обучения. В качестве состояния нейросеть получает на вход текущее изображение игрового экрана. Набор действий соответствует возможному набору игровых действий.

Ключевой проблемой при использовании Q-обучения с нейросетями является то, что метод Q-обучения в применении к Марковским процессам с бесконечным числом состояний, вообще говоря, не сходится к оптимальной стратегии. Алгоритм DQN борется с этим обстоятельством с помощью experience replay — буфера из всех встреченных четверок (s, a, r, s'), где s — начальное состояние, a — действие, предпринятое в этом состоянии, r — полученное вознаграждение, s' — следующее состояние. Этот буфер работает как "память" нейросети: в каждый момент времени из буфера выбирается случайное подмножество встреченных ситуаций, и нейросеть заново обучается на них, "вспоминая" игровой опыт.

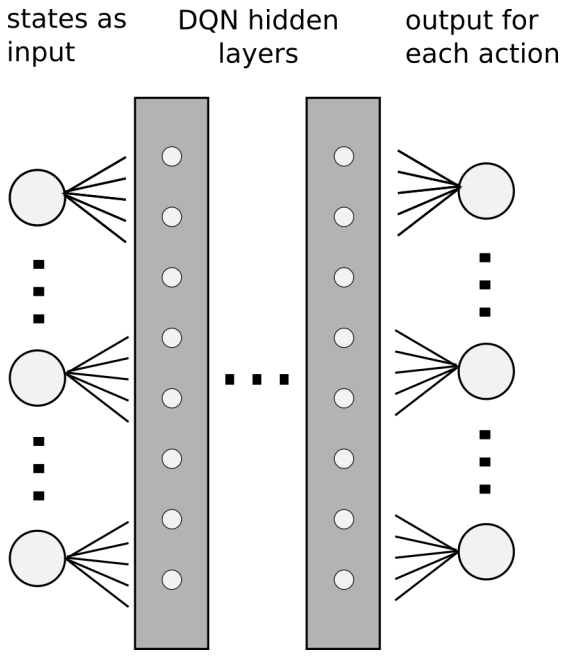

Рисунок 4 – Устройство нейросети для приближения Q-функции

TBD: закончить

Выводы по главе 1

В главе 1 были рассмотрены существующие алгоритмы маршрутизации, их преимущества и недостатки. Задача маршрутизации была сформулирована в терминах обучения с подкреплением и были намечен подход к ее решению.

ГЛАВА 2. ОПИСАНИЕ РАЗРАБОТАННОГО АЛГОРИТМА

Идея алгоритма заключается в объединении метода Q-routing1.2.3 с обучением нейросетей. Однако очевидно, что простая замена табличной Q-функции на нейросеть не даст положительных результатов. Чтобы использовать преимущества нейросети, будут применены такие подходы, как *предобучение* и *расширение наблюдаемого состояния*.

2.1. Используемая архитектура нейросети

Базовой частью любого текущего состояния s (или, точнее *наблюдения* $o \in \Omega$ будем считать кортеж $(n, d, y_1 ... y_m)$, где n — это идентификатор (номер) текущего узла, d — номер узла назначения текущего пакета, а $y_1 ... y_m$ — номера соседей текущего узла. Чтобы избежать взаимозависимости оценок Q-функции для узлов с близкими номерами, будем использовать *унитарный код*, т. е. для сети с семью узлами номер 3 будет кодироваться как 0010000, номер 5 — как 0000100, и т. д. Для кодирования множества соседей будем использовать тот же принцип, т. е. множество соседей 1, 3, 4 будет закодировано как 1011000. Таким образом, размер базового входного состояния для нейросети составит $3n$, где n — количество узлов в сети.

Нейросеть имеет 2 скрытых полносвязных слоя с ReLU активаторами. Выходной слой состоит из n нейронов с линейными функциями активации, где i-ый нейрон соответствует i-ому узлу в сети. К выходам узлов, не являющихся соседями текущего, отдельно прибавляется $-\inf$ (на практике — большое отрицательное число, например -1000000). Таким образом, на выходе нейросети образуются оценки функции $Q(s, a)$ для всех действий a, где для действий, недоступных в текущем состоянии (узлов, не являющихся соседями), $Q(s, a) = -\inf$.

TBD: добавить иллюстрации

2.2. Предобучение модели

Как было указано в разделе 1.3.2, мы будем моделировать задачу маршрутизации как частично наблюдаемый Марковский процесс принятия решений и рассматривать каждый пакет как независимого обучающегося агента, взаимодействующего со средой. Однако такая постановка задачи неизбежно ведет к нестационарности поведения среды по сразу двум причинам:

— Другие агенты, рассматриваемые как часть среды, тоже обучаются и меняют собственное поведение
— Меняется характер трафика пакетов в сети

Нестационарность среды приводит к тому, что использование experience replay не дает положительных результатов при обучении, так как прошлый опыт становится неактуальным в новых условиях. Это приводит к тому, что процесс Q-обучения перестает сходиться к оптимальному решению даже в случае равномерного распределения трафика и низкой нагрузки. Чтобы частично справиться с этой проблемой, будем использовать предварительное *обучение с учителем* (supervised learning). В качестве опорных данных будут использованы данные работы алгоритма кратчайших путей в условиях низкой нагрузки и равномерно распределенного трафика между узлами сети. В таких условиях алгоритм кратчайших путей является оптимальным.

Здесь стоит заметить, что, вообще говоря, не во всех вариантах постановки задачи маршрутизации можно вычислить стоимости прохождения пакета через ребра и узлы сети даже в условиях низкой стационарной нагрузки. В таких случаях будем довольствоваться некоторыми оптимистичными оценками на эти стоимости.

Благодаря тому, что мы используем номер текущего узла как часть базового состояния, мы можем предобучить одну нейросеть для работы во всех узлах сети. Это не только сократит время на предобучение, но и даст нам модель, которая имеет информацию о стратегии каждого из узлов в условиях низкой нагрузки, что мы сможем использовать в дальнейшем.

2.3. Расширение наблюдаемого состояния

Интуитивно понятно, что чем больше релевантной информации мы используем для оценки Q-функции, тем точнее в теории мы можем ее оценить. Очевидно, что информация о текущей топологии в сети довольно важна для точности оценки. Эту информацию мы можем эффективно передавать, используя link-state протокол.

Имея версию текущей топологии графа, закодируем ее как *матрицу смежности* и подадим на вход нейросети вместе с базовым состоянием. Таким образом мы получим возможность быстро реагировать на

обрывы и восстановление соединений. Эксперименты показывают, что предобученная нейросеть с матрицей смежности на входе и link-state протоколом работает в условиях низкой нагрузки и изменяющейся топологии почти аналогично link-state алгоритму кратчайших путей.

2.4. Использование softmax-стратегии

Алгоритмы обучения с подкреплением, особенно в нестационарных средах, сталкиваются с проблемой застревания в локальном максимуме. Она заключается в том, что алгоритм перестает менять свою стратегию, так как она “кажется” ему наиболее выгодной, и не пробует никакие другие стратегии, хотя они на деле могут оказаться выгоднее текущей. При решении задачи роутинга это чаще всего проявляется в неоптимальности поведения алгоритма при переходе от высокой нагрузки к низкой. Это является одним из основных недостатков оригинального алгоритма Q-routing.

Частично решить эту проблему поможет *softmax-стратегия*: выбор следующего действия не детерминированно, а случайно, однако с распределением вероятности, полученным применением к оценкам Q-функции функции *softmax*:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1 .. K$$

Это приводит к тому, что агент с некоторой вероятностью выбирает на очередном шаге не самое оптимальное с его точки зрения действие, а что-то чуть менее оптимальное, однако вероятность выбрать действие, оцениваемое как крайне неоптимальное, крайне мала. Это помогает бороться с застреванием в локальном максимуме.

TBD: придумать что-то поумнее

ГЛАВА 3. ЭКСПЕРИМЕНТЫ

3.1. Эксперименты в упрощенной модели сети

Эксперименты проводились на симуляции сети из 10 узлов. Служебные сообщения (такие как сообщения о вознаграждении от соседей) передаются по отдельному каналу, не зависящему от времени симуляции.

Нейросеть предварительно обучалась на эпизодах поведения роутера, использующего link-state протокол и алгоритм поиска кратчайшего пути (алгоритм Дейкстры). Всего в датасете порядка 250 тыс. эпизодов.

Работа предложенного алгоритма сравнивалась с работой link-state алгоритма и алгоритма Q-routing. Такой выбор кандидатов для сравнения обусловлен тем, что link-state алгоритмы являются доминирующими в сетевом роутинге, а алгоритм Q-routing является идейным предком нашего алгоритма.

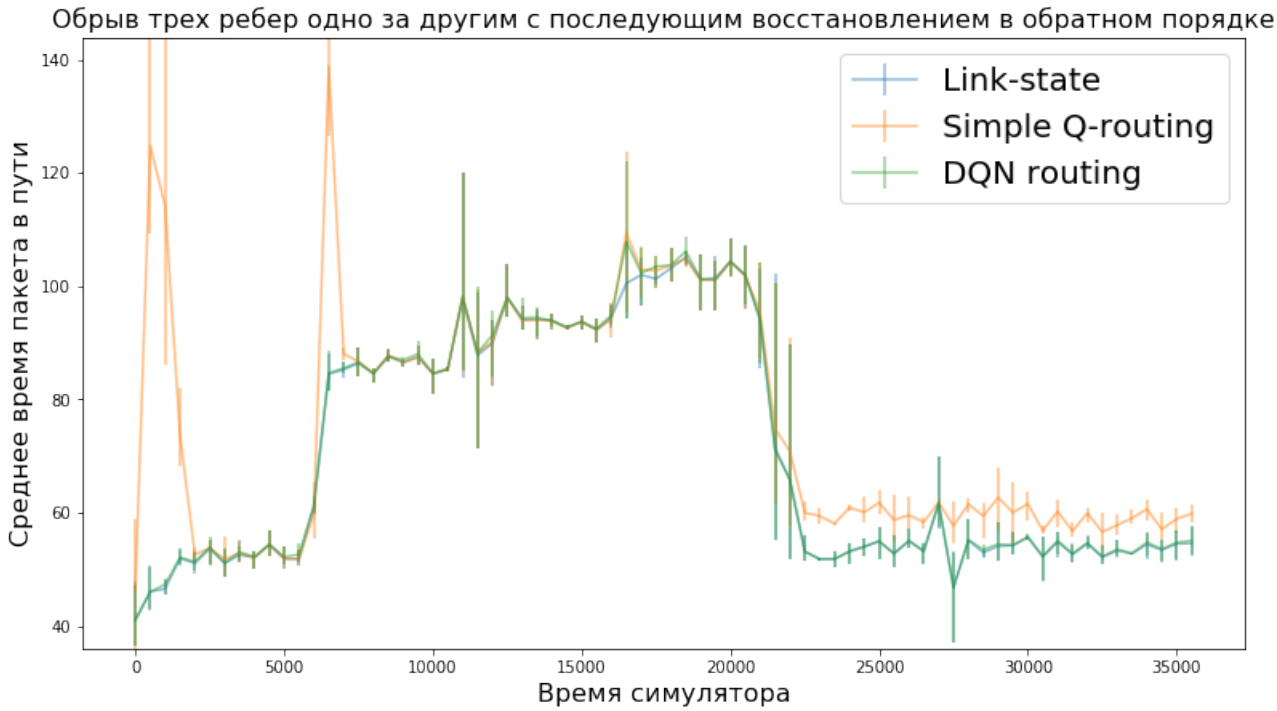

Рисунок 5 – Сценарий изменения топологии сети

Из результатов экспериментов видно, что предложенный алгоритм сильно превосходит link-state алгоритм в плане адаптивности под меняющуюся нагрузку, а также превосходит Q-routing как по показате-

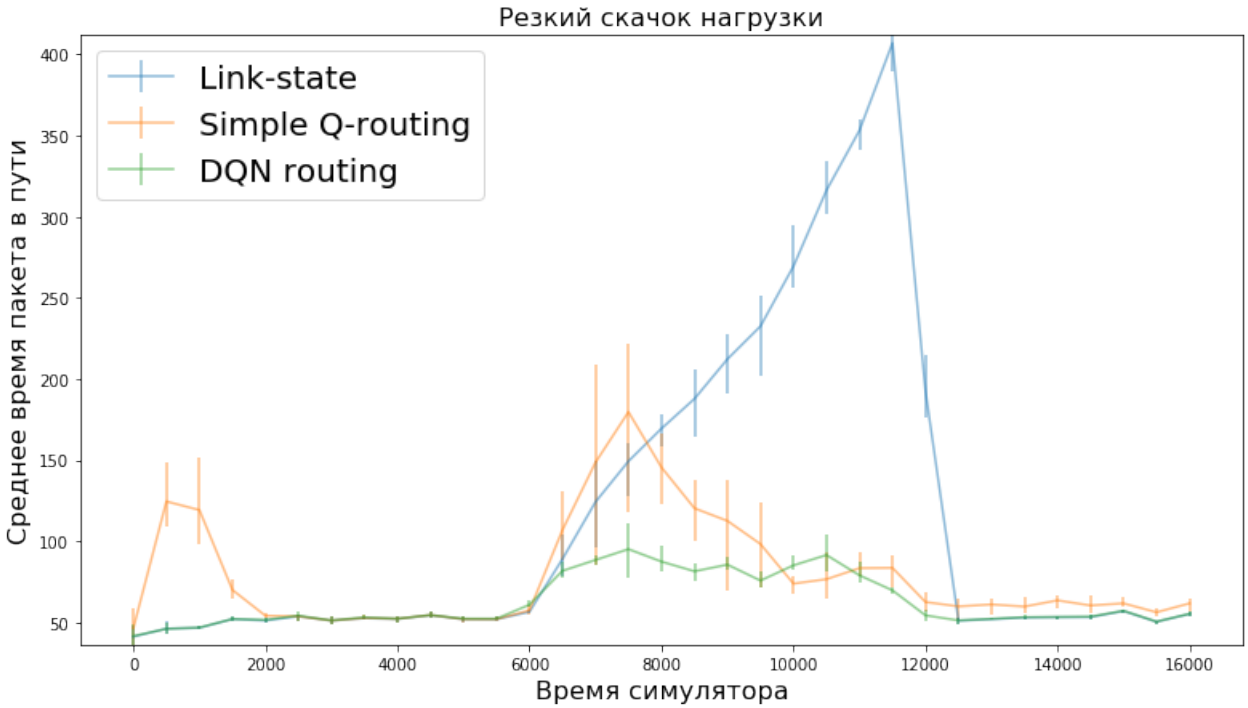

Рисунок 6 – Сценарий увеличения и уменьшения нагрузки

лям адаптивности под нагрузку, так и по показателям адаптивности под изменяющуюся топологию сети.

Дополнительно приведем график, показывающий плохое влияние использования experience replay на адаптивность к нагрузке.

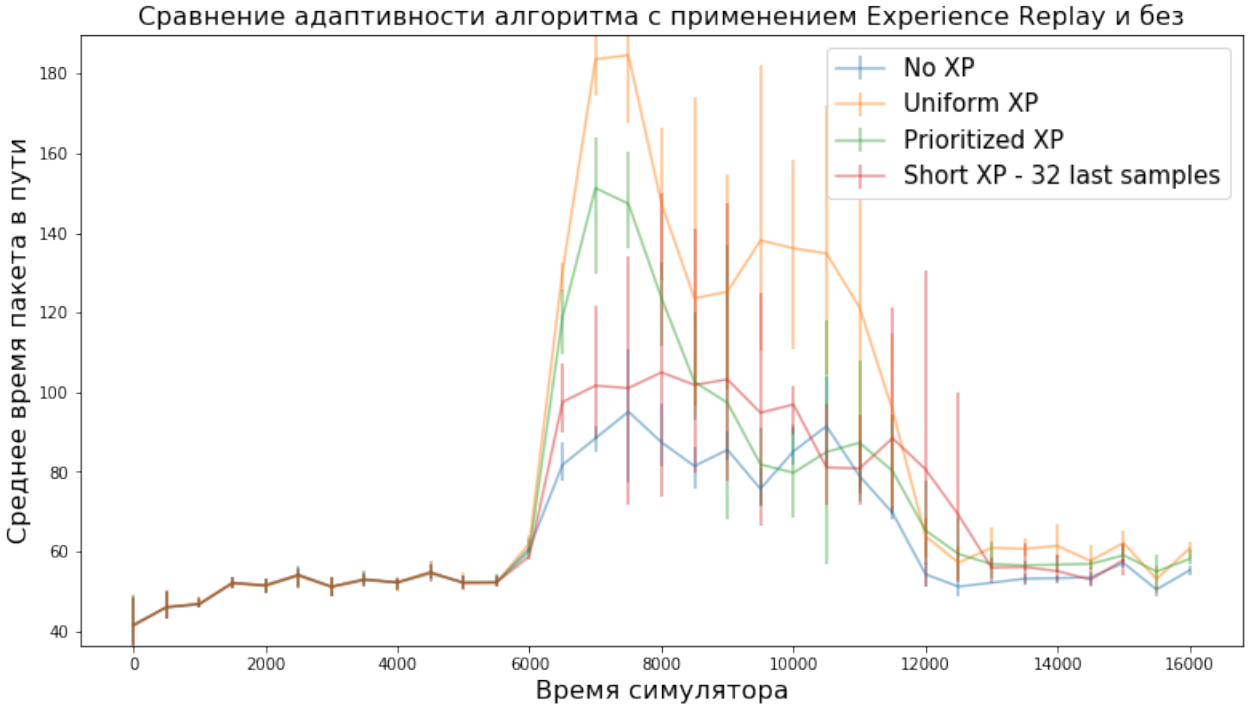

Рисунок 7 – Сравнение эффективности версий алгоритма с использованием различных модификаций experience replay

TBD: больше экспериментов

3.2. Эксперименты в модели системы транспортировки багажа

ЗАКЛЮЧЕНИЕ

TBD

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *McQuillan J. M.*, *Walden D. C.* The ARPA network design decisions // Computer Networks. — 1977. — Т. 1, № 5. — С. 243–289.

2 *McQuillan J. M.*, *Richer I.*, *Rosen E. C.* The New Routing Algorithm for the ARPANet // IEEE Trans. on Comm. — 1980. — Т. 28, № 5. — С. 711–719.

3 *Hendrick C.* Routing Information Protocol: RFC. — Июнь 1988. — № 1058.

4 *Moy J.* OSPF Version 2: RFC. — Апр. 1998. — № 1058.

5 *Boyan J. A.*, *Littman M. L.* Packet routing in dynamically changing networks: a reinforcement learning approach // Advances in Neural Information Processing Systems. — 1994. — No. 6. — P. 671–678.

6 *Watking C.* Learning from Delayed Rewards: дис. ... канд. / Watking C. — Cambridge : King's College, 1989.

7 *Choi S. P. . M.*, *Yeung D.-Y.* Predictive Q-Routing: A Memory-based Reinforcement Learning Approach to Adaptive Traffic Control // Advances in Neural Information Processing Systems. — 1996. — No. 8. — P. 945–951.

8 *Kumar S.*, *Miikkulainen R.* Dual reinforcement Q-routing: An on-line adaptive routing algorithm // Artificial Neural Networks in Engineering. — 1997. — No. 7. — P. 231–238.

9 *Yan J.*, *Vyatkin V.* Distributed Software Architecture Enabling Peer to Peer Communicating Controllers // IEEE Transactions on Industrial Informatics. — 2013. — Vol. 9, no. 4. — P. 2200–2209.

10 Human-level control through deep reinforcement learning / V. Mnih [et al.] // Nature. — 2015. — No. 518. — P. 529–533.

11 *Bellman R.* On a routing problem // Quarterly of Applied Mathematics. — 1958. — № 16. — С. 87–90.

12 *Bosack L.* Method and apparatus for routing communications among computer networks. — Февр. 1992. — URL: `https://www.google.com/patents/US5088032` ; US Patent 5,088,032.

13 *Dijkstra E. W.* A note on two problems in connexion with graphs // Numerische Mathematik. — 1959. — № 1. — С. 269–271.

14 *Ali M. K. M.*, *Kamoun F.* Neural networks for shortest path computation and routing in computer networks // IEEE Transactions on Neural Networks. — 1993. — Т. 4, № 6. — С. 941–953.

15 *Araujo F.*, *Ribeiro B.*, *Rodrigues L.* A neural network for shortest path computation // IEEE Transactions on Neural Networks. — 2001. — Т. 12, № 5. — С. 1067–1073.

16 *Di Caro G.*, *Dorigo M.* AntNet: Distributed stigmergetic control for communications networks // Journal of Artificial Intelligence Research. — 1998. — Т. 9. — С. 317–365.

17 *Tan M.* Multi-agent reinforcement learning: Independent vs. cooperative agents // Proceedings of the tenth international conference on machine learning. — 1993. — С. 330–337.